Auction Algorithms for Path Planning, Network Transport, and Reinforcement Learning

by

Dimitri P. Bertsekas

Chapter 4 Auction Algorithms for Network Transport

This monograph represents "work in progress," and will be periodically updated. It more than likely contains errors (hopefully not serious ones). Furthermore, the references to the literature are incomplete. Your comments and suggestions to the author at dbertsek@asu.edu are welcome.

October 15, 2022

4

Auction Algorithms for Network Transport

Contents

4.1. The ϵ-Relaxation Method . . . . . p. 2
4.1.1. Implementation Issues . . . . . p. 8
4.2. Auction Algorithms for Max-Flow . . . . . p. 9
4.2.1. Path Construction Algorithms . . . . . p. 11
4.2.2. An Improved Version of Path Construction . . . p. 15
4.2.3. The Auction Max-Flow Algorithm . . . . . p. 18
4.2.4. Efficient Implementation . . . . . p. 25
4.3. Auction/Sequential Shortest Path Algorithms . . . . p. 28
4.4. Convex Separable Network Optimization . . . . . p. 33
4.4.1. Convex Functions of a Single Variable . . . . . p. 34
4.4.2. Optimality Conditions . . . . . p. 37
4.4.3. Duality . . . . . p. 39
4.4.4. Auction Algorithms . . . . . p. 48
4.4.5. The ϵ-Relaxation Method . . . . . p. 57
4.4.3. Auction/Sequential Shortest Path Algorithm . . p. 61
4.5. Theoretical Aspects . . . . . p. 64
4.6. Notes and Sources . . . . . p. 74

In this chapter we discuss auction algorithms for the transhipment problem. The auction algorithms for transhipment that we will discuss in this chapter involve adaptations of the algorithmic ideas of Chapters 2 and 3 for assignment and shortest path problems, respectively. They are based on two general approaches:

(a) Transforming the transhipment problem to an equivalent assignment problem, using one of the methods discussed in Section 1.2, and then applying one of the assignment auction algorithms of Chapter 2.

(b) Using the path construction algorithms of Chapter 3 within the framework of classical primal dual/sequential shortest path methods for transhipment. The latter methods involve constructions of a large number of augmenting paths, which are similar and hence can benefit from one of the principal advantages of auction algorithms for path construction: reusing prices as a favorable initialization from one path construction to the next.

In Section 4.1, we discuss the ϵ-relaxation method for the transhipment problem, which is based on the first approach above. We then discuss auction algorithms based on the second approach above, first for the max-flow problem in Section 4.2, and then for the transhipment problem in Section 4.3. In Section 4.4, we extend the algorithms of Sections 4.1-4.3 to convex separable network optimization problems. In Section 4.5, we provide proofs of various propositions, and discuss some theoretical aspects of the algorithms of this chapter, including computational complexity estimates. The analysis of Section 4.5 requires some additional background on directed graphs, and associated notions of paths and flows. We have provided this background in the appendix to Chapter 1, and we will use it in the main body of the chapter.

4.1 THE ϵ-RELAXATION METHOD

In this section, we will introduce and analyze an auction algorithm for the transhipment problem

$$\begin{aligned} &\text{minimize} \quad \sum_{(i,j)\in\mathcal{A}} a_{ij}x_{ij} \\ &\text{subject to} \quad \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} = s_i, \qquad \forall\ i \in \mathcal{N}, \\ &\qquad\qquad b_{ij} \le x_{ij} \le c_{ij}, \qquad \forall\ (i,j) \in \mathcal{A}, \end{aligned} \tag{4.1}$$

where a_{ij}, b_{ij}, c_{ij}, and s_i are given. Throughout this section, we assume that *a_{ij}, b_{ij}, c_{ij}, and s_i are integer, and that the problem is feasible.*

The method of this section, called *ϵ-relaxation*, can be interpreted as a special case of the auction algorithm for the assignment problem; see

Sections 7.3.3 and 7.4 of the book [Ber98]. We will not go into the details of this equivalence here, but instead we will provide an intuitive interpretation of the algorithm.

Let us consider a graph that represents a network of cities connected with transportation links (i,j), along which persons can move at cost a_{ij} per person. We think of each node i as a city, of each arc (i,j) as a link between cities i and j, and of s_i as the (positive or negative) excess persons at city i. We want to bring the excess of each city to 0 by moving the persons between the cities while observing the capacity bounds b_{ij}, c_{ij} of the transportation links, while minimizing the total cost of the transfer.

Consider the possibility of accomplishing the minimum cost transfer by providing an economic incentive for persons to move between cities. In particular, suppose that we *charge each person a price p_i for being in city i*. Taking into account the transportation cost a_{ij} for crossing link (i,j), we stipulate that persons will move from city i to city j if $p_i > a_{ij} + p_j$, to the extent that the capacity of link (i,j) allows. For a given set of capacity feasible link flows x_{ij}, let

$$g_i = \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} - \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} \tag{4.2}$$

denote the corresponding excess of node i. In the spirit of the auction algorithm, we introduce a mechanism by which the price p_i tends to increase as $g_i > 0$. This in turn stimulates a movement of persons from higher price cities i to lower price cities j, taking into account the transportation cost a_{ij} and link capacities b_{ij}, c_{ij}, and also giving priority to cities that offer a most favorable price differential.

The ϵ-relaxation method provides a precise algorithmic mechanism for effecting the changes in flows in response to current prices. It relies on a notion of *ϵ-complementary slackness* (ϵ-CS for short), which extends the corresponding ϵ-CS notions for assignment and path planning problems, discussed in Chapters 2 and 3, respectively. For a given ϵ, we say that a capacity-feasible flow vector x and a price vector p satisfy ϵ-CS if

$$p_i - p_j \le a_{ij} + \epsilon \qquad \text{for all } (i,j) \in \mathcal{A} \text{ with } x_{ij} < c_{ij}, \tag{4.3}$$

$$p_i - p_j \ge a_{ij} - \epsilon \qquad \text{for all } (i,j) \in \mathcal{A} \text{ with } b_{ij} < x_{ij}, \tag{4.4}$$

(see Fig. 4.1.1). The usefulness of ϵ-CS is due in large measure to the following proposition, which generalizes a corresponding result for the assignment problem. The proposition relies on the integrality of the cost coefficients a_{ij}. Its proof is given in Section 4.5.

Proposition 4.1.1: If $\epsilon < 1/N$, where N is the number of nodes, x is feasible, and x and p satisfy ϵ-CS, then x is optimal for the transhipment problem (4.1).

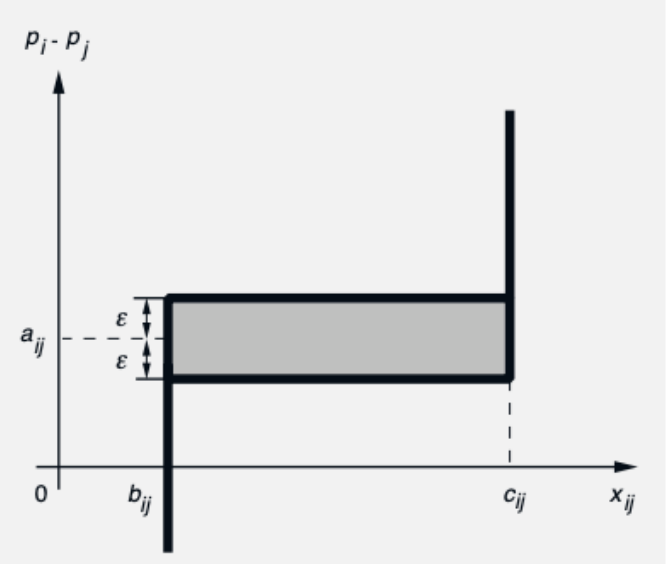

Figure 4.1.1: Illustration of ϵ-CS. All pairs of arc flows x_{ij} and price differences $p_i - p_j$ should either lie on the thick lines or in the shaded area between the thick lines.

In the ϵ-relaxation method, flows and prices are changed in a way that maintains ϵ-CS and tends to drive the nonzero node excesses towards zero. Furthermore, flow is allowed to change along certain types of arcs, which we now introduce. Given a flow-price pair (x, p) satisfying ϵ-CS, we say that an arc (i, j) is ϵ^+-*unblocked* if

$$p_i = p_j + a_{ij} + \epsilon \qquad \text{and} \qquad x_{ij} < c_{ij}.$$

We say that an arc (j, i) is ϵ^--*unblocked* if

$$p_i = p_j - a_{ji} + \epsilon \qquad \text{and} \qquad b_{ji} < x_{ji}.$$

The *candidate list* of a node i is the (possibly empty) set of outgoing arcs (i, j) that are ϵ^+- unblocked, and incoming arcs (j, i) that are ϵ^--unblocked.

We use a fixed positive value of ϵ, and we start with a pair (x, p) satisfying ϵ-CS. Furthermore, the starting arc flows are integer, and it will be seen that the integrality of the arc flows is preserved thanks to the integrality of the node supplies and the arc flow bounds. Implementations that have good worst case complexity also require that all initial arc flows be at either their upper or their lower bound, as will be explained later. This can be easily enforced.

At the start of a typical iteration we have a flow-price vector pair (x, p) satisfying ϵ-CS and we select a node i with $g_i > 0$; if no such node can be found, the algorithm terminates.

Iteration of the ϵ-Relaxation Method

Step 1: (Scan incident arc) If the candidate list of node i is empty, go to Step 4; else select from the candidate list of i either an arc (i,j) and go to Step 2, or an arc (j,i) and go to Step 3.

Step 2: (Push flow forward along arc (i,j)) Increase x_{ij} by $\delta = \min\{g_i, c_{ij} - x_{ij}\}$. If now $g_i = 0$ and $x_{ij} < c_{ij}$, stop; else go to Step 1.

Step 3: (Push flow backward along arc (j,i)) Decrease x_{ji} by $\delta = \min\{g_i, x_{ji} - b_{ji}\}$. If now $g_i = 0$ and $b_{ji} < x_{ji}$, stop; else go to Step 1.

Step 4: (Increase price of node i) Raise p_i to the level

$$\overline{p}_i = \min\Big\{ \min_{\{(i,j)\in\mathcal{A} \mid x_{ij} < c_{ij}\}} \big\{a_{ij} + p_j + \epsilon\big\}, \min_{\{(j,i)\in\mathcal{A} \mid b_{ji} < x_{ji}\}} \big\{p_j - a_{ji} + \epsilon\big\}\Big\}. \tag{4.5}$$

Go to Step 1.

To describe the algorithm in words, at each iteration, the selected node i reduces its excess g_i by pushing flow along its candidate list arcs. i.e., the ones that are ϵ^+-unblocked and ϵ^--unblocked. If after pushing all possible flow along these arcs, node i still has positive excess, it increases its price p_i to create new ϵ^+-unblocked and/or ϵ^--unblocked arcs, and corresponding candidate list. This is done as many times as necessary, until the excess g_i is reduced to 0.

Figure 4.1.2 illustrates the sequence of price rises in an ϵ-relaxation iteration, and shows how it can be interpreted as an approximate coordinate ascent or Gauss-Seidel relaxation iteration. This interpretation parallels the approximate coordinate descent interpretation of the mathematically equivalent auction algorithm (cf. Fig. 2.1.1).

To see that the iteration is well-defined in the sense that it stops after a finite number of computational operations, observe the following:

(a) Integrality of the arc flows is maintained by the algorithm, since the starting arc flows, the node supplies, and the arc flow bounds are integer. In particular, the flow increments δ in Steps 2 and 3 are integer throughout the algorithm.

(b) At most one flow change per incident arc of node i is performed at each iteration since a flow change either sets the flow to one of its bounds,

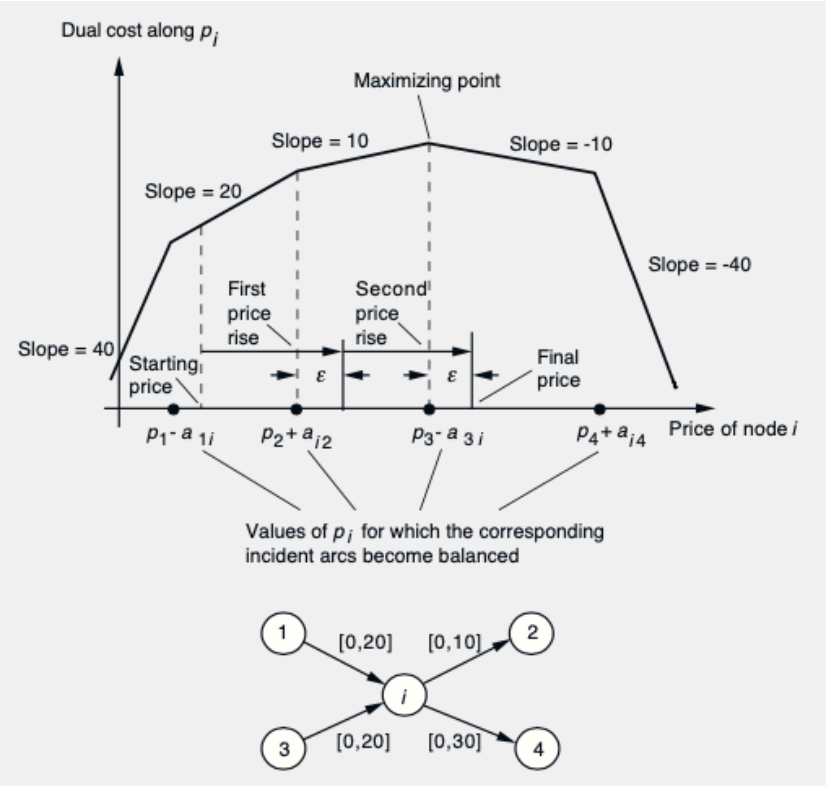

Figure 4.1.2: Illustration of the price rises of the ϵ-relaxation iteration. Here, node i has four incident arcs $(1,i)$, $(3,i)$, $(i,2)$, and $(i,4)$ with flow ranges $[0,20]$, $[0,20]$, $[0,10]$, and $[0,30]$, respectively, and supply $s_i = 0$. The arc costs and current prices are such that

$$p_1 - a_{1i} \leq p_2 + a_{i2} \leq p_3 - a_{3i} \leq p_4 + a_{i4},$$

as shown in the figure. The break points of the dual cost along the price p_i correspond to the values of p_i at which one or more incident arcs to node i become balanced. For values between two successive break points, there are no balanced arcs. Each price rise of the ϵ-relaxation iteration increases p_i to the point which is ϵ to the right of the next break point larger than p_i, (assuming that the starting price of node i is to the left of the maximizing point by more than ϵ). In the example of the figure, there are two price rises, the second of which sets p_i at the point which is ϵ to the right of the maximizing point, leading to the approximate (within ϵ) coordinate ascent interpretation.

which causes the corresponding arc to drop out of the candidate list of i through the end of the iteration, or else results in $g_i = 0$, which leads the iteration to branch to Step 4 and subsequently stop. Therefore, the number of flow changes per iteration is finite. In addition we have

$g_i > 0$ at the start and $g_i = 0$ at the end of an iteration, so at least one flow change must occur before an iteration can stop.

(c) After each price rise with $g_i > 0$ at least one flow change must be performed, so from (b) it follows that the number of price changes per iteration is finite.

Thus the method's iteration is guaranteed to stop after a finite number of operations.

There is, however, an exceptional situation in Step 4, which requires special handling. This is the case where in Eq. (4.5) we have $x_{ij} = c_{ij}$ for all outgoing arcs (i, j) and $b_{ji} = x_{ji}$ for all incoming arcs (j, i); that is, the cut separating i from the remainder of the graph is saturated, while $g_i \geq 0$. This can arise under two circumstances: (1) $g_i > 0$, in which case, the problem must be infeasible, or (2) $g_i = 0$. To deal with the situation, we stop the algorithm in case (1), and we keep p_i at its current level and stop the iteration in case (2).

The following proposition establishes the validity of the ϵ-relaxation method. The proof is given in Section 4.5.

Proposition 4.1.2: Assume that the transhipment problem is feasible and that a_{ij}, b_{ij}, c_{ij}, and s_i are integer, and that $s_i \geq 0$ for all i. Then the ϵ-relaxation method terminates with a pair (x, p) satisfying ϵ-CS. The flow vector x is feasible, and is optimal if $\epsilon < 1/N$.

Note the feasibility of the problem is a requirement for the algorithm to terminate. In practice, the algorithm may be supplemented with additional mechanisms to detect infeasibility, as will be discussed later in this section. Note also it is necessary to assume that the problem data is integer to assert optimality of the obtained solution with $\epsilon < 1/N$. A version of the method that can deal with noninteger data will be developed in Section 4.4, in the context of the more general convex separable network problem.

ϵ-Scaling

Let us now apply an ϵ-scaling approach to the ϵ-relaxation method. Similar to the case of the auction algorithms of Chapter 2, the idea is to use repeated applications of the method, called *scaling phases*, with progressively smaller values of ϵ. Each scaling phase uses price and flow information obtained from the preceding one. The kth scaling phase consists of applying the ϵ-relaxation method with $\epsilon = \epsilon^k$, where ϵ^k is updated by

$$\epsilon^{k+1} = \max\left\{\frac{\epsilon^k}{\theta}, \frac{1}{N+1}\right\}, \qquad k = 0, 1, \ldots,$$

where θ is an integer with $\theta > 1$. The first scaling phase is started with zero initial prices and an ϵ^0 that is a fixed fraction of the arc cost range $C = \max_{(i,j)\in\mathcal{A}} a_{ij}$. The total number of scaling phases is $\overline{k}$, which is the first positive integer k for which ϵ^{k-1} is equal to $1/(N+1)$. Thus the number of scaling phases is $O\big(\log(NC)\big)$.

With some special technical refinements to the algorithm, we can show that the $(k+1)$st scaling phase has a running time of $O(N^3)$. Since the number of scaling phases is $O\big(\log(NC)\big)$, we obtain the following proposition. The proof is given in Section 4.5.

Proposition 4.1.3: The running time of the ϵ-relaxation method using the sweep implementation and ϵ-scaling as described above is $O\big(N^3 \log(NC)\big)$.

4.1.1 Implementation Issues

The efficient implementation of the ϵ-relaxation method requires a number of techniques that while not suggested by the complexity analysis of Section 4.5, are essential for good practical performance.

Surplus Scaling

When applying ϵ-scaling, except for the last scaling phase, it is not essential to reduce the excesses of all nodes to zero; it is possible to terminate a scaling phase prematurely, and reduce ϵ further, in an effort to economize on computation. A technique that is typically quite effective is to iterate only on nodes whose excess exceeds some threshold, which is gradually reduced to zero with each scaling phase. The threshold is usually set by some heuristic scheme.

Negative Surplus Node Iterations

It is possible to define a symmetric form of the ϵ-relaxation iteration that starts from a node with negative excess and decreases (rather than increases) the price of that node. Furthermore, one can mix positive excess and negative excess iterations in the same algorithm; this is analogous to the combined forward/reverse auction algorithm for assignment and the forward/reverse auction algorithm for shortest paths. However, if the two types of iterations are mixed arbitrarily, the algorithm is not guaranteed to terminate even for a feasible problem; for an example, see Bertsekas and Tsitsiklis [1989], p. 373. For this reason, some care must be exercised in mixing the two types of iterations in order to guarantee that the algorithm eventually makes progress.

Dealing with Infeasibility

The issues and methods relating to infeasibility are similar to those discussed in Section 2.1, in connection with the assignment problem. One possibility is to monitor infeasibility by checking the price levels. If the problem is infeasible, the ϵ-relaxation method will either terminate with $g_i \leq 0$ for all i and $g_i < 0$ for at least one i, in which case infeasibility will be detected, or else it will perform an infinite number of iterations and, consequently, an infinite number of flow pushes and price rises. In the latter case, it can be shown that the prices of some of the nodes will diverge to infinity. This can be used to detect infeasibility. For further discussion we refer to the book [Ber98].

4.2 AN AUCTION ALGORITHM FOR MAX-FLOW

In this section, we consider the classical max-flow problem, where we are given a directed graph $(\mathcal{N}, \mathcal{A})$, and we want to push a maximum amount of flow from a source node 1 to a sink node N, subject to the constraint that the flow of each arc $(i,j) \in \mathcal{A}$ should lie in an interval $[0, c_{ij}]$, where c_{ij} is a given positive scalar, called the *capacity* of (i,j).

Here the number of nodes is N and the nodes are denoted $1, 2, \ldots, N$. To facilitate the presentation we assume that there is at most one arc (i,j) starting at i and ending at j, so that we can unambiguously refer to an arc as (i,j). A flow vector $x = \{x_{ij} \mid (i,j) \in \mathcal{A}\}$ is said to be *capacity feasible* if $0 \leq x_{ij} \leq c_{ij}$ for all $(i,j) \in \mathcal{A}$. The associated *surplus* of each node is defined by

$$g_i = \sum_{\{j \mid (j,i) \in \mathcal{A}\}} x_{ji} - \sum_{\{j \mid (i,j) \in \mathcal{A}\}} x_{ij}, \qquad \forall\ i \in \mathcal{N}. \tag{4.6}$$

The flow vector is said to be *feasible* if it is capacity feasible and the node surpluses satisfy

$$g_i = 0, \qquad \forall\ i \in \mathcal{N},\ i \neq 1,\ i \neq\ N. \tag{4.7}$$

The problem is to find a feasible flow such that g_N is maximized.

The most popular algorithms for the max-flow problem consist of a sequence of flow increases along paths from the source node to the sink node, also called *augmenting paths*. In the most efficient of these methods, the augmenting paths are shortest, in the sense that they consist of a minimum number of arcs. We discuss an auction algorithm for the max-flow problem, whereby the augmenting paths are constructed by one of the auction/path construction algorithms of Chapter 3. While, in the auction algorithm the

augmenting paths are not necessarily shortest, they typically can be found with much less computation than the shortest augmenting paths used by competing methods. Moreover, the path computations benefit from the reuse of the auction algorithm prices. For these reasons, the algorithm has excellent performance and outperforms other competing methods by a very large margin in tests with standard randomly generated problems.

More specifically, the classical approach to the max-flow problem is the Ford-Fulkerson algorithm [FoF56], which consists of successive augmentations; it sequentially moves flow from the source to the sink along augmenting paths, until a saturated cut separating the source and the sink is created. In its original form, this algorithm had two drawbacks:

(a) If the augmenting paths are arbitrarily constructed, the number of augmentations can be very large. In fact if the arc capacities are irrational, the algorithm may fail to terminate (see e.g. [FoF62], [PaS82], [Ber91a]).

(b) No mechanism is provided to pass helpful information from one augmenting path construction to the next.

These two drawbacks have been addressed by much subsequent research. The traditional approach to keep the number of augmentations small is to ensure that the augmenting paths are shortest, in the sense that they contain the smallest possible number of arcs. In fact all polynomial augmenting path methods that we are aware of use this approach. The simplest way to construct the shortest augmenting paths is to use a breadth-first search method, leading to an $O(NA^2)$ running time [EdK72], where A is the number of arcs. In order to reuse information from one shortest augmenting path construction to the next, the idea of a layered network implementation was also suggested [Din70], and resulted in an $O(N^2A)$ running time.

The algorithm of this section is of the Ford-Fulkerson type, but do not use shortest augmenting paths. Instead it constructs (possibly nonshortest) augmenting paths using the ideas of the auction algorithm for the assignment problem [Ber79], [Ber91a], [Ber92a]. In particular, our path construction algorithm is obtained by converting the path construction problem to a special type of unweighted matching problem, applying the auction algorithms of Section 3.2, and streamlining the computations. A key feature here is that the price mechanism of the auction algorithm is used to pass valuable information from one augmenting path construction to the next.

Another relevant class of max-flow algorithms is the class of preflow-push methods, which originated with the work of [Kar74], [ShV82], and has been the subject of much subsequent development [Gol85], [GoT86], [AhO89], [AMO89], [ChM89], [DeM89], [MPS91]. These methods move flow along single-arc paths, and they share with the auction algorithm the idea of using a price mechanism (within this context, prices are also

called labels). This connection is not accidental, and in fact it is shown in [Ber94] that a generic preflow-push method for the max-flow problem [GoT86] can be derived as a special case of the auction algorithm for the assignment problem, using the reformulation of the max-flow problem as an assignment problem. Preflow-push methods have excellent theoretical worst-case complexity [$O\big(N^2A^{1/2}\big)$ with relatively simple implementation [ChM89], and even better through the use of sophisticated but somewhat impractical data structures].

Our algorithm of this section has an $O\big(N^2A\big)$ worst-case running time, but according to our experiments, it is substantially faster than both shortest augmenting path and preflow-push methods. There is a two-fold explanation for this. First, the auction algorithm solves simpler path construction problems than the competing shortest augmenting path methods, while at the same time it passes useful price information from one path construction to the next. Second, because flow changes take place over multiple-arc paths, the phenomenon of ping-ponging of flow between pairs of nodes that is characteristic of preflow-push methods is largely avoided. Indeed experiments have shown that the number of arc flow changes required to solve the problem is generally far smaller in our method than in preflow-push methods.

4.2.1 Path Construction Algorithms

In this section we describe a method for finding a path between two nodes of a graph. This method lies at the heart of our max-flow algorithm, which will be presented in the next section. We give two versions of the algorithm. The first is simple and easy to understand. The second is a more complex variation of the first, but is apparently more efficient in practice. We first introduce some terminology.

Given the directed graph $(\mathcal{N}, \mathcal{A})$, the set of arcs outgoing from node i is denoted by $A(i)$ and the corresponding set of nodes $\{j \mid (i,j) \in A(i)\}$ is denoted by $N(i)$. We will use the extended definition of a path, described in the appendix to Chapter 1, which involves both forward and backward arcs. In particular, in this section, a *path* P is a sequence of nodes $(n_1, n_2, \ldots, n_t)$ with $t \geq 2$, and a corresponding sequence of $t-1$ arcs such that the ith arc in the sequence is either (n_i, n_{i+1}) (in which case it is called a *forward* arc of the path) or (n_{i+1}, n_i) (in which case it is called a *backward* arc of the path). Node n_1 is called the *start node* of P and node n_t is called the *terminal node* of P. By slight abuse of terminology, we consider $P = (n_1)$ to be a path, in which case n_1 is both the start and the terminal node of P. For $i = 2, \ldots, t$, the node n_{i-1} is called the *predecessor* of n_i, and is denoted by $pred(n_i)$.

We denote by P^+ and P^- the sets of forward and backward arcs of P, respectively. The path P is said to be *forward* if all its arcs are forward. The path P is said to be *simple* if it contains no cycles, that is, if the nodes

$n_1, \ldots, n_t$ are distinct. In the context of the max-flow problem, the *length* of a path is the number of its arcs. All paths in this subsection will be forward paths. The paths to be considered in the context of the max-flow problem, starting with the next subsection, may contain both forward and backward arcs.

The following algorithm aims at finding a simple forward path that starts at a given node n_1 and ends at node N. It maintains a simple forward path $P = (n_1, \ldots, n_t)$ and a set of integer node prices satisfying

$$p(i) \leq p(j) + 1, \qquad \forall\ (i,j) \in \mathcal{A}, \tag{4.8}$$

$$p(n_1) < N, \qquad p(N) = 0, \tag{4.9}$$

$$p(i) \geq p(j), \qquad \forall\ (i,j) \in P. \tag{4.10}$$

The conditions (4.8) and (4.10) are related to the ϵ-complementary slackness conditions with $\epsilon = 1$ and arc costs equal to 0.

The algorithm is motivated by the max-flow context, where the objective is not to find a single path, but rather to find a sequence of paths each in a graph that differs slightly from its predecessor. Within this context, prices are helpful in guiding the search for new paths. Loosely speaking, prices are modified by the algorithm in a way that the desired paths have an approximate downhill direction, that is, they proceed from high price nodes to low price nodes. Thus, if a set of prices is roughly appropriate for guiding the search for a path in a given graph, it is also roughly appropriate for guiding the search for a path in a slightly different graph.

At the start of the algorithm, we require that $P = (n_1)$, and that p is such that Eqs. (4.8) and (4.9) hold. The path P is modified repeatedly using the following two operations:

(a) A *contraction* of P, which deletes the last arc of P, that is, replaces the path $P = (n_1, \ldots, n_t)$ by the path $P = (n_1, \ldots, n_{t-1})$. [In the degenerate case where $P = (n_1)$, a contraction leaves P unchanged.]

(b) An *extension* of P, which adds to P an arc outgoing from its terminal node, that is, replaces the path $P = (n_1, \ldots, n_t)$ by a path $P = (n_1, \ldots, n_t, n_{t+1})$, where (n_t, n_{t+1}) is an arc.

The prices $p(i)$ may also be increased in the course of the algorithm so that, together with P, they satisfy the conditions (4.8)(4.10). A contraction always involves a price increase of the terminal node n_t. An extension may or may not involve such a price increase. An extension of P is always done to a neighbor node of n_t that has minimal price. The algorithm terminates if either node N becomes the terminal node of P (then P is the desired path), or else $p(n_1) \geq N$ [in view of $p(N) = 0$ and $p(i) \leq p(j) + 1$ for all arcs (i,j) as per Eqs. (4.8) and (4.9), this means that there is no forward path from n_1 to N].

Path Construction Algorithm

Set $P = (n_1)$, and select p such that Eqs. (4.8) and (4.9) hold.

Step 1 (Check for contraction or extension): Let n_t be the terminal node of P. If the set $N(n_t)$ is empty, set $p(n_t) = N$ and go to Step 3. Otherwise, find a node in $N(n_t)$ with minimal price and denote it $succ(n_t)$,

$$succ(n_t) = \arg \min_{j \in N(n_t)} p(j). \tag{4.11}$$

Set

$$p(n_t) = p\big(succ(n_t)\big) + 1. \tag{4.12}$$

If $n_t = n_1$, or if

$$n_t \neq n_1 \qquad \text{and} \qquad p\big(pred(n_t)\big) > p\big(succ(n_t)\big), \tag{4.13}$$

go to Step 2; otherwise go to Step 3.

Step 2 (Extend path): Extend P by node $succ(n_t)$ and the corresponding arc of $A(n_t)$. If $succ(n_t) = N$, terminate the algorithm; otherwise go to Step 1.

Step 3 (Contract path): If $P = (n_1)$ and $p(n_1) \geq N$, terminate the algorithm; otherwise, contract P and go to Step 1.

We note, that maintaining a path that is extended or contracted at each iteration, while maintaining a price vector that satisfies complementary slackness conditions, is a central feature of the auction algorithm for shortest paths [Ber91a], [Ber91b], and its embedding in a sequential shortest path algorithm for the minimum cost flow problem [Ber92b]. However, as mentioned earlier, our path construction algorithm does not necessarily generate a shortest path. Instead, we show later that it just solves a special type of unweighted matching problem by means of the auction algorithm.

In the special case where all initial prices are zero and there is a path from each node to N, by tracing the steps, it can be seen that the algorithm will work like depth-first search, raising to 1 the prices of the nodes of some path from n_1 to N in a sequence of extensions with no intervening contractions. More generally, the algorithm terminates without performing any contractions if the initial prices satisfy $p(i) \geq p(j)$ for all arcs (i, j) and there is a path from each node to N.

We make the following observations:

(1) The prices remain integer throughout the algorithm [cf. Eq. (4.12)].

(2) The conditions (4.8)-(4.10) are satisfied each time Step 1 is entered. The proof is by induction. These conditions hold initially by assumption. Condition (4.9) is maintained by the algorithm, since we have termination as soon as $p(n_1) \geq N$. To verify conditions (4.8) and (4.10), we note that only the price of n_t can change in Step 1, and by Eqs. (4.11) and (4.12), this price change maintains condition (4.8) for all arcs, and condition (4.10) for all arcs of P, except possibly for the arc $\big(pred(n_t), n_t\big)$ in the case of an extension with the condition $p\big(pred(n_t)\big) > p\big(succ(n_t)\big)$ holding. In the latter case, we must have $p\big(pred(n_t)\big) \geq p\big(succ(n_t)\big) + 1$ because the prices are integer, so by Eq. (4.12), we have $p\big(pred(n_t)\big) \geq p(n_t)$ at the next entry to Step 1. This completes the induction.

(3) A contraction is always accompanied by a price increase. Indeed by Eq. (4.10), which was just established, upon entering Step 1 with $n_t \neq n_1$, we have $p(n_t) \leq p\big(pred(n_t)\big)$, and to perform a contraction, we must have $p\big(pred(n_t)\big) \leq p\big(succ(n_t)\big)$. Hence $p(n_t) \leq p\big(succ(n_t)\big)$, implying by Eq. (4.12) that $p(n_t)$ must be increased to the level $p\big(succ(n_t)\big) + 1$. It can be seen, however, by example, that an extension may or may not be accompanied by a price increase.

(4) Upon return to Step 1 following an extension, the terminal node n_t satisfies [cf. Eq. (4.12)]

$$p\big(pred(n_t)\big) = p(n_t) + 1.$$

This, together with the condition $p(i) \geq p(j)$ for all $(i, j) \in P$ [cf. Eq. (4.10)], implies that the path P will not be extended to a node that already belongs to P. Thus P remains a simple path throughout the algorithm.

To facilitate the presentation, let us introduce some additional terminology. For a given integer price vector p, we say that an arc (i, j) is *uphill* if $p(i) < p(j)$, *downhill* if $p(i) \geq p(j)$, and *strictly downhill* if $p(i) = p(j)+1$. The following proposition summarizes the conclusions of the preceding discussion and establishes the termination properties of the algorithm.

Proposition 4.2.1:

(a) Throughout the algorithm, the prices satisfy the conditions (4.8) and (4.9), the path P is simple, its arcs are downhill, and following an extension, the last arc of P is strictly downhill.

(b) If there exists a forward path from n_1 to N, the algorithm terminates via Step 2 with such a path. Otherwise, the algorithm terminates via Step 3.

Proof: Part (a) was established above, so we prove part (b). We first note that the prices of the nodes of P are upper bounded by N in view of Eqs. (4.9) and (4.10). Next we observe that there is a price change of at least one unit with each contraction, and since the prices of the nodes of P are upper bounded by N, there can be only a finite number of contractions. Since there can be at most $N-1$ successive extensions without a contraction, the algorithm must terminate. Since, throughout the algorithm, $p(N) = 0$ and the condition $p(i) \leq p(j) + 1$ holds for all arcs (i, j), the existence of a forward path starting at a node n_1 and ending at N implies that $p(n_1) < N$ throughout the algorithm. Therefore, if termination occurs via Step 3, there cannot exist a path from n_1 to N. **Q.E.D.**

4.2.2 An Improved Version of Path Construction

Most of the calculation in the preceding algorithm is needed to determine the nodes $succ(n_t)$ attaining the minimum in Eq. (4.11) of Step 1. On the other hand, typically some of these nodes and the corresponding arcs do not change frequently during the algorithm. Thus it makes sense to save them in a data structure and try to reuse them as much as is possible without affecting the essential properties of the algorithm [maintaining conditions (4.8)-(4.10) and precluding the formation of a cycle within the path P]. This leads to a modification of the algorithm, where in addition to the price p, we maintain for each node $i \neq N$, a subset of outgoing arcs of i denoted $Cand(i)$, and called the *candidate* set of arcs of node i. The set of end nodes of arcs in $Cand(i)$ which are opposite to i is denoted $Succ(i)$.

The sets of arcs $Cand(i)$ together with the set of prices $p(i)$, define a graph, called the *admissible graph*, whose node set is $\mathcal{N} = \{1, \ldots, N\}$ and arc set is

$$\big\{(i, j) \mid j \in Succ(i),\, p(i) \geq p(j),\, i = 1, \ldots, N\big\}.$$

As the sets $Succ(i)$ and the prices $p(i)$ change in the course of the algorithm, the admissible graph also changes. We require that the initial sets $Cand(i)$ and prices $p(i)$ are such that the admissible graph is acyclic. This condition is satisfied in particular if we select the sets $Cand(i)$ to be empty. The algorithm is as follows:

Path Construction Algorithm: Second Version

Set $P = (n_1)$, and select p such that Eqs. (4.8) and (4.9) hold.

Step 1 (Check for contraction or extension): Let n_t be the terminal node of P. If there is a node $\overline{j} \in Succ(n_t)$ such that

$$p(n_t) \geq p(\overline{j}), \tag{4.14}$$

select such a node $\overline{j}$ and go to Step 2. Otherwise, if the set $N(n_t)$ is empty, set $p(n_t) = N$ and go to Step 3; otherwise set

$$Succ(n_t) = \left\{ \overline{j} \mid p(\overline{j}) = \min_{j \in N(n_t)} p(j) \right\}, \tag{4.15}$$

$$Cand(n_t) = \big\{ (n_t, \overline{j}) \in A(n_t) \mid \overline{j} \in Succ(n_t) \big\}, \tag{4.16}$$

and select a node $\overline{j} \in Succ(n_t)$. Set

$$p(n_t) = p(\overline{j}) + 1. \tag{4.17}$$

If $n_t = n_1$, or if

$$n_t \neq n_1 \qquad \text{and} \qquad p\big(pred(n_t)\big) > p(\overline{j}), \tag{4.18}$$

go to Step 2; otherwise go to Step 3.

Step 2 (Extend path): Extend P by node $\overline{j}$ and the corresponding arc of $Cand(n_t)$. If $\overline{j} = N$, terminate the algorithm, and otherwise go to Step 1.

Step 3 (Contract path): If $P = (n_1)$ and $p(n_1) \geq N$, terminate the algorithm; otherwise, contract P and go to Step 1.

Note that, similar to the first version of the algorithm, each contraction is accompanied by an increase of the price $p(n_t)$, while each extension may or may not be accompanied by an increase of $p(n_t)$. Note also that if the "downhill test" $p(n_t) \geq p(\overline{j})$ of Eq. (4.14) were to be replaced by the "strictly downhill test" $p(n_t) = p(\overline{j}) + 1$, the two versions of the algorithm would have been essentially identical [the sets $Cand(i)$ would just provide a specific implementation of the successor node selection of Eq. (4.11)]. However, because of the difference in the test for making an extension to a node of $Succ(n_t)$, the two versions of the algorithm are not mathematically equivalent. In particular, in the second version we perform an extension

when upon entering Step 1, we have $p(n_t) = p(\overline{j})$ for some $\overline{j} \in Succ(n_t)$, in which case the last arc of the path P is not strictly downhill following the extension. For this reason it is not obvious that an extension will not create a cycle in P with an attendant breakdown of the algorithm.

It turns out, however, that such a cycle cannot be closed because it can be proved that throughout the algorithm:

(a) The arcs of P belong to the admissible graph.

(b) The admissible graph remains acyclic.

Both of these properties can be shown by induction. In particular, property (a) is maintained because a contraction that deletes the terminal arc of P does not affect the prices of the end nodes of the other arcs of P. Furthermore, each extension is done along an arc of $Cand(n_t)$ and whether the test (4.14) is passed or $p(n_t)$ is set via Eq. (4.17), this arc is downhill and its predecessor arc continues to be downhill following the extension. Also, to show that property (b) is maintained, suppose that property (b) holds at the start of Step 1, and consider the two cases where a node $\overline{j} \in Cand(n_t)$ satisfying the downhill test (4.14) can be found, and cannot be found. In the first case, the admissible graph remains unchanged. In the second case, the only potentially new arcs of the admissible graph are the arcs of the set $Cand(n_t)$, after this set is recalculated. However, following the price setting of Eq. (4.17), all the arcs of $Cand(n_t)$ are strictly downhill, so these arcs cannot be part of a cycle of the admissible graph, all the arcs of which are downhill by definition. Thus the admissible graph remains acyclic following Step 2 or 3, which shows that P remains a simple path at all times. We have the following proposition.

Proposition 4.2.2: Assume that the initial admissible graph is acyclic. Then:

(a) Throughout the algorithm, the admissible graph remains acyclic.

(b) The flow-price pairs generated by the algorithm satisfy the conditions (4.8) and (4.9), the path P is simple, and the arcs of P are downhill.

(c) If there exists a path from n_1 to N, the algorithm terminates via Step 2 with such a path. Otherwise, the algorithm terminates via Step 3.

Proof: Part (a) was shown above and the remaining parts are proved similar to the corresponding parts of Prop. 4.2.1. **Q.E.D.**

4.2.3 The Auction/Max-Flow Algorithm

We now consider the max-flow problem. We introduce some additional terminology.

Given a capacity feasible flow vector x, for each node i, we introduce the set of *eligible arcs* of i

$$A(i,x) = \big\{(i,j) \mid x_{ij} < c_{ij}\big\} \cup \big\{(j,i) \mid 0 < x_{ji}\big\},$$

and the corresponding set of *eligible neighbors* of i

$$N(i,x) = \big\{j \mid (i,j) \in A(i,x) \text{ or } (j,i) \in A(i,x)\big\}.$$

The *reduced graph* is the graph with node set $\mathcal{N}$ which contains an arc (i,j) if and only if j is an eligible neighbor of i. Thus eligible arcs of a node i in the original graph correspond to outgoing arcs from i in the reduced graph. For a given capacity feasible x, a path P in the original graph is said to be *unblocked* if it corresponds to a forward path of the reduced graph, that is, if $x_{ij} < c_{ij}$ for all forward arcs $(i,j) \in P^+$ and $0 < x_{ij}$ for all backward arcs $(i,j) \in P^-$. An unblocked path is said to be *augmenting* if its start node has positive surplus and its terminal node is the sink N. If P is an augmenting path, an *augmentation* is an operation that increases the flow of all arcs $(i,j) \in P^+$ and decreases the flow of all arcs $(i,j) \in P^-$ by a common increment $\delta > 0$.

Following standard terminology, a *cut* is a partition $\big(\mathcal{N}^+, \mathcal{N}^-\big)$ of the set of nodes $\mathcal{N}$ into two subsets $\mathcal{N}^+$ and $\mathcal{N}^-$ with $1 \in \mathcal{N}^+$ and $N \in \mathcal{N}^-$. The *capacity* of this cut is the sum of the capacities of all arcs (i,j) with $i \in \mathcal{N}^+$ and $j \in \mathcal{N}^-$. The max flow-min cut theorem states that the maximum flow is equal to the minimal cut capacity. For a given flow vector x, a cut $\big(\mathcal{N}^+, \mathcal{N}^-\big)$ is said to be *saturated* if $x_{ij} = c_{ij}$ for all arcs (i,j) with $i \in \mathcal{N}^+$ and $j \in \mathcal{N}^-$, and $x_{ij} = 0$ for all arcs (i,j) with $i \in \mathcal{N}^-$ and $j \in \mathcal{N}^+$. The algorithm of this section terminates with a capacity feasible flow vector x and a cut $\big(\mathcal{N}^+, \mathcal{N}^-\big)$ that is saturated and is such that the surpluses g_i, given by Eq. (4.6), satisfy

$$g_1 \leq 0, \qquad g_i \geq 0, \quad \forall\, i \neq 1, \qquad g_i = 0, \quad \forall\, i \in \mathcal{N}^-,\ i \neq N.$$

It is well known that such a cut is a minimum cut, and we will show how it can be used together with x to obtain a maximum flow (see the remarks following the proof of Prop. 3, which also prove that the cut obtained upon termination is minimum).

A capacity feasible flow vector x together with a price vector $p = \{p(i) \mid i \in \mathcal{N}\}$ are said to be a *valid pair* if

$$p(i) \leq p(j) + 1, \qquad \forall\ j \text{ that are eligible neighbors of } i. \tag{4.19}$$

Our algorithm starts with and maintains a valid flow-price pair (x, p) such that

$$g_1 \leq 0, \qquad g_i \geq 0, \quad \forall\, i \neq 1,$$

$$p(1) = N, \qquad p(N) = 0, \qquad p(i) \geq 0, \quad \forall\, i \neq 1,\, N. \tag{4.20}$$

A possible initial choice is the flow vector x given by

$$x_{ij} = \begin{cases} c_{ij} & \text{if } i = 1 \\ 0 & \text{if } i \neq 1 \end{cases} \tag{4.21}$$

together with the price vector p given by

$$p(i) = \begin{cases} N & \text{if } i = 1 \\ \text{length of a shortest unblocked path from } i \text{ to } N & \text{if } i \neq 1 \end{cases} \tag{4.22}$$

which can be obtained by a breadth-first search starting from N. (If there is no forward path of the original graph from i to N, the above length is taken to be equal to N.)

Our algorithm maintains a flow-price pair (x, p) satisfying the conditions (4.19)-(4.20), performs a sequence of iterations, and terminates with a minimum cut. At the start of each iteration, a node n_1 with $n_1 \neq N$, $p(n_1) < N$, and $g_{n_1} > 0$ is selected. The iteration tries to construct an augmenting path starting at n_1 by using the second path construction algorithm of the preceding section, applied to the reduced graph and using the price vector p. If an augmenting path is found, the iteration concludes with a corresponding augmentation. If an augmenting path cannot be found, the path construction algorithm terminates with $p(n_1) \geq N$, so that node n_1 will not be chosen as the starting node at any subsequent iteration. Consistently with the second path construction algorithm of Section 2, we maintain for each node i, a set of incident arcs of i denoted $Cand(i)$. The set $Cand(i)$ is empty for $i = 1$, $i = N$, and all i with $p(i) = N$. The set of end nodes of $Cand(i)$ which are opposite to i is denoted $Succ(i)$.

We require that initially, we have

$$p(i) = p(j) + 1, \qquad \text{if} \qquad j \in Succ(i),$$

which will be true if all sets $Cand(i)$ are empty, or for all i we have

$$Cand(i) = \big\{(i,j) \in A(i,x) \mid p(i) = p(j)+1\big\} \cup \big\{(j,i) \in A(i,x) \mid p(i) = p(j)+1\big\}, \tag{4.23}$$

where (x, p) is the initial flow-price pair. If the shortest path initialization of Eqs. (4.21)-(4.22) is used, then $Cand(i)$ as given by Eq. (4.23), is the set of arcs outgoing from i in a shortest augmenting path from i. The typical iteration is as follows:

Typical Iteration of the Auction/Max-Flow Algorithm

Select a node n_1 with $n_1 \neq N$, $p(n_1) < N$, and $g_{n_1} > 0$ (if no such node exists, the algorithm terminates). Set $P = (n_1)$.

Step 1 (Check for contraction or extension): Let n_t be the terminal node of P. If there is a node $\overline{j} \in Succ(n_t) \cap N(n_t, x)$ such that

$$p(n_t) \geq p(\overline{j}), \tag{4.24}$$

select such a node $\overline{j}$ and go to Step 2. Otherwise, if the set $N(n_t, x)$ is empty, set $p(n_t) = N$ and go to Step 3; otherwise set

$$Succ(n_t) = \left\{ \overline{j} \mid p(\overline{j}) = \min_{j \in N(n_t, x)} p(j) \right\}, \tag{4.25}$$

$$\begin{aligned} Cand(n_t) = {} & \big\{(n_t, j) \in A(n_t, x) \mid j \in Succ(n_t)\big\} \\ & \cup \big\{(j, n_t) \in A(n_t, x) \mid j \in Succ(n_t)\big\}, \end{aligned} \tag{4.26}$$

and select a node $\overline{j} \in Succ(n_t)$. Set

$$p(n_t) = p(\overline{j}) + 1. \tag{4.27}$$

If $n_t = n_1$, or if

$$n_t \neq n_1 \qquad \text{and} \qquad p\big(pred(n_t)\big) > p(\overline{j}),$$

go to Step 2; otherwise go to Step 3.

Step 2 (Extend path): Extend P by the node $\overline{j}$ and the corresponding arc of $Cand(n_t)$. If $\overline{j}$ is the sink N, go to Step 4, and otherwise go to Step 1.

Step 3 (Contract path): If $P = (n_1)$ and $p(n_1) \geq N$, terminate the iteration; otherwise, contract P and go to Step 1.

Step 4 (Augmentation): Perform an augmentation along P with flow increment

$$\delta = \min\big\{g_{n_1},\, \{c_{ij} - x_{ij} \mid (i,j) \in P^+\},\, \{x_{ij} \mid (i,j) \in P^-\}\big\}, \tag{4.28}$$

and terminate the iteration.

Note that, except for the at most $N - 2$ contractions in which $p(n_t)$ is set to N, all contractions involve an increase of the price $p(n_t)$ and a

recalculation of the set $Succ(n_t)$. Extensions can either occur through a discovery of a node $\overline{j} \in Succ(n_t) \cap N(n_t, x)$ such that $p(n_t) \geq p(\overline{j})$, or through a recalculation of the set $Succ(n_t)$, in which case an increase of $p(n_t)$ may or may not occur.

We assume that the search through the set $Succ(n_t) \cap N(n_t, x)$ for a node $\overline{j}$ such that $p(n_t) \geq p(\overline{j})$ is organized as follows: When a set $Cand(i)$ is initially calculated, via for example Eq. (4.23), or is recalculated via Eq. (4.26), it is organized as a queue, which allows the deletion of its top element with $O(1)$ work. Each iteration is started by sequentially retrieving arcs from the top of $Cand(n_t)$ and checking to see if these arcs are eligible and their endnode $\overline{j}$ opposite to n_t satisfies $p(n_t) \geq p(\overline{j})$. Each arc not passing these tests is deleted from $Cand(n_t)$, and the checking is stopped when either a node $\overline{j}$ with the required properties is found or the set $Cand(n_t)$ becomes empty. To simplify the following complexity accounting, the work for checking and deleting the arcs of $Cand(n_t)$ is lumped into the work for calculating $Cand(n_t)$. With this convention, the work involved in an extension for which we recalculate the set $Cand(n_t)$ via Eq. (4.26) is proportional to the degree of n_t, while the work involved in an extension where after checking and possibly deleting enough arcs of $Cand(n_t)$, we find an eligible neighbor node $\overline{j}$ that passes the test $p(n_t) \geq p(\overline{j})$ is $O(1)$. Similarly, the work involved in a contraction is proportional to the degree of n_t.

The next proposition establishes the basic properties of the algorithm:

Proposition 4.2.3: The following hold for the max-flow algorithm of this section:

(a) Each iteration of the max-flow algorithm up to the discovery of the corresponding augmenting path, consists of an application of the second path construction algorithm of the preceding section to the reduced graph, with the start node of the path being the chosen node n_1 for this iteration, and the end node of the path being N.

(b) The algorithm terminates and upon termination, there is a saturated cut separating the sink from all nodes with nonzero surplus, which is a minimum capacity cut.

(c) The running time of the algorithm is $O(N^2A)$.

Proof: (a) By comparing the descriptions of the second path construction algorithm and the iteration of the max-flow algorithm, we see that the condition (4.8) that is maintained by the path construction algorithm is equivalent to the condition (4.19) for the pair (x, p) to be valid, the price

change (4.17)corresponds to the price change (4.27), and the downhill test (4.14) for an extension corresponds to the downhill test (4.24). Let us define the *admissible graph of the max-flow algorithm* as the graph whose node set is $\mathcal{N} = \{1, \ldots, N\}$ and arc set is

$$\big\{(i,j) \mid j \in Succ(i) \cap N(i,x),\, p(i) \geq p(j),\, i = 1, \ldots, N\big\}.$$

Then the sets $Succ(i)$ and the admissible graph of the path construction algorithm correspond to the sets $Succ(i) \cap N(i,x)$ and the admissible graph in the max-flow algorithm, respectively.

Based on the preceding associations, it is seen that if at the start of an iteration of the max-flow algorithm the admissible graph is acyclic, then the iteration up to the discovery of an augmenting path is equivalent to the application of the path construction algorithm to the reduced graph. Thus, to prove the result we must show that the admissible graph of the max-flow algorithm remains acyclic throughout the algorithm.

To this end, we note that, in view of the initial restriction $p(i) = p(j) + 1$ for all $j \in Succ(i)$, the admissible graph is acyclic at the start of the algorithm. Furthermore, if the admissible graph is acyclic at the start of an iteration, the same is true during the iteration up to the discovery of the augmenting path, since the path construction algorithm maintains the acyclicity of the admissible graph. We claim that an augmentation does not add any new arcs to the admissible graph, and thus maintains its acyclicity. Indeed, suppose that an augmentation occurs along the path $(i_1, i_2, \ldots, i_k, N)$, and that one of the arcs (i_m, i_{m-1}), $m = 2, \ldots, k$, is added to the reduced graph and to the admissible graph as a result of the augmentation. Then, we must have $p(i_m) \geq p(i_{m-1})$, $i_{m-1} \in Succ(i_m)$ [by the definition of the admissible graph], and also $p(i_{m-1}) \geq p(i_m)$, $i_m \in Succ(i_{m-1})$ [since the arc (i_{m-1}, i_m) belongs to the augmenting path], so that $p(i_{m-1}) = p(i_m)$. This implies that $p(i_{m-1})$ and $p(i_m)$ have been increased at least once since the start of the algorithm [since we have $p(i) = p(j) + 1$ for all $j \in Succ(i)$ at the start of the algorithm and also following each recalculation of the set $Succ(i)$]. Furthermore, the conditions $p(i_{m-1}) < p(i_m) + 1$ and $i_m \in Succ(i_{m-1})$ imply that the last increase of $p(i_m)$ occured after the last recalculation of $Succ(i_{m-1})$ [since following a recalculation of $Succ(i)$ at a node i, we have $p(i) = p(j) + 1$ for all $j \in Succ(i)$]. Therefore the last increase of $p(i_m)$ occured after the last increase of $p(i_{m-1})$ [since each increase of $p(i)$ involves a recalculation of $Succ(i)$]. Similarly, the conditions $p(i_m) < p(i_{m-1}) + 1$ and $i_{m-1} \in Succ(i_m)$ imply that the opposite is true. We thus reach a contradiction.

(b) From part (a) and Prop. 4.2.2, it follows that each iteration terminates. At the end of an iteration, either we have $p(n_1) \geq N$, indicating that there is no augmenting path starting at n_1, or we have an augmentation. In the former case the number of nodes i with $p(i) \geq N$ increases strictly, so there

can be at most $N-2$ iterations of this type. To show that the number of augmentations is finite, we first note that there are at most N price increases per node, since prices take nonnegative integer values, and once the price of a node exceeds $N-1$, it increases no further. We next observe that each augmentation either exhausts the surplus of n_1, or saturates at least one arc (that is, it drives the flow of the arc to zero or its upper bound). When an arc with end nodes i and j is saturated in the direction from i to j, there are two possibilities: (1) $p(i) = p(j)+1$, or (2) $p(i) = p(j)$, in which case in view of $j \in Succ(i)$, we cannot have $i \in Succ(j)$, since this would violate the acyclicity of the admissible graph. In either case (1) or case (2), we see that one of the at most N increases of $p(j)$ must occur before this arc can become unsaturated and then saturated again in the direction from i to j. Thus the number of arc saturations is $O(N)$ per arc, and the total number of arc saturations is $O(NA)$, leading to an $O(NA)$ bound in the number of iterations and the number of augmentations.

We thus see that the algorithm terminates, and since augmentations preserve the condition $g_i \geq 0$ for all $i \neq 1$, upon termination, we must have $g_i \geq 0$ for all $i \neq 1$, $p(1) = N$, $p(i) \geq N$ for all $i \neq N$ with $g_i > 0$, and $p(N) = 0$. It follows that there can be no augmenting path starting at node 1 or at a node i with $g_i > 0$, implying that there is a saturated cut $\big(\mathcal{N}^+, \mathcal{N}^-\big)$ such that $1 \in \mathcal{N}^+$, $N \in \mathcal{N}^-$, $g_i \geq 0$ for all $i \neq 1$, and $g_i = 0$ for all $i \neq N$ with $i \in \mathcal{N}^-$. As discussed earlier, this is a minimum cut.

(c) We first note that as shown in the proof of part (b):

(1) There are at most N price increases per node.

(2) There are at most $O(NA)$ iterations and at most $O(NA)$ augmentations.

In view of (1) above, there can be at most N contractions and extensions that involve a price increase at each node, and the work for each is proportional to the degree of n_t. Thus the work for these contractions and extensions is $O(NA)$. Also, since each augmentation involves a flow change for each of at most $N-1$ arcs, the work for augmentations is $O(N^2A)$.

There remains to bound the work for extensions that do not involve a price increase. We argue by contradiction that each such extension does not involve the recalculation of the set $Succ(n_t)$, that is, either it involves the first calculation of $Succ(n_t)$ or the downhill test (4.24) is failed for all $\overline{j} \in Succ(n_t) \cap N(n_t, x)$. Indeed suppose that the set $Succ(n_t)$ is recalculated via Eq. (4.25) and we find that $p(n_t) = p(j) + 1$ for all $j \in Succ(n_t)$, so that an extension is performed without an increase of $p(n_t)$. Then, every $j \in Succ(n_t)$ must have been an eligible neighbor of n_t and its price must have remained unchanged continuously since the preceding time $Succ(n_t)$ was calculated [and $p(n_t)$ was set to $p(j) + 1$]. But this is a contradiction, since in order for $Succ(n_t)$ to be recalculated, all nodes j in the set $Succ(n_t) \cap N(n_t, x)$ must satisfy $p(n_t) < p(j)$. Thus if an extension at

n_t does not involve a price increase, it also does not involve a recalculation of $Succ(n_t)$, and therefore (using the accounting method described in the paragraph preceding Prop. 4.2.3) it requires only $O(1)$ work, unless it involves the calculation of $Succ(n_t)$ for the first time. Now the total number of extensions is $O(N^2A)$ because in each iteration, the number of extensions exceeds the number of contractions by at most $N-1$, the total number of contractions in the entire algorithm is $O(N^2)$, while the total number of iterations is $O(NA)$. Thus the total work for extensions that do not involve a price increase is $O(N^2A)$. **Q.E.D.**

Given the cut $(\mathcal{N}^+, \mathcal{N}^-)$ and the flow vector x obtained upon termination of the algorithm, we can obtain a maximum flow by applying the same algorithm to a certain feasibility problem, that aims to return to the source the excess flow that has entered the graph from the source and has accumulated at the other nodes of $\mathcal{N}^+$. In particular, we delete all nodes in $\mathcal{N}^-$ and all arcs with at least one endnode in $\mathcal{N}^-$, and for each node $i \neq 1$ with $i \in \mathcal{N}^+$ and

$$\sum_{\{(i,j)|j\in\mathcal{N}^-\}} c_{ij} > 0,$$

we introduce an arc $(i,1)$ with flow and capacity

$$x_{i1} = c_{i1} = \sum_{\{(i,j)|j\in\mathcal{N}^-\}} c_{ij} \tag{4.29}$$

[if the arc $(i,1)$ already exists, we just change its capacity and flow to the above value]. In the resulting graph, call it $\mathcal{G}$, we pose the problem of finding a flow vector $\overline{x}$ such that the corresponding surpluses are all zero. It can be seen that the surpluses corresponding to the flow vector x restricted to $\mathcal{G}$ are equal to the nonnegative surpluses g_i obtained upon termination for all $i \neq 1$. We can thus apply the max-flow algorithm of this section starting with this flow vector, and the prices

$$p(i) = \begin{cases} 0 & \text{if } i = 1, \\ \text{length of a shortest unblocked path from } i \text{ to } 1 & \text{if } i \neq 1, \end{cases}$$

which together with x form a valid pair for the graph $\mathcal{G}$. It can be shown then that each iteration of the algorithm will terminate with an augmentation from some node i with $g_i > 0$ to the source 1. [Given any capacity feasible flow vector in a graph with arc capacities, and a node i with positive surplus, there is always an augmenting path starting at i and ending at some node with negative surplus; this follows from the conformal realization theorem (see e.g. [Ber91a], p. 7). Here node 1 is the only node with negative surplus.] Thus the algorithm will terminate when the surpluses of all the nodes $i \neq 1$ will be reduced to 0, while upon termination the flows of the arcs $(i,1)$ will still be equal to their initial values given by Eq.

(4.29), since these arcs cannot participate in an augmenting path. If $\overline{x}_{ij}$ is the final flow of each arc (i,j) of $\mathcal{G}$, it can be seen, using also the fact $g_i = 0$ for all $i \in \mathcal{N}^-$ with $i \neq N$, that the flow vector x^* defined for each arc $(i,j) \in \mathcal{A}$ by

$$x^*_{ij} = \begin{cases} \overline{x}_{ij} & \text{if } i \notin \mathcal{N}^-,\ j \notin \mathcal{N}^-, \\ x_{ij} & \text{otherwise,} \end{cases}$$

will have surpluses g^*_i satisfying $g^*_i = 0$ for all $i \neq 1, N$, $g^*_1 < 0$, $g^*_N > 0$, while saturating the cut $(\mathcal{N}^+, \mathcal{N}^-)$. Thus, by the max flow-min cut theorem, x^* must be a maximum flow and $(\mathcal{N}^+, \mathcal{N}^-)$ must be a minimum cut.

Note from the proof of Prop. 3 that the complexity bottleneck is the $O(N^2A)$ bound for augmentations and for extensions that do not involve a price increase. Our computational experience, however, indicates that the $O(NA)$ work for price increases is at least as much of a bottleneck. This is similar to preflow-push methods where the $O(NA)$ work for price increases usually dominates the computation, even though the worst case complexity bound is worse than $O(NA)$. It thus appears that the practical computation bottlenecks are comparable for preflow-push methods and our method.

We finally note two variants of the max-flow algorithm. In the first variant, we use the first version of the path construction algorithm, given in Section 2, in place of the second version. The statement of the typical iteration of this algorithm is identical with the one given above, except that the downhill test $p(n_t) \geq p(\overline{j})$ of Eq. (4.24) is replaced by the strictly downhill $p(n_t) = p(\overline{j}) + 1$. Proposition 3 can also be proved for this variant of the algorithm using a similar (in fact simpler) proof.

In the second variant of the max-flow algorithm, instead of maintaining the entire set $Cand(i)$, we maintain just one arc of $Cand(i)$. The iteration of the algorithm is modified so that if the unique arc of $Cand(n_t)$ passes the downhill test of Eq. (4.24), it is used as earlier. Otherwise [assuming $N(n_t, x)$ is nonempty] the set $Succ(n_t)$ is computed and a single arc of $Cand(n_t)$ is retained. This variant can be shown to terminate with a minimum cut as stated in Prop. 3. Its complexity analysis is similar to the one given in the proof of Prop. 3(c), except that the work for extensions that do not involve a price increase can be estimated as $O(NA^2)$ rather than $O(N^2A)$, raising the complexity bound to $O(NA^2)$. However, when combined with the second best data structure given in the next section, this second variant of the max-flow algorithm proved the most effective in our computational results.

4.2.4 Efficient Implementation

In this section we describe a number of variations of the auction/max-flow algorithm of the preceding section, which we have empirically found to improve performance.

Tests for a Saturated Cut

It has been observed that for some problems (particularly those involving a sparse graph), our method can create a saturated cut very quickly and may then spend a great deal of additional time to raise to the level N the prices of the nodes that are left with positive surplus. This characteristic is shared with preflow-push methods. Computational studies [DeM89], [MPS91], [AnS93], [NgV93] of preflow-push methods have shown that it is extremely important to use a procedure that detects early the presence of a saturated cut. Several schemes have been suggested in the literature.

One possibility is to test periodically for a saturated cut by an $O(A)$ breadth-first search from the sink, which tries to find the set $\mathcal{S}$ of nodes from which there is an unblocked path to the sink. If all nodes in $\mathcal{S}$ have zero surplus, then $\mathcal{S}$ defines a minimum cut. Note that once a node of $\mathcal{S}$ with positive surplus is found, the breadth-first search can be terminated. However, in an alternative version of this scheme, one can also perform *global repricing*, whereby all the nodes in $\mathcal{S}$ are obtained, and their prices are recalculated and are set to their shortest distances from the sink. Furthermore, all the nodes not in $\mathcal{S}$ can effectively be purged from the computation by setting their price equal to N. While global repricing can be costly, it is known to be beneficial for several problem types [MPS91], [AnS93], [NgV93]. It is important to use an appropriate heuristic scheme that ensures that global repricing is not too frequent, in view of the associated overhead. In practice, repeating the test after a number of contractions, which is of the order of N, seems to work well.

Another possibility, suggested in the context of preflow-push methods in [DeM89], is to maintain in a suitable data structure, for each integer k in the range $[1, N-1]$, the number of nodes $m(k)$ whose price is equal to k. If for some k we have $m(k) = 0$ (this is called a *gap at price* k), then there is a saturated cut separating all nodes with price greater than k from all nodes whose price is less than k. All the nodes with price greater than k can effectively be purged from the computation by setting their price equal to N. Furthermore, if all nodes with price less than k have zero surplus, the separating saturated cut is a minimum cut. In our experiments, we have found this second procedure in conjunction with the highest price selection rule to be more effective than the first. Note an advantage of both of these procedures: they can purge from the computation a significant number of nodes before finding a minimum cut.

Method for Selecting the Starting Node of the Path

Our algorithm leaves unspecified the choice of the positive surplus node used as the starting node of the path P. One possibility is to select a node with the highest price among all positive surplus nodes i with $p(i) < N$. Each time the path P degenerates to its start node, following a contraction,

it is possible to make a new start node selection based on the highest price criterion without affecting the termination properties of the algorithm.

An alternative is to maintain all nodes i with positive surplus and $p(i) < N$ in a FIFO queue, and use as starting node the first node in the queue. Note that the preflow-push method that uses a highest price scheme is superior to the method that uses a FIFO scheme in terms of worst-case complexity [$O(N^2A^{1/2})$ versus $O(N^3)$].

Greedy Augmentations

Once an augmenting path is constructed, instead of pushing the same amount of flow along each arc of the path, it is possible to push along each arc (i, j) the maximum possible amount of flow, that is, $\max\{g_i, c_{ij} - x_{ij}\}$ if (i, j) is a forward arc of the path, or $\max\{g_j, x_{ij}\}$ if (i, j) is a backward arc of the path. We call this a *greedy augmentation*. For examples where such augmentations are helpful, we refer to the paper [Ber95a].

Using a Second Best Candidate

Consider the variant of the algorithm, where only one node of the set $Succ(i)$, call it $j_1(i)$, is maintained for each i, together with a corresponding arc of $Cand(i)$. Suppose that for the terminal node n_t of the current path P we have available a lower bound $\beta(n_t)$ on the prices of all the nodes in $N(n_t, x)$ except for the price of node $j_1(n_t)$. Suppose also that in Step 1, the downhill test $p(n_t) \geq p\big(j_1(n_t)\big)$ of Eq. (4.24) for an extension is failed. Then we can check to see whether we have

$$p\big(j_1(n_t)\big) \leq \beta(n_t),$$

and if this is so, we know that $p\big(j_1(n_t)\big)$ is still less or equal to the prices of all nodes in $N(x, n_t)$, thereby making the computation of this minimum as per Eq. (4.25) unnecessary. A lower bound of this type can be obtained by calculating, together with $j_1(n_t)$, the second best node in $N(n_t, x)$, that is, a node $j_2(n_t)$ given by

$$j_2(n_t) = \arg \min_{j \in N(n_t, x),\, j \neq j_1(n_t)} p(j).$$

Then, as long as $j_1(n_t)$ remains unchanged and no new node is added to $N(n_t, x)$, we can use

$$\beta(n_t) = p\big(j_2(n_t)\big)$$

as a suitable lower bound [if a new node is added to $N(n_t, x)$ due to an augmentation, we must suitably modify $\beta(n_t)$ and $j_2(n_t)$]. This idea can be further strengthened by checking to see if $j_2(n_t)$ still belongs to $N(n_t, x)$ and whether its price is still $\beta(n_t)$, in the case where the test $p\big(j_1(n_t)\big) \leq$

$p\big(j_2(n_t)\big)$ is failed. If this is so, we can set $j_1(n_t)$ to $j_2(n_t)$, thereby obviating again the calculation of the minimum in Eq. (4.25).

The idea of using a second best candidate arc and node is known to be very effective in auction algorithms for the assignment problem and the shortest path problem (see Chapters 2 and 3). It similarly improves the performance of our max-flow algorithm.

4.3 THE AUCTION/SEQUENTIAL SHORTEST PATH ALGORITHM

In this section, we develop an auction algorithm for the solution of the transhipment problem, based on augmentations along paths, which are guided by an ϵ-CS condition. An important feature of the auction approach is that it allows useful information to be passed from one path construction to the next in the form of prices.

We use an approach that blends the auction/shortest path construction process with the remainder of the algorithm. In this approach, we use ϵ-perturbations of the arc lengths, related to ϵ-CS, which ensure that the path generated by the auction/shortest path method does not close a cycle through an extension. We first introduce some terminology.

We recall from Section 4.1 that given a flow-price pair (x, p) satisfying ϵ-CS, an arc (i, j) is said to be ϵ^+-*unblocked* if

$$p_i = p_j + a_{ij} + \epsilon \qquad \text{and} \qquad x_{ij} < c_{ij},$$

and an arc (j, i) is said to be ϵ^--*unblocked* if

$$p_i = p_j - a_{ji} + \epsilon \qquad \text{and} \qquad b_{ji} < x_{ji}.$$

The *admissible graph* corresponding to (x, p) is defined as $G^* = (\mathcal{N}, \mathcal{A}^*)$, where the arc set $\mathcal{A}^*$ consists of an arc (i, j) for each ϵ^+-unblocked arc $(i, j) \in \mathcal{A}$, and an arc (i, j) for each ϵ^--unblocked arc $(j, i) \in \mathcal{A}$.

We recall that a path P is a sequence of nodes $(n_1, n_2, \ldots, n_k)$ and a corresponding sequence of $k-1$ arcs such that the ith arc in the sequence is either (n_i, n_{i+1}) or (n_{i+1}, n_i). For any path P, we denote by $s(P)$ and $t(P)$ the start and terminal nodes of P, respectively, and by P^+ and P^- the sets of forward and backward arcs of P, respectively. The path P is said to be ϵ-*unblocked* if all arcs of P^+ are ϵ^+-unblocked, and all arcs of P^- are ϵ^--unblocked. If P is ϵ-unblocked, and the start node $s(P)$ has positive excess and the terminal node $t(P)$ has negative excess, then P is an *augmenting path*. An augmentation along such a path consists of increasing the flow of all arcs in P^+ and reducing the flow of all arcs in P^- by the common increment

$$\delta = \min \left\{ g_{s(P)},\ -g_{t(P)},\ \min_{(i,j) \in P^+} \{c_{ij} - x_{ij}\},\ \min_{(i,j) \in P^-} \{x_{ij} - b_{ij}\} \right\}.$$

Given a path $P = (n_1, n_2, \ldots, n_k)$, a *contraction* of P is the operation that deletes the terminal node of P together with the corresponding terminal arc. An *extension* of P by an arc (n_k, n_{k+1}) or an arc (n_{k+1}, n_k), replaces P by the path $(n_1, n_2, \ldots, n_k, n_{k+1})$ and adds to P the corresponding arc. For convenience, we allow a path P to consist of a single node i, in which case extension by an arc (i, j) or (j, i) gives a path with start node i and terminal node j.

The algorithm to be presented will be called *auction/sequential shortest path algorithm* (abbreviated ASSP). It uses a fixed $\epsilon > 0$, and maintains a flow-price pair (x, p) satisfying ϵ-CS and also a simple path P (possibly consisting of a single node). It terminates when all nodes have nonnegative excess; then either all nodes have zero excess and x is feasible, or else some node has negative excess showing that the problem is infeasible. Throughout the algorithm, x is integer, and (x, p) and P satisfy:

(a) The admissible graph corresponding to (x, p) is acyclic.

(b) P belongs to the admissible graph, i.e., it is ϵ-unblocked. Furthermore, P starts at a node with positive excess, and all its nodes have nonnegative excess.

We assume that at the start of the algorithm we have a pair (x, p) satisfying ϵ-CS, as well as the above two properties. In particular, initially one may choose any price vector p, select x according to

$$x_{ij} = \begin{cases} c_{ij} & \text{if } p_i \geq a_{ij} + p_j, \\ b_{ij} & \text{if } p_i < a_{ij} + p_j, \end{cases}$$

and choose P to consist of a single node with positive excess. For these choices, ϵ-CS is satisfied and the corresponding admissible graph is acyclic, since its arc set is empty.

At each iteration, the path P is either extended or contracted. In the case of a contraction, the price of the terminal node of P is strictly increased. In the case of an extension, no price rise occurs, but if the new terminal node has negative excess, P becomes augmenting, and an augmentation along P is performed. Then the path P is replaced by the degenerate path that consists of a single node with positive excess, and the process is repeated.

Iteration of the ASSP Algorithm

Let i be the terminal node of P. If

$$p_i < \min\Big\{ \min_{\{(i,j)\in\mathcal{A}|x_{ij}<c_{ij}\}} \big\{a_{ij}+p_j+\epsilon\big\}, \min_{\{(j,i)\in\mathcal{A}|b_{ji}<x_{ji}\}} \big\{p_j-a_{ji}+\epsilon\big\}\Big\} \tag{4.30}$$

go to Step 1; else go to Step 2.

Step 1 (Contract path): Set

$$p_i := \min\Big\{ \min_{\{(i,j)\in\mathcal{A}|x_{ij}<c_{ij}\}} \big\{a_{ij}+p_j+\epsilon\big\}, \min_{\{(j,i)\in\mathcal{A}|b_{ji}<x_{ji}\}} \big\{p_j-a_{ji}+\epsilon\big\}\Big\} \tag{4.31}$$

and if $i \neq s(P)$, contract P. Go to the next iteration.

Step 2 (Extend path): Extend P by an arc (i, j_i) or an arc (j_i, i) that attains the minimum in Eq. (4.30). If the excess of j_i is negative go to Step 3; otherwise, go to the next iteration.

Step 3 (Augmentation): Perform an augmentation along P. If all nodes have nonpositive excess, terminate the algorithm; otherwise, replace P by a path that consists of a single node with positive excess and go to the next iteration.

The following proposition establishes that some basic properties are maintained by the algorithm.

Proposition 4.3.1: Suppose that at the start of an iteration of the ASSP algorithm the following two conditions hold:

(1) (x, p) satisfies ϵ-CS and the corresponding admissible graph is acyclic.

(2) P belongs to the admissible graph, starts at a node with positive excess, and all its nodes have nonnegative excess.

Then these two conditions hold at the start of the next iteration.

Proof: Suppose the iteration involves a contraction. Then it can be seen that the price increase (4.31) preserves ϵ-CS. Furthermore, since only the

price of node i changes and no arc flow changes, the admissible graph remains unchanged except for the incident arcs of node i. In particular, all the incident arcs of i in the admissible graph at the start of the iteration are deleted and the arcs of the admissible graph corresponding to the arcs (i,j) and (j,i) that attain the minimum in Eq. (4.31) are added. Since all these arcs are outgoing from i in the admissible graph, a cycle cannot be closed. Finally, following a contraction, P does not contain the terminal node i, so it belongs to the admissible graph that we had before the iteration. Thus P consists of arcs that belong to the admissible graph that we obtain after the iteration.

Suppose the iteration involves an extension. Then by ϵ-CS, we must have

$$p_i = \min\left\{\min_{\{(i,j)\in\mathcal{A}|x_{ij}<c_{ij}\}}\big\{a_{ij}+p_j+\epsilon\big\}, \min_{\{(j,i)\in\mathcal{A}|b_{ji}<x_{ji}\}}\big\{p_j-a_{ji}+\epsilon\big\}\right\},$$

at the start of the iteration. It follows that the path P obtained by extension is simple and ϵ-unblocked, since the extension arc (i,j_i) must belong to the admissible graph. Since no price or flow changes with an extension, the ϵ-CS conditions and the admissible graph stay unchanged following the extension. If there is a subsequent augmentation at Step 3 because the new terminal node j_i has negative excess, the ϵ-CS conditions will not be affected, while the admissible graph will not gain any new arcs, so it will remain acyclic. **Q.E.D.**

Note that if we were to take $\epsilon = 0$ (rather than $\epsilon > 0$), the preceding proof would break down, because we would not be able to prove that the admissible graph remains acyclic following an augmentation. In particular, if following an augmentation, the flow of some arc (i,j) lies strictly between its lower and upper bound, the arcs (i,j) and (j,i) would both belong to the admissible graph, each with zero length, thereby closing a zero length cycle.

A sequence of iterations between two successive augmentations (or the sequence of iterations up to the first augmentation) will be called an *augmentation cycle*. Let us fix an augmentation cycle and let $\overline{p}$ be the price vector at the start of the cycle. The reduced graph $G_R = (\mathcal{N}, \mathcal{A}_\mathcal{R})$, defined earlier, will not change in the course of this augmentation cycle, since no arc flow will change during the cycle, except for the augmentation at the end. Suppose that we take as arc lengths of the reduced graph the reduced costs at the start of the cycle plus ϵ. In particular, during the cycle, the arc set $\mathcal{A}_\mathcal{R}$ consists of an arc (i,j) with length $a_{ij}+\overline{p}_j-\overline{p}_i+\epsilon$ for each arc $(i,j)\in\mathcal{A}$ with $x_{ij}<c_{ij}$, and an arc (j,i) with length $\overline{p}_i-a_{ij}-\overline{p}_j+\epsilon$ for each arc $(i,j)\in\mathcal{A}$ with $b_{ij}<x_{ij}$. Note that, because $(x,\overline{p})$ satisfies ϵ-CS, the arc lengths of the reduced graph are nonnegative. However, the reduced graph does not contain zero length cycles, since any such cycle must belong to the admissible graph, which is acyclic.

Using these observations, it can now be seen that the augmentation cycle is just the auction/shortest path algorithm of Section 2.6 applied to the problem of finding a shortest path from the starting node $s(P)$ to some node with negative excess in the reduced graph G_R, using the preceding ϵ-perturbed arc lengths. To understand this, one should view $p_i - \overline{p}_i$ during the augmentation cycle as the price of node i that is maintained by the auction/shortest path algorithm. The price increments $p_i - \overline{p}_i$ obtained by the auction/shortest path algorithm are added in effect to the starting prices $\overline{p}_i$ at the end of the augmentation cycle to form the new prices that will be used for the shortest path construction of the next augmentation cycle.

By the theory of the auction/shortest path algorithm, a shortest path in the reduced graph will be found in a finite number of iterations if there exists at least one path from the starting node $s(P)$ to some node with negative excess. Such a path is guaranteed to exist if the problem is feasible. Since the augmentation will change all the flows of the final path P by a positive integer amount, we see that each augmentation cycle reduces the total absolute excess $\sum_{i \in \mathcal{N}} |g_i|$ by a positive integer. Therefore, there can be only a finite number of augmentation cycles, and we have shown the following proposition.

Proposition 4.3.2: Assume that the minimum cost flow problem is feasible. Then the ASSP algorithm terminates with a pair (x, p) satisfying ϵ-CS. The flow vector x is feasible and is optimal if $\epsilon < 1/N$.

It is interesting to try to relate the iterations of the algorithm with iterations of the ϵ-relaxation method. Each iteration of the algorithm involving a contraction can be viewed as an iteration of an ϵ-relaxation method, except that the iterating terminal node i may have zero excess. Each iteration involving an extension without an augmentation changes neither the flow nor the price vectors; it merely extends the path P by a single arc. Finally, each iteration involving an augmentation can be viewed as a sequence of ϵ-relaxation iterations, each pushing the flow increment δ along the ϵ^+-unblocked forward arcs and the ϵ^--unblocked backward arcs of P. Thus we may view the algorithm as a variant of the ϵ-relaxation method.

ϵ-Scaling

As in all auction algorithms, the practical performance of the algorithm may be degraded by "price wars," that is, prolonged sequences of iterations involving small price increases. There is a built-in potential for price

wars here because with a small ϵ, the reduced graph may contain cycles with small length, which slow down the underlying auction/shortest path algorithm. (There is a cycle of length 2ϵ for every arc whose flow lies strictly between the corresponding flow bounds.) This difficulty can be addressed by ϵ-scaling, that is, by applying the algorithm several times, each time decreasing ϵ by a constant factor, up to the threshold value of $1/(N+1)$, while using the final prices obtained for one value of ϵ as starting prices for the next value of ϵ. A polynomial complexity bound of $O\big(N^2 A \log(NC)\big)$, where C is the cost range

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

can be proved for the resulting method, after we introduce modifications similar to the ones of Section ??? for the ϵ-relaxation method. The unscaled version of the method, where ϵ is kept fixed at $1/(N+1)$, is pseudopolynomial.

In addition to ϵ-scaling, there are several implementation techniques, which have been found to improve performance in practice. We refer to Bertsekas [1995b] for further details and computational results.

4.4 AUCTION ALGORITHMS FOR CONVEX SEPARABLE NETWORK OPTIMIZATION

In this section we develop auction algorithms for the separable convex network flow problem introduced in Section 1.2. It has the form

$$\begin{aligned} \text{minimize} \quad & \sum_{(i,j)\in\mathcal{A}} f_{ij}(x_{ij}) \\ \text{subject to} \quad & \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} = s_i, \qquad \forall\ i \in \mathcal{N}, \\ & x_{ij} \in X_{ij}, \qquad \forall\ (i,j) \in \mathcal{A}, \end{aligned} \tag{4.32}$$

where x is a flow vector in a given directed graph $(\mathcal{N}, \mathcal{A})$, s_i are given supply scalars, X_{ij} are nonempty intervals of scalars, and each function $f_{ij} : X_{ij} \mapsto \Re$ is convex.

We begin with a development of the mathematical properties of convex functions of one variable in Section 4.4.1. We then derive, in Section 4.4.2, optimality conditions that do not require differentiability of the cost function. In Section 4.4.3, we develop a duality theory that generalizes the one of Section 1.3 for the transhipment problem.

We then proceed with the development of auction algorithms for convex separable problems. These algorithms can deal with nondifferentiabilities in the dual problem and are also very efficient in practice. There is a

solid theoretical basis for this efficiency, as we show with a computational complexity analysis. Based on complexity analysis and experimentation, these algorithms are very efficient. With proper implementation, they appear to be minimally affected by ill-conditioning in the dual problem.

4.4.1 Convex Functions of a Single Variable

In this section, we introduce some mathematical properties of convex functions of one variable, defined over an interval of the real line $\Re$. We recall that in our terminology, an interval is a nonempty and convex subset of the real line. The supremum (infimum) of an interval is called the *right endpoint* (the *left endpoint*, respectively). Thus, an interval is a set that has one of the forms (a,b), $(a,b]$, $[a,b)$, $[a,b]$, $(-\infty,b)$, $(-\infty,b]$, (a,∞), $[a,\infty)$, $(-\infty,\infty)$, where a and b are scalars. The left endpoint is a (or $-\infty$) and the right endpoint is b (or ∞). The *interior* of an interval is the set (a,b) where a and b are the left and right endpoints, respectively.

Let $f : X \mapsto \Re$ be a convex function defined on an interval X.† The subset

$$\big\{(x,\gamma) \mid x \in X,\ f(x) \leq \gamma\big\}$$

of $\Re^2$ is called the *epigraph* of f, and is convex if and only if f is convex. It can be shown (as a consequence of convexity) that f is continuous at all points in the interior of X; that is, $\lim_{k\to\infty} f(x_k) = f(x)$ for all sequences $\{x_k\} \subset X$ converging to an interior point x of X. At an endpoint of X that is included in X, f may or may not be continuous. A condition that guarantees continuity of f over the entire interval X is that the epigraph of f is a closed subset of $\Re^2$. If this condition holds, we say that f is *closed*. *Throughout this chapter, we assume that the convex functions f_{ij} involved in the convex separable network problem (4.32) are closed*. This assumption facilitates the analysis and is practically always satisfied.

---

† Much of the literature of convex analysis treats convex functions as extended real-valued functions, which are defined over the entire real line but take the value ∞ outside their (effective) domain. In this format, a function $f : X \mapsto \Re$ that is convex over the convex interval X is represented by the function $\hat{f} : \Re \mapsto (-\infty,\infty]$ defined by

$$\hat{f}(x) = \begin{cases} f(x) & \text{if } x \in X, \\ \infty & \text{if } x \notin X. \end{cases}$$

There are notational advantages to this format, particularly for functions of several variables, as it is not necessary to keep track of the domains of various functions explicitly. It is simpler for our limited purposes, however, to maintain the more common framework of real-valued functions.

The *right derivative* of f at a point $x \in X$ that is not the right endpoint of X is defined by

$$f^+(x) = \lim_{\alpha_k \to 0^+} \frac{f(x+\alpha_k) - f(x)}{\alpha_k},$$

where the limit is taken over any positive sequence $\{\alpha_k\}$ such that $x+\alpha_k \in X$ for all k. If X contains its right endpoint b, we define $f^+(b) = \infty$. Similarly, the *left derivative* of f at a point $x \in X$ that is not the left endpoint of X is defined by

$$f^-(x) = \lim_{\alpha_k \to 0^+} \frac{f(x-\alpha_k) - f(x)}{\alpha_k},$$

where the limit is taken over any positive sequence $\{\alpha_k\}$ such that $x-\alpha_k \in X$ for all k. If X contains its left endpoint a, we define $f^-(a) = -\infty$. In the degenerate case where X consists of a single point a, we define $f^-(a) = -\infty$ and $f^+(a) = \infty$. Note that the only point of X where f^+ may equal ∞ is the right endpoint (assuming it belongs to X), and the only point of X where f^- may equal $-\infty$ is the left endpoint (assuming it belongs to X).

It can be shown, as a consequence of convexity, that the right and left derivatives are monotonically nondecreasing and satisfy

$$f^-(x) \leq f^+(x) \leq f^-(y) \leq f^+(y), \qquad \forall\ x, y \in X \text{ with } x < y. \tag{4.33}$$

Furthermore, f^- is left continuous (f^+ is right continuous) over the interval where it is finite. If f is differentiable at a point $x \in X$, we have

$$f^-(x) = f^+(x) = \nabla f(x),$$

where $\nabla f(x)$ is the gradient of f at x. The right and left derivatives define the subset

$$\Gamma = \big\{(x,t) \mid x \in X,\ f^-(x) \leq t \leq f^+(x)\big\}$$

of $\Re^2$, which is called the *characteristic curve of* f, and is illustrated in Fig. 4.4.1.

Directional Derivatives of Separable Convex Functions

Consider now a general convex set F in $\Re^n$, and a function $f : F \mapsto \Re$ that is convex. The *directional derivative* $f'(x;y)$ of f at a vector $x \in F$ in the direction y is defined to be the right derivative of the convex function $f(x+\alpha y)$ of the scalar α at $\alpha = 0$ (this function is defined over the interval of all α such that $x + \alpha y \in F$). In other words,

$$f'(x;y) = \lim_{\alpha \to 0^+} \frac{f(x+\alpha y) - f(x)}{\alpha}, \tag{4.34}$$

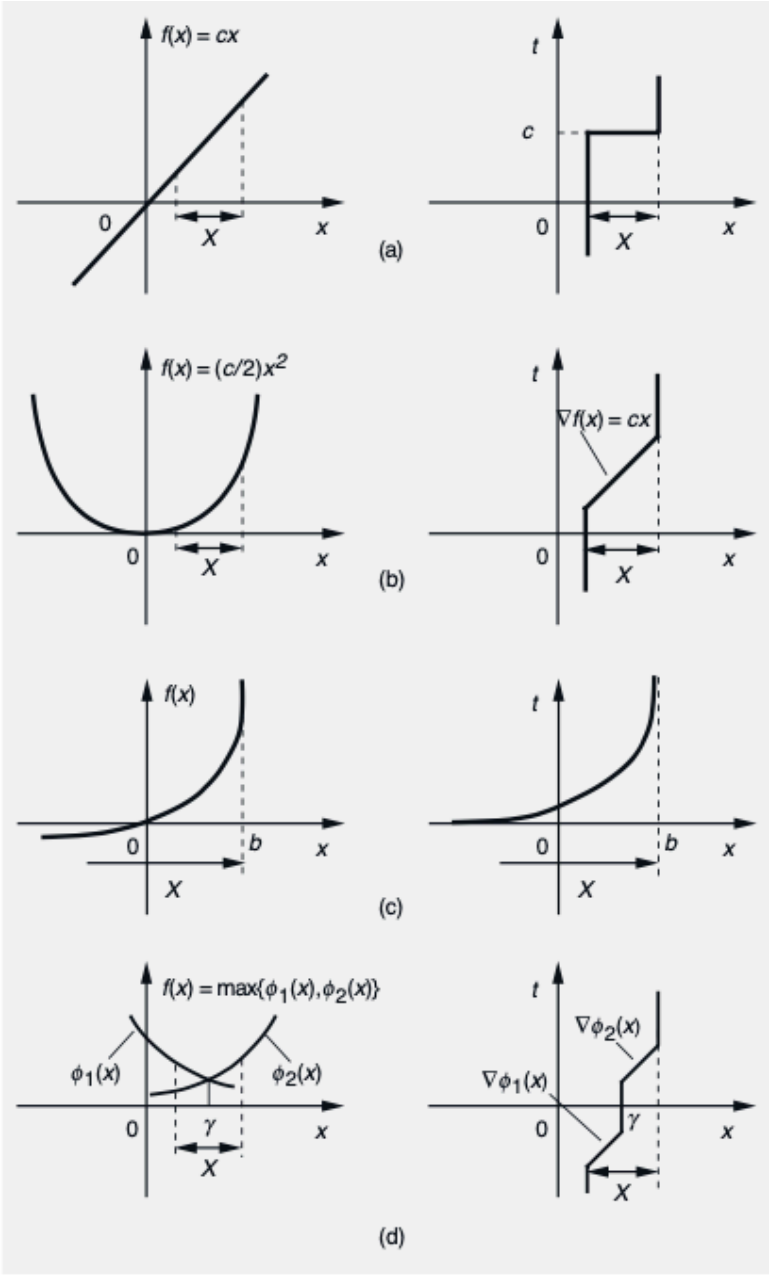

Figure 4.4.1: Illustration of various convex functions $f : X \mapsto \Re$ (on the left-hand side) and their right and left derivatives, and characteristic curves

$$\Gamma = \Big\{(x,t) \mid x \in X,\, f^-(x) \le t \le f^+(x)\Big\}$$

(on the right-hand side). In example (c), X contains its right endpoint b, but we have $f^-(b) = f^+(b) = \infty$.

where we use the convention $f(x+\alpha y)=\infty$ if $x+\alpha y \notin F$. Note that a vector $x^* \in F$ minimizes f over F if and only if

$$f'(x^*;y) \geq 0, \qquad \forall\ y. \tag{4.35}$$

Let us consider the special case of a separable function of the flow vector x:

$$f(x) = \sum_{(i,j)\in\mathcal{A}} f_{ij}(x_{ij}),$$

where each f_{ij} is a closed convex function over an interval X_{ij}. Then, by applying the definition (4.34), we see that the directional derivative is given by

$$f'(x;y) = \sum_{\{(i,j)\in\mathcal{A}\mid y_{ij}>0\}} f_{ij}^+(x_{ij})y_{ij} + \sum_{\{(i,j)\in\mathcal{A}\mid y_{ij}<0\}} f_{ij}^-(x_{ij})y_{ij}, \tag{4.36}$$

where $f_{ij}^-(x_{ij})$ and $f_{ij}^+(x_{ij})$ denote the left and the right derivative of f_{ij} at an arc flow $x_{ij} \in X_{ij}$. There is an ambiguity in the above equation when $f_{ij}^+(x_{ij})=\infty$ for some (i,j) with $y_{ij}>0$ and $f_{ij}^-(x_{ij})=\infty$ for some (i,j) with $y_{ij}<0$, in which case the sum $\infty-\infty$ appears. We resolve this ambiguity by adopting the convention

$$\infty - \infty = \infty.$$

It can be shown by using the definition (4.34) that with this convention, the directional derivative formula of Eq. (4.36) is correct even in cases where the ambiguity arises. To see this, note that if $f_{ij}^+(x_{ij})=\infty$ for some (i,j), x_{ij} must be the right endpoint of the interval X_{ij}, so that if in addition $y_{ij}>0$, it follows that $x_{ij}+\alpha y_{ij} \notin X_{ij}$ for all $\alpha>0$. Thus $x+\alpha y$ is outside the domain of f for all $\alpha>0$, so that, according to our convention, $f(x+\alpha y)=\infty$ for all $\alpha>0$ and, from Eq. (4.34), $f'(x;y)=\infty$.

4.4.2 Optimality Conditions

In this and the next two sections, we discuss the main analytical aspects of convex separable problems. The optimality conditions derived in Section 8.6 require differentiability of the cost function. However, the approach used there can be extended to a nondifferentiable separable convex cost by using directional differentiability. In particular, by arguing that the directional derivative of f cannot be negative along any feasible direction at x^* [cf. Eq. (4.35)], we obtain a generalization of the nonnegative cycle condition for optimality of Props. 1.2 and 8.2.

Proposition 4.4.1: (Nonnegative Cycle Condition) Consider the separable convex network problem. A vector x^* is optimal if and only if x^* is feasible and for every simple cycle C that is unblocked with respect to x^* there holds

$$\sum_{(i,j)\in C^+} f_{ij}^+(x_{ij}^*) - \sum_{(i,j)\in C^-} f_{ij}^-(x_{ij}^*) \geq 0. \tag{4.37}$$

Proof: Let x^* be an optimal flow vector and let C be a simple cycle that is unblocked with respect to x^*. Consider the flow vector $d(C)$ with components

$$d_{ij}(C) = \begin{cases} 1 & \text{if } (i,j) \in C^+, \\ -1 & \text{if } (i,j) \in C^-, \\ 0 & \text{otherwise.} \end{cases} \tag{4.38}$$

Then $d(C)$ is a feasible direction at x^* and using Eq. (4.36), it is seen that the directional derivative of f at x^* in the direction $d(C)$ is the left-hand side of Eq. (4.37). Since x^* is optimal, this directional derivative must be nonnegative [cf. Eq. (4.35)].

Conversely, suppose that x^* is feasible but not optimal. Let $\overline{x}$ be a feasible flow vector with cost smaller that the one of x^*. Consider a conformal decomposition of the circulation $\overline{x} - x^*$ into simple cycles $C_1, \ldots, C_M$, and the corresponding cycle flow vectors $d(C_1), \ldots, d(C_M)$ as per Eq. (4.38):

$$\overline{x} - x^* = \sum_{m=1}^{M} \gamma_m d(C_m), \qquad \gamma_m > 0,\ m = 1, \ldots, M. \tag{4.39}$$

Using Eqs. (4.36) and (4.39), we see that the directional derivative of f in

the direction $\overline{x} - x^*$ is given by

$$\begin{aligned} f'(x^*; \overline{x} - x^*) &= \sum_{\{(i,j)|\overline{x}_{ij} - x^*_{ij} > 0\}} f^+_{ij}(x^*_{ij})(\overline{x}_{ij} - x^*_{ij}) \\ &\quad + \sum_{\{(i,j)|\overline{x}_{ij} - x^*_{ij} < 0\}} f^-_{ij}(x^*_{ij})(\overline{x}_{ij} - x^*_{ij}) \\ &= \sum_{\{(i,j)|\overline{x}_{ij} - x^*_{ij} > 0\}} f^+_{ij}(x^*_{ij}) \sum_{m=1}^{M} \gamma_m d_{ij}(C_m) \\ &\quad + \sum_{\{(i,j)|\overline{x}_{ij} - x^*_{ij} < 0\}} f^-_{ij}(x^*_{ij}) \sum_{m=1}^{M} \gamma_m d_{ij}(C_m) \\ &= \sum_{m=1}^{M} \gamma_m \Bigg(\sum_{\{(i,j)|d_{ij}(C_m) > 0\}} f^+_{ij}(x^*_{ij}) d_{ij}(C_m) \\ &\quad + \sum_{\{(i,j)|d_{ij}(C_m) < 0\}} f^-_{ij}(x^*_{ij}) d_{ij}(C_m) \Bigg) \\ &= \sum_{m=1}^{M} \gamma_m f'\big(x^*; d(C_m)\big). \end{aligned}$$

[The last equality holds using the definition (4.36) of a directional derivative. The next-to-last inequality holds because for any arc (i,j) the sign of each nonzero arc flow $d_{ij}(C_m)$ is the same as the sign of $\overline{x}_{ij} - x^*_{ij}$, since the decomposition is conformal.] Since $f'(x^*; \overline{x} - x^*) < 0$ and $\gamma_m > 0$ for all m, we must have $f'\big(x^*; d(C_m)\big) < 0$ for at least one m, or

$$\sum_{(i,j) \in C^+_m} f^+_{ij}(x^*_{ij}) - \sum_{(i,j) \in C^-_m} f^-_{ij}(x^*_{ij}) < 0.$$

Thus if Eq. (4.37) holds, x^* must be optimal. **Q.E.D.**

4.4.3 Duality

As in earlier developments of duality, we obtain a dual problem by introducing a price p_i for each node i and by forming the Lagrangian function

$$\begin{aligned} L(x,p) &= \sum_{(i,j) \in \mathcal{A}} f_{ij}(x_{ij}) + \sum_{i \in \mathcal{N}} p_i \left(\sum_{\{j|(j,i) \in \mathcal{A}\}} x_{ji} - \sum_{\{j|(i,j) \in \mathcal{A}\}} x_{ij} + s_i \right) \\ &= \sum_{(i,j) \in \mathcal{A}} \big(f_{ij}(x_{ij}) - (p_i - p_j) x_{ij} \big) + \sum_{i \in \mathcal{N}} s_i p_i. \end{aligned} \tag{4.40}$$

The dual function value $q(p)$ at a price vector p is obtained by minimizing $L(x,p)$ over all x satisfying the constraint $x_{ij} \in X_{ij}$. Thus,

$$q(p) = \inf_{x \in X} L(x,p) = \sum_{(i,j) \in \mathcal{A}} q_{ij}(p_i - p_j) + \sum_{i \in \mathcal{N}} s_i p_i,$$

where

$$q_{ij}(p_i - p_j) = \inf_{x_{ij} \in X_{ij}} \big\{ f_{ij}(x_{ij}) - (p_i - p_j) x_{ij} \big\}. \tag{4.41}$$

The problem

$$\text{maximize} \quad q(p)$$

$$\text{subject to no constraint on } p,$$

is referred to as the *dual problem*, while the original problem of minimizing f subject to the conservation of flow constraints and $x \in X$ is referred to as the *primal problem*. The dual function is also referred to as the *dual cost function* or *dual cost*, and the optimal value of the dual problem is referred to as the *optimal dual cost*.

Note that q_{ij} is concave since it is the pointwise infimum of linear functions [the epigraph of $-q_{ij}$ is a convex set, since it is the intersection of the epigraphs of the linear functions $(p_i - p_j)x_{ij} - f_{ij}(x_{ij})$ as x_{ij} ranges over X_{ij}]. If X_{ij} is a compact set, then since f_{ij} is assumed closed and hence continuous over X_{ij}, the infimum in the definition (4.41) of q_{ij} is attained (by Weierstrass' theorem), and it follows that q_{ij} is real-valued; that is, $q(p)$ is a real number for all p. If X_{ij} is not compact, it is possible that q_{ij} is not real-valued. Thus the dual problem embodies the implicit constraint $p \in Q$, where Q is the "effective domain" of q given by

$$Q = \big\{ p \mid q(p) > -\infty \big\}.$$

We consequently say that a price vector p is *feasible* if $q(p) > -\infty$. The dual problem is said to be *infeasible* if there is no feasible price vector. The form of q_{ij} is illustrated in Fig. 4.4.2.†

Our objective is to generalize the duality theorems given in Chapter 4 for the minimum cost flow cost problem. For this, we must first generalize the conditions for complementary slackness.

---

† The relation between the primal and dual arc cost functions f_{ij} and q_{ij} is a special case of a conjugacy relation that is central in the theory of convex functions (see e.g., Rockafellar [1970], [1984]). There is a rich theory around this relation. Here, we will prove only those facts about conjugacy that we will need in our analysis.

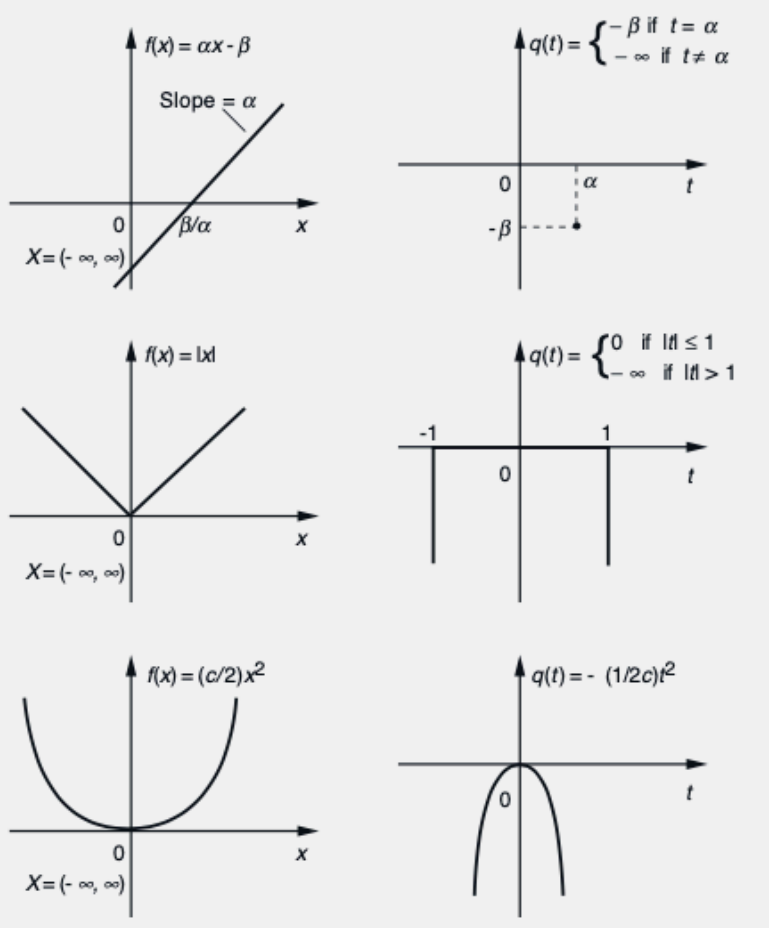

Figure 4.4.2: Illustration of primal and dual arc cost function pairs. Points where the primal function is nondifferentiable correspond to linear segments of the dual function.

Definition 4.4.1: A flow-price vector pair (x, p) is said to satisfy *complementary slackness* (CS for short) if for all arcs (i, j), we have $x_{ij} \in X_{ij}$ and

$$f_{ij}^-(x_{ij}) \leq p_i - p_j \leq f_{ij}^+(x_{ij}).$$

Thus a pair (x, p) satisfies CS if for every arc (i, j), the pair $(x_{ij}, p_i - p_j)$ lies on the characteristic curve of the function f_{ij} (see Fig. 4.4.3). Note

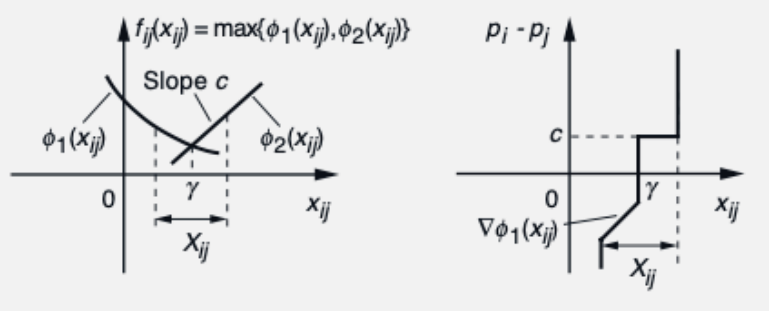

Figure 4.4.3: Illustration of CS. The pairs $(x_{ij}, p_i - p_j)$ must lie on the corresponding characteristic curves

$$\Gamma_{ij} = \Big\{(x_{ij}, t_{ij}) \mid x_{ij} \in X_{ij},\, f_{ij}^-(x_{ij}) \leq t_{ij} \leq f_{ij}^+(x_{ij})\Big\},$$

shown in the right-hand side.

that an equivalent definition of CS is that x_{ij} attains the infimum in the definition of q_{ij} for all arcs (i, j):

$$f_{ij}(x_{ij}) - (p_i - p_j)x_{ij} = \min_{z_{ij} \in X_{ij}} \big\{f_{ij}(z_{ij}) - (p_i - p_j)z_{ij}\big\}.$$

It can be seen that these conditions generalize the corresponding CS conditions for the minimum cost flow problem.

We are now ready to derive the basic duality results for separable problems.

Proposition 4.4.2: (Complementary Slackness Theorem) A feasible flow vector x^* and a price vector p^* satisfy CS if and only if x^* and p^* are optimal primal and dual solutions, respectively, and the optimal primal and dual costs are equal.

Proof: We first show that for any feasible flow vector x and any price vector p, the primal cost of x is no less than the dual cost of p. Indeed, using the definition of $q(p)$ and $L(x, p)$, we have

$$\begin{aligned} q(p) &\leq L(x, p) \\ &= \sum_{(i,j) \in \mathcal{A}} f_{ij}(x_{ij}) + \sum_{i \in \mathcal{N}} p_i \left(s_i - \sum_{\{j \mid (i,j) \in \mathcal{A}\}} x_{ij} + \sum_{\{j \mid (j,i) \in \mathcal{A}\}} x_{ji} \right) \\ &= \sum_{(i,j) \in \mathcal{A}} f_{ij}(x_{ij}), \end{aligned} \tag{4.42}$$

where the last equality follows from the feasibility of x.

If x^* is feasible and satisfies CS together with p^*, we have by the definition of q

$$\begin{aligned} q(p^*) &= \inf_x \big\{ L(x,p^*) \mid x_{ij} \in X_{ij}, (i,j) \in \mathcal{A} \big\} \\ &= L(x^*,p^*) \\ &= \sum_{(i,j)\in\mathcal{A}} f_{ij}(x^*_{ij}) + \sum_{i\in\mathcal{N}} p^*_i \left(s_i - \sum_{\{j \mid (i,j)\in\mathcal{A}\}} x^*_{ij} + \sum_{\{j \mid (j,i)\in\mathcal{A}\}} x^*_{ji} \right) \\ &= \sum_{(i,j)\in\mathcal{A}} f_{ij}(x^*_{ij}), \end{aligned}$$

where the last equality follows from the feasibility of x^*, and the second equality holds because (x^*, p^*) satisfies CS if and only if

$$f_{ij}(x^*_{ij}) - (p^*_i - p^*_j)x^*_{ij} = \min_{x_{ij}\in X_{ij}} \big\{ f_{ij}(x_{ij}) - (p^*_i - p^*_j)x_{ij} \big\}, \quad \forall\ (i,j) \in \mathcal{A},$$

and $L(x^*, p^*)$ can be written as in Eq. (4.40). Therefore, x^* attains the minimum of the primal cost on the right-hand side of Eq. (4.42), and p^* attains the maximum of $q(p)$ on the left-hand side of Eq. (4.42), while the optimal primal and dual costs are equal.

Conversely, suppose that x^* and p^* are optimal flow and price vectors for the primal and dual problems, respectively, and the two optimal costs are equal; that is,

$$q(p^*) = \sum_{(i,j)\in\mathcal{A}} f_{ij}(x^*_{ij}).$$

We have by definition

$$q(p^*) = \inf_x \big\{ L(x,p^*) \mid x_{ij} \in X_{ij}, (i,j) \in \mathcal{A} \big\},$$

and also, using the Lagrangian expression (4.40) and the feasibility of x^*,

$$\sum_{(i,j)\in\mathcal{A}} f_{ij}(x^*_{ij}) = L(x^*,p^*).$$

Combining the last three equations, we obtain

$$L(x^*,p^*) = \min_x \big\{ L(x,p^*) \mid x_{ij} \in X_{ij}, (i,j) \in \mathcal{A} \big\}.$$

Using the Lagrangian expression (4.40), it follows that for all arcs (i,j), we have

$$f_{ij}(x^*_{ij}) - (p^*_i - p^*_j)x^*_{ij} = \min_{x_{ij}\in X_{ij}} \big\{ f_{ij}(x_{ij}) - (p^*_i - p^*_j)x_{ij} \big\}.$$

This is equivalent to the pair (x^*, p^*) satisfying CS. **Q.E.D.**

An important question, which is left open by Prop. 4.4.2, is whether there exists a price vector that satisfies CS together with an optimal flow vector. For the minimum cost flow problem, this is always true, as we have seen in Chapter 1. However, answering this question for convex but nonlinear problems requires some qualifying condition. We introduce such a condition in the following definition.

Definition 4.4.2: (Regularity) A flow vector x is called *regular* if for all arcs (i,j), we have

$$f_{ij}^-(x_{ij}) < \infty, \qquad -\infty < f_{ij}^+(x_{ij}).$$

It is quite unusual for a flow vector x *not* to be regular. For this to happen, there must exist an arc flow x_{ij} that lies at the right (left) endpoint of the corresponding constraint interval X_{ij} while both the left and the right slopes of f_{ij} at that endpoint are ∞ (or $-\infty$, respectively) [see Fig. 4.4.1(c) for an example]. In particular, if x_{ij} belongs to the interior of X_{ij} for all arcs (i,j), then x is regular. Furthermore, all flow vectors are regular if each f_{ij} is the restriction to the interval X_{ij} of some function that is convex over the entire real line, such as for example a linear function.

While nonregularity is unusual for a feasible flow vector, it is far more rare for an *optimal* flow vector. In particular, we claim that *if there exists at least one regular feasible solution, all optimal solutions must be regular*. To show this, note that if x^* is an optimal solution and $\overline{x}$ is another feasible solution, we have

$$x_{ij}^* < \overline{x}_{ij} \qquad \Rightarrow \qquad f_{ij}^+(x_{ij}^*) < \infty,$$

since if $x_{ij}^* < \overline{x}_{ij}$, then x_{ij}^* cannot be the right endpoint of the interval X_{ij}. Similarly, we have

$$\overline{x}_{ij} < x_{ij}^* \qquad \Rightarrow \qquad f_{ij}^-(x_{ij}^*) > -\infty.$$

It follows from the preceding two relations that

$$f_{ij}^+(x_{ij}^*)(\overline{x}_{ij} - x_{ij}^*) < \infty, \quad f_{ij}^-(x_{ij}^*)(\overline{x}_{ij} - x_{ij}^*) < \infty, \qquad \forall\, (i,j) \in \mathcal{A}. \quad (4.43)$$

Now if $\overline{x}$ is regular and x^* is not regular but optimal, there must exist an arc (i,j) such that either (a) $f_{ij}^-(x_{ij}^*) = \infty$, or (b) $f_{ij}^+(x_{ij}^*) = -\infty$. In case (a), x_{ij}^* must be the right endpoint of X_{ij} and $\overline{x}_{ij} < x_{ij}^*$ (since $\overline{x}$ is regular).

Hence the product $f_{ij}^-(x_{ij}^*)(\overline{x}_{ij} - x_{ij}^*)$ is $-\infty$, and in view of Eq. (4.43), we have

$$f'(x^*; \overline{x} - x^*) = -\infty,$$

contradicting the optimality of x^*. We similarly obtain a contradiction in case (b), completing the proof that regularity of at least one feasible flow vector implies regularity of every optimal flow vector. We use this to show the following proposition.

Proposition 4.4.3: Suppose that there exists at least one primal feasible solution that is regular. Then, if x^* is an optimal solution of the primal problem, there exists an optimal solution p^* of the dual problem that satisfies CS together with x^*.

Proof: By Prop. 4.4.1, for every simple cycle C that is unblocked with respect to x^* there holds

$$\sum_{(i,j)\in C^+} f_{ij}^+(x_{ij}^*) - \sum_{(i,j)\in C^-} f_{ij}^-(x_{ij}^*) \geq 0.$$

The discussion preceding the present proposition, implies that x^* must be regular. Using this fact, it is seen that the assumptions for the use of the feasible differential theorem (Exercise 5.11 in Chapter 5) are fulfilled with $a_{ij}^+ = f_{ij}^+(x_{ij}^*)$ and $a_{ij}^- = f_{ij}^-(x_{ij}^*)$. Using the conclusion of this theorem, we can assert that there exists a price vector p^* satisfying

$$f_{ij}^-(x_{ij}) \leq p_i^* - p_j^* \leq f_{ij}^+(x_{ij}),$$

for all arcs (i, j). Thus p^* satisfies CS together with x^*. **Q.E.D.**

Figure 4.4.4 gives an example where the assertion of Prop. 4.4.3 does not hold in the absence of a regular feasible solution.

An important question, which is left open by Props. 4.4.2 and 4.4.3, relates to the equality of the optimal primal and dual costs in the absence of an optimal primal solution that is regular. Generally, for convex programs, it is possible that the optimal primal cost is strictly greater that the optimal dual cost, in which case we say that there is a *duality gap*. Using the machinery of the simplex method, we showed that for linear cost problems, this cannot happen (see Props. 4.2 and 5.8). However, the equality of the optimal primal and dual costs is a characteristic property of linear programs and the corresponding proof methods do not easily generalize to the case of a general convex cost function. It is thus somewhat unexpected that for the *separable* problem of this chapter the optimal primal and dual costs are

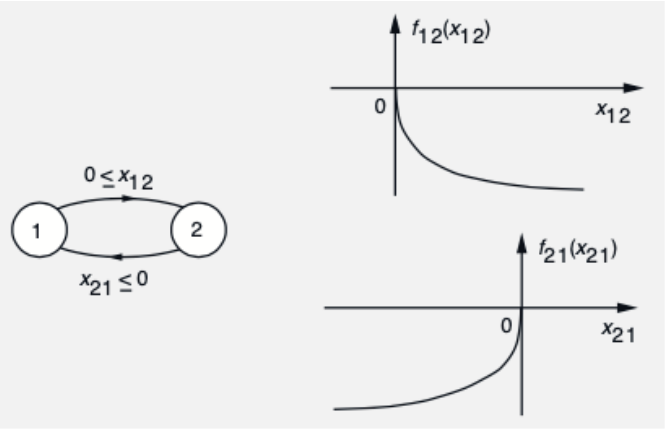

Figure 4.4.4: An example of a problem where there is no regular primal feasible solution, and the dual problem has no optimal solution (cf. Prop. 4.4.3). The primal problem is

$$\begin{aligned} &\text{minimize} \quad f_{12}(x_{12}) + f_{21}(x_{21}) \\ &\text{subject to} \quad x_{12} = x_{21}, \qquad 0 \le x_{12} < \infty, \quad -\infty < x_{21} \le 0, \end{aligned}$$

where

$$\begin{aligned} f_{12}(x_{12}) &= -\sqrt{x_{12}}, \qquad x_{12} \in [0, \infty), \\ f_{21}(x_{21}) &= -\sqrt{-x_{21}}, \qquad x_{21} \in (-\infty, 0]. \end{aligned}$$

The dual arc functions can be calculated to be

$$q_{12}(t_{12}) = \inf_{0 \le x_{12} < \infty} \left\{ -\sqrt{x_{12}} - t_{12} x_{12} \right\} = \begin{cases} \frac{1}{4t_{12}} & \text{if } t_{12} < 0, \\ -\infty & \text{otherwise,} \end{cases}$$

and

$$q_{21}(t_{21}) = \inf_{-\infty < x_{21} \le 0} \left\{ -\sqrt{-x_{21}} - t_{21} x_{21} \right\} = \begin{cases} -\frac{1}{4t_{21}} & \text{if } t_{21} > 0, \\ -\infty & \text{otherwise.} \end{cases}$$

The only primal feasible solution is the zero flow vector, which is nonregular. The optimal primal cost is 0. The dual problem is to maximize

$$\frac{1}{4(p_1 - p_2)} - \frac{1}{4(p_2 - p_1)}$$

over all (p_1, p_2) with $p_1 < p_2$, and has no optimal solution. The dual optimal cost is 0. Note that the optimal primal and dual costs are equal, consistently with the following Prop. 4.4.4.

equal under comparable assumptions to those for linear programs. This is

a remarkable result due to Minty [1960] and Rockafellar ([1967] or [1970] or [1984]), and requires a fairly sophisticated proof.

Proposition 4.4.4: (Duality Theorem for Separable Problems) If there exists at least one feasible solution to the primal problem, or at least one feasible solution to the dual problem, the optimal primal and dual costs are equal.

Note that part of the assertion of Prop. 4.4.4 is that if the primal problem is feasible but unbounded, then the dual problem is infeasible (the optimal costs of both problems are equal to $-\infty$), and that if the dual problem is feasible but unbounded, the primal problem is infeasible (the optimal costs of both problems are equal to ∞).

Duality and the Equilibrium Problem

We can use duality and CS to introduce a problem, which is referred to as the *equilibrium problem*. The name stems from the association with some classical problems of finding equilibrium solutions to various physical systems, as we will explain shortly.

Network Equilibrium Problem

Find a flow-price pair (x, p) such that x satisfies the conservation of flow equations, and for each arc (i, j), the pair $(x_{ij}, p_i - p_j)$ lies on the characteristic curve

$$\Gamma_{ij} = \big\{(x_{ij}, t_{ij}) \mid x_{ij} \in X_{ij},\, f_{ij}^-(x_{ij}) \leq t_{ij} \leq f_{ij}^+(x_{ij})\big\}. \qquad (4.44)$$

Thus, the pair (x, p) is an equilibrium solution if and only if x is feasible and (x, p) satisfies CS. We have the following result:

Proposition 4.4.5: (Network Equilibrium Theorem) A flow-price pair (x^*, p^*) solves the equilibrium problem if and only if x^* and p^* are optimal primal and dual solutions, respectively

Proof: If (x^*, p^*) solve the equilibrium problem, then (x^*, p^*) satisfy CS, so by the forward part of Prop. 4.4.2, x^* is primal optimal and p^* is dual optimal. Conversely, if x^* is primal optimal and p^* is dual optimal, then x^* is primal feasible, so by Prop. 4.4.4, the optimal primal and dual costs are equal. It follows using the reverse part of Prop. 4.4.2 that x^* and p^* satisfy CS, and since x^* is feasible, they also solve the equilibrium problem. **Q.E.D.**

4.4.4 Auction Algorithms

We first develop an appropriate extension of the notion of ϵ-complementary slackness (ϵ-CS for short). We then derive and analyze generalizations of the ϵ-relaxation and auction/sequential shortest path methods. Throughout this section, *we assume that the problem is feasible*.

Definition 4.4.3: Given $\epsilon \geq 0$, a flow-price vector pair (x, p) is said to satisfy ϵ-CS if for all arcs (i, j), we have $x_{ij} \in X_{ij}$ and

$$f_{ij}^-(x_{ij}) - \epsilon \leq p_i - p_j \leq f_{ij}^+(x_{ij}) + \epsilon.$$

Figure 4.4.5 illustrates the definition of ϵ-CS. The intuition behind the ϵ-CS conditions is that a feasible flow-price pair is "approximately" primal and dual optimal if the ϵ-CS conditions are satisfied. This intuition is quantified in the following proposition:

Proposition 4.4.6: Let $\big(x(\epsilon), p(\epsilon)\big)$ be a flow-price pair satisfying ϵ-CS such that $x(\epsilon)$ is feasible, and let $\xi(\epsilon)$ be any flow vector satisfying CS together with $p(\epsilon)$ [note that $\xi(\epsilon)$ need not satisfy the conservation of flow constraints].

(a)

$$0 \leq f\big(x(\epsilon)\big) - q\big(p(\epsilon)\big) \leq \epsilon \sum_{(i,j)\in\mathcal{A}} \left|x_{ij}(\epsilon) - \xi_{ij}(\epsilon)\right|. \qquad (4.45)$$

(b) Assume that all the dual arc cost functions q_{ij} are real-valued. Then

$$\lim_{\epsilon\to 0}\big(f\big(x(\epsilon)\big) - q\big(p(\epsilon)\big)\big) = 0.$$

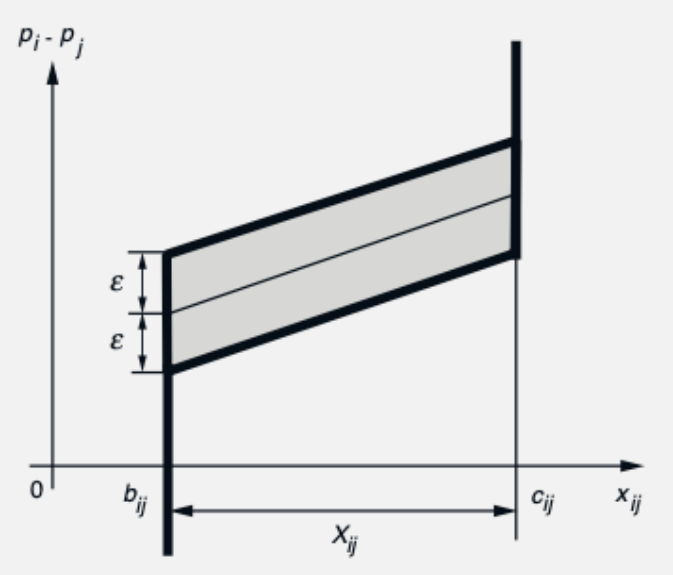

Figure 4.4.5: A visualization of the ϵ-CS conditions in terms of a "cylinder" around the characteristic curve. The shaded area represents flow-price differential pairs that satisfy the ϵ-CS conditions. In this figure, f_{ij} is a quadratic function whose curvature is the slope shown, and the arc flow range X_{ij} is the interval $[b_{ij}, c_{ij}]$.

Proof: (a) To simplify notation, let us replace $x(\epsilon)$, $p(\epsilon)$, and $\xi(\epsilon)$, by x, p, and ξ, respectively. Denote $t_{ij} = p_i - p_j$. Since ξ and p satisfy CS, we have

$$f_{ij}(x_{ij}) = \xi_{ij} t_{ij} + q_{ij}(t_{ij}), \qquad \forall\ (i,j) \in \mathcal{A}.$$

Take an arc (i,j) such that $x_{ij} \geq \xi_{ij}$. Then

$$f_{ij}(x_{ij}) + (\xi_{ij} - x_{ij}) f_{ij}^-(x_{ij}) \leq f_{ij}(\xi_{ij}) = \xi_{ij} t_{ij} + q_{ij}(t_{ij}).$$

Hence

$$f_{ij}(x_{ij}) - q_{ij}(t_{ij}) \leq (x_{ij} - \xi_{ij})\big(f_{ij}^-(x_{ij}) - t_{ij}\big) + x_{ij} t_{ij} \leq |x_{ij} - \xi_{ij}|\epsilon + x_{ij} t_{ij},$$

where the second inequality follows from ϵ-CS. This inequality is similarly obtained when $x_{ij} \leq \xi_{ij}$, so we have

$$f_{ij}(x_{ij}) - q_{ij}(t_{ij}) \leq |x_{ij} - \xi_{ij}|\epsilon + x_{ij} t_{ij}, \qquad \forall\ (i,j) \in \mathcal{A}.$$

From the definition of q_{ij}, we also have

$$x_{ij} t_{ij} \leq f_{ij}(x_{ij}) - q_{ij}(t_{ij}), \qquad \forall\ (i,j) \in \mathcal{A}.$$

By combining these two inequalities and adding over all arcs, we obtain

$$\sum_{(i,j)\in\mathcal{A}} x_{ij}t_{ij} \leq \sum_{(i,j)\in\mathcal{A}} \big(f_{ij}(x_{ij})-q_{ij}(t_{ij})\big) \leq \epsilon \sum_{(i,j)\in\mathcal{A}} |x_{ij}-\xi_{ij}|+\sum_{(i,j)\in\mathcal{A}} x_{ij}t_{ij}.$$

Since x is feasible, we have

$$\sum_{(i,j)\in\mathcal{A}} x_{ij}t_{ij} = \sum_{i\in\mathcal{N}} p_i \left(\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} \right) = \sum_{i\in\mathcal{N}} p_i s_i.$$

Combining the last two relations, we obtain

$$0 \leq \sum_{(i,j)\in\mathcal{A}} \big(f_{ij}(x_{ij}) - q_{ij}(t_{ij})\big) - \sum_{i\in\mathcal{N}} p_i s_i \leq \epsilon \sum_{(i,j)\in\mathcal{A}} |x_{ij} - \xi_{ij}|.$$

Using the definitions of $f(x)$ and $q(p)$, this relation is seen to be equivalent to the desired Eq. (4.45).

(b) We first argue by contradiction that $x(\epsilon)$ remains bounded as $\epsilon \to \infty$. Indeed, if this is not so, then since $x(\epsilon)$ is feasible for all ϵ, there exists a cycle C and a sequence ϵ_k converging to 0 such that $x_{ij}(\epsilon_k) \to \infty$ for all $(i,j) \in C^+$ and $x_{ij}(\epsilon_k) \to -\infty$ for all $(i,j) \in C^-$. Since all q_{ij} are assumed real-valued, we must have

$$\lim_{\xi\to\infty} f_{ij}^-(\xi) = \infty, \qquad \forall\ (i,j) \in C^+,$$

$$\lim_{\xi\to-\infty} f_{ij}^+(\xi) = -\infty, \qquad \forall\ (i,j) \in C^+.$$

This implies that for k sufficiently large,

$$t_{ij}(\epsilon_k) \geq f_{ij}^-\big(x_{ij}(\epsilon_k)\big) - \epsilon_k > t_{ij}(\epsilon_0), \qquad \forall\ (i,j) \in C^+, \tag{4.46}$$

$$t_{ij}(\epsilon_k) \leq f_{ij}^+\big(x_{ij}(\epsilon_k)\big) - \epsilon_k < t_{ij}(\epsilon_0), \qquad \forall\ (i,j) \in C^-. \tag{4.47}$$

On the other hand, since $t_{ij}(\epsilon_k) = p_i(\epsilon_k) - p_j(\epsilon_k)$, we have

$$\sum_{(i,j)\in C^+} t_{ij}(\epsilon_k) - \sum_{(i,j)\in C^-} t_{ij}(\epsilon_k) = 0, \qquad \forall\ k,$$

which contradicts Eqs. (4.46) and (4.47). Therefore $x(\epsilon)$ is bounded as $\epsilon \to 0$.

We will now show that $\xi_{ij}(\epsilon) - x_{ij}(\epsilon)$ is bounded for all arcs (i,j) as $\epsilon \to 0$, where $\xi(\epsilon)$ is any flow vector satisfying CS together with $p(\epsilon)$, i.e., for all arcs (i,j), we have

$$\xi_{ij}(\epsilon) \in X_{ij}, \qquad f_{ij}^-\big(\xi_{ij}(\epsilon)\big) \leq t_{ij}(\epsilon) \leq f_{ij}^+\big(\xi_{ij}(\epsilon)\big).$$

If the interval X_{ij} is unbounded above, we have $f_{ij}^-(\xi) \to \infty$ as $\xi \to \infty$. Since $x_{ij}(\epsilon)$ is bounded, we have that $t_{ij}(\epsilon)$ is bounded from above, which in turn implies that $\xi_{ij}(\epsilon)$ is bounded from above. Similarly, we can argue that $\xi_{ij}(\epsilon)$ is bounded from below. Therefore, $\xi_{ij}(\epsilon)$ is bounded for all arcs (i,j) as $\epsilon \to 0$, and it follows that $|x_{ij}(\epsilon) - \xi_{ij}(\epsilon)|$ is also bounded for all arcs (i,j) as $\epsilon \to 0$. This, together with Eq. (4.45), which was shown earlier, completes the proof. **Q.E.D.**

Proposition 4.4.6 does not tell us how small ϵ must be to achieve a certain tolerance for the sum $f\big(x(\epsilon)\big) - q\big(p(\epsilon)\big)$. On the other hand, if the the lengths of the intervals X_{ij} are bounded by some constant $L > 0$, then from Eq. (4.45), we obtain

$$f\big(x(\epsilon)\big) - q\big(p(\epsilon)\big) \leq \epsilon AL,$$

where A is the number of arcs.

For the remainder of this section, *we assume that the dual arc cost functions q_{ij} are real-valued*, as in Prop. 9.7(b). This is true in particular if the intervals X_{ij} are compact, or if $\lim_{x_{ij}\to\infty} f^+(x_{ij}) = \infty$ and $\lim_{x_{ij}\to-\infty} f^-(x_{ij}) = -\infty$ for all arcs (i,j).

We introduce a *generic auction algorithm*, whereby x and p are alternately adjusted so as to drive the excesses

$$g_i = \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} - \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} + s_i$$

to zero while maintaining ϵ-CS at all iterations. The only additional requirements are that nodes with nonnegative excess continue to have nonnegative excess and that price changes are effected by increasing the price of a node with positive excess by the maximum amount possible. We then consider two special cases of this generic algorithm. The first is the ϵ-relaxation method, which generalizes the method of Section 7.4; the second is the auction/sequential shortest path algorithm, which generalizes the method of Section 7.5.

Given a flow-price vector pair (x,p) satisfying ϵ-CS, an iteration of the generic auction algorithm updates (x,p) as follows:

Iteration of the Generic Auction Algorithm

If there is no node with positive excess, terminate the algorithm. Otherwise, perform one of the following two operations:

(a) **(Flow change)** Adjust the flow vector x in a way that ϵ-CS is maintained and all nodes with nonnegative excess continue to have nonnegative excess. (Here p is unchanged.)

(b) **(Price rise)** Increase the price p_i of some node i with positive excess by the maximum amount that maintains ϵ-CS. (Here x and all other coordinates of p are unchanged.)

Upon termination of the generic auction algorithm, the flow-price vector pair (x, p) satisfies ϵ-CS and all nodes have excess that is non-positive (and is equal to 0 since the problem is assumed to feasible). Thus, the validity of the method rests on whether it terminates finitely. The following proposition shows that the total number of price rises is finite under a suitable assumption.

Proposition 4.4.7: Let r be any nonnegative scalar such that the initial price vector p^0 for the generic auction algorithm satisfies $r\epsilon$-CS together with some feasible flow vector x^0. Also, assume that each price rise on a node increases the price of that node by at least $\beta\epsilon$, for some fixed $\beta \in (0, 1)$. Then, the method performs at most $(r + 1)(N - 1)/\beta$ price rises on each node.

Proof: Consider the pair (x, p) at the beginning of an iteration of the generic method. Since the excess vector $g = (g_1, \ldots, g_N)$ is not zero, and the flow vector x^0 is feasible, we conclude that for each node s with $g_s > 0$ there exists a node t with $g_t < 0$ and a simple path P from t to s such that:

$$x_{ij} > x^0_{ij}, \qquad \forall\ (i, j) \in P^+, \tag{4.48}$$

$$x_{ij} < x^0_{ij}, \qquad \forall\ (i, j) \in P^-, \tag{4.49}$$

where P^+ is the set of forward arcs of P and P^- is the set of backward arcs of P. [This can be seen from the conformal realization theorem (Prop. 1.1) as follows. For the flow vector $x - x^0$, the excess of node t is $-g_t > 0$ and the excess of node s is $-g_s < 0$. Hence, by the conformal realization

theorem, there is a simple path P from t to s that conforms to the flow $x - x^0$, that is, $x_{ij} - x^0_{ij} > 0$ for all $(i,j) \in P^+$ and $x_{ij} - x^0_{ij} < 0$ for all $(i,j) \in P^-$.]

From Eqs. (4.48) and (4.49), and the convexity of the functions f_{ij} for all $(i,j) \in \mathcal{A}$, we have

$$f^-_{ij}(x_{ij}) \geq f^+_{ij}(x^0_{ij}), \qquad \forall\ (i,j) \in P^+, \tag{4.50}$$

$$f^+_{ij}(x_{ij}) \leq f^-_{ij}(x^0_{ij}), \qquad \forall\ (i,j) \in P^-. \tag{4.51}$$

Since the pair (x,p) satisfies ϵ-CS, we also have that

$$p_i - p_j \in [f^-_{ij}(x_{ij}) - \epsilon, f^+_{ij}(x_{ij}) + \epsilon], \qquad \forall\ (i,j) \in \mathcal{A}. \tag{4.52}$$

Similarly, since the pair (x^0, p^0) satisfies $r\epsilon$-CS, we have

$$p^0_i - p^0_j \in [f^-_{ij}(x^0_{ij}) - r\epsilon, f^+_{ij}(x^0_{ij}) + r\epsilon], \qquad \forall\ (i,j) \in \mathcal{A}. \tag{4.53}$$

Combining Eqs. (4.50), (4.52), and (4.53), we obtain for all $(i,j) \in P^+$,

$$p_i - p_j \geq f^-_{ij}(x_{ij}) - \epsilon \geq f^+_{ij}(x^0_{ij}) - \epsilon \geq p^0_i - p^0_j - (r+1)\epsilon.$$

Similarly, combining Eqs. (4.51)-(4.53), we obtain for all $(i,j) \in P^-$,

$$p_i - p_j \leq p^0_i - p^0_j + (r+1)\epsilon.$$

Applying the above inequalities for all arcs of the path P, we get

$$p_t - p_s \geq p^0_t - p^0_s - (r+1)|P|\epsilon, \tag{4.54}$$

where $|P|$ denotes the number of arcs of the path P. Since only nodes with positive excess can change their prices and nodes with nonnegative excess continue to have nonnegative excess, it follows that if a node has negative excess at some time, then its price is unchanged from the beginning of the method until that time. Thus $p_t = p^0_t$. Since the path is simple, we also have that $|P| \leq N - 1$. Therefore, Eq. (4.54) yields

$$p_s - p^0_s \leq (r+1)|P|\epsilon \leq (r+1)(N-1)\epsilon. \tag{4.55}$$

Since only nodes with positive excess can increase their prices and, by assumption, each price rise increment is at least $\beta\epsilon$, we conclude from Eq. (4.55) that the total number of price rises that can be performed for node s is at most $(r+1)(N-1)/\beta$. **Q.E.D.**

The preceding proposition shows that the bound on the number of price rises is independent of the cost functions, but depends only on

$$r^0 = \min \big\{ r \in [0,\infty) \mid (x^0,p^0) \text{ satisfies } r\epsilon\text{-CS} \\ \text{for some feasible flow vector } x^0 \big\},$$

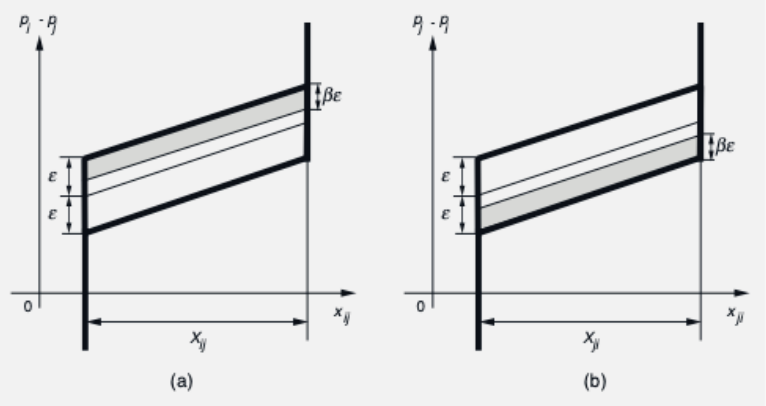

Figure 4.4.6: Visualization of the conditions satisfied by a candidate-list arc. The shaded area represents flow-price differential pairs corresponding to a candidate-list arc $(i,j) \in L^+(i)$ in figure (a), and to a candidate-list arc $(j,i) \in L^-(i)$ in figure (b). Note that at the right endpoint of X_{ij} the right derivative f_{ij}^+ is ∞, so at the right endpoint, $L^+(i)$ is empty. Similarly, at the left endpoint, $L^-(i)$ is empty.

which is the minimum multiplicity of ϵ with which CS is violated by the initial price vector together with some feasible flow vector. Note that r^0 is well defined for any p^0 because, for all r sufficiently large, $r\epsilon$-CS is satisfied by p^0 and any feasible flow vector.

To ensure that the number of flow changes between successive price rises is finite and that each price rise is at least $\beta\epsilon$, we need to further specify how the price rises and flow changes should be effected. We thus proceed to introduce the key mechanisms for achieving this.

For any $\epsilon > 0$, any $\beta \in (0,1)$, and any flow-price vector pair (x,p) satisfying ϵ-CS, we define for each node $i \in \mathcal{N}$ its *candidate list* as the union of the following two sets of arcs

$$L^+(i) = \left\{(i,j) \in \mathcal{A} \mid \ (1-\beta)\epsilon < p_i - p_j - f_{ij}^+(x_{ij}) \leq \ \epsilon\right\}, \tag{4.56}$$

$$L^-(i) = \left\{(j,i) \in \mathcal{A} \mid -(1-\beta)\epsilon > p_j - p_i - f_{ji}^-(x_{ji}) \geq -\epsilon\right\}. \tag{4.57}$$

The arcs of the candidate list can be visualized in terms of the characteristic curves

$$\Gamma_{ij} = \left\{(x_{ij}, t_{ij}) \in \Re^2 \mid f_{ij}^-(x_{ij}) \leq t_{ij} \leq f_{ij}^+(x_{ij})\right\}.$$

Thus, (i,j) is in the candidate list of i (respectively, j) if $(x_{ij}, p_i - p_j)$ belongs to the "strip" at height between $(1-\beta)\epsilon$ and ϵ above (respectively, below) Γ_{ij} (see Fig. 4.4.6).

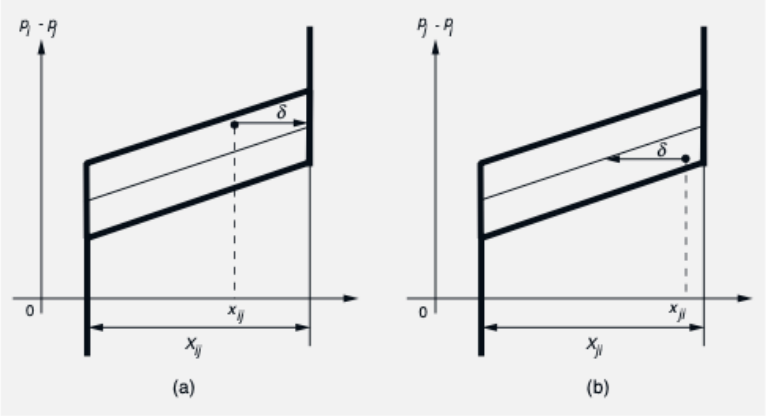

Figure 4.4.7: Illustration of the flow margin δ of a candidate-list arc $(i,j) \in L^+(i)$ in figure (a), and to a candidate-list arc $(j,i) \in L^-(i)$ in figure (b).

For each arc (i,j) [respectively, (j,i)] in the candidate list of i, the supremum of δ for which

$$p_i - p_j \geq f_{ij}^+(x_{ij} + \delta)$$

[respectively, $p_j - p_i \leq f_{ji}^-(x_{ji} - \delta)$] is called the *flow margin* of the arc (see Fig. 4.4.7). An important fact, shown below, is that the flow margins of these arcs are always positive.

Proposition 4.4.8: All arcs in the candidate list of a node have positive flow margins.

Proof: Assume that for an arc $(i,j) \in \mathcal{A}$ the flow margin is not positive; that is, we have

$$p_i - p_j < f_{ij}^+(x_{ij} + \delta), \quad \forall\ \delta > 0.$$

Since the function f_{ij}^+ is right continuous, this yields

$$p_i - p_j \leq \lim_{\delta \downarrow 0} f_{ij}^+(x_{ij} + \delta) = f_{ij}^+(x_{ij}),$$

and thus, based on the definition of Eq. (4.56), (i,j) cannot be in the candidate list of node i. A similar argument shows that an arc $(j,i) \in \mathcal{A}$ such that

$$p_j - p_i > f_{ji}^-(x_{ji} - \delta), \quad \forall\ \delta > 0,$$

cannot be in the candidate list of node i. **Q.E.D.**

The method that we will use for flow changes is to decrease the excess of a node with positive excess by changing the flow of candidate-list arcs. This can be done either one arc at a time, as in the case of the ϵ-relaxation method of Section 4.1, or one path of arcs at a time, as in the case of the auction/sequential-shortest-path algorithm of Section 4.3. When the candidate list of the node is empty, we perform a price rise on the node. An important fact, shown below, is that the price rise increment for a node with empty candidate list is at least $\beta\epsilon$.

Proposition 4.4.9: If we perform a price rise on a node whose candidate list is empty, then the price of that node will increase by at least $\beta\epsilon$.

Proof: If the candidate list of a node i is empty, then for every arc $(i,j) \in \mathcal{A}$ we have $p_i - p_j - f_{ij}^+(x_{ij}) \leq (1-\beta)\epsilon$, and for every arc $(j,i) \in \mathcal{A}$ we have $p_j - p_i - f_{ji}^-(x_{ji}) \geq -(1-\beta)\epsilon$. This implies that the numbers

$$p_j - p_i + f_{ij}^+(x_{ij}) + \epsilon, \qquad \forall\ (i,j) \in \mathcal{A},$$

$$p_j - p_i - f_{ji}^-(x_{ji}) + \epsilon, \qquad \forall\ (j,i) \in \mathcal{A},$$

are all greater than or equal to $\beta\epsilon$. Since a price rise on i adds to p_i the minimum of all these numbers, the result follows. **Q.E.D.**

For any $\epsilon > 0$, any $\beta \in (0,1)$, and any flow-price vector pair (x,p) satisfying ϵ-CS, let us consider the arc set $\mathcal{A}^*$ that contains all candidate list arcs oriented in the direction of flow change. In particular, for each arc (i,j) in the forward portion $L^+(i)$ of the candidate list of a node i, we introduce an arc (i,j) in $\mathcal{A}^*$ and for each arc (j,i) in the backward portion $L^-(i)$ of the candidate list of node i, we introduce an arc (i,j) in $\mathcal{A}^*$ (thus the direction of the latter arc is reversed). The set of nodes $\mathcal{N}$ and the set $\mathcal{A}^*$ define the *admissible graph* $\mathcal{G}^* = (\mathcal{N}, \mathcal{A}^*)$. We will consider methods that keep $\mathcal{G}^*$ acyclic at all iterations. Intuitively, because we move flow in the direction of the arcs in $\mathcal{G}^*$, keeping $\mathcal{G}^*$ acyclic helps to limit the number of flow changes between price rises, as we have seen in Section 7.4. To ensure that initially the admissible graph is acyclic, one possibility is to choose, for any initial price vector p^0, the initial flow vector x^0 such that (x^0, p^0) satisfies 0-CS, that is,

$$f_{ij}^-(x_{ij}^0) \leq p_i^0 - p_j^0 \leq f_{ij}^+(x_{ij}^0), \qquad \forall\ (i,j) \in \mathcal{A}. \tag{4.58}$$

With this choice, ϵ-CS is satisfied by (x^0, p^0) for any $\epsilon > 0$, and the initial admissible graph is empty and thus acyclic.

In the next two sections, we will study two specializations of the generic auction algorithm. These methods perform flow changes by moving flow out of nodes with positive excess along candidate-list arcs and they perform price rises only on nodes with empty candidate lists. In addition, they keep the admissible graph acyclic at all iterations and have favorable complexity bounds.

4.4.5 The ϵ-Relaxation Method

For fixed $\epsilon > 0$ and $\beta \in (0, 1)$, and a given flow-price vector pair (x, p) satisfying ϵ-CS, an iteration of the ϵ-relaxation method updates (x, p) as follows:

Iteration of the ϵ-Relaxation Method

Step 1: Select a node i with positive excess g_i; if no such node exists, terminate the method.

Step 2: (δ-Flow push) If the candidate list of i is empty, go to Step 3. Otherwise, choose an arc from the candidate list of i, and let

$$\delta = \min\{g_i, \text{ flow margin of the chosen arc}\}.$$

Increase x_{ij} by δ if (i, j) is the arc, or decrease x_{ji} by δ if (j, i) is the arc. If as a result the excess of i becomes zero, go to the next iteration; otherwise, go to Step 2.

Step 3: (Price rise) Increase the price p_i by the maximum amount that maintains ϵ-CS. Go to the next iteration.

To see that the ϵ-relaxation method is a specialization of the generic auction method of Section 2, note that Step 3 is a price rise on node i and that Step 2 adjusts the flows in such a way that ϵ-CS is maintained and nodes with nonnegative excess continue to have nonnegative excess for all subsequent iterations. The reason for the latter is that when iterating at a node i, a flow push cannot make the excess of i negative (by the choice of δ in Step 2), and cannot decrease the excess of neighboring nodes. Furthermore, the ϵ-relaxation method performs a price rise only on nodes with empty candidate list. Then, by Prop. 4.4.9, each price rise increment is at least $\beta\epsilon$ and, by Prop. 4.4.7, the number of price rises (i.e., Step 3) on each node is at most $(r + 1)(N - 1)/\beta$, where r is any nonnegative scalar such that the initial price vector satisfies $r\epsilon$-CS together with some feasible flow vector. Thus, to prove finite termination of the ϵ-relaxation method, it suffices to show that the number of flow pushes (i.e., Step 2) performed between successive price rises is finite. We show this by first showing that the method maintains the acyclicity of the admissible graph.

Proposition 4.4.10: If the admissible graph is initially acyclic, then it remains acyclic at all iterations of the ϵ-relaxation method.

Proof: We use induction. Initially, the admissible graph $\mathcal{G}^*$ is acyclic by assumption. Assume that $\mathcal{G}^*$ remains acyclic for all subsequent iterations up to the mth iteration for some m. We will prove that after the mth iteration $\mathcal{G}^*$ remains acyclic. Clearly, after a flow push in Step 2, the admissible graph remains acyclic, since it either remains unchanged, or some arcs are deleted from it. Thus we only have to prove that after a price rise on a node i, no cycle involving i is created. We note that, after a price rise on node i, all incident arcs to i in the admissible graph at the start of the mth iteration are deleted and new arcs incident to i are added. We claim that i cannot have any incoming arcs that belong to the admissible graph. To see this, note that just before a price rise on node i, we have

$$p_j - p_i - f_{ji}^+(x_{ji}) \leq \epsilon, \qquad \forall\ (j,i) \in \mathcal{A},$$

and since each price rise increment is at least $\beta\epsilon$, we must have

$$p_j - p_i - f_{ji}^+(x_{ji}) \leq (1-\beta)\epsilon, \qquad \forall\ (j,i) \in \mathcal{A},$$

after the price rise. Then, by Eq. (4.56), (j,i) cannot be in the candidate list of node j. By a similar argument, we have that (i,j) cannot be in the candidate list of j for all $(i,j) \in \mathcal{A}$. Thus, after a price rise on node i, we see that i cannot have any incoming arcs belonging to the admissible graph, so no cycle involving i can be created. **Q.E.D.**

We say that a node i is a *predecessor* of a node j in the admissible graph $\mathcal{G}^*$ if a directed path (i.e., a path having no backward arc) from i to j exists in $\mathcal{G}^*$. Node j is then called a *successor* of i. Observe that, in the ϵ-relaxation method, flow is pushed towards the successors of a node and if $\mathcal{G}^*$ is acyclic, flow cannot be pushed from a node to any of its predecessors. A δ-flow push along an arc in $\mathcal{A}$ is said to be *saturating* if the flow increment δ is equal to the flow margin of the arc. By our choice of δ in the ϵ-relaxation method, a nonsaturating flow push always exhausts (i.e., sets to zero) the excess of the starting node of the arc. Then, by using Prop. 9.11, we obtain the following result.

Proposition 4.4.11: If the admissible graph is initially acyclic, then the number of flow pushes between two successive price rises (not necessarily at the same node) performed by the ϵ-relaxation method is finite. Furthermore, the algorithm terminates with a flow-price pair satisfying ϵ-CS.

Proof: We observe that a saturating flow push along an arc removes the arc from the admissible graph, while a nonsaturating flow push does not add a new arc to the admissible graph. Thus the number of saturating flow pushes that can be performed between successive price rises is at most A. It will thus suffice to show that the number of nonsaturating flow pushes that can be performed between saturating flow pushes is finite. Assume the contrary, that is, there is an infinite sequence of successive nonsaturating flow pushes, with no intervening saturating flow push. Then the admissible graph remains fixed throughout this sequence. Furthermore, the excess of some node i_0 must be exhausted infinitely often during this sequence. This can happen only if the excess of some predecessor i_1 of i_0 is exhausted infinitely often during the sequence. Continuing in this manner, we construct an infinite sequence of predecessor nodes $\{i_k\}$. Thus, some node in this sequence must be repeated, which is a contradiction since the admissible graph is acyclic. Hence, the number of flow pushes between two successive price rises is finite. Since the number of price rises is finite (cf. Props. 4.4.7 and 4.4.9), termination of the algorithm follows. **Q.E.D.**

By refining the proof of Prop. 9.12, we can further show that the number of flow pushes between successive price rises is at most $(N+1)A$, from which a complexity bound for the ϵ-relaxation method may be readily derived. However, we will focus on a special implementation of the method for which we will derive a more favorable running time.

Efficient Implementation

Let us consider a generalization of the sweep implementation, discussed in Section 7.4. This implementation defines the order in which nodes are selected for an ϵ-relaxation iteration. In particular, the nodes are maintained in a linked list T, which is traversed from the first to the last element. The order of the nodes in the list is consistent with the successor order implied by the admissible graph; that is, if a node j is a successor of a node i, then j must appear after i in the list. If the initial admissible graph is empty, as is the case with the initialization of Eq. (4.58), the initial list is arbitrary. Otherwise, the initial list must be consistent with the successor order of

the initial admissible graph. The list is updated in a way that maintains the consistency with the successor order. In particular, let i be the node chosen in Step 1 of the iteration, and let N_i be the subset of nodes of T that are after i in T. If the price of i changes in this iteration, then node i is removed from its position in T and placed in the first position of T. The node chosen in the next iteration, if N_i is nonempty, is the node $i' \in N_i$ with positive excess which ranks highest in T. Otherwise, the positive excess node ranking highest in T is chosen. It can be seen as in Section 7.4 that, with this rule of repositioning the nodes following a price change, the list order is consistent with the successor order implied by the admissible graph at all iterations.

The next proposition gives a bound on the number of flow pushes made by the sweep implementation of the ϵ-relaxation method. This result is based on the observations that (a) between successive saturating flow pushes on an arc, there is at least one price rise performed on one of the end nodes of the arc, and (b) between successive price rises (not necessarily at the same node), the number of nonsaturating flow pushes is at most N. The proof parallels the one given in Section 7.4, and will be omitted.

Proposition 4.4.12: Let r be any nonnegative scalar such that the initial price vector for the sweep implementation of the ϵ-relaxation method satisfies $r\epsilon$-CS together with some feasible flow vector. Then, the number of price rises on each node, the number of saturating flow pushes, and the number of nonsaturating flow pushes up to termination of the method are $O(rN)$, $O(rNA)$, and $O(rN^3)$, respectively.

We now derive the running time for the sweep implementation of the ϵ-relaxation method. The dominant computational requirements are:

(1) The computation required for price rises.

(2) The computation required for saturating flow pushes.

(3) The computation required for nonsaturating flow pushes.

In contrast to the linear cost case, we cannot express the running time in terms of the size of the problem data since the latter is not well defined for convex cost functions. Instead, we introduce a set of simple operations performed by the ϵ-relaxation method, and we estimate the number of these operations. In particular, in addition to the usual arithmetic operations with real numbers, we consider the following operations:

(a) Given the flow x_{ij} of an arc (i,j), calculate the cost $f_{ij}(x_{ij})$, the left derivative $f_{ij}^-(x_{ij})$, and the right derivative $f_{ij}^+(x_{ij})$.

(b) Given the price differential $t_{ij} = p_i - p_j$ of an arc (i,j), calculate $\sup\{\xi \mid f_{ij}^+(\xi) \leq t_{ij}\}$ and $\inf\{\xi \mid f_{ij}^-(\xi) \geq t_{ij}\}$.

Operation (a) is needed to compute the candidate list of a node and a price increase increment; operation (b) is needed to compute the flow margin of an arc and the flow initialization of Eq. (4.58). Complexity will thus be measured in terms of the total number of operations performed by the method, as in the following proposition, which follows from Prop. 9.13.

Proposition 4.4.13: Let r be any nonnegative scalar such that the initial price vector for the sweep implementation of the ϵ-relaxation method satisfies $r\epsilon$-CS together with some feasible flow vector. Then, the method requires $O(rN^3)$ operations up to termination.

The theoretical and the practical performance of the ϵ-relaxation method can be further improved by ϵ-*scaling*, whereby we apply the ϵ-relaxation method several times, starting with a large value of ϵ, say ϵ^0, and successively reduce ϵ up to a final value, say $\bar{\epsilon}$, that will give the desirable degree of accuracy to our solution. Furthermore, the price and flow information from one application of the method is passed to the next. Similar to Section 7.4, it can be shown that if ϵ^0 is chosen sufficiently large so that the initial price vector satisfies ϵ^0-CS together with some feasible flow vector, then the running time of the ϵ-relaxation method using the sweep implementation and ϵ-scaling is $O\big(N^3 \ln(\epsilon^0/\bar{\epsilon})\big)$ operations.

On the other hand, contrary to what complexity analysis suggests, it is not clear whether the candidate list organization of the sweep implementation improves the practical performance, in view of the additional overhead it requires.

4.4.6 Auction/Sequential Shortest Path Algorithm

We now consider the extension of the auction/sequential shortest path (ASSP) algorithm of Section 7.5. The algorithm is a special case of the generic auction method, and differs from the ϵ-relaxation method in that instead of pushing flow along a candidate-list arc to any node, it pushes flow along a path of candidate-list arcs ending at a node with negative excess. In fact, whereas a flow push in the ϵ-relaxation method may increase the excess of a node in absolute value (e.g., when flow is pushed to a neighboring node with nonnegative excess), in the ASSP algorithm, the excess of each node is nonincreasing in absolute value.

We first introduce some definitions. For a path P, we denote by $s(P)$ and $t(P)$ the starting node and the terminal node, respectively, of P. For any $\epsilon > 0$ and $\beta \in (0, 1)$, and any flow-price vector pair (x, p) satisfying ϵ-CS, we say that a path P of a graph $(\mathcal{N}, \mathcal{A})$ is *augmenting* if each forward (respectively, backward) arc (i, j) of P is in the candidate

list of i (respectively, j) and $s(P)$ is a *source* (i.e., has positive excess) and $t(P)$ is a *sink* (i.e., has negative excess). As in Section 7.5, we define two operations on a given path $P = (n_1, n_2, \ldots, n_k)$:

(a) A *contraction* of P, which deletes the terminal node of P and the arc incident to this node.

(b) An *extension* of P by an arc (n_k, n_{k+1}) or an arc (n_{k+1}, n_k), which replaces P by the path $(n_1, n_2, \ldots, n_k, n_{k+1})$ and adds to P the corresponding arc.

For a fixed $\epsilon > 0$ and $\beta \in (0, 1)$, and a given flow-price vector pair (x, p) satisfying ϵ-CS, an iteration of the ASSP algorithm updates (x, p) as follows:

Iteration of the ASSP Algorithm

Step 1: Select a node i with positive excess and let the path P consist of only this node; if no such node exists, terminate the algorithm.

Step 2: Let i be the terminal node of the path P. If the candidate list of i is empty, then go to Step 3; otherwise, go to Step 4.

Step 3: (Contract Path) Increase the price p_i by the maximum amount that maintains ϵ-CS. If $i \neq s(P)$, contract P. Go to Step 2.

Step 4: (Extend Path) Select an arc (i, j) [or (j, i)] from the candidate list of i and extend P by this arc. If the excess of j is negative, go to Step 5; otherwise, go to Step 2.

Step 5: (Augmentation) Perform an *augmentation* along the path P by the amount

$$\delta = \min\left\{g_{s(P)}, -g_{t(P)}, \text{minimum of flow margins of the arcs of } P\right\},$$

(i.e., increase the flow of all forward arcs of P and decrease the flow of all backward arcs of P by δ). Go to the next iteration.

Roughly speaking, at each iteration of the ASSP algorithm, the path P starts as a single source and is successively extended or contracted until the terminal node of P is a sink. Then an augmentation along P is performed so as to decrease (respectively, increase) the excess of the starting node (respectively, terminal node), while leaving the excess of the remaining nodes unchanged. In case of a contraction, the price of the terminal node of P is strictly increased.

We note that the ASSP algorithm is a special case of the generic auction algorithm. To see this, note that Step 2 is a price rise on node i

and that Step 5 adjusts the flows in such a way that ϵ-CS is maintained and nodes with nonnegative excess continue to have nonnegative excess for all subsequent iterations. The reason for the latter is that an augmentation along P changes the excess of only two nodes $s(P)$ and $t(P)$, and by our choice of δ, the excess of the node $s(P)$ remains nonnegative after the augmentation.

We also note that the ASSP algorithm performs price rises only on nodes with empty candidate list. Thus, by Prop. 4.4.9, each price rise increment is at least $\beta\epsilon$ and, by Prop. 4.4.7, the number of price rises (i.e., path contractions) on each node is at most $(r+1)(N-1)/\beta$, where r is any nonnegative scalar such that the initial price vector satisfies $r\epsilon$-CS together with some feasible flow vector. It follows that to prove finite termination of the ASSP algorithm, it suffices to show that the number of path extensions (cf. Step 4) and the number of augmentations (cf. Step 5) performed between successive path contractions is finite. Similar to the case of the ϵ-relaxation method, we show this by first showing that the algorithm keeps the admissible graph acyclic and that the path P, when its backward arcs are reversed in direction, belongs to the admissible graph.

Proposition 4.4.14: If initially the admissible graph is acyclic, then the admissible graph remains acyclic at all iterations of the ASSP algorithm. Moreover, the path P maintained by the algorithm, when its backward arcs are reversed in direction, belongs to the admissible graph at all times.

Proof: The admissible graph can change either by a price rise (Step 3) or by an augmentation (Step 5). An augmentation keeps the admissible graph acyclic because, after an augmentation, the admissible graph either remains unchanged or some arcs are deleted from it. A price rise keeps the admissible graph acyclic, as was shown in the proof of Prop. 9.11.

To show that P, when its backward arcs are reversed in direction, belongs to the admissible graph at all times, we simply observe that a path extension maintains this property (since the arc added to P is in the candidate list of the terminal node of P) and that a path contraction also maintains this property (since a price rise on the terminal node of P changes the admissible graph only by adding/deleting arcs incident to this node and, after the contraction, this node and its incident arc in P are both deleted from P). **Q.E.D.**

We now use Prop. 4.4.14 to bound the number of augmentations and path extensions performed by the ASSP algorithm between successive path contractions. This shows that the algorithm terminates with a flow-price pair satisfying ϵ-CS.

Proposition 4.4.15: If initially the admissible graph is acyclic, then the number of augmentations and path extensions between two successive path contractions (not necessarily at the same node) performed by the ASSP algorithm is finite. Furthermore, the algorithm terminates with a flow-price pair satisfying ϵ-CS.

Proof: We observe that an augmentation does not increase the number of nodes with nonzero excess and does not add any arc to the admissible graph. Moreover, after an augmentation, either an arc is removed from the admissible graph or a node has its excess set to zero. Thus, the number of arcs in the admissible graph plus the number of nodes with nonzero excess is decreased by at least one after each augmentation. It follows that the number of augmentations between successive path contractions is at most $A + N$.

By Prop. 4.4.14, the path P always belongs to the admissible graph which is acyclic, so P cannot have repeated nodes and hence the number of successive extensions of P (before a contraction or an augmentation is performed) is at most N. Thus, the number of path extensions between successive path contractions is at most $N \cdot$ (number of augmentations between successive path contractions) $\leq N(A + N)$. Since the number of contractions is finite (cf. Props. 4.4.7 and 4.4.9), termination of the algorithm follows. **Q.E.D.**

4.5 THEORETICAL ASPECTS

In this section, we provide a more detailed analysis of the algorithms of this chapter, including some of theoretical results that are needed for this analysis;see also the book [Ber98], and the paper by Bertsekas, Polymanakos, and Tseng [BPT97].

Proof of Prop. 4.1.1

We prove Prop. 4.1.1, stated below:

Proposition: If $\epsilon < 1/N$, where N is the number of nodes, x is feasible, and x and p satisfy ϵ-CS, then x is optimal for the transhipment problem (4.1).

Proof: If x is not optimal, then by Prop. 1.2 in Section 1.2, there exists a simple cycle Y that has negative cost, i.e.,

$$\sum_{(i,j)\in Y^+} a_{ij} - \sum_{(i,j)\in Y^-} a_{ij} < 0, \tag{4.59}$$

and is unblocked with respect to x, i.e.,

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

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

By ϵ-CS [cf. Eqs. (4.3) and (4.4)], the preceding relations imply that

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

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

By adding these relations over all arcs of Y (whose number is no more than N), and by using the hypothesis $\epsilon < 1/N$, we obtain

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

Since the arc costs a_{ij} are integer, we obtain a contradiction of Eq. (4.59). **Q.E.D.**

Proof of Prop. 4.1.2

We prove Prop. 4.1.2, stated below:

> **Proposition:** Assume that the transhipment problem is feasible and that a_{ij}, b_{ij}, c_{ij}, and s_i are integer, and that $s_i \ge 0$ for all i. Then the ϵ-relaxation method terminates with a pair (x,p) satisfying ϵ-CS. The flow vector x is feasible, and is optimal if $\epsilon < 1/N$.

Proof: We first make a few observations.

(a) The algorithm preserves ϵ-CS; this can be verified from the price change formula (4.5).

(b) The prices of all nodes are monotonically nondecreasing during the algorithm [this follows from the ϵ-CS property of (x,p) and Eq. (4.5)].

(c) Once a node has nonnegative excess, its excess stays nonnegative thereafter, since a flow change in Step 2 or 3 at a node i cannot drive the excess of i below zero (since $\delta \leq g_i$), and cannot decrease the excess of neighboring nodes.

(d) If at some time a node has negative excess, its price must have never been increased up to that time, and must be equal to its initial price. This is a consequence of (c) above and of the assumption that only nodes with nonnegative excess can be chosen for iteration.

Suppose, to arrive at a contradiction, that the method does not terminate. Then, since there is at least one flow change per iteration, an infinite number of flow changes must be performed at some node i on some arc (i,j). Since for each flow change, the increment δ is integer, an infinite number of flow changes must also be performed at node j on the arc (i,j). This means that arc (i,j) becomes alternately ϵ^+-unblocked with $g_i > 0$ and ϵ^--unblocked with $g_j > 0$ an infinite number of times, which implies that p_i and p_j must increase by amounts of at least 2ϵ an infinite number of times. Thus we have $p_i \to \infty$ and $p_j \to \infty$, while either $g_i > 0$ or $g_j > 0$ at the start of an infinite number of flow changes.

Let $\mathcal{N}^\infty$ be the set of nodes whose prices increase to ∞. To preserve ϵ-CS, we must have, after a sufficient number of iterations,

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

$$x_{ji} = b_{ji} \qquad \text{for all } (j,i) \in \mathcal{A} \text{ with } i \in \mathcal{N}^\infty, j \notin \mathcal{N}^\infty.$$

After some iteration, by (d) above, every node in $\mathcal{N}^\infty$ must have nonnegative excess, so the sum of excesses of the nodes in $\mathcal{N}^\infty$ must be positive at the start of the flow changes where either $g_i > 0$ or $g_j > 0$. It follows that

$$0 < \sum_{i \in \mathcal{N}^\infty} s_i - \sum_{\{(i,j) \in \mathcal{A} \mid i \in \mathcal{N}^\infty,\, j \notin \mathcal{N}^\infty\}} c_{ij} + \sum_{\{(j,i) \in \mathcal{A} \mid i \in \mathcal{N}^\infty,\, j \notin \mathcal{N}^\infty\}} b_{ji}.$$

For any feasible vector, the above relation implies that the sum of the excesss of nodes in $\mathcal{N}^\infty$ exceeds the capacity of the cut $[\mathcal{N}^\infty, \mathcal{N} - \mathcal{N}^\infty]$, which is impossible. It follows that there is no feasible flow vector, contradicting the hypothesis. Thus the algorithm must terminate. Since upon termination we have $g_i \leq 0$ for all i and the problem is assumed feasible, it follows that $g_i = 0$ for all i. Hence the final flow vector x is feasible and by (a) above it satisfies ϵ-CS together with the final p. By Prop. 7.10, if $\epsilon < 1/N$, x is optimal. **Q.E.D.**

Computational Complexity Analysis of ϵ-Relaxation – ϵ-Scaling

We now discuss the running time of the ϵ-relaxation method. We first introduce some technical changes to the algorithm.

Let p^k denote the initial price vector for the $(k+1)$st scaling phase. We have $p^0 = 0$, and we assume that for $k \geq 1$, p^k is the price vector obtained at the end of the kth scaling phase. As in Chapter 2, at the beginning of the $(k+1)$st scaling phase, we make a correction of size at most ϵ^k to each a_{ij} so that it is divisible by ϵ^k [no correction is made in the last phase since the a_{ij} are integer and the final value of ϵ is $1/(N+1)$]. Thus the arc cost coefficients in the $(k+1)$st scaling phase, denoted a_{ij}^k, are all divisible by ϵ^k, and satisfy

$$|a_{ij}^k - a_{ij}| \leq \epsilon^k, \qquad \forall\ (i,j) \in \mathcal{A}.$$

The correction of the arc cost coefficients guarantees that all price rise increments and prices are integer multiples of the prevailing value of ϵ. The initial flow of each arc (i,j) for the $(k+1)$st scaling phase is

$$x_{ij} = \begin{cases} b_{ij} & \text{if } p_i^k - p_j^k \leq a_{ij}^k, \\ c_{ij} & \text{if } p_i^k - p_j^k > a_{ij}^k. \end{cases}$$

With this choice, the initial admissible graph is empty and is therefore acyclic.

We first focus on the case where ϵ is fixed, and we subsequently consider the ϵ-scaling case where ϵ is progressively reduced. We continue to assume that the problem data and the starting flows are integer. As in Section 2.4 (??), *for the case where ϵ is fixed, we assume that the cost coefficients a_{ij}, and all the initial node prices are integer multiples of ϵ.* Under this assumption, it is seen from the price change operation (4.5) in Step 4 that all node prices will be integer multiples of ϵ throughout the algorithm, implying that *each price rise is of size at least ϵ.*

For purposes of easy reference, let us call the operation of Step 4 a *price rise* at node i, and let us call the operation of Step 2 (or Step 3) a *flow push* on arc (i,j) [a flow push on arc (j,i), respectively]. A flow push on arc (i,j) [or arc (j,i)] is said to be *saturating* if it results in setting the flow x_{ij} to its upper bound c_{ij} (the flow x_{ji} to its lower bound b_{ij}, respectively); otherwise, the flow push is said to be *nonsaturating*. The complexity analysis revolves around bounding the number of price rises, and saturating and nonsaturating flow pushes. We first bound the number of price rises. The proof is given in Section 4.5.

Proposition 4.5.1: Assume that for some scalar $r \geq 1$, the initial price vector p^0 for the ϵ-relaxation method satisfies $r\epsilon$-CS together with some feasible flow vector x^0. Then, the ϵ-relaxation method performs at most $(r+1)(N-1)$ price rises per node.

Proof: Consider the pair (x,p) at the beginning of an ϵ-relaxation iteration. Since the excess vector $g=(g_1,\ldots,g_N)$ is not zero, and the flow vector x^0 is feasible, we conclude that for each node s with $g_s>0$ there exists a node t with $g_t<0$ and a path H from t to s that contains no cycles and is such that:

$$b_{ij}\leq x^0_{ij}<x_{ij}\leq c_{ij},\qquad \forall\ (i,j)\in H^+, \tag{4.60}$$

$$b_{ij}\leq x_{ij}<x^0_{ij}\leq c_{ij},\qquad \forall\ (i,j)\in H^-, \tag{4.61}$$

where H^+ is the set of forward arcs of H and H^- is the set of backward arcs of H. [This can be seen from the conformal realization theorem (Prop. 1.1) as follows. For the flow vector $x-x^0$, the net outflow from node t is $-g_t>0$ and the net outflow from node s is $-g_s<0$ (here we ignore the flow supplies), so by the conformal realization theorem, there is a path H from t to s that contains no cycle and conforms to the flow $x-x^0$, that is, $x_{ij}-x^0_{ij}>0$ for all $(i,j)\in H^+$ and $x_{ij}-x^0_{ij}<0$ for all $(i,j)\in H^-$. Equations (4.60) and (4.61) then follow.]

Since the pair (x,p) satisfies ϵ-CS, we have using Eqs. (4.60) and (4.61),

$$p_i-p_j\leq a_{ij}+\epsilon,\qquad \forall\ (i,j)\in H^+, \tag{4.62}$$

$$p_i-p_j\geq a_{ij}-\epsilon,\qquad \forall\ (i,j)\in H^-. \tag{4.63}$$

Similarly, since the pair (x^0,p^0) satisfies $r\epsilon$-CS, we have

$$p^0_i-p^0_j\geq a_{ij}+r\epsilon,\qquad \forall\ (i,j)\in H^+, \tag{4.64}$$

$$p^0_i-p^0_j\leq a_{ij}-r\epsilon,\qquad \forall\ (i,j)\in H^-. \tag{4.65}$$

Combining Eqs. (4.62)-(4.65), we obtain

$$p_i-p_j\geq p^0_i-p^0_j-(r+1)\epsilon,\qquad \forall\ (i,j)\in H^+,$$

$$p_i-p_j\leq p^0_i-p^0_j+(r+1)\epsilon,\qquad \forall\ (i,j)\in H^-.$$

Applying the above inequalities for all arcs of the path H, we get

$$p_t-p_s\geq p^0_t-p^0_s-(r+1)|H|\epsilon, \tag{4.66}$$

where $|H|$ denotes the number of arcs of the path H. We observed earlier that if a node has negative excess at some time, then its price is unchanged from the beginning of the method until that time. Thus $p_t=p^0_t$. Since the path contains no cycles, we also have that $|H|\leq N-1$. Therefore, Eq. (4.66) yields

$$p_s-p^0_s\leq (r+1)|H|\epsilon\leq (r+1)(N-1)\epsilon. \tag{4.67}$$

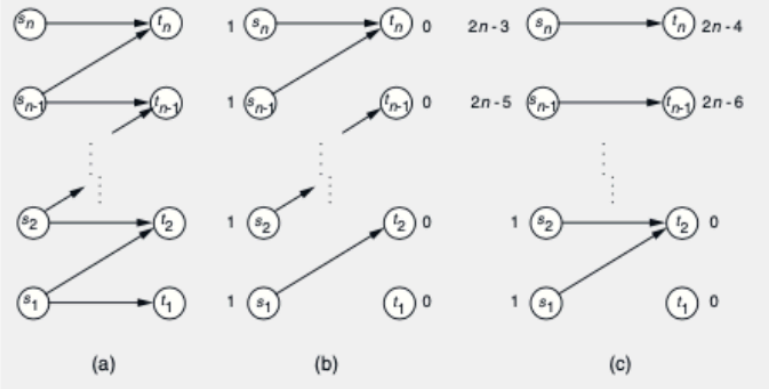

Figure 4.5.1: (a) An assignment example in which the number of price rises required by the ϵ-relaxation method is proportional to N^2. Note that the only feasible solution has each s_k assigned to the corresponding t_k. (b) The assignment example after n price rises, starting with zero prices. Prices are shown next to the corresponding node. Only arcs with positive flow are depicted. (c) The intermediate result after $(n-1)^2+1$ price rises.

Since only nodes with positive excess can increase their prices and each price rise increment is at least ϵ, we conclude from Eq. (4.67) that the total number of price rises that can be performed for node s is at most $(r+1)(N-1)$. **Q.E.D.**

The upper bound on the number of price rises given in Prop. 4.5.1 turns out to be tight, in the sense that examples can be found where rN price rises occur at a number of nodes that is proportional to N. Under these circumstances, the total number of price rises performed by the ϵ-relaxation method is no better than $O(rN^2)$. The following example, from Bertsekas and Tsitsiklis [1989], illustrates that the bound $O(rN^2)$ cannot be improved.

Example 4.5.1:

Consider an assignment problem with $2n$ nodes, nodes $s_1, ..., s_n$ being sinks (persons) and $t_1, ..., t_n$ being sources (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 ϵ-relaxation method is applied to the assignment version of this example, with $\epsilon = 1$, zero initial prices, 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. 4.5.1(a) is selected. The nodes are chosen for iteration in the order $1, 2, ..., n$.

We claim that the ϵ-relaxation algorithm as applied to the example of Fig. 4.5.1(a) requires n^2 price rises. The final price of node s_k is $2k-1$, and that of t_k is $2k-2$. We prove this 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. 4.5.1(b) 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 rises in a problem of size $n-1$. By induction, after another

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

price rises, the algorithm reaches the configuration of Fig. 4.5.1(c). Following the rules of ϵ-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 rises is

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

This establishes the induction.

The total number of nodes here is $N = 2n$. Hence the number of price rises is $(N/2)^2 = N^2/4$, and increases with N at the same rate as its theoretical bound.

We now introduce the notion of the *admissible graph*, which will play an important role in the subsequent complexity analysis. For a given pair (x, p) satisfying ϵ-CS, consider an arc set $\mathcal{A}^*$ that contains all candidate list arcs oriented in the direction of flow change. In particular, for each arc (i, j) in the forward portion of the candidate list of a node i, we introduce an arc (i, j) in $\mathcal{A}^*$, and for each arc (j, i) in the backward portion of the candidate list of node i, we introduce an arc (i, j) in $\mathcal{A}^*$ (thus the direction of the latter arc is reversed). The set of nodes $\mathcal{N}$ and the set $\mathcal{A}^*$ define the *admissible graph* $\mathcal{G}^* = (\mathcal{N}, \mathcal{A}^*)$. Note that an arc can be in the candidate list of at most one node, so the admissible graph is well-defined.

For good performance of the ϵ-relaxation method, it may be important to start with a flow-price vector pair (x, p) satisfying ϵ-CS, and such that the corresponding admissible graph $\mathcal{G}^*$ is acyclic. One possibility is to select an initial price vector p and to set the initial arc flow x_{ij} for every arc $(i, j) \in \mathcal{A}$ so that the flow-price pair (x, p) satisfies 0-CS; for example

$$x_{ij} = \begin{cases} b_{ij} & \text{if } p_i - p_j \leq a_{ij}, \\ c_{ij} & \text{if } p_i - p_j > a_{ij}, \end{cases} \qquad \forall\ (i, j) \in \mathcal{A}. \tag{4.68}$$

It can be seen that with this choice, ϵ-CS is satisfied for every arc $(i, j) \in \mathcal{A}$, and that the initial admissible graph is empty and thus acyclic. Figure 4.5.2

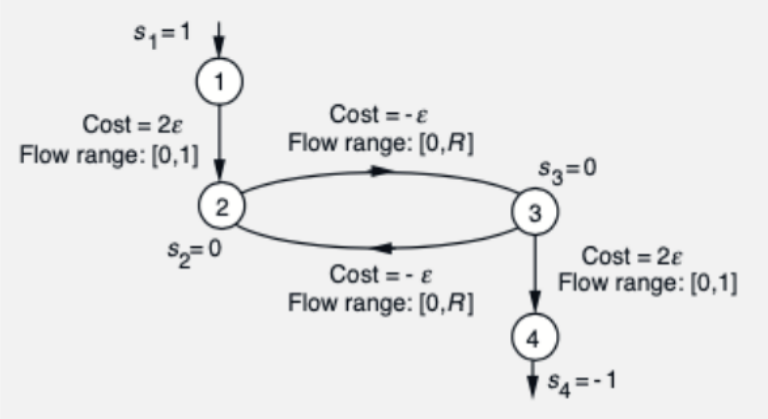

Figure 4.5.2: Example showing the importance of starting with an admissible graph that is acyclic. Initially, we choose $x = 0$ and $p = 0$, which do satisfy ϵ-CS. The initial admissible graph consists of arcs $(2, 3)$ and $(3, 2)$. The algorithm will start with a price rise of node 1 to $p_1 = 2\epsilon$, followed by a flow push of 1 unit from node 1 to node 2. Following this, node 2 will push 1 unit of flow to node 3, node 3 will push 1 unit of flow to node 2, and this will be repeated R times, until the arcs $(2, 3)$ and $(3, 2)$ become saturated. Thus the running time is proportional to the capacity R.

provides an example illustrating the importance of starting with an acyclic admissible graph.

On the other hand, it turns out that if we choose the initial flow-price pair so that the admissible graph is initially acyclic, the algorithm cannot create cycles in this graph, and the type of poor performance illustrated in Fig. 4.5.2 cannot occur. This is shown in the following proposition.

Proposition 4.5.2: If the admissible graph is initially acyclic, it remains acyclic throughout the ϵ-relaxation method.

Proof: We use induction. Assume that the admissible graph $\mathcal{G}^*$ is acyclic up to the start of the mth iteration, for some $m \geq 1$. We will prove that following the mth iteration $\mathcal{G}^*$ remains acyclic. Clearly, after a flow push the admissible graph remains acyclic, since it either remains unchanged, or some arcs are deleted from it. Thus we only have to prove that after a price rise at a node i, no cycle involving i is created. We note that, after a price rise at node i, all incident arcs to i in the admissible graph at the start of the mth iteration are deleted and new arcs incident to i are added. We claim that i cannot have any incoming arcs which belong to the admissible

graph. To see this, note that, just before a price rise at node i, we have

$$p_j - p_i \leq a_{ji} + \epsilon, \qquad \forall\ (j,i) \in \mathcal{A},$$

and since each price rise is at least ϵ, we must have

$$p_j - p_i - a_{ji} \leq 0, \qquad \forall\ (j,i) \in \mathcal{A},$$

after the price rise. Then, (j,i) cannot be in the candidate list of node j. By a similar argument, we have that (i,j) cannot be in the candidate list of j for all $(i,j) \in \mathcal{A}$. Thus, after a price rise at i, node i cannot have any incoming incident arcs belonging to the admissible graph, so no cycle involving i can be created. **Q.E.D.**

In order to obtain a sharper complexity result, we introduce a special implementation of the ϵ-relaxation method, called the *sweep implementation*, whereby nodes are chosen for iteration in a way that enhances computational efficiency (for an illustration, see Fig. 4.5.3). We assume here that the initial admissible graph is acyclic. We introduce an order in which the nodes are chosen in iterations. All the nodes are kept in a list T, which is traversed from the first to the last element. The order of the nodes in the list is consistent with the successor order implied by the admissible graph, that is, if a node j is a successor of a node i, then j must appear after i in the list. If the initial admissible graph is empty, as is the case with the initialization of Eq. (4.68), the initial list is arbitrary. Otherwise, the initial list must be consistent with the successor order of the initial admissible graph. The list is updated in a way that maintains the consistency with the successor order. In particular, let i be a node on which we perform an ϵ-relaxation iteration, and let N_i be the subset of nodes of T that are after i in T. If the price of i changes, then node i is removed from its position in T and placed in the first position of T. The next node chosen for iteration, if N_i is nonempty, is the node $i' \in N_i$ with positive excess which ranks highest in T. Otherwise, the positive excess node ranking highest in T is picked. It can be seen that with this rule of repositioning nodes following a price rise, the list order is consistent with the successor order implied by the admissible graph throughout the method.

A *sweep cycle* is a set of iterations whereby all nodes are chosen once from the list T and an ϵ-relaxation iteration is performed on those nodes that have positive excess. The idea of the sweep implementation is that an ϵ-relaxation iteration at a node i that has predecessors with positive excess may be wasteful, since the excess of i will be set to zero and become positive again through a flow push at a predecessor node.

We have the following proposition that estimates the number of sweep cycles required for termination.

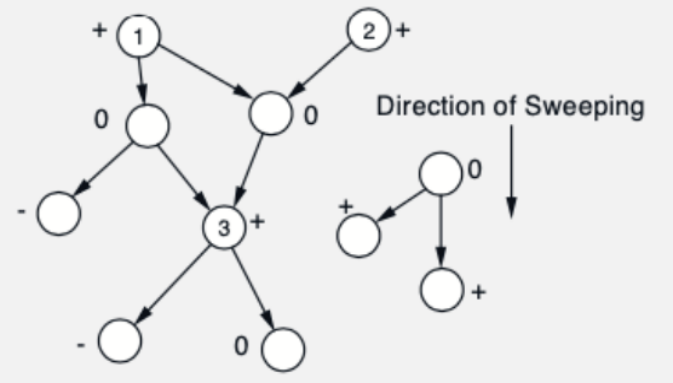

Figure 4.5.3: Illustration of the admissible graph consisting of the ϵ^+ - unblocked arcs and the ϵ^- - unblocked arcs with their directions reversed. These arcs specify the direction along which flow can be changed according to the rules of the algorithm. A "+" (or "-" or "0") indicates a node with positive (or negative or zero) excess. The algorithm is operated so that the admissible graph is acyclic at all times. The sweep implementation 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).

Proposition 4.5.3: Assume that for some scalar $r \geq 1$, the initial price vector for the sweep implementation of the ϵ-relaxation method satisfies $r\epsilon$-CS together with some feasible flow vector. Then, the number of sweep cycles up to termination is $O(rN^2)$.

Proof: Consider the start of any sweep cycle. Let $\mathcal{N}^+$ be the set of nodes with positive excess that have no predecessor with positive excess; let $\mathcal{N}^0$ be the set of nodes with nonpositive excess that have no predecessor with positive excess. Then, as long as no price rise takes place during the cycle, all nodes in $\mathcal{N}^0$ remain in $\mathcal{N}^0$, and an iteration on a node $i \in \mathcal{N}^+$ moves i from $\mathcal{N}^+$ to $\mathcal{N}^0$. So if no node changed price during the cycle, then all nodes in $\mathcal{N}^+$ will be moved to $\mathcal{N}^0$ and the method terminates. Therefore, there is a price rise in every cycle except possibly the last one. Since by Prop. 4.5.1 there are $O(rN^2)$ price rises, the result follows. **Q.E.D.**

We now bound the running time for the sweep implementation of the ϵ-relaxation method.

Proposition 4.5.4: Consider the ϵ-relaxation method with the sweep implementation, and assume that for some scalar $r \geq 1$ the initial price vector p^0 satisfies $r\epsilon$-CS together with some feasible flow vector x^0. Then, the method requires $O(rN^3)$ operations up to termination.

Proof: The dominant computational requirements are:

(1) The computation required for price rises.

(2) The computation required for saturating flow pushes.

(3) The computation required for nonsaturating flow pushes.

According to Prop. 4.5.1, there are $O(rN)$ price rises per node, so the requirements for (1) above are $O(rNA)$ operations. Furthermore, whenever a flow push at an arc is saturating, it takes at least one price rise at one of the end nodes of the arc before the arc's flow can be changed again. Thus the total requirement for (2) above is $O(rNA)$ operations also. Finally, for (3) above we note that for each sweep cycle there can be only one nonsaturating flow push per node. Thus an estimate for (3) is $O(N \cdot$ total number of sweep cycles) which, by Prop. 4.5.1, is $O(rN^3)$ operations. Adding the computational requirements for (1), (2), and (3), and using the fact $A \leq N^2$, the result follows. **Q.E.D.**