4

Auction Algorithms

In this chapter we will first focus on the assignment problem. We will discuss and analyze the auction algorithm described in Section 1.2.3, and some of its variations. We will then present an auction-like algorithm for shortest path problems. Finally, we will extend the auction algorithm to the minimum cost flow problem and some of its special cases.

4.1 THE AUCTION ALGORITHM FOR THE ASSIGNMENT PROBLEM

Recall the assignment problem where we want to match n persons and n objects on a one-to-one basis. We are given a value or benefit a_{ij} for matching person i with object j, and we want to assign persons to objects so as to maximize the total benefit. The set of objects to which person i can be assigned is a nonempty set denoted $A(i)$. An *assignment* S is a (possibly empty) set of person-object pairs (i, j) such that $j \in A(i)$ for all $(i, j) \in S$; for each person i there can be at most one pair $(i, j) \in S$; and for every object j there can be at most one pair $(i, j) \in S$. Given an assignment S, we say that person i is *assigned* if there exists a pair $(i, j) \in S$; otherwise we say that i is *unassigned*. We use similar terminology for objects. An assignment is said to be *feasible* if it contains n pairs, so that every person and every object is assigned; otherwise the assignment is called *partial*.

4.1.1 The Main Auction Algorithm

The auction algorithm, described somewhat loosely in Section 1.2.3, proceeds iteratively and terminates when a feasible assignment is obtained. At the start of the generic iteration we have a partial assignment S and a price vector p satisfying *ϵ-complementary slackness* (ϵ-CS). This is the condition

$$a_{ij} - p_j \geq \max_{k \in A(i)} \{a_{ik} - p_k\} - \epsilon, \qquad \forall\ (i,j) \in S \tag{1.1}$$

introduced in Section 1.2.3. As an initial choice, one can use an arbitrary set of prices together with the empty assignment, which trivially satisfies ϵ-CS. The iteration consists of two phases: the *bidding phase* and the *assignment phase* described in the following. We will show later that the iteration preserves the ϵ-CS condition.

Bidding Phase

Let I be a nonempty subset of persons i that are unassigned under the assignment S. For each person $i \in I$:

1. Find a "best" object j_i having maximum value, that is,

$$j_i = \arg\max_{j \in A(i)} \{a_{ij} - p_j\},$$

and the corresponding value

$$v_i = \max_{j \in A(i)} \{a_{ij} - p_j\}, \tag{1.2}$$

and find the best value offered by objects other than j_i

$$w_i = \max_{j \in A(i),\, j \neq j_i} \{a_{ij} - p_j\}. \tag{1.3}$$

[If j_i is the only object in $A(i)$, we define w_i to be $-\infty$, or for computational purposes, a number that is much smaller than v_i.]

2. Compute the "bid" of person i given by

$$b_{ij_i} = p_{j_i} + v_i - w_i + \epsilon = a_{ij_i} - w_i + \epsilon. \tag{1.4}$$

[We characterize this situation by saying that person i bid for object j_i, and that object j_i received a bid from person i. The algorithm works if the bid has any value between $p_{j_i} + \epsilon$ and $p_{j_i} + v_i - w_i + \epsilon$, but it tends to work fastest for the maximal choice of Eq. (1.4).]

Assignment Phase

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

$$p_j := \max_{i \in P(j)} b_{ij}, \tag{1.5}$$

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

Note that there is some freedom in choosing the subset of persons I that bid during an iteration. One possibility is to let I consist of a single unassigned person. This version, known as the *Gauss-Seidel version* because of its similarity with Gauss-Seidel methods for solving systems of nonlinear equations, usually works best in a serial computing environment. The version where I consists of all unassigned persons, is the one best suited for parallel computation; it is known as the *Jacobi version* because of its similarity with Jacobi methods for solving systems of nonlinear equations.

During an iteration, the objects whose prices are changed are the ones that received a bid during the iteration. Each price change involves an increace of at least ϵ. To see this, note that from Eqs. (1.2) to (1.4) we have

$$b_{ij_i} = a_{ij_i} - w_i + \epsilon \geq a_{ij_i} - v_i + \epsilon = p_{j_i} + \epsilon,$$

so each bid for an object, including the winning bid, exceeds the object's current price by at least ϵ. At the end of the iteration, we have a new assignment that differs from the preceding one in that each object that received a bid is now assigned to some person that was unassigned at the start of the iteration. However, the assignment at the end of the iteration need not have more pairs than the one at the start of the iteration, because it is possible that all objects that received a bid were assigned at the start of the iteration.

The choice of bidding increment $v_i - w_i + \epsilon$ for a person i [cf. Eq. (1.4)] is such that ϵ-CS is preserved by the algorithm, as shown by the following proposition (in fact, it can be seen that it is the largest bidding increment for which this is so).

Proposition 1.1: The auction algorithm preserves ϵ-CS throughout its execution; that is, if the assignment and the price vector available at the start of an iteration satisfy ϵ-CS, the same is true for the assignment and the price vector obtained at the end of the iteration.

Proof: Suppose that object j^* received a bid from person i and was assigned to i during the iteration. Let p_j and p'_j be the object prices before and after the assignment phase, respectively. Then we have [see Eqs. (1.4) and (1.5)]

$$p'_{j^*} = a_{ij^*} - w_i + \epsilon.$$

Using this equation, we obtain

$$a_{ij^*} - p'_{j^*} = w_i - \epsilon = \max_{j \in A(i), j \neq j^*} \{a_{ij} - p_j\} - \epsilon.$$

Since $p'_j \geq p_j$ for all j, this equation implies that

$$a_{ij^*} - p'_{j^*} \geq \max_{j \in A(i)} \{a_{ij} - p'_j\} - \epsilon, \tag{1.6}$$

which shows that the ϵ-CS condition (1.1) continues to hold after the assignment phase of an iteration for all pairs (i, j^*) that entered the assignment during the iteration.

Consider also any pair (i, j^*) that belonged to the assignment just before the iteration, and also belongs to the assignment after the iteration. Then, j^* must not have received a bid during the iteration, so $p'_{j^*} = p_{j^*}$. Therefore, Eq. (1.6) holds in view of the ϵ-CS condition that held prior to the iteration and the fact $p'_j \geq p_j$ for all j. Hence, the ϵ-CS condition (1.1) holds for all pairs (i, j^*) that belong to the assignment after the iteration, proving the result. **Q.E.D.**

The next result establishes the validity of the algorithm. The proof relies on the following facts:

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

(b) Each time an object receives a bid, its price increases by at least ϵ [see Eqs. (1.4) and (1.5)]. Therefore, if the object receives a bid an infinite number of times, its price increases to ∞.

(c) Every $|A(i)|$ bids by person i, where $|A(i)|$ is the number of objects in the set $A(i)$, the scalar v_i defined by the equation

$$v_i = \max_{j \in A(i)} \{a_{ij} - p_j\} \tag{1.7}$$

decreases by at least ϵ. The reason is that a bid by person i either decreases v_i by at least ϵ, or else leaves v_i unchanged because there is more than one object j attaining the maximum in Eq. (1.7). However, in the latter case, the price of the object j_i receiving the bid will increase by at least ϵ, and object j_i will not receive another bid by person i until

v_i decreases by at least ϵ. The conclusion is that if a person i bids an infinite number of times, v_i must decrease to $-\infty$.

Proposition 1.2: If at least one feasible assignment exists, the auction algorithm terminates with a feasible assignment that is within $n\epsilon$ of being optimal (and is optimal if the problem data are integer and $\epsilon < 1/n$).

Proof: We argue by contradiction. If termination did not occur, the subset J^∞ of objects that received an infinite number of bids is nonempty. Also, the subset of persons I^∞ that bid an infinite number of times is nonempty. As argued in (b) above, the prices of the objects in J^∞ must tend to ∞, while as argued in (c) above, the scalars $v_i = \max_{j \in A(i)}\{a_{ij} - p_j\}$ must decrease to $-\infty$ for all persons $i \in I^\infty$. This implies that

$$A(i) \subset J^\infty, \qquad \forall\ i \in I^\infty. \tag{1.8}$$

The ϵ-CS condition (1.1) states that $a_{ij} - p_j \geq v_i - \epsilon$ for every assigned pair (i, j), so after a finite number of iterations, each object in J^∞ can only be assigned to a person from I^∞. Since after a finite number of iterations at least one person from I^∞ will be unassigned at the start of each iteration, it follows that the number of persons in I^∞ is strictly larger than the number of objects in J^∞. This contradicts the existence of a feasible assignment, since by Eq. (1.8), persons in I^∞ can only be assigned to objects in J^∞. Therefore, the algorithm must terminate. The feasible assignment obtained upon termination satisfies ϵ-CS by Prop. 1.1, so by Prop. 2.3 of Section 1.2.3, this assignment is within $n\epsilon$ of being optimal. **Q.E.D.**

Consider now the case of an infeasible assignment problem. In this case, the auction algorithm cannot possibly terminate; it will keep on increasing the prices of some objects by increments of at least ϵ. Furthermore, some persons i will be submitting bids infinitely often, and the corresponding maximum values

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

will be decreasing toward $-\infty$. One can detect this situation by making use of a precomputable lower bound on the above values v_i, which holds when the problem is feasible; see Exercise 1.5. Once v_i gets below this bound for some i, we know that the problem is infeasible. Unfortunately, it may take many iterations for some v_i to reach this bound. An alternative method to detect infeasibility is to convert the problem to a feasible problem by adding artificial arcs to the assignment graph. The values of these arcs should be very small (i.e. large negative), so that they never participate in an optimal assignment unless the problem is infeasible. Exercise 1.6 quantifies the appropriate threshold for the values of the artificial arcs.

4.1.2 The Approximate Coordinate Descent Interpretation

The Gauss-Seidel version of the auction algorithm resembles coordinate descent algorithms, and the relaxation method of the previous chapter in particular, because it involves the change of a single object price with all other prices held fixed. In contrast with the relaxation method, however, such a price change may worsen strictly the value of the dual function

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

which was introduced in Prop. 2.4 of Section 1.2.

Generally we can interpret the bidding and assignment phases as a simultaneous "approximate" coordinate descent step for all price coordinates that increase during the iteration. The coordinate steps are aimed at minimizing approximately the dual function. In particular, we claim that *the price p_j of each object j that received a bid during the assignment phase is increased to either a value that minimizes $q(p)$ when all other prices are kept constant or else exceeds the largest such value by no more than ϵ.* Figure 1.1 illustrates this property and outlines its proof.

Figure 1.1 suggests that the amount of deterioration of the dual cost is at most ϵ. Indeed, for the Gauss-Seidel version of the algorithm this can be deduced from the argument given in Figure 1.1 and is left for the reader as Exercise 1.1.

4.1.3 Computational Aspects – ϵ-Scaling

The auction algorithm can be shown to have an $O\big(A(n + nC/\epsilon)\big)$ worst-case running time, where A is the number of arcs of the assignment graph and

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

is the maximum absolute object value; see [BeE88], [BeT89]. Thus, the amount of work to solve the problem can depend strongly on the value of ϵ as well as C. In practice, the dependence of the running time on ϵ and C is often significant, particularly for sparse problems; this dependence can also be seen in the example of Section 1.2.3 (cf. Fig. 2.14), and in Exercise 1.4.

The practical performance of the auction algorithm is often considerably improved by using *ϵ-scaling*, which consists of applying the algorithm several times, starting with a large value of ϵ and successively reducing ϵ up to an ultimate value that is less than $1/n$; cf. the discussion in Section 1.2.3. Each application of the algorithm, called a *scaling phase*, provides good initial prices for the next application. In the auction code of Appendix A.4, the integer

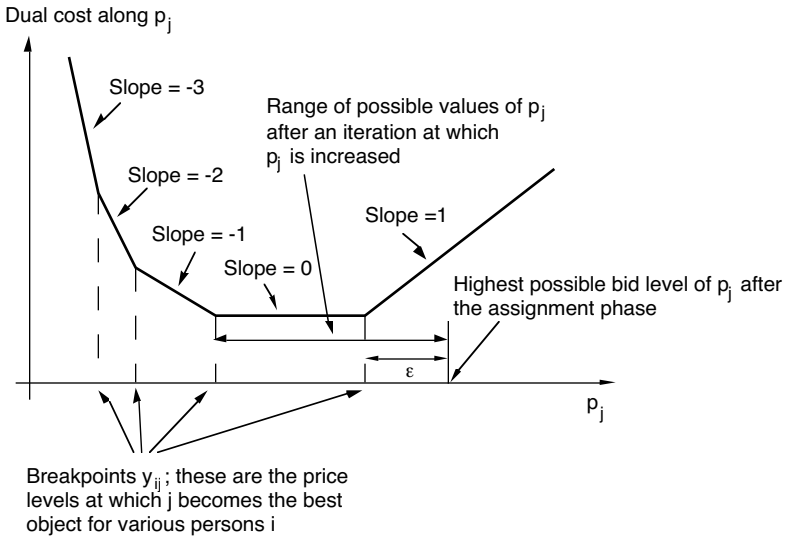


Figure 1.1 Form of the dual cost along the price coordinate p_j. From the definition of the dual cost q, the right directional derivative of q along p_j is

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

where

$$y_{ij} = a_{ij} - \max_{k \in A(i),\, k \neq j} \{a_{ik} - p_k\}$$

is the level of p_j below which j is the best person for person i. The break points are y_{ij} for all i such that $j \in A(i)$. Let $\overline{y} = \max_{\{i \mid j \in A(i)\}} \{a_{ij} - p_j\}$, let $\overline{i}$ be a person such that $\overline{y} = y_{\overline{i}j}$, let $\hat{y} = \max_{\{i \mid j \in A(i),\, i \neq \overline{i}\}} \{a_{ij} - p_j\}$, let $\hat{i}$ be a person such that $\hat{i} \neq \overline{i}$ and $\hat{y} = y_{\hat{i}j}$. Note that the interval $[\hat{y}, \overline{y}]$ is the set of points that minimize q along the coordinate p_j.

Let p_j be the price of j just before an iteration at which j receives a bid and let p'_j be the price of j after the iteration. We claim that $\hat{y} \leq p'_j \leq \overline{y} + \epsilon$. Indeed, if i is the person that bids and wins j during the iteration, then $p'_j = y_{ij} + \epsilon$, implying that $p'_j \leq \overline{y} + \epsilon$. To prove that $p'_j \geq \hat{y}$, we note that if $p_j \geq \hat{y}$, we must also have $p'_j \geq \hat{y}$, since $p'_j \geq p_j$. On the other hand, if $p'_j < \hat{y}$, there are two possibilities:

1. At the start of the iteration, $\overline{i}$ was not assigned to j. In this case, either $\overline{i}$ was unassigned in which case i will bid for j so that $p'_j = \overline{y} + \epsilon$, or else $\overline{i}$ was assigned to an object $\overline{j} \neq j$, in which case by ϵ-CS,

$$a_{\overline{i}j} - p_j - \epsilon \leq a_{\overline{i}\overline{j}} - p_{\overline{j}} \leq \max_{k \in A(\overline{i}),\, k \neq j} \{a_{\overline{i}k} - p_k\} = a_{\overline{i}j} - \overline{y}.$$

Thus, $p_j \geq \overline{y} - \epsilon$, implying that $p'_j \geq \overline{y}$ (since a bid increases a price by at least ϵ). In both cases we have $p'_j \geq \overline{y} \geq \hat{y}$.

2. At the start of the iteration, $\overline{i}$ was assigned to j. In this case, $\hat{i}$ was not assigned to j, so by repeating the argument of the preceding paragraph with $\hat{i}$ and $\hat{y}$ replacing $\overline{i}$ and $\overline{y}$, respectively, we obtain $p'_j \geq \hat{y}$.

benefits a_{ij} are first multiplied by $n+1$ and the auction algorithm is applied with progressively lower values of ϵ, to the point where ϵ becomes 1 or smaller (because a_{ij} has been scaled by $n+1$, it is sufficient for optimality of the final assignment to have $\epsilon \leq 1$). The sequence of ϵ values used is

$$\epsilon(k) = \max\{1, \Delta/\theta^k\}, \qquad k = 0, 1, \ldots,$$

where Δ and θ are parameters set by the user with $\Delta > 0$ and $\theta > 1$. (Typical values for sparse problems are $C/5 \leq \Delta \leq C/2$ and $4 \leq \theta \leq 10$. For nonsparse problems, sometimes $\Delta = 1$, which in effect bypasses ϵ-scaling, works quite well.) The auction code of Appendix A.4 also uses an *adaptive* form of ϵ-scaling, whereby, within the kth scaling phase, the value of ϵ is gradually *increased* to the value $\epsilon(k)$ given above, starting from a relatively small value, based on the results of the computation.

For integer data, it can be shown that the worst-case running time of the auction algorithm using scaling and appropriate data structures is $O\big(nA\log(nC)\big)$; see [BeE88], [BeT89]. For randomly generated problems, the running time of the algorithm seems to grow proportionally to something like $A\log n$ or $A\log n\log(nC)$; see also Exercise 1.3.

E X E R C I S E S

Exercise 1.1

Consider the Gauss-Seidel version of the auction algorithm, where only one person can bid at each iteration. Show that, as a result of a bid, the dual cost can be degraded by at most ϵ.

Exercise 1.2 (A Refinement of the Termination Tolerance [Ber79])

Show that the assignment obtained upon termination of the auction algorithm is within $(n-1)\epsilon$ of being optimal (rather than $n\epsilon$). Also, for every $n \geq 2$, construct an example of an assignment problem with integer data such that the auction algorithm terminates with a nonoptimal assignment when $\epsilon = 1/(n-1)$. (Try first $n = 2$ and $n = 3$, and generalize.) *Hint*: Modify slightly the algorithm so that when the last object is assigned, its price is increased by $v_i - w_i$ (rather than $v_i - w_i + \epsilon$). Then the assignment obtained upon termination satisfies the ϵ-CS condition for $n-1$ objects and the CS condition ($\epsilon = 0$) for the last object. Modify the proof of Prop. 2.6 in Section 1.2.

Exercise 1.3

This problem uses a rough (and flawed) argument to estimate the average complexity of the auction algorithm. We assume that at each iteration, only one person submits a bid (that is, the Gauss-Seidel version of the algorithm is used). Furthermore, every object is the recipient of a bid with equal probability $(1/n)$, independently of the results of earlier bids. (This assumption clearly does not hold, but seems to capture somewhat the real situation where the problem is fairly dense and ϵ-scaling is used.)

(a) Show that when k objects are unassigned the average number of iterations needed to assign a new object is n/k.

(b) Show that, on the average, the number of iterations is $n(1 + 1/2 + \cdots + 1/n)$, which can be estimated as $O(n \log n)$.

(c) Assuming that the average number of bids submitted by each person is the same for all persons, show that the average running time is $O(A \log n)$.

Exercise 1.4

Consider the auction algorithm applied to assignment problems with benefits in the range $[0, C]$, starting with zero prices.

(a) Show that for dense problems (every person can bid for every object) an object can receive a bid in at most $1 + C/\epsilon$ iterations.

(b) [Cas91] Use the example of Fig. 1.2 to show that, in general, some objects may receive a bid in a number of iterations that is proportional to nC/ϵ.

Exercise 1.5 (Detecting Infeasibility)

Consider application of the auction algorithm to a feasible assignment problem with initial object prices $\{p_j^0\}$. Let

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

be the maximum object value for person i in the course of the algorithm. Show that for any unassigned person i we have at all times

$$v_i \geq -(2n-1)C - (n-1)\epsilon - \max_j \{p_j^0\},$$

where $C = \max_{(i,j) \in \mathcal{A}} |a_{ij}|$, and describe how this lower bound can be used to detect that a problem is infeasible. *Hint*: Show that if the problem is feasible and i is unassigned, there must exist an augmenting path starting from i and ending at some unassigned object. Add the ϵ-CS condition along this path.

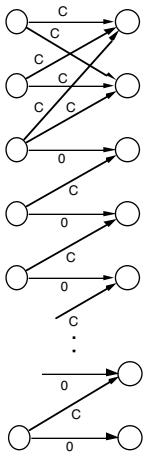


Figure 1.2 Assignment problem for which some objects receive a number of bids that is proportional to nC/ϵ. The arc values are shown next to the corresponding arcs.

Exercise 1.6 (Dealing with Infeasibility by Using Artificial Arcs)

Suppose that we add to the arc set $\mathcal{A}$ of an assignment problem a set $\overline{\mathcal{A}}$ of artificial arcs (possibly for the purpose of guaranteeing that the problem becomes feasible). Suppose also that we obtain an optimal assignment for the modified problem using the auction algorithm with initial object prices $\{p_j^0\}$. Show that if the original problem was feasible, no arc $(i,j) \in \overline{\mathcal{A}}$ will participate in the optimal assignment, provided

$$a_{ij} < -(2n-1)C - (n-1)\epsilon + p_j^0 - \max_k\{p_k^0\}, \qquad \forall\ (i,j) \in \overline{\mathcal{A}},$$

where $C = \max_{(i,j)\in\mathcal{A}} |a_{ij}|$. *Hint*: Use the result of Exercise 1.5.

Exercise 1.7 (Implementation of the Auction Algorithm [BeT91])

Frequently in the auction algorithm the two best objects for a given person do not change between two successive bids of that person. This exercise develops an implementation idea that attempts to exploit this fact by using a test to check whether the two best objects from the previous bid continue to be best. If the test is passed, the computation of the values $a_{ij} - p_j$ of the remaining objects is unnecessary. The implementation is used in the code for asymmetric assignment of Appendix A.4.

Suppose that at a given iteration, when we calculate the bid of the person i on the basis of a price vector p we compute the best value $v_i = \max_{j\in A(i)}\{a_{ij} -$

$p_j\}$, the best object $j_1 = \arg\max_{j \in A(i)}\{a_{ij} - p_j\}$, the second best value $w_i = \max_{j \in A(i), j \neq j_1}\{a_{ij} - p_j\}$, the second best object $j_2 = \arg\max_{j \in A(i), j \neq j_1}\{a_{ij} - p_j\}$, and the *third best value* $y_i = \max_{j \in A(i), j \neq j_1, j \neq j_2}\{a_{ij} - p_j\}$. Suppose that at a subsequent iteration when person i bids based on a price vector $\overline{p}$, we have $a_{ij_1} - \overline{p}_{j_1} \geq y_i$ and $a_{ij_2} - \overline{p}_{j_2} \geq y_i$. Show that j_1 and j_2 continue to be the two best objects for i (although j_1 need not be better than j_2).

4.2 REVERSE AUCTION AND INEQUALITY CONSTRAINED ASSIGNMENT PROBLEMS

In the auction algorithm, persons compete for objects by bidding and raising the price of their best object. It is possible to use an alternative form of the auction algorithm, called *reverse auction*, where, roughly, the *objects* compete for persons by essentially offering discounts.

To describe this algorithm, we introduce a *profit* variable π_i for each person i. Profits play for persons a role analogous to the role prices play for objects. We can describe reverse auction in two equivalent ways: one where unassigned objects lower their prices as much as possible to attract an unassigned person or lure a person away from its currently held object without violating ϵ-CS, and another where unassigned objects select a best person and raise his or her profit as much as possible without violating ϵ-CS. For analytical convenience, we will adopt the second description rather than the first, leaving the proof of their equivalence as Exercise 2.1 for the reader.

Let us consider the following ϵ-CS condition for a (partial) assignment S and a profit vector π:

$$a_{ij} - \pi_i \geq \max_{k \in B(j)}\{a_{kj} - \pi_k\} - \epsilon, \qquad \forall\ (i,j) \in S, \tag{2.1}$$

where $B(j)$ is the set of persons that can be assigned to object j,

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

For feasibility, we assume that this set is nonempty for all j. Note the symmetry of this condition with the corresponding one for prices; cf. Eq. (1.1). The reverse auction algorithm starts with and maintains an assignment and a profit vector π satisfying the above ϵ-CS condition. It terminates when the assignment is feasible. At the beginning of each iteration, we have an assignment S and a profit vector π satisfying the ϵ-CS condition (2.1).

Typical Iteration of Reverse Auction

Let J be a nonempty subset of objects j that are unassigned under the assignment S. For each object $j \in J$:

1. Find a "best" person i_j such that

$$i_j = \arg\max_{i \in B(j)} \{a_{ij} - \pi_i\},$$

and the corresponding value

$$\beta_j = \max_{i \in B(j)} \{a_{ij} - \pi_i\}, \tag{2.2}$$

and find

$$\omega_j = \max_{i \in B(j), i \neq i_j} \{a_{ij} - \pi_i\}. \tag{2.3}$$

[If i_j is the only person in $B(j)$, we define ω_j to be $-\infty$ or, for computational purposes, a number that is much smaller than β_j.]

2. Each object $j \in J$ bids for person i_j an amount

$$b_{i_j j} = \pi_{i_j} + \beta_j - \omega_j + \epsilon = a_{i_j j} - \omega_j + \epsilon. \tag{2.4}$$

3. For each person i that received at least one bid, increase π_i to the highest bid,

$$\pi_i := \max_{j \in P(i)} b_{ij}, \tag{2.5}$$

where $P(i)$ is the set of objects from which i received a bid; remove from the assignment S any pair (i, j) (if i was assigned to some j under S), and add to S the pair (i, j_i), where j_i is an object in $P(i)$ attaining the maximum above.

Note that reverse auction is identical to (forward) auction with the roles of persons and objects and those of profits and prices interchanged. Thus, by using the corresponding (forward) auction result (cf. Prop. 1.2), we have the following proposition.

Proposition 2.1: If at least one feasible assignment exists, the reverse auction algorithm terminates with a feasible assignment that is within $n\epsilon$ of being optimal (and is optimal if the problem data are integer and $\epsilon < 1/n$).

Combined Forward and Reverse Auction

One of the reasons we are interested in reverse auction is to construct algorithms that switch from forward to reverse auction and back. Such algorithms must simultaneously maintain a price vector p satisfying the ϵ-CS condition (1.1) and a profit vector π satisfying the ϵ-CS condition (2.1). To this end we introduce an ϵ-CS condition for the *pair* (π, p), which (as we will see) implies the other two. Maintaining this condition is essential for switching gracefully between forward and reverse auction.

Definition 2.1: An assignment S and a pair (π, p) are said to satisfy ϵ-CS if

$$\pi_i + p_j \geq a_{ij} - \epsilon, \qquad \forall\ (i,j) \in \mathcal{A}, \tag{2.6a}$$

$$\pi_i + p_j = a_{ij}, \qquad \forall\ (i,j) \in S. \tag{2.6b}$$

We have the following proposition.

Proposition 2.2: Suppose that an assignment S together with a profit-price pair (π, p) satisfy ϵ-CS. Then:

(a) S and π satisfy the ϵ-CS condition

$$a_{ij} - \pi_i \geq \max_{k \in B(j)} \{a_{kj} - \pi_k\} - \epsilon, \qquad \forall\ (i,j) \in S. \tag{2.7}$$

(b) S and p satisfy the ϵ-CS condition

$$a_{ij} - p_j \geq \max_{k \in A(i)} \{a_{ik} - p_k\} - \epsilon, \qquad \forall\ (i,j) \in S. \tag{2.8}$$

(c) If S is feasible, then S is within $n\epsilon$ of being an optimal assignment.

Proof: (a) In view of Eq. (2.6b), for all $(i,j) \in S$, we have $p_j = a_{ij} - \pi_i$, so Eq. (2.6a) implies that $a_{ij} - \pi_i \geq a_{kj} - \pi_k - \epsilon$ for all $k \in B(j)$. This shows Eq. (2.7).

(b) The proof is the same as the one of part (a) with the roles of π and p interchanged.

(c) Since by part (b) the ϵ-CS condition (2.8) is satisfied, by Prop. 2.6 of Section 1.2, S is within $n\epsilon$ of being optimal. **Q.E.D.**

We now introduce a combined forward/reverse auction algorithm. The algorithm starts with and maintains an assignment S and a profit-price pair (π, p) satisfying the ϵ-CS condition (2.6). It terminates when the assignment is feasible.

Combined Forward/Reverse Auction Algorithm

Step 1 (Run forward auction): Execute several iterations of the forward auction algorithm (subject to the termination condition), and at the end of each iteration (after increasing the prices of the objects that received a bid) set

$$\pi_i = a_{ij_i} - p_{j_i} \tag{2.9}$$

for every person-object pair (i, j_i) that entered the assignment during the iteration. Go to Step 2.

Step 2 (Run reverse auction): Execute several iterations of the reverse auction algorithm (subject to the termination condition), and at the end of each iteration (after increasing the profits of the persons that received a bid) set

$$p_j = a_{i_j j} - \pi_{i_j} \tag{2.10}$$

for every person-object pair (i_j, j) that entered the assignment during the iteration. Go to Step 1.

Note that the additional overhead of the combined algorithm over the forward or the reverse algorithm is minimal; just one update of the form (2.9) or (2.10) is required per iteration for each object or person that received a bid during the iteration. An important property is that these updates maintain the ϵ-CS condition (2.6) for the pair (π, p), and therefore, by Prop. 2.2, maintain the required ϵ-CS conditions (2.7) and (2.8) for π and p, respectively. This is shown in the following proposition.

Proposition 2.3: If the assignment and the profit-price pair available at the start of an iteration of either the forward or the reverse auction algorithm satisfy the ϵ-CS condition (2.6), the same is true for the assignment and the profit-price pair obtained at the end of the iteration, provided Eq. (2.9) is used to update π (in the case of forward auction), and Eq. (2.10) is used to update p (in the case of reverse auction).

Proof: Assume for concreteness that forward auction is used, and let (π, p) and $(\overline{\pi}, \overline{p})$ be the profit-price pair before and after the iteration, respectively. Then, $\overline{p}_j \geq p_j$ for all j (with strict inequality if and only if j received a bid during the iteration). Therefore, we have $\overline{\pi}_i + \overline{p}_j \geq a_{ij} - \epsilon$ for all (i, j) such that $\pi_i = \overline{\pi}_i$. Furthermore, we have $\overline{\pi}_i + \overline{p}_j = \pi_i + p_j = a_{ij}$ for all (i, j) that belong to the assignment before as well as after the iteration. Also, in view of the update (2.9), we have $\overline{\pi}_i + \overline{p}_{j_i} = a_{ij_i}$ for all pairs (i, j_i) that entered the assignment during the iteration. What remains is to verify that the condition

$$\overline{\pi}_i + \overline{p}_j \geq a_{ij} - \epsilon, \qquad \forall\ j \in A(i) \tag{2.11}$$

holds for all persons i that submitted a bid and were assigned to an object, say j_i, during the iteration. Indeed, for such a person i, we have, by Eq. (1.4),

$$\overline{p}_{j_i} = a_{ij_i} - \max_{j \in A(i),\, j \neq j_i} \{a_{ij} - p_j\} + \epsilon,$$

which implies that

$$\overline{\pi}_i = a_{ij_i} - \overline{p}_{j_i} \geq a_{ij} - p_j - \epsilon \geq a_{ij} - \overline{p}_j - \epsilon, \qquad \forall\, j \in A(i).$$

This shows the desired relation (2.11). **Q.E.D.**

Note that during forward auction the object prices p_j increase while the profits π_i decrease, but exactly the opposite happens in reverse auction. For this reason, the termination proof that we used for forward and for reverse auction does not apply to the combined method. Indeed, it is possible to construct examples of feasible problems where the combined method never terminates if the switch between forward and reverse auctions is done arbitrarily. However, it is easy to guarantee that the combined algorithm terminates for a feasible problem; it is sufficient to ensure that some "irreversible progress" is made before switching between forward and reverse auction. One easily implementable possibility is to refrain from switching until the number of assigned person-object pairs increases by at least one.

The combined forward/reverse auction algorithm often works substantially faster than the forward version. It seems to to be affected less by "price wars," that is, protracted sequences of small price rises by a number of persons bidding for a smaller number of objects (cf. Fig. 2.13 in Section 1.2). Price wars can still occur in the combined algorithm, by they arise through more complex and unlikely problem structures than in the forward algorithm. For this reason the combined forward/reverse auction algorithm depends less on ϵ-scaling for good performance than its forward counterpart; in fact, starting with $\epsilon = 1/(n+1)$, thus bypassing ϵ-scaling, is often the best choice.

4.2.1 Auction Algorithms for Asymmetric Assignment Problems

Reverse auction can be used in conjunction with forward auction to provide algorithms for solving the asymmetric assignment problem, where the number of objects n is larger than the number of persons m. Here we still require that each person be assigned to some object, but we allow objects to remain unassigned. As before, an assignment S is a (possibly empty) set of person-object pairs (i, j) such that $j \in A(i)$ for all $(i, j) \in S$; for each person i there can be at most one pair $(i, j) \in S$; and for every object j there can be at most one pair $(i, j) \in S$. The assignment S is said to be feasible if all persons are assigned under S.

The corresponding linear programming problem is

$$\begin{aligned}
&\text{maximize} \quad \sum_{(i,j)\in\mathcal{A}} a_{ij}x_{ij} \\
&\text{subject to} \\
&\qquad \sum_{j\in A(i)} x_{ij} = 1, \qquad \forall\ i = 1,\ldots,m, \\
&\qquad \sum_{i\in B(j)} x_{ij} \le 1, \qquad \forall\ j = 1,\ldots,n, \\
&\qquad 0 \le x_{ij}, \qquad \forall\ (i,j)\in\mathcal{A}.
\end{aligned} \tag{2.12}$$

We can convert this program to the minimum cost flow problem

$$\begin{aligned}
&\text{minimize} \quad \sum_{(i,j)\in\mathcal{A}} \bigl(-a_{ij}\bigr)x_{ij} \\
&\text{subject to} \\
&\qquad \sum_{j\in A(i)} x_{ij} = 1, \qquad \forall\ i = 1,\ldots,m, \\
&\qquad \sum_{i\in B(j)} x_{ij} + x_{sj} = 1, \qquad \forall\ j = 1,\ldots,n, \\
&\qquad \sum_{j=1}^{n} x_{sj} = n - m, \\
&\qquad 0 \le x_{ij}, \qquad \forall\ (i,j)\in\mathcal{A}, \\
&\qquad 0 \le x_{sj}, \qquad \forall\ j = 1,\ldots,n,
\end{aligned} \tag{2.13}$$

by replacing maximization by minimization, by reversing the sign of a_{ij}, and by introducing a supersource node s, which is connected to each object node j by an arc (s,j) of zero cost and feasible flow range $[0,\infty)$ (see Fig. 2.1).

Using the theory of Section 1.2 (cf. Prop. 2.5 and Exercise 2.11 of that section), it can be seen that the corresponding dual problem is

$$\begin{aligned}
&\text{minimize} \quad \sum_{i=1}^{m} \pi_i + \sum_{j=1}^{n} p_j - (n-m)\lambda \\
&\text{subject to} \\
&\qquad \pi_i + p_j \ge a_{ij}, \qquad \forall\ (i,j)\in\mathcal{A}, \\
&\qquad \lambda \le p_j, \qquad \forall\ j = 1,\ldots,n,
\end{aligned} \tag{2.14}$$

where we have converted maximization to minimization, we have used $-\pi_i$ in place of the price of each person node i, and we have denoted by λ the price of the supersource node s.

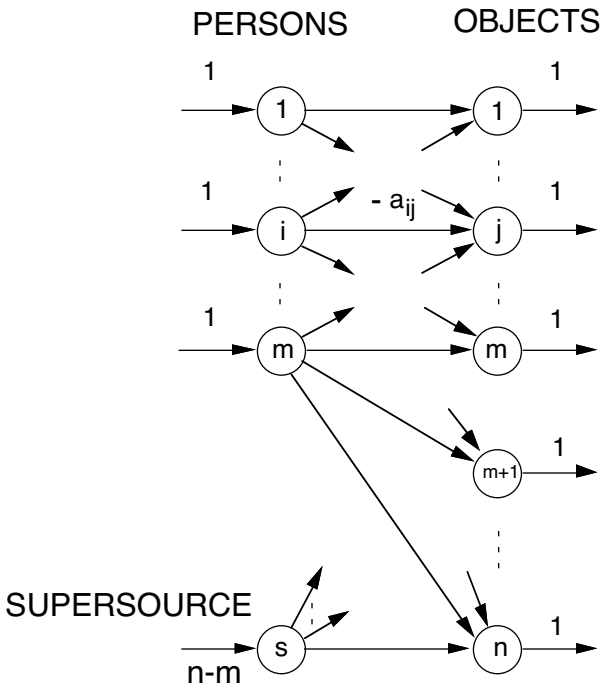


Figure 2.1 Converting an asymmetric assignment problem into a minimum cost flow problem involving a supersource node s and a zero cost artificial arc (s, j) with feasible flow range $[0, \infty)$ for each object j.

We now introduce an ϵ-CS condition for an assignment S and a pair (π, p).

Definition 2.2: An assignment S and a pair (π, p) are said to satisfy ϵ-CS if

$$\pi_i + p_j \geq a_{ij} - \epsilon, \qquad \forall\ (i,j) \in \mathcal{A}, \tag{2.15a}$$

$$\pi_i + p_j = a_{ij}, \qquad \forall\ (i,j) \in S, \tag{2.15b}$$

$$p_j \leq \min_{k:\ \text{assigned under } S} p_k, \qquad \forall\ j:\ \text{unassigned under } S. \tag{2.15c}$$

The following proposition clarifies the significance of the preceding ϵ-CS condition.

Proposition 2.4: If a feasible assignment S satisfies the ϵ-CS conditions (2.15) together with a pair (π, p), then S is within $m\epsilon$ of being optimal for the asymmetric assignment problem. The triplet $(\hat{\pi}, \hat{p}, \lambda)$, where

$$\lambda = \min_{k:\ \text{assigned under } S} p_k, \tag{2.16a}$$

$$\hat{\pi}_i = \pi_i + \epsilon, \qquad \forall\ i = 1, \ldots, m, \tag{2.16b}$$

$$\hat{p}_j = \begin{cases} p_j, & \text{if } j \text{ is assigned under } S, \\ \lambda, & \text{if } j \text{ is unassigned under } S \end{cases} \qquad \forall\ j = 1, \ldots, n, \tag{2.16c}$$

is within $m\epsilon$ of being an optimal solution of the dual problem (2.14).

Proof: For any feasible assignment $\{(i, k_i) \mid i = 1, \ldots, m\}$ and for any triplet $(\overline{\pi}, \overline{p}, \lambda)$ satisfying the dual feasibility constraints $\overline{\pi}_i + \overline{p}_j \geq a_{ij}$ for all $(i, j) \in \mathcal{A}$ and $\lambda \leq \overline{p}_j$ for all j, we have

$$\sum_{i=1}^{m} a_{ik_i} \leq \sum_{i=1}^{m} \overline{\pi}_i + \sum_{i=1}^{m} \overline{p}_{k_i} \leq \sum_{i=1}^{m} \overline{\pi}_i + \sum_{j=1}^{n} \overline{p}_j - (n-m)\lambda.$$

By maximizing over all feasible assignments $\{(i, k_i) \mid i = 1, \ldots, m\}$ and by minimizing over all dual-feasible triplets $(\overline{\pi}, \overline{p}, \lambda)$, we see that

$$A^* \leq D^*,$$

where A^* is the optimal assignment value and D^* is the minimal dual cost.

Let now $S = \{(i, j_i) \mid i = 1, \ldots, m\}$ be the given assignment satisfying ϵ-CS together with (π, p), and consider the triplet $(\hat{\pi}, \hat{p}, \lambda)$ defined by Eq. (2.16). Since for all i we have $\hat{\pi}_i + \hat{p}_{j_i} = a_{ij} + \epsilon$, we obtain

$$\begin{aligned} A^* &\geq \sum_{i=1}^{m} a_{ij_i} = \sum_{i=1}^{m} \hat{\pi}_i + \sum_{i=1}^{m} \hat{p}_{j_i} - m\epsilon \geq \sum_{i=1}^{m} \hat{\pi}_i + \sum_{j=1}^{n} \hat{p}_j - (n-m)\lambda - m\epsilon \\ &\geq D^* - m\epsilon, \end{aligned}$$

where the last inequality holds because the triplet $(\hat{\pi}, \hat{p}, \lambda)$ is feasible for the dual problem. Since we showed earlier that $A^* \leq D^*$, the desired conclusion follows. **Q.E.D.**

Consider now trying to solve the asymmetric assignment problem by means of auction. We can start with any assignment S and pair (π, p) satisfying the first two ϵ-CS conditions (2.15a) and (2.15b), and perform a forward auction (as defined earlier for the symmetric assignment problem) up to the point where each person is assigned to a distinct object. For a feasible problem, by essentially repeating the proof of Prop. 1.2 for the symmetric case, it can be seen that this will yield, in a finite number of iterations, a feasible assignment S satisfying the first two conditions (2.15a) and (2.15b). However, this assignment may not be optimal, because the prices of the unassigned objects j are not minimal; that is, they do not satisfy the third ϵ-CS condition (2.15c).

To remedy this situation, we use a modified form of reverse auction to lower the prices of the unassigned objects so that, after several iterations in

which persons may be reassigned to other objects, the third condition, (2.15c), is satisfied. We will show that the assignment thus obtained satisfies all the ϵ-CS conditions (2.15a)-(2.15c), and by Prop. 2.4, is optimal within $m\epsilon$ (and thus optimal if the problem data are integer and $\epsilon < 1/m$).

The modified reversed auction starts with a feasible assignment S and with a pair (π, p) satisfying the first two ϵ-CS conditions (2.15a) and (2.15b). [For a feasible problem, such an S and (π, p) can be obtained by regular forward or reverse auction, as discussed earlier.] Let us denote by λ the minimal assigned object price under the initial assignment,

$$\lambda = \min_{j:\ \text{assigned under the initial assignment } S} p_j. \tag{2.17}$$

The typical iteration of modified reverse auction is the same as the one of reverse auction, except that only unassigned objects j with $p_j > \lambda$ participate in the auction. In particular, the algorithm maintains a feasible assignment S and a pair (π, p) satisfying Eqs. (2.15a) and (2.15b), and terminates when all unassigned objects j satisfy $p_j \leq \lambda$, in which case it will be seen that the third ϵ-CS condition (2.15c) is satisfied as well. The scalar λ is kept fixed throughout the algorithm.

Typical Iteration of Modified Reverse Auction for Asymmetric Assignment:

Select an object j that is unassigned under the assignment S and satisfies $p_j > \lambda$ (if no such object can be found, the algorithm terminates). Find a "best" person i_j such that

$$i_j = \arg\max_{i \in B(j)} \{a_{ij} - \pi_i\},$$

and the corresponding value

$$\beta_j = \max_{i \in B(j)} \{a_{ij} - \pi_i\}, \tag{2.18}$$

and find

$$\omega_j = \max_{i \in B(j),\, i \neq i_j} \{a_{ij} - \pi_i\}. \tag{2.19}$$

[If i_j is the only person in $B(j)$, we define ω_j to be $-\infty$.] If $\lambda \geq \beta_j - \epsilon$, set $p_j := \lambda$ and go to the next iteration. Otherwise, let

$$\delta = \min\{\beta_j - \lambda, \beta_j - \omega_j + \epsilon\}. \tag{2.20}$$

Set

$$p_j := \beta_j - \delta, \tag{2.21}$$

$$\pi_{i_j} := \pi_{i_j} + \delta, \tag{2.22}$$

add to the assignment S the pair (i_j, j), and remove from S the pair (i_j, j'), where j' is the object that was assigned to i_j under S at the start of the iteration.

Note that the formula (2.20) for the bidding increment δ is such that the object j enters the assignment at a price which is no less that λ [and is equal to λ if and only if the minimum in Eq. (2.20) is attained by the first term]. Furthermore, when δ is calculated (that is, when $\lambda > \beta_j - \epsilon$) we have $\delta \geq \epsilon$, so it can be seen from Eqs. (2.21) and (2.22) that, throughout the algorithm, prices are monotonically decreasing and profits are monotonically increasing. The following proposition establishes the validity of the method.

Proposition 2.5: The modified reverse auction algorithm for the asymmetric assignment problem terminates with an assignment that is within $m\epsilon$ of being optimal.

Proof: In view of Prop. 2.4, the result will follow once we prove the following:

(a) The modified reverse auction iteration preserves the first two ϵ-CS conditions (2.15a) and (2.15b), as well as the condition

$$\lambda \leq \min_{j:\ \text{assigned under the current assignment } S} p_j, \tag{2.23}$$

so upon termination of the algorithm (necessarily with the prices of all unassigned objects less or equal to λ) the third ϵ-CS condition, (2.15c), is satisfied.

(b) The algorithm terminates.

We will prove these facts in sequence.

We assume that the conditions (2.15a), (2.15b), and (2.23) are satisfied at the start of an iteration, and we will show that they are also satisfied at the end of the iteration. First consider the case where there is no change in the assignment, which happens when $\lambda \geq \beta_j - \epsilon$. Then Eqs. (2.15b), and (2.23) are automatically satisfied at the end of the iteration; only p_j changes in the iteration according to

$$p_j := \lambda \geq \beta_j - \epsilon = \max_{i \in B(j)} \{a_{ij} - \pi_i\} - \epsilon,$$

so the condition (2.15a) is also satisfied at the end of the iteration.

Next consider the case where there is a change in the assignment during the iteration. Let (π, p) and $(\overline{\pi}, \overline{p})$ be the profit-price pair before and after the iteration, respectively, and let j and i_j be the object and person involved in the iteration. By construction [cf. Eqs. (2.21) and (2.22)], we have $\overline{\pi}_{i_j} + \overline{p}_j = a_{i_j j}$ and since $\overline{\pi}_i = \pi_i$ and $\overline{p}_k = p_k$ for all $i \neq i_j$ and $k \neq j$, we see that the

condition (2.15b) ($\overline{\pi}_i + \overline{p}_k = a_{ik}$) is satisfied for all assigned pairs (i,k) at the end of the iteration.

To show that the condition (2.15a) is satisfied at the end of the iteration, that is,

$$\overline{\pi}_i + \overline{p}_k \geq a_{ik} - \epsilon, \qquad \forall\ (i,k) \in \mathcal{A}, \tag{2.24}$$

consider first objects $k \neq j$. Then, $\overline{p}_k = p_k$ and since $\overline{\pi}_i \geq \pi_i$ for all i, the above condition holds, since our hypothesis is that at the start of the iteration we have $\pi_i + p_k \geq a_{ik} - \epsilon$ for all (i,k). Consider next the case $k = j$. Then condition (2.24) holds for $i = i_j$, since $\overline{\pi}_{i_j} + \overline{p}_j = a_{i_j j}$. Also using Eqs. (2.18)-(2.21) and the fact $\delta \geq \epsilon$, we have for all $i \neq i_j$

$$\begin{aligned}\overline{\pi}_i + \overline{p}_j &= \pi_i + \overline{p}_j \geq \pi_i + \beta_j - (\beta_j - \omega_j + \epsilon) \\ &= \pi_i + \omega_j - \epsilon \geq \pi_i + (a_{ij} - \pi_i) - \epsilon = a_{ij} - \epsilon,\end{aligned}$$

so condition (2.24) holds for $i \neq i_j$ and $k = j$, completing the proof of Eq. (2.24). To see that condition (2.23) is maintained by the iteration, note that by Eqs. (2.18), (2.19), and (2.21), we have

$$\overline{p}_j = \beta_j - \delta \geq \beta_j - (\beta_j - \lambda) = \lambda.$$

Finally, to show that the algorithm terminates, we note that in the typical iteration involving object j and person i_j there are two possibilities:

(1) The price of object j is set to λ without the object entering the assignment; this occurs if $\lambda \geq \beta_j - \epsilon$.

(2) The profit of person i_j increases by at least ϵ [this is seen from the definition (2.20) of δ; we have $\lambda < \beta_j - \epsilon$ and $\beta_j \geq \omega_j$, so $\delta \geq \epsilon$].

Since only objects j with $p_j > \lambda$ can participate in the auction, possibility (1) can occur only a finite number of times. Thus, if the algorithm does not terminate, the profits of some persons will increase to ∞. This is impossible, since when person i is assigned to object j we must have by Eqs. (2.15b) and (2.23)

$$\pi_i = a_{ij} - p_j \leq a_{ij} - \lambda,$$

so the profits are bounded from above by $\max_{(i,j)\in\mathcal{A}} a_{ij} - \lambda$. Thus the algorithm must terminate. **Q.E.D.**

Note that one may bypass the modified reverse auction algorithm by starting the forward auction with all object prices equal to zero. Upon termination of the forward auction, the prices of the unassigned objects will still be at zero, while the prices of the assigned objects will be nonnegative. Therefore the ϵ-CS condition (2.15c) will be satisfied, and the modified reverse auction will be unnecessary (see Exercise 2.2).

Unfortunately the requirement of zero initial object prices is incompatible with ϵ-scaling. The principal advantage offered by the modified reverse auction algorithm is that it allows arbitrary initial object prices for the forward auction, thereby also allowing the use of ϵ-scaling. This can be shown to improve the theoretical worst-case complexity of the method, and is often beneficial in practice, particularly for sparse problems.

Reverse auction can be used also in the context of other types of network flow problems. One example is the variation of the asymmetric assignment problem where persons (as well as objects) need not be assigned if this degrades the assignment's value (see Exercise 2.3). Another class of assignment-like problems is described in the next subsection.

4.2.2 Auction Algorithms for Multiassignment Problems

An interesting type of assignment problem is described by the linear program

$$\begin{aligned} &\text{maximize} \quad \sum_{(i,j)\in\mathcal{A}} a_{ij}x_{ij} \\ &\text{subject to} \\ &\quad \sum_{j\in A(i)} x_{ij} \geq 1, \qquad \forall\ i = 1,\ldots,m, \\ &\quad \sum_{i\in B(j)} x_{ij} = 1, \qquad \forall\ j = 1,\ldots,n, \\ &\quad 0 \leq x_{ij}, \qquad \forall\ (i,j) \in \mathcal{A}, \end{aligned} \tag{2.25}$$

where $m < n$. For feasibility, we assume that the sets $A(i)$ and $B(j)$ are nonempty for all i and j. This is known as the *multiassignment* problem, and is characterized by the possibility of assignment of more than one object to a single person. Problems of this type arise in military applications such as multi-target tracking with sensors of limited resolution [Bla86], [BaF88], where objects correspond to tracked moving objects and persons correspond to data points each representing at least one object (but possibly more than one because of the sensor's limited resolution). The multiassignment problem results when we try to associate data points with moving objects so as to match as closely as possible these data points with our prior knowledge of the objects' positions.

We can convert the multiassignment problem to the minimum cost flow problem

$$\begin{aligned} &\text{minimize} \quad \sum_{(i,j)\in\mathcal{A}} \big(-a_{ij}\big)x_{ij} \\ &\text{subject to} \end{aligned}$$

$$\sum_{j\in A(i)} x_{ij} - x_{si} = 1, \qquad \forall\ i = 1,\ldots,m, \tag{2.26}$$

$$\sum_{i\in B(j)} x_{ij} = 1, \qquad \forall\ j = 1,\ldots,n,$$

$$\sum_{i=1}^{m} x_{si} = n - m,$$

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

$$0 \le x_{si}, \qquad \forall\ i = 1,\ldots,n,$$

by replacing maximization by minimization, by reversing the sign of a_{ij}, and by introducing a supersource node s, which is connected to each person node i by an arc (s,i) of zero cost and feasible flow range $[0,\infty)$ (see Fig. 2.2).

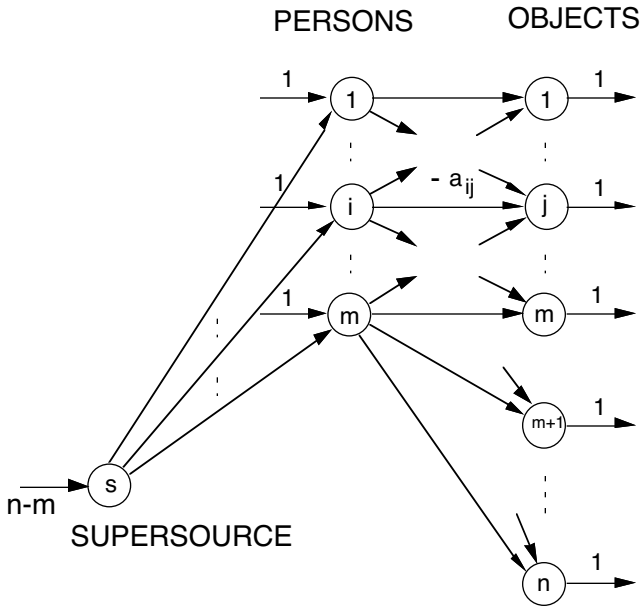


Figure 2.2 Converting a multiassignment problem into a minimum cost flow problem involving a supersource node s and a zero cost artificial arc (s,i) with feasible flow range $[0,\infty)$ for each person i.

Again using the theory of Section 1.2 and appropriately redefining the price variables corresponding to the nodes, it can be seen that the correspond-

ing dual problem is

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{m} \pi_i + \sum_{j=1}^{n} p_j + (n-m)\lambda \\ \text{subject to} \quad & \\ & \pi_i + p_j \geq a_{ij}, \qquad \forall\ (i,j) \in \mathcal{A}, \\ & \lambda \geq \pi_i, \qquad \forall\ i = 1, \ldots, m. \end{aligned} \tag{2.27}$$

We define a *multiassignment* S to be a set of pairs $(i,j) \in \mathcal{A}$ such that for each object j, there is at most one pair (i,j) in S. A person i for which there are more than one pairs (i,j) in S is said to be *multiassigned* under S. We now introduce an ϵ-CS condition for a multiassignment S and a pair (π, p).

Definition 2.3: A multiassignment S and a pair (π, p) are said to satisfy ϵ-CS if

$$\pi_i + p_j \geq a_{ij} - \epsilon, \qquad \forall\ (i,j) \in \mathcal{A}, \tag{2.28a}$$

$$\pi_i + p_j = a_{ij}, \qquad \forall\ (i,j) \in S, \tag{2.28b}$$

$$\pi_i = \max_{k=1,\ldots,m} \pi_k, \qquad \text{if } i \text{ is multiassigned under } S. \tag{2.28c}$$

We have the following result.

Proposition 2.6: If a feasible multiassignment S satisfies the ϵ-CS conditions (2.28) together with a pair (π, p), then S is within $n\epsilon$ of being optimal for the multiassignment problem. The triplet $(\hat{\pi}, p, \hat{\lambda})$, where

$$\hat{\pi}_i = \pi_i + \epsilon, \qquad \forall\ i = 1, \ldots, m,$$

$$\hat{\lambda} = \max_{k=1,\ldots,m} \hat{\pi}_k,$$

is within $n\epsilon$ of being an optimal solution of the dual problem (2.27).

Proof: Very similar to the proof of Prop. 2.4 – left for the reader. **Q.E.D.**

Consider now trying to solve the multiassignment problem by means of auction. We can start with any multiassignment S and profit-price pair (π, p) satisfying the first two ϵ-CS conditions (2.28a) and (2.28b), and perform a forward auction up to the point where each person is assigned to a (single) distinct object, while satisfying the conditions (2.28a) and (2.28b). However, this multiassignment will not be feasible, because some objects will still be unassigned.

To make further progress, we use a modified reverse auction that starts with the final results of the forward auction (that is, a multiassignment S,

where each person is assigned to a single distinct object) and with a pair (π, p) satisfying the first two ϵ-CS conditions (2.28a) and (2.28b). Let us denote by λ the maximal initial person profit,

$$\lambda = \max_{i=1,\ldots,m} \pi_i. \tag{2.29}$$

The typical iteration, given below, is the same as the one of reverse auction, except that unassigned objects j that bid for a person may not necessarily displace the object assigned to the person but may instead *share* the person with its already assigned object(s); this will happen if and only if the person's profit has reached the upper bound λ.

The algorithm maintains a multiassignment S, for which each person is assigned to at least one object, and a pair (π, p) satisfying Eqs. (2.28a) and (2.28b); it terminates when all unassigned objects j have been assigned. It will be seen that upon termination, the third ϵ-CS condition (2.28c) will be satisfied as well. The scalar λ is kept fixed throughout the algorithm.

Typical Iteration of Modified Reverse Auction for Multiassignment

Select an object j that is unassigned under the multiassignment S (if all objects are assigned, the algorithm terminates). Find a "best" person i_j such that

$$i_j = \arg\max_{i \in B(j)} \{a_{ij} - \pi_i\}, \tag{2.30}$$

and the corresponding value

$$\beta_j = \max_{i \in B(j)} \{a_{ij} - \pi_i\}, \tag{2.31}$$

and find

$$\omega_j = \max_{i \in B(j), i \neq i_j} \{a_{ij} - \pi_i\}. \tag{2.32}$$

[If i_j is the only person in $B(j)$, we define ω_j to be $-\infty$.] Let

$$\delta = \min\{\lambda - \pi_{i_j}, \beta_j - \omega_j + \epsilon\}. \tag{2.33}$$

Add (i_j, j) to the multiassignment S, set

$$p_j := \beta_j - \delta, \tag{2.34}$$

$$\pi_{i_j} := \pi_{i_j} + \delta, \tag{2.35}$$

and, if $\delta > 0$, remove from the multiassignment S the pair (i_j, j'), where j' was assigned to i_j under S.

Note that in an iteration the number of assigned objects increases by one if and only if $\delta = 0$ [which is equivalent to $\pi_{i_j} = \lambda$, since the second

term $\beta_j - \omega_j + \epsilon$ in Eq. (2.33) is always greater or equal to ϵ]. The following proposition establishes the validity of the method.

Proposition 2.7: The modified reverse auction algorithm for the multiassignment problem terminates with a feasible multiassignment that is within $n\epsilon$ of being optimal.

Proof: In view of Prop. 2.6, the result will follow once we prove the following:

(a) The modified reverse auction iteration preserves the ϵ-CS conditions (2.28), as well as the condition

$$\lambda = \max_{i=1,\ldots,m} \pi_i. \tag{2.36}$$

(b) The algorithm terminates (necessarily with a feasible multiassignment).

To show (a) we use induction. In particular, we will show that if the conditions (2.28) and (2.36) are satisfied at the start of an iteration, they are also satisfied at the end of the iteration. This is easily seen to be true for Eqs. (2.28a) and (2.28b). Equations (2.28c) and (2.36) are preserved, since we have $\lambda = \max_{i=1,\ldots,m} \pi_i$ at the start of the iteration and the only profit that changes is π_{i_j}, which by Eqs. (2.33) and (2.35) is set to something that is less or equal to λ, and is set to λ if and only if i_j is multiassigned at the end of the iteration.

To show termination, we observe that a person i can receive a bid only a finite number of times after the profit π_i is set to λ, since at each of these times the corresponding object will get assigned to i without any object already assigned to i becoming unassigned. On the other hand, by Eqs. (2.33) and (2.35), at an iteration where a person receives a bid, his or her profit is either set equal to λ or else increases by at least ϵ. Since profits are bounded above by λ throughout the algorithm, it follows that each person can receive only a finite number of bids; this proves termination. **Q.E.D.**

When the problem data are integer, Prop. 2.7 shows that the auction algorithm terminates with an optimal multiassignment provided $\epsilon < 1/n$. It is possible to strengthen this result and show that it is sufficient that $\epsilon < 1/m$ for optimality of the final multiassignment. This, however, requires a somewhat different proof argument than the one we have used so far; see Prop. 4.1 and Exercises 4.6 and 4.7 in Section 4.4.

E X E R C I S E S

Exercise 2.1 (Equivalence of Two Forms of Reverse Auction)

Show that the iteration of the Gauss-Seidel version of the reverse auction algorithm for the (symmetric) assignment problem can equivalently be described

by the following iteration, which maintains an assignment and a pair (π, p) satisfying the ϵ-CS condition (2.6):

Step 1: Choose an unassigned object j.

Step 2: Decrease p_j to the highest level for which two or more persons will increase their profit by at least ϵ after assignment to j, that is, set p_j to the highest level for which $a_{ij} - p_j \geq \pi_i + \epsilon$ for at least two persons i, where π_i is the profit of i at the start of the iteration.

Step 3: From the persons in Step 2, assign to j a person i_j that experiences maximum profit increase after assignment to j, and cancel the prior assignment of i_j if he or she was assigned at the start of the iteration. Set the profit of i_j to $a_{i_j j} - p_j$.

Exercise 2.2

Consider the asymmetric assignment problem and apply forward auction starting with the zero price vector and the empty assignment. Show that, for a feasible problem, the algorithm terminates with a feasible assignment that is within $m\epsilon$ of being optimal. *Note*: Because this method must start with the zero price vector, it does not admit ϵ-scaling.

Exercise 2.3 (A Variation of the Asymmetric Assignment Problem)

Consider a problem which is the same as the asymmetric assignment problem with the exception that in a feasible assignment S there can be at most one incident arc for every person and at most one incident arc for every object (that is, there is no need for every person, as well as for every object, to be assigned). The corresponding linear program is

$$\begin{aligned}
&\text{maximize} \quad \sum_{(i,j)\in\mathcal{A}} a_{ij}x_{ij} \\
&\text{subject to} \\
&\quad \sum_{j\in A(i)} x_{ij} \leq 1, \qquad \forall\ i = 1,\ldots,m, \\
&\quad \sum_{i\in B(j)} x_{ij} \leq 1, \qquad \forall\ j = 1,\ldots,n, \\
&\quad 0 \leq x_{ij}, \qquad \forall\ (i,j) \in \mathcal{A}.
\end{aligned}$$

(a) Show that this problem can be converted to an asymmetric assignment problem where all persons must be assigned. *Hint:* For each person i introduce an artificial object i' and a zero cost arc (i, i').

(b) Adapt and streamline the auction algorithm of Section 4.2.1 to solve the problem.

Exercise 2.4

Consider the multiassignment problem. Derive a combined forward/reverse auction algorithm similar to the one for the symmetric assignment problem. Forward auction iterations should be used only when there are unassigned persons, and reverse auction iterations should be such that the quantity $\lambda = \max_i \pi_i$ is never increased.

Exercise 2.5 (A Refinement of the Optimality Conditions)

(a) Consider the asymmetric assignment problem with integer data, and suppose that we have a feasible assignment S and a pair (π, p) satisfying the first two ϵ-CS conditions (2.15a) and (2.15b) with $\epsilon < 1/m$. Show that in order for S to be optimal it is sufficient that

$$p_k \leq p_t$$

for all k and t such that k is unassigned under S, t is assigned under S, and there exists a path $(k, i_1, j_1, \ldots, i_q, j_q, i_{q+1}, t)$ such that $(i_r, j_r) \in S$ for $r = 1, \ldots, q$, and $(i_{q+1}, t) \in S$. *Hint*: Consider the existence of cycles with positive value along which S can be modified.

(b) Consider the multiassignment problem. Derive a result analogous to the one of part (a), with the condition $p_k \leq p_t$ replaced by the condition $\pi_k \geq \pi_t$, where k is any multiassigned person and t is any person for which there exists a path $(k, j_1, i_1, \ldots, j_q, i_q, j_{q+1}, t)$ such that $(k, j_1) \in S$ and $(i_r, j_{r+1}) \in S$ for $r = 1, \ldots, q$.

4.3 AN AUCTION ALGORITHM FOR SHORTEST PATHS

In this section we consider an algorithm for finding a shortest path from several origins to a single destination in a directed graph $(\mathcal{N}, \mathcal{A})$. We will see later that this algorithm can also be viewed as an application of the *naive* auction algorithm (this is the auction algorithm with $\epsilon = 0$, discussed in Section 1.2.4) to a special type of assignment problem that is equivalent to the shortest path problem.

We will assume throughout this section that *all cycles have positive length*. When all the arc lengths are nonnegative, the cycle positivity assumption can be weakened to a nonnegativity assumption at the expense of complicating the algorithm somewhat; see Exercise 3.3.

To simplify the presentation, we also assume that each node except for the destination has at least one outgoing incident arc; any node not satisfying this condition can be connected to the destination with a very high length arc without materially changing the problem and the subsequent algorithm.

For the single origin case the algorithm is very simple. It maintains a single path starting at the origin. At each iteration, the path is either *extended* by adding a new node or *contracted* by deleting its terminal node. When the destination becomes the terminal node of the path, the algorithm terminates.

To get an intuitive sense of the algorithm, think of a person moving in a graph-like maze, trying to reach a destination. The person criss-crosses the maze, either advancing or backtracking along the current path. Each time the person backtracks from a node, he or she records a measure of the desirability of revisiting and advancing from that node in the future (this will be implemented with the help of a price variable). The person revisits and proceeds forward from a node when the node's measure of desirability is judged superior to those of other nodes. The algorithm of this section emulates this search process efficiently, using simple data structures.

Similar to the algorithms of Section 1.3, complementary slackness conditions are fundamental for the algorithm of this section. However, it is helpful for our purposes to reformulate these conditions in terms of node prices p_i rather than the node labels d_i used in Section 1.3.

In particular, given a simple (forward) path P and a price vector p consisting of prices p_i, we say that the pair (P, p) satisfies *complementary slackness* (CS) if

$$p_i \leq a_{ij} + p_j, \qquad \forall\ (i,j) \in \mathcal{A}, \tag{3.1a}$$

$$p_i = a_{ij} + p_j, \qquad \text{for all pairs of successive nodes } i \text{ and } j \text{ of } P. \tag{3.1b}$$

[When we say that the pair (P, p) satisfies CS, we implicitly assume that P is simple.]

The CS conditions given above are equivalent to the CS conditions for the shortest path problem given in Prop. 3.1 in Section 1.3, with the labels d_i of that proposition replaced by the negative prices $-p_i$. It follows that if a pair (P, p) satisfies CS, then the portion of P between any node $i \in P$ and any node $k \in P$ is a shortest path from i to k, while $p_i - p_k$ is the corresponding shortest distance. This can also be seen directly by observing that by Eq. (3.1b), $p_i - p_k$ is the length of the portion of P between i and k, and every path connecting i to k must have length at least equal to $p_i - p_k$ [add Eq. (3.1a) along the arcs of the path].

There is an interesting interpretation of the CS conditions in terms of a mechanical model [Min57]. Think of each node as a ball, and for every arc $(i,j) \in \mathcal{A}$, connect i and j with a string of length a_{ij}. (This requires that $a_{ij} = a_{ji} > 0$, which we assume.) Let the resulting balls-and-strings model be at an arbitrary position in three-dimensional space, and let p_i be the vertical coordinate of node i. Then the CS condition $p_i - p_j \leq a_{ij}$ clearly holds for all arcs (i,j), as illustrated in Fig. 3.1(b). If the model is picked up and left to hang from the origin node (by gravity – strings that are tight are perfectly vertical), then for all the tight strings (i,j) we have $p_i - p_j = a_{ij}$, so any tight chain of strings corresponds to a shortest path between the endnodes of the chain, as illustrated in Fig. 3.1(c). In particular, the length of the tight chain connecting the origin node 1 to any other node i is $p_1 - p_i$ and is also equal to the shortest distance from 1 to i. (This result is essentially the min path/max tension theorem described in Exercise 3.5 of Chapter 1.)

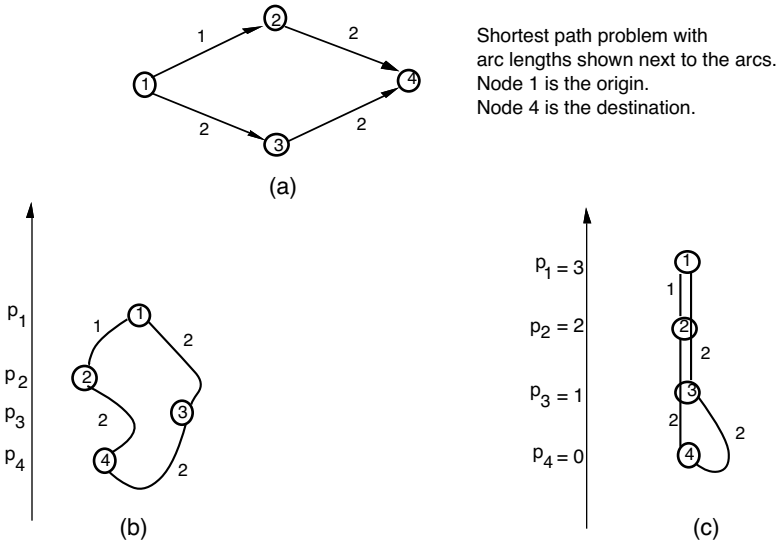


Figure 3.1 Illustration of the CS conditions. If each node is a ball, and for every arc $(i,j) \in \mathcal{A}$, nodes i and j are connected with a string of length a_{ij}, the vertical coordinates p_i of the nodes satisfy $p_i - p_j \leq a_{ij}$, as shown in (b) for the problem given in (a). If the model is picked up and left to hang from the origin node 1, then $p_1 - p_i$ gives the shortest distance to each node i, as shown in (c).

The algorithm of this section can be interpreted in terms of the balls-and-strings model, as we will see shortly. As a prelude to this, it is interesting to note that Dijkstra's algorithm can also be interpreted in terms of this

model, as shown in Fig. 3.2. At each iteration of the algorithm, the model is lifted by the origin node to the level where at least one more string becomes tight. Note that this interpretation leads to an interesting two-sided version of Dijkstra's algorithm for the single origin/single destination problem. In particular, it can be seen that a solution can be obtained by lifting the model upward from the origin, and simultaneously pulling the model downward from the destination. The corresponding algorithm is given in Exercise 3.5.

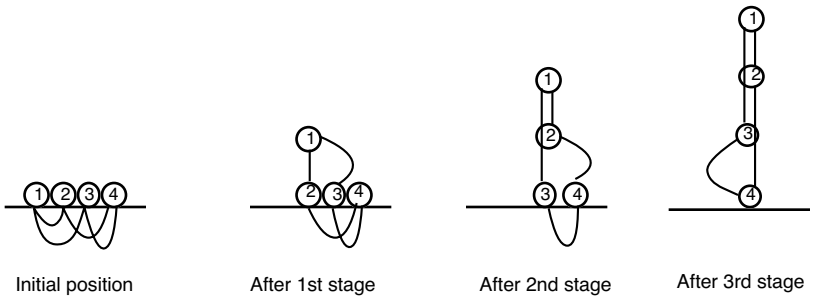


Figure 3.2 Interpretation of Dijkstra's algorithm in terms of the balls-and-strings model for the shortest path problem of Fig. 3.1. The model initially rests on a flat surface. It is then picked up from the origin and lifted in stages. At each stage the origin is raised to the next higher level at which one more node is ready to be lifted off the surface. Thus at each stage the shortest distance to at least one more node is found. Furthermore, the shortest distances of the nodes are obtained in the order of the nodes' proximity to the origin.

4.3.1 Algorithm Description and Analysis

We describe the algorithm in its simplest form for the case of a single origin and a single destination, and we defer the discussion of other and more efficient versions.

Let node 1 be the origin node and let t be the destination node. The algorithm maintains at all times a simple path $P = (1, i_1, i_2, \ldots, i_k)$. (When we refer to a path in this section, we implicitly assume that the path is *forward*; that is, all the arcs of the path are forward arcs.) The node i_k is called the *terminal* node of P. The degenerate path $P = (1)$ may also be obtained in the course of the algorithm. If i_{k+1} is a node that does not belong to a path $P = (1, i_1, i_2, \ldots, i_k)$ and (i_k, i_{k+1}) is an arc, *extending P by i_{k+1}* means replacing P by the path $(1, i_1, i_2, \ldots, i_k, i_{k+1})$, called the *extension of P by i_{k+1}*. If P does not consist of just the origin node 1, *contracting* P means replacing P by the path $(1, i_1, i_2, \ldots, i_{k-1})$.

The algorithm maintains also a price vector p satisfying CS together with P. We assume that an initial pair (P, p) satisfying CS is available. This is not a restrictive assumption when all arc lengths are nonnegative, since then one can use the default pair

$$P = (1), \qquad p_i = 0, \quad \forall\, i.$$

When some arcs have negative lengths, an initial choice of a pair (P, p) satisfying CS may not be obvious or available, but Exercise 3.2 provides a general method for finding such a pair.

We now describe the algorithm. Initially, (P, p) is any pair satisfying CS. The algorithm proceeds in iterations, transforming a pair (P, p) satisfying CS into another pair satisfying CS. At each iteration, the path P is either extended by a new node or else contracted by deleting its terminal node. In the latter case the price of the terminal node is increased strictly. A degenerate case occurs when the path consists by just the origin node 1; in this case the path is either extended or is left unchanged with the price p_1 being strictly increased. The iteration is as follows.

Typical Iteration

Let i be the terminal node of P. If

$$p_i < \min_{(i,j)\in\mathcal{A}} \big\{a_{ij} + p_j\big\}, \tag{3.2}$$

go to Step 1; else go to Step 2.

Step 1 (Contract path): Set

$$p_i := \min_{(i,j)\in\mathcal{A}} \big\{a_{ij} + p_j\big\}, \tag{3.3}$$

and if $i \neq 1$, contract P. Go to the next iteration.

Step 2 (Extend path): Extend P by node j_i where

$$j_i = \arg\min_{(i,j)\in\mathcal{A}} \big\{a_{ij} + p_j\big\}. \tag{3.4}$$

If j_i is the destination t, stop; P is the desired shortest path. Otherwise, go to the next iteration.

Note that following an extension (Step 2), P is a simple path from 1 to j_i; if this were not so, then adding j_i to P would create a cycle, and for every arc (i, j) of this cycle we would have $p_i = a_{ij} + p_j$. By adding this condition along the cycle, we see that the cycle should have zero length, which is not possible by our assumptions.

Figure 3.3 illustrates the algorithm. As can be seen from the example of this figure, the terminal node traces the tree of shortest paths from the origin to the nodes that are closer to the origin than the given destination. This behavior is typical when the initial prices are all zero as we will show shortly. We now derive the properties of the algorithm and establish its validity.

Proposition 3.1: The pairs (P, p) generated by the algorithm satisfy CS. Furthermore, for every pair of nodes i and j, and at all iterations, $p_i - p_j$ is an underestimate of the shortest distance from i to j.

Proof: We first show by induction that (P, p) satisfies CS. Indeed, the initial pair satisfies CS by assumption. Consider an iteration that starts with a pair (P, p) satisfying CS and produces a pair $(\bar{P}, \bar{p})$. Let i be the terminal node of P. If

$$p_i = \min_{(i,j)\in\mathcal{A}} \{a_{ij} + p_j\}, \tag{3.5}$$

then $\bar{P}$ is the extension of P by a node j_i and $\bar{p} = p$, implying that the CS condition (3.1b) holds for all arcs of P as well as arc (i, j_i) [since j_i attains the minimum in Eq. (3.5); cf. Eq. (3.4)].

Suppose next that

$$p_i < \min_{(i,j)\in\mathcal{A}} \{a_{ij} + p_j\}.$$

Then if P is the degenerate path (1), the CS condition holds vacuously. Otherwise, $\bar{P}$ is obtained by contracting P, we have $\bar{p}_i > p_i$, and for all nodes $j \in \bar{P}$, we have $\bar{p}_j = p_j$, implying the CS conditions (3.1a) and (3.1b) for arcs outgoing from nodes of $\bar{P}$. Also, for the terminal node i, we have

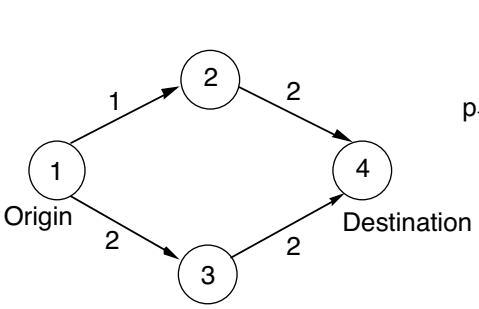
$$\bar{p}_i = \min_{(i,j)\in\mathcal{A}} \{a_{ij} + p_j\},$$

implying the CS condition (3.1a) for arcs outgoing from that node as well. Furthermore, since $\bar{p}_i > p_i$ and $\bar{p}_k = p_k$ for all $k \neq i$, we have $\bar{p}_k \leq a_{kj} + \bar{p}_j$ for all arcs (k, j) outgoing from nodes $k \notin P$. This completes the induction proof that (P, p) satisfies CS.

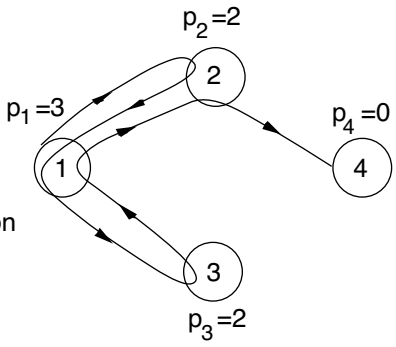
Finally consider any path from a node i to a node j. By adding the CS condition (3.1a) along that path, we see that the length of the path is at least $p_i - p_j$, proving the last assertion of the proposition. **Q.E.D.**

Proposition 3.2: If P is a path generated by the algorithm, then P is a shortest path from the origin to the terminal node of P.

Proof: This follows from the CS property of the pair (P, p) shown in Prop. 3.1; see the remarks following the CS conditions (3.1). In particular, by the CS condition (3.1b), P has length $p_1 - p_i$, and by the CS condition (3.1a),



Shortest path problem with arc lengths as shown



Trajectory of terminal node and final prices generated by the algorithm

Iteration #	**Path P prior to the iteration**	**Price vector p prior to the iteration**	**Type of action during the iteration**
1	(1)	$(0,0,0,0)$	contraction at 1
2	(1)	$(1,0,0,0)$	extension to 2
3	$(1,2)$	$(1,0,0,0)$	contraction at 2
4	(1)	$(1,2,0,0)$	contraction at 1
5	(1)	$(2,2,0,0)$	extension to 3
6	$(1,3)$	$(2,2,0,0)$	contraction at 3
7	(1)	$(2,2,2,0)$	contraction at 1
8	(1)	$(3,2,2,0)$	extension to 2
9	$(1,2)$	$(3,2,2,0)$	extension to 4
10	$(1,2,4)$	$(3,2,2,0)$	stop

Figure 3.3 An example illustrating the algorithm starting with $P = (1)$ and $p = 0$.

every path connecting 1 and i must have length at least equal to $p_1 - p_i$. **Q.E.D.**

Interpretations of the Algorithm

The algorithm can be interpreted in terms of a balls-and-strings model where nodes are raised in stages as illustrated in Fig. 3.4. All nodes are resting initially on a flat surface. At each stage, we raise the *last* node in a tight chain that starts at the origin to the level at which at least one more string becomes tight. This should be contrasted with Dijkstra's algorithm (cf. Fig. 3.2), where we raise the entire set of nodes that are connected with the origin via a tight chain.

For an alternative interpretation, denote for each node i,

$$D_i = \text{shortest distance from the origin 1 to node } i, \tag{3.6}$$

with $D_1 = 0$ by convention. By Prop. 3.1, we have throughout the algorithm

$$p_1 - p_j \leq D_j, \qquad \forall\ j \in \mathcal{N},$$

while by Prop. 3.2, we have

$$p_1 - p_i = D_i, \qquad \text{for all } i \in P.$$

The preceding two relations imply that

$$D_i + p_i - p_t \leq D_j + p_j - p_t, \qquad \forall\ i \in P, \text{ and } j \in \mathcal{N}.$$

Since by Prop. 3.1 $p_j - p_t$ is an estimate of the shortest distance from j to t, we may view the quantity

$$D_j + p_j - p_t$$

as *an estimate of the shortest distance from 1 to t using only paths passing through j*. Thus, intuitively, it makes sense to consider a node j as "most desirable" for inclusion in the algorithm's path if $D_j + p_j - p_t$ is minimal.

Based on the preceding interpretation, it can be seen that:

(a) The algorithm maintains a path consisting of "most desirable" candidates for participation in a shortest path from 1 to t.

(b) The algorithm extends P by a node j if and only if j is a "most desirable" candidate.

(c) The algorithm contracts P if the terminal node i has no neighbor that is "most desirable." Then, the estimate of i's shortest distance to t is

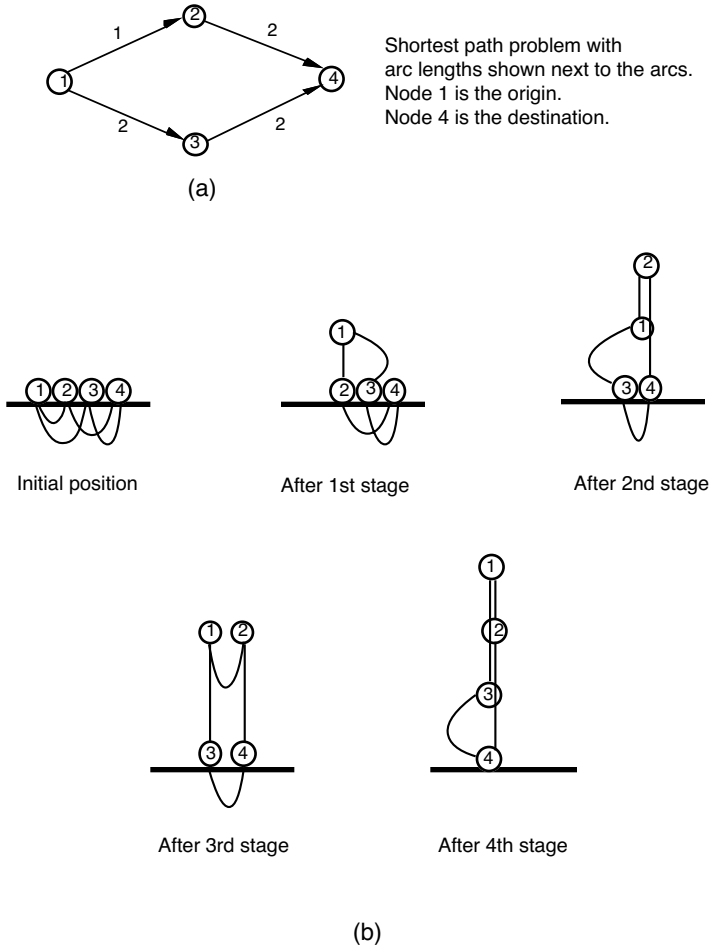


Figure 3.4 Illustration of the algorithm of this section in terms of the balls-and-strings model for the problem shown in (a). The model initially rests on a flat surface, and various balls are then raised in stages. At each stage we raise a single ball $i \neq t$, which is at a lower level than the origin and can be reached from the origin through a sequence of tight strings; i should not have any tight string connecting it to another ball, which is at a lower level, that is, i should be the last ball in a tight chain hanging from the origin. (If the origin does not have any tight string connecting it to another ball, which is at a lower level, we use $i =$ origin.) We then raise i to the first level at which one of the strings connecting it to a ball at a lower level becomes tight. Each stage corresponds to a contraction. The ball i, which is being raised, corresponds to the terminal node of the path.

improved (i.e., is increased), and i is not "most desirable" (since $D_i+p_i-p_t$ is not minimal anymore), thus justifying its deletion from P. Node i will be revisited only after $D_i + p_i - p_t$ becomes minimal again, after sufficiently large increases of the prices of the currently "most desirable" nodes.

The preceding interpretation suggests also that the nodes become terminal for the first time in the order of the initial values $D_j + p_j^0 - p_t^0$, where

$$p_i^0 = \text{initial price of node } i. \tag{3.7}$$

To formulate this property, denote for every node i

$$d_i = D_i + p_i^0. \tag{3.8}$$

Index the iterations by $1, 2, \ldots$, and let

$$k_i = \text{the first iteration at which node } i \text{ becomes the terminal node}, \tag{3.9}$$

where by convention, $k_1 = 0$ and $k_i = \infty$ if i never becomes a terminal node.

Proposition 3.3:

(a) At the end of iteration k_i we have $p_1 = d_i$.

(b) If $k_i < k_j$, then $d_i \leq d_j$.

Proof: (a) At the end of iteration k_i, P is a shortest path from 1 to i by Prop. 3.2, while the length of P is $p_1 - p_i^0$.

(b) By part (a), at the end of iteration k_i we have $p_1 = d_i$, while at the end of iteration k_j we have $p_1 = d_j$. Since p_1 is monotonically nondecreasing during the algorithm and $k_i < k_j$, the result follows. **Q.E.D.**

Note that the preceding proposition shows that when all arc lengths are nonnegative, and the default initialization $p = 0$ is used, the nodes become terminal for the first time in the order of their proximity to the origin. This property is also evident from the interpretation of the algorithm in terms of the balls-and-strings model; cf. Fig. 3.4.

Termination – Running Time of the Algorithm

The following proposition establishes the validity of the algorithm.

Proposition 3.4: If there exists at least one path from the origin to the destination, the algorithm terminates with a shortest path from the origin to the destination. Otherwise the algorithm never terminates and $p_1 \to \infty$.

Proof: Assume first that there is a path from node 1 to the destination t. Since by Prop. 3.1 $p_1 - p_t$ is an underestimate of the (finite) shortest distance from 1 to t, p_1 is monotonically nondecreasing, and p_t is fixed throughout the algorithm, it follows that p_1 must stay bounded. We next claim that p_i must stay bounded for all i. Indeed, in order to have $p_i \to \infty$, node i must become the terminal node of P infinitely often, implying (by Prop. 3.1) that $p_1 - p_i$ must be equal to the shortest distance from 1 to i infinitely often, which is a contradiction since p_1 is bounded.

We next show that the algorithm terminates. Indeed, it can be seen with a straightforward induction argument that for every node i, either p_i is equal to its initial value or else p_i is the length of some path starting at i plus the initial price of the final node of the path; we call this the *modified length* of the path. Every path starting at i can be decomposed into a simple path and a finite number of cycles, each having positive length by assumption (Exercise 1.5 in Section 1.1), so the number of distinct modified path lengths within any bounded interval is bounded. Now, p_i was shown earlier to be bounded. Furthermore, each time i becomes the terminal node by extension of the path P, p_i is strictly larger over the preceding time i became the terminal node of P, corresponding to a strictly larger modified path length. It follows that the number of times i can become a terminal node by extension of the path P is bounded. Since the number of path contractions between two consecutive path extensions is bounded by the number of nodes in the graph, the number of iterations of the algorithm is bounded, implying that the algorithm terminates.

Assume now that there is no path from node 1 to the destination. Then the algorithm will never terminate, so by the preceding argument some node i will become the terminal node by extension of the path P infinitely often, and $p_i \to \infty$. At the end of iterations where this happens, $p_1 - p_i$ must be equal to the shortest distance from 1 to i, implying that $p_1 \to \infty$. **Q.E.D.**

We will now estimate the running time of the algorithm, assuming that all the arc lengths and initial prices are integer. Our estimate involves the set of nodes

$$\mathcal{I} = \{i \mid d_i \leq d_t\}; \tag{3.10}$$

by Prop. 3.3, these are the only nodes that ever become terminal nodes of the paths generated by the algorithm. Let us denote

$$I = \text{number of nodes in } \mathcal{I}, \tag{3.11}$$

$$G = \max_{i \in \mathcal{I}} g_i, \tag{3.12}$$

where g_i is the number of outgoing incident arcs of node i, and let us also denote by E the product

$$E = I \cdot G. \tag{3.13}$$

Proposition 3.5: Assume that there exists at least one path from the origin 1 to the destination t, and that the arc lengths and initial prices are all integer. The worst case running time of the algorithm is $O\big(E\big(D_t + p_t^0 - p_1^0\big)\big)$.

Proof: Each time a node i becomes the terminal node of the path, we have $p_i = p_1 - D_i$ (cf. Prop. 3.2). Since at all times we have $p_1 \leq D_t + p_t^0$ (cf. Prop. 3.1), it follows that

$$p_i = p_1 - D_i \leq D_t + p_t^0 - D_i.$$

Using the definitions $d_t = D_t + p_t^0$ and $d_i = D_i + p_i^0$, and the fact $d_i \geq d_1$ (cf. Prop. 3.3), we see that throughout the algorithm we have

$$p_i - p_i^0 \leq d_t - d_i \leq d_t - d_1 = D_t + p_t^0 - p_1^0, \qquad \forall\ i \in \mathcal{I}.$$

Therefore, since prices increase by integer amounts, $D_t + p_t^0 - p_1^0 + 1$ bounds the number of times that each price p_i increases (with an attendant path contraction if $i \neq 1$). The computation per iteration is bounded by a constant multiple of the number of outgoing arcs of the terminal node of the path, so the computation corresponding to contractions and price increases is $O\big(E\big(D_t + p_t^0 - p_1^0\big)\big)$.

The number of path extensions with $i \in \mathcal{I}$ becoming the terminal node of the path is bounded by the number of increases of p_i, which in turn is bounded by $D_t + p_t^0 - p_1^0 + 1$. Thus the computation corresponding to extensions is also $O\big(E\big(D_t + p_t^0 - p_1^0\big)\big)$. **Q.E.D.**

The actual running time of the algorithm can indeed, in the worst case, depend strongly on the shortest distance D_t, as suggested by the estimate of the preceding proposition. This is illustrated in Fig. 3.5 with a graph involving a cycle with relatively small length. It is possible to use scaling to turn the algorithm into one that is polynomial (see [Ber90]), but in practice this device does not seem particularly effective, because the practical performance of the algorithm is typically much better than suggested by the preceding running time estimate. In fact, for randomly generated problems, it appears that the number of iterations can be estimated quite reliably (within a small multiplicative factor) by

$$n_t - 1 + \sum_{i \in \mathcal{I},\, i \neq t} (2n_i - 1), \tag{3.14}$$

where n_i is the number of nodes in a shortest path from 1 to i. For example, for the problem of Fig. 3.3 the above estimate is exact; see also Exercise 3.4. Note also that the number of iterations is reduced substantially when the algorithm is implemented in a forward/reverse mode, as discussed in the next subsection.

Assuming that the estimate (3.14) on the number of iterations is correct (within a constant factor) the running time of the algorithm depends critically on the number of nodes n_i in the shortest path to node i averaged over all nodes i. If the shortest paths are very long as in graphs with large diameter, the average number of arcs on a shortest path is $O(N)$, and the running time of the algorithm is usually $O(NA)$, where A is the number of arcs [a more accurate estimate is $O(NE)$, where E bounds the number of arcs in the subgraph of nodes that are closer to the origin than the destination t, cf. Eqs. (3.10)-(3.13)]. If on the other hand the shortest paths are short as in graphs with small diameter, the average number of arcs on a shortest path is $O(1)$, and the running time of the algorithm is usually $O(A)$ [a more accurate estimate is $O(E)$].

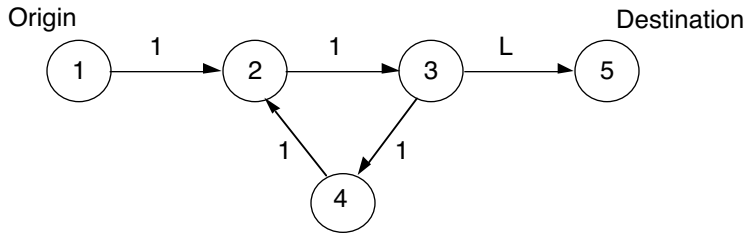


Figure 3.5 Example graph for which the number of iterations of the algorithm is not polynomially bounded. The lengths are shown next to the arcs and $L > 1$. By tracing the steps of the algorithm starting with $P = (1)$ and $p = 0$, we see that the price of node 3 will be first increased by 1 and then it will be increased by increments of 3 (the length of the cycle) as many times as is necessary for p_3 to reach or exceed L.

The Case of Multiple Destinations or Multiple Origins

To solve the problem with multiple destinations and a single origin, one can simply run the algorithm until every destination becomes the terminal node of the path at least once. Also, to solve the problem with multiple origins and a single destination, one can combine several versions of the algorithm – one for each origin. However, the different versions can share a common price vector, since regardless of the origin considered, the condition $p_i \leq a_{ij} + p_j$ is always maintained. There are several ways to operate such a method; they differ in the policy used for switching between different origins. One possibility is to run the algorithm for one origin and, after the shortest path is obtained, to switch to the next origin (without changing the price vector), and so on, until all origins are exhausted. Another possibility, which is probably preferable in

most cases, is to rotate between different origins, switching from one origin to another, if a contraction at the origin occurs or the destination becomes the terminal node of the current path.

4.3.2 Efficient Implementation – Forward/Reverse Algorithm

The main computational bottleneck of the algorithm is the calculation of

$$\min_{(i,j)\in\mathcal{A}}\{a_{ij}+p_j\},$$

which is done every time node i becomes the terminal node of the path. We can reduce the number of these calculations using the following observation. Since the CS condition $p_i \le a_{ij}+p_j$ is maintained at all times for all arcs (i,j), if some (i,j_i) satisfies

$$p_i = a_{ij_i}+p_{j_i}$$

it follows that

$$a_{ij_i}+p_{j_i} = \min_{(i,j)\in\mathcal{A}}\{a_{ij}+p_j\},$$

so the path can be extended by j_i if i is the terminal node of the path. This suggests the following implementation strategy: each time a path contraction occurs with i being the terminal node, we calculate

$$\min_{(i,j)\in\mathcal{A}}\{a_{ij}+p_j\}$$

together with an arc (i,j_i) such that

$$j_i = \arg\min_{(i,j)\in\mathcal{A}}\{a_{ij}+p_j\}.$$

At the next time node i becomes the terminal node of the path, we check whether the condition $p_i = a_{ij_i}+p_{j_i}$ is satisfied, and if it is we extend the path by node j_i without going through the calculation of $\min_{(i,j)\in\mathcal{A}}\{a_{ij}+p_j\}$. In practice this device is very effective, typically saving from a third to a half of the calculations of the preceding expression. The reason is that the test $p_i = a_{ij_i}+p_{j_i}$ rarely fails; the only way it can fail is if the price p_{j_i} is increased between the two successive times i became the terminal node of the path.

The preceding idea can be strengthened further. Suppose that whenever we compute the "best neighbor"

$$j_i = \arg\min_{(i,j)\in\mathcal{A}}\{a_{ij}+p_j\}$$

we also compute the "second best neighbor" k_i, given by

$$k_i = \arg \min_{(i,j)\in\mathcal{A},\ j\neq j_i} \{a_{ij} + p_j\},$$

and the corresponding "second best level"

$$w_i = a_{ik_i} + p_{k_i}.$$

Then, at the next time node i becomes the terminal node of the path, we can check whether the condition $a_{ij_i} + p_{j_i} \leq w_i$ is satisfied, and if it is we know that j_i still attains the minimum in the expression

$$\min_{(i,j)\in\mathcal{A}} \{a_{ij} + p_j\},$$

thereby obviating the calculation of this minimum. If on the other hand we have $a_{ij_i} + p_{j_i} > w_i$ (due to an increase of p_{j_i} subsequent to the calculation of w_i), we can check to see whether we still have $w_i = a_{ik_i} + p_{k_i}$; if this is so, then k_i becomes the "best neighbor,"

$$k_i = \arg \min_{(i,j)\in\mathcal{A}} \{a_{ij} + p_j\},$$

thus again obviating the calculation of the minimum.

With proper implementation the devices outlined above can typically reduce the number of calculations of the expression $\min_{(i,j)\in\mathcal{A}}\{a_{ij} + p_j\}$ by a factor that is typically in the range from 3 to 5, thereby dramatically reducing the total computation time.

Forward/Reverse Algorithm

In shortest path problems, one can exchange the roles of origins and destinations by reversing the directions of all arcs. It is therefore possible to use a destination-oriented version of our algorithm that maintains a path R that *ends* at the destination and changes at each iteration by means of a contraction or an extension. This algorithm, presented below and called the *reverse algorithm*, is equivalent to the earlier algorithm, which will henceforth be referred to as the *forward algorithm*. The CS conditions for the problem with arc directions reversed are

$$\overline{p}_j \leq a_{ij} + \overline{p}_i, \qquad \forall\ (i,j) \in \mathcal{A},$$

$$\overline{p}_j = a_{ij} + \overline{p}_i, \qquad \text{for all pairs of successive nodes } i \text{ and } j \text{ of } R,$$

where $\overline{p}$ is the price vector. By replacing $\overline{p}$ by $-p$, we obtain the CS conditions in the form $p_i \leq a_{ij} + p_j$, thus maintaining a common CS condition for both the

forward and the reverse algorithm. The following description of the reverse algorithm also replaces $\overline{p}$ by $-p$, with the result that the prices are *decreasing* instead of increasing. To be consistent with the assumptions made regarding the forward algorithm, we assume that each node except for the origin has at least one incoming arc.

In the reverse algorithm, initially, R is any path ending at the destination, and p is any price vector satisfying the CS conditions (3.1) together with R; for example,

$$R = (t), \qquad p_i = 0, \quad \forall\ i$$

if all arc lengths are nonnegative.

Typical Iteration of the Reverse Algorithm

Let j be the starting node of R. If

$$p_j > \max_{(i,j)\in\mathcal{A}} \{p_i - a_{ij}\},$$

go to Step 1; else go to Step 2.

Step 1 (Contract path): Set

$$p_j := \max_{(i,j)\in\mathcal{A}} \{p_i - a_{ij}\}$$

and, if $j \neq t$, contract R (that is, delete the starting node j of R). Go to the next iteration.

Step 2 (Extend path): Extend R by node i_j, (that is, make i_j the starting node of R, preceding j), where

$$i_j = \arg\max_{(i,j)\in\mathcal{A}} \{p_i - a_{ij}\}.$$

If i_j is the origin 1, stop; R is the desired shortest path. Otherwise, go to the next iteration.

The reverse algorithm is really the forward algorithm applied to a reverse shortest path problem, so by the results of Section 4.3.1, it is valid and terminates with a shortest path, if at least one path exists from 1 to t.

We now consider combining the forward and the reverse algorithms into one. In this combined algorithm, we initially have a price vector p, and two paths P and R, satisfying CS together with p, where P starts at the origin and R ends at the destination. The paths P and R are extended and contracted according to the rules of the forward and the reverse algorithms, respectively, and the combined algorithm terminates when P and R have a common node. Since P and R satisfy CS together with p throughout the algorithm, it is seen that when P and R meet, say at node i, the composite path consisting of the portion of P from 1 to i and the portion of R from i to t will be shortest.

Combined Algorithm

Step 1 (Run forward algorithm): Execute several iterations of the forward algorithm (subject to the termination condition), at least one of which leads to an increase of the origin price p_1. Go to Step 2.

Step 2 (Run reverse algorithm): Execute several iterations of the reverse algorithm (subject to the termination condition), at least one of which leads to a decrease of the destination price p_t. Go to Step 1.

To justify the combined algorithm, note that p_1 can only increase and p_t can only decrease during its course, and that the difference $p_1 - p_t$ can be no more than the shortest distance between 1 and t. Assume that the arc lengths and the initial prices are integer, and that there is at least one path from 1 to t. Then, p_1 and p_t can change only by integer amounts, and $p_1 - p_t$ is bounded. Hence, p_1 and p_t can change only a finite number of times, guaranteeing that there will be only a finite number of executions of Steps 1 and 2 of the combined algorithm. By the results of Section 4.3.1, each Step 1 and Step 2 must contain only a finite number of iterations of the forward and the reverse algorithms, respectively. It follows that the algorithm must terminate. Note that this argument relies on the requirement that p_1 increases at least once in Step 1 and p_t decreases at least once in Step 2. Without this requirement, one can construct examples showing that the combined algorithm may never terminate.

In practice, it appears that the combined algorithm is typically much faster than either the forward or the reverse algorithm (often by a factor of the order of ten or more). In particular, the running time of the (exclusively) forward algorithm is typically proportional to the product $m_F h_F$, where m_F is the number of nodes reached by the algorithm, and h_F is the average number of nodes on the shortest paths from the origin to these nodes [cf. Eq. (3.14)]. Similarly, the running time of the (exclusively) reverse algorithm is typically proportional to the product $m_R h_R$, where m_R is the number of nodes reached by the algorithm, and h_R is the average number of nodes on the shortest paths from these nodes to the destination. The running time of the forward/reverse algorithm is typically proportional to $\overline{m}_F \overline{h}_F + \overline{m}_R \overline{h}_R$, where the terms $\overline{m}_F$, $\overline{h}_F$, and $\overline{m}_R$, $\overline{h}_R$ are analogously defined, and correspond to the forward and the reverse portions of the algorithm, respectively. For many types of problems it appears that $\overline{m}_F + \overline{m}_R$ is much less than both m_F and m_R, while $\overline{h}_F + \overline{h}_R$ is roughly comparable to h_F and h_R. This explains the experimentally observed faster running time of the forward/reverse algorithm.

Note that the forward/reverse algorithm can also be interpreted in terms of the balls-and-strings model. Just as the forward algorithm can be viewed as a sequence of stages where some ball is lifted upward as in Fig. 3.4, the reverse algorithm can be viewed as a sequence of stages where some ball is

pulled downward. In the forward/reverse algorithm, we switch from raising to lowering balls and reversely. It is apparent that the algorithm works provided we make sure that, once in a while, the vertical distance between the origin and the destination increases either because the origin is raised or because the destination is lowered.

Forward/Reverse Algorithm for Multiple Origins

One may use the combined algorithm for the problem with multiple origins and a single destination using an algorithm that combines a separate forward version of the algorithm for each origin, and a reverse algorithm, which is common for all origins. The same price vector can be used for all forward versions, since the condition $p_i \leq a_{ij}+p_j$ is always maintained. One possibility is to rotate between different origins and the destination, switching from a forward algorithm for one origin to the reverse algorithm, then to another origin, and so on. The switch is made if a contraction at the origin (in the forward algorithm case) or the destination (in the reverse algorithm case) occurs, or if the destination becomes the terminal node of the current path (in the forward algorithm case). The code given in Appendix A.2 uses this scheme.

4.3.3 Relation to Naive Auction and Dual Coordinate Ascent

We now explain how our (forward) single origin/single destination algorithm can be viewed as an instance of application of the naive auction algorithm to a special type of assignment problem.

The naive auction algorithm was described in Section 1.2.4 for maximization assignment problems, where we want to maximize the benefit of matching n persons and n objects on a one-to-one basis. It is convenient here to reformulate the problem and the algorithm in terms of minimization by reversing the signs of the cost coefficients and the prices, and by replacing maximization by minimization. In particular, suppose that there is a cost c_{ij} for assigning person i with object j and we want to assign persons to objects so as to minimize the total cost. Mathematically, we want to find a feasible assignment that minimizes the total cost $\sum_{i=1}^{n} c_{ij_i}$, where by a feasible assignment we mean a set of person-object pairs $(1, j_1), \ldots, (n, j_n)$ such that the objects $j_1, \ldots, j_n$ are all distinct and $(i, j_i) \in \mathcal{A}$ for all i.

The naive auction algorithm proceeds in iterations and generates a sequence of price vectors p and (partial) assignments. At the beginning of each iteration, the complementary slackness condition

$$c_{ij_i} + p_{j_i} = \min_{(i,j)\in\mathcal{A}} \{c_{ij} + p_j\} \tag{3.15}$$

is satisfied for all pairs (i, j_i) of the assignment [cf. Eq. (2.8) in Section 1.2.3]. The initial price vector–assignment pair is required to satisfy this condition, but is otherwise arbitrary. If all persons are assigned, the algorithm terminates. If not, some person who is unassigned, say i, is selected. This person finds an object j_i, which is best in the sense

$$j_i = \arg \min_{(i,j)\in\mathcal{A}} \{c_{ij} + p_j\},$$

and then:

(a) Gets assigned to the best object j_i; the person that was assigned to j_i at the beginning of the iteration (if any) becomes unassigned.

(b) Sets the price of j_i to the level at which he or she is indifferent between j_i and the second best object – that is, he or she sets p_{j_i} to

$$p_{j_i} + w_i - v_i,$$

where v_i is the cost for acquiring the best object (including payment of the corresponding price),

$$v_i = \min_{(i,j)\in\mathcal{A}} \{c_{ij} + p_j\},$$

and w_i is the cost for acquiring the second best object,

$$w_i = \min_{(i,j)\in\mathcal{A},\, j\neq j_i} \{c_{ij} + p_j\}.$$

This process is repeated in a sequence of iterations until each person is assigned to an object.

The naive auction algorithm differs from the auction algorithm in the choice of the increment of the price increase. In the auction algorithm the price p_{j_i} is increased by $w_i - v_i + \epsilon$, where ϵ is a positive constant. Thus, the naive auction algorithm is the same as the auction algorithm, except that $\epsilon = 0$. This is, however, a significant difference. As shown in Section 1.2.4 (cf. Fig. 2.10), whereas the auction algorithm is guaranteed to terminate if at least one feasible assignment exists, the naive auction algorithm may cycle indefinitely, with some objects remaining unassigned. If, however, the naive auction algorithm terminates, the feasible assignment obtained upon termination is optimal (cf. Prop. 2.4 in Section 1.2.3).

Formulation of the Shortest Path Problem as an Assignment Problem

Given the shortest path problem of this section with node 1 as origin and node t as destination, we formulate the following assignment problem.

Let $2, \ldots, N$ be the "object" nodes, and for each node $i \neq t$ introduce a "person" node i'. For every arc (i, j) of the shortest path problem with $i \neq t$ and $j \neq 1$, introduce the arc (i', j) with cost a_{ij} in the assignment problem. Introduce also the zero cost arc (i', i) for each $i \neq 1, t$. Figure 3.6 illustrates the assignment problem and shows how, given the partial assignment that assigns object i to person i' for $i \neq 1, t$, paths from 1 to t can be associated with augmenting paths that start at $1'$ and end at t.

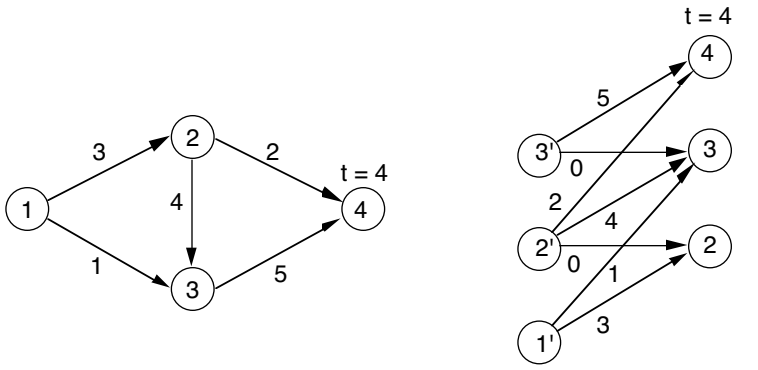


Figure 3.6 A shortest path problem (the origin is 1, the destination is $t = 4$) and its corresponding assignment problem. The arc lengths and the assignment costs are shown next to the arcs. Consider the partial assignment that assigns object i to person i' for $i \neq 1, t$. Then a shortest path can be associated with an optimal augmenting path that starts at $1'$ and ends at t.

Consider now applying the naive auction algorithm starting from a price vector p satisfying the CS condition (3.1a), that is,

$$p_i \leq a_{ij} + p_j, \qquad \forall\ (i, j) \in \mathcal{A} \tag{3.16}$$

and the partial assignment

$$(i', i), \qquad \forall\ i \neq 1, t.$$

This initial pair satisfies the corresponding complementary slackness condition (3.15), because the cost of the assigned arcs (i', i) is zero.

We impose an additional rule for breaking ties in the naive auction algorithm: if at some iteration involving the unassigned person i' the arc (i', i) is the best arc and is equally desirable with some other arc (i', j_i) (i.e., $p_i = a_{ij_i} + p_{j_i} = \min_{(i,j)\in\mathcal{A}}\{a_{ij} + p_j\}$), then the latter arc is preferred, that is, (i', j_i) is added to the assignment rather than (i', i). Furthermore, we introduce an inconsequential modification of the naive auction iteration involving a bid of

person $1'$, in order to account for the special way of handling a contraction at the origin in the shortest path algorithm. In particular, the bid of $1'$ will consist of finding an object j_1 attaining the minimum in

$$\min_{(1,j)\in\mathcal{A}}\{a_{1j}+p_j\},$$

assigning j_1 to $1'$, and deassigning the person assigned to j_1 (in the case $j_1 \neq t$), but *not* changing the price p_{j_1}.

It can now be shown that the naive auction algorithm with the preceding modifications is equivalent to the (forward) shortest path algorithm of Section 4.3.1. In particular, the following can be verified by induction:

(a) The CS condition (3.16) is preserved by the naive auction algorithm.

(b) Each assignment generated by the naive auction algorithm consists of a sequence of the form

$$(1', i_1), (i_1', i_2), \ldots, (i_{k-1}', i_k),$$

together with the additional arcs

$$(i', i), \quad \text{for } i \neq i_1, \ldots, i_k, t;$$

this sequence corresponds to a path $P = (1, i_1, \ldots, i_k)$ generated by the shortest path algorithm. As long as $i_k \neq t$, the (unique) unassigned person in the naive auction algorithm is person i_k', corresponding to the terminal node of the path. When $i_k = t$, a feasible assignment results, in which case the naive auction algorithm terminates, consistently with the termination criterion for the shortest path algorithm.

(c) In an iteration corresponding to an unassigned person i' with $i \neq 1$, the arc (i', i) is always a best arc; this is a consequence of the complementary slackness condition (3.16). Furthemore, there are three possibilities:

(1) (i', i) is the unique best arc, in which case (i', i) is added to the assignment, and the price p_i is increased by

$$\min_{(i,j)\in\mathcal{A}}\{c_{ij}+p_j\}-p_i;$$

this corresponds to contracting the current path by the terminal node i.

(2) There is an arc (i', j_i) with $j_i \neq t$, which is equally preferred to (i', i), that is,

$$p_i = a_{ij_i} + p_{j_i},$$

in which case, in view of the tie-breaking rule specified earlier, (i', j_i) is added to the assignment and the price p_{j_i} remains the

same. Furthermore, the object j_i must have been assigned to j_i' at the start of the iteration, so adding (i', j_i) to the assignment [and removing (j_i', j_i)] corresponds to extending the current path by node j_i. (The positivity assumption on the cycle lengths is crucial for this property to hold.)

(3) The arc (i', t) is equally preferred to (i', i), in which case the heretofore unassigned object t is assigned to i', thereby terminating the naive auction algorithm; this corresponds to the destination t becoming the terminal node of the current path, thereby terminating the shortest path algorithm.

We have thus seen that the shortest path algorithm may be viewed as an instance of the naive auction algorithm. However, the properties of the former algorithm do not follow from generic properties of the latter. As shown in Section 1.2.4 (see Fig. 2.12), the naive auction algorithm need not terminate in general. In the present context it does terminate, thanks to the special structure of the corresponding assignment problem, and also thanks to the positivity assumption on all cycle lengths.

We finally note that the forward/reverse version of the shortest path algorithm is equivalent to a combined forward/reverse version of naive auction, with the minor modifications described earlier; see the algorithm of Section 4.2 with $\epsilon = 0$.

Relation to Dual Coordinate Ascent

We next explain how the single origin/single destination algorithm can be viewed as a dual coordinate ascent method.

As was seen in Section 1.3 [see Eq. (1.3) of that section], the shortest path problem can be written in the minimum cost flow format as follows:

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

$$\text{subject to} \quad \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} = s_i, \qquad \forall\ i \in \mathcal{N}, \tag{3.17}$$

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

where

$$s_1 = 1, \qquad s_t = -1$$
$$s_i = 0, \qquad \forall\ i \ne 1, t.$$

The dual problem is (cf. Exercise 2.11 in Section 1.2)

$$\text{maximize } p_1 - p_t$$
$$\text{subject to } p_i - p_j \le a_{ij}, \qquad \forall\ (i,j) \in \mathcal{A}.$$

Let us associate with a given path $P = (1, i_1, i_2, \ldots, i_k)$ the flow

$$x_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are successive nodes in } P \\ 0 & \text{otherwise.} \end{cases}$$

Then, the CS conditions (3.1a) and (3.1b) are equivalent to the complementary slackness conditions

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

$$p_i = a_{ij} + p_j, \qquad \text{for all } (i,j) \in \mathcal{A} \text{ with } 0 < x_{ij}$$

for the preceding minimum cost flow problem. For a pair (x, p), the above conditions together with primal feasibility [the conservation of flow constraint (3.17) for all $i \in \mathcal{N}$, which in our case translates to the terminal node of the path P being the destination node] are necessary and sufficient for x to be primal-optimal and p to be dual-optimal. Thus, upon termination of the shortest path algorithm, the price vector p is an optimal dual solution.

To interpret the algorithm as a dual ascent method, note that a path contraction and an attendant price increase of the terminal node i of P, corresponds to a step along the price coordinate p_i that leaves the dual cost $p_1 - p_t$ unchanged if $i \neq 1$. Furthermore, an increase of the origin price p_1 by an increment δ improves the dual cost by δ. Thus, the algorithm may be viewed as a dual coordinate ascent algorithm, except that true ascent steps occur only when the origin price increases; all other ascent steps are "degenerate," producing a price increase but no change in dual cost.

The above interpretation can also be visualized in terms of the balls-and-strings model of Fig. 3.4. The dual cost is the vertical distance $p_1 - p_t$ between the balls representing the origin and the destination. In the forward algorithm, the destination stays fixed at its initial position, and this vertical distance increases only at the stages where the origin is raised; these are the 1st, 3rd, and 4th stages in the example of Fig. 3.4. In the forward/reverse version of the algorithm, the vertical distance increases only at the stages where either the origin is raised or the destination is lowered; at all other stages it stays unchanged.

E X E R C I S E S

Exercise 3.1

Apply the forward/reverse algorithm to the example of Fig. 3.5, and show that it terminates in a number of iterations that does not depend on the large arc length L. Construct a related example for which the number of iterations of the forward/reverse algorithm is not polynomially bounded.

Exercise 3.2 (Finding an Initial Price Vector [Ber90])

In order to initialize the shortest path algorithm of this section, one needs a price vector p satisfying the condition

$$p_i \leq a_{ij} + p_j, \qquad \forall\ (i,j) \in \mathcal{A}. \tag{3.18}$$

Such a vector may not be available if some arc lengths are negative. Furthermore, even if all arc lengths are nonnegative, there are many cases where it is important to use a favorable initial price vector in place of the default choice $p = 0$. This possibility arises in a reoptimization context with slightly different arc length data, or with a different origin and/or destination. This exercise gives an algorithm to obtain a vector p satisfying the condition (3.18), starting from another vector $\overline{p}$ satisfying the same condition for a different set of arc lengths $\overline{a}_{ij}$.

Suppose that we have a vector $\bar{p}$ and a set of arc lengths $\{\bar{a}_{ij}\}$, satisfying $\overline{p}_i \leq \overline{a}_{ij} + \overline{p}_j$ for all arcs (i,j), and we are given a new set of arc lengths $\{a_{ij}\}$. (For the case where some arc lengths a_{ij} are negative, this situation arises with $\overline{p} = 0$ and $\overline{a}_{ij} = \max\{0, a_{ij}\}$.) Consider the following algorithm that maintains a subset of arcs $\mathcal{E}$ and a price vector p, and terminates when $\mathcal{E}$ is empty. Initially

$$\mathcal{E} = \{(i,j) \in \mathcal{A} \mid a_{ij} < \bar{a}_{ij},\ i \neq t\}, \qquad p = \bar{p}.$$

The typical iteration is as follows:

Step 1 (Select arc to scan): If $\mathcal{E}$ is empty, stop; otherwise, remove an arc (i,j) from $\mathcal{E}$ and go to Step 2.

Step 2 (Add affected arcs to $\mathcal{E}$): If $p_i > a_{ij} + p_j$, set

$$p_i := a_{ij} + p_j$$

and add to $\mathcal{E}$ every arc (k,i) with $k \neq t$ that does not already belong to $\mathcal{E}$.

Assuming that each node i is connected to the destination t with at least one path, and that all cycle lengths are positive, show that the algorithm terminates with a price vector p satisfying

$$p_i \leq a_{ij} + p_j, \qquad \forall\ (i,j) \in \mathcal{A} \text{ with } i \neq t.$$

Exercise 3.3 (Extension for the Case of Zero Length Cycles)

Extend the algorithms of this section for the case where all arcs have nonnegative length but some cycles may consist exclusively of zero length arcs. *Hint:* Any cycle of zero length arcs generated by the algorithm can be treated as a single node.

Exercise 3.4

Consider the two single origin/single destination shortest path problems shown in Fig. 3.7.

(a) Show that the number of iterations required by the forward algorithm is estimated accurately by the formula given in Section 4.3.1,

$$n_t - 1 + \sum_{i \in \mathcal{I},\, i \neq t} (2n_i - 1),$$

where n_i is the number of nodes in a shortest path from 1 to i. Show also that the corresponding running times are $O(N^2)$.

(b) Show that for the problem of Fig. 3.7(a) the running time of the forward/reverse algorithm (with a suitable "reasonable" rule for switching between the forward and reverse algorithms) is $O(N^2)$ (the number of iterations is roughly half the corresponding number for the forward algorithm). Show also that for the problem of Fig. 3.7(b) the running time of the forward/reverse algorithm is $O(N)$.

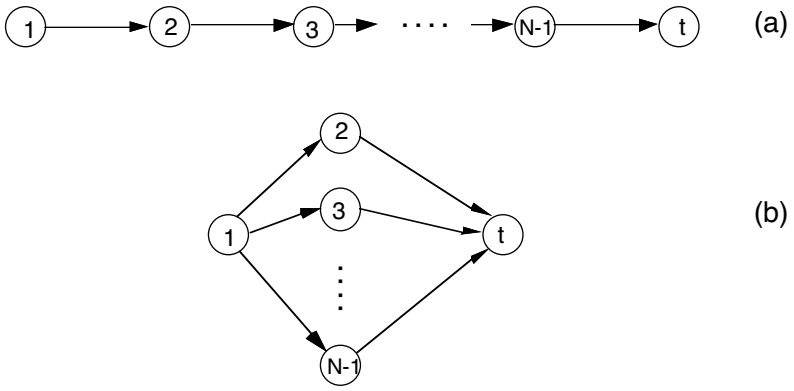


Figure 3.7 Shortest path problems for Exercise 3.4. In problem (a) arc lengths are equal to 1. In problem (b), the length of each arc $(1, i)$ is i, and the length of each arc (i, t) is N.

Exercise 3.5 (A Forward/Reverse Version of Dijkstra's Algorithm)

Consider the single origin/single destination shortest path problem and assume that all arc lengths are nonnegative. Let node 1 be the origin, let node t be the destination, and assume that there exists at least one path from 1 to

t. This exercise provides a forward/reverse version of Dijkstra's algorithm, which is motivated by the balls-and-strings model analogy of Figs. 3.1 and 3.2. In particular, the algorithm may be interpreted as alternately lifting the model upward from the origin (the following Step 1), and pulling the model downward from the destination (the following Step 2).

The algorithm maintains a price vector p and two node subsets W_1 and W_t. Initially, p satisfies the CS condition

$$p_i \leq a_{ij} + p_j, \qquad \forall\ (i,j) \in \mathcal{A}, \tag{3.19}$$

$W_1 = \{1\}$, and $W_t = \{t\}$. One may view W_1 and W_t as the sets of permanently labeled nodes from the origin and from the destination, respectively. The algorithm terminates when W_1 and W_t have a node in common. The typical iteration is as follows:

Step 1 (Forward Step): Find

$$\gamma^+ = \min\{a_{ij} + p_j - p_i \mid (i,j) \in \mathcal{A},\ i \in W_1,\ j \notin W_1\}$$

and let

$$V_1 = \{j \notin W_1 \mid \gamma^+ = a_{ij} + p_j - p_i \text{ for some } i \in W_1\}.$$

Set

$$p_i := \begin{cases} p_i + \gamma^+, & \text{if } i \in W_1 \\ p_i, & \text{if } i \notin W_1. \end{cases}$$

Set

$$W_1 := W_1 \cup V_1.$$

If W_1 and W_t have a node in common, terminate the algorithm; otherwise, go to Step 2.

Step 2 (Backward Step): Find

$$\gamma^- = \min\{a_{ji} + p_i - p_j \mid (j,i) \in \mathcal{A},\ i \in W_t,\ j \notin W_t\}$$

and let

$$V_t = \{j \notin W_t \mid \gamma^+ = a_{ji} + p_i - p_j \text{ for some } i \in W_t\}.$$

Set

$$p_i := \begin{cases} p_i - \gamma^-, & \text{if } i \in W_t \\ p_i, & \text{if } i \notin W_t. \end{cases}$$

Set

$$W_t := W_t \cup V_t.$$

If W_1 and W_t have a node in common, terminate the algorithm; otherwise, go to Step 1.

(a) Show that throughout the algorithm, the condition (3.19) is maintained. Furthermore, for all $i \in W_1$, $p_1 - p_i$ is equal to the shortest distance from 1 to i. Similarly, for all $i \in W_t$, $p_i - p_t$ is equal to the shortest distance from i to t. *Hint*: Show that if $i \in W_1$, there exists a path from 1 to i such that $p_m = a_{mn} + p_n$ for all arcs (m, n) of the path.

(b) Show that the algorithm terminates and that upon termination, $p_1 - p_t$ is equal to the shortest distance from 1 to t.

(c) Show how the algorithm can be implemented so that its running time is $O(N^2)$. *Hint*: Let d_{mn} denote the shortest distance from m to n. Maintain the labels

$$v_j^+ = \min\{d_{1i} + a_{ij} \mid i \in W_1,\ (i,j) \in \mathcal{A}\}, \qquad \forall\ j \notin W_1,$$

$$v_j^- = \min\{a_{ji} + d_{it} \mid i \in W_t,\ (j,i) \in \mathcal{A}\}, \qquad \forall\ j \notin W_t.$$

Let p_j^0 be the initial price of node j. Show that

$$\gamma^+ = \min\left\{\min_{j \notin W_1,\, j \notin W_t}\left(v_j^+ + p_j^0\right),\ p_t + \min_{j \notin W_1,\, j \in W_t}\left(v_j^+ + d_{jt}\right)\right\} - p_1, \tag{3.20}$$

$$\gamma^- = \min\left\{\min_{j \notin W_1,\, j \notin W_t}\left(v_j^- - p_j^0\right),\ -p_1 + \min_{j \in W_1,\, j \notin W_t}\left(v_j^- + d_{1j}\right)\right\} + p_t. \tag{3.21}$$

Use these relations to calculate γ^+ and γ^- in $O(N)$ time.

(d) Show how the algorithm can be implemented using binary heaps so that its running time is $O(A \log N)$. *Hint*: One possibility is to use four heaps to implement the minimizations in Eqs. (3.20) and (3.21).

(e) Apply the two-sided version of Dijkstra's algorithm of Exercise 3.8 of Section 3.1 with arc lengths $a_{ij} + p_j - p_i$ and with the termination criterion of part (c) of that exercise. Show that the resulting algorithm is equivalent to the one of the present exercise.

Exercise 3.6 (A Generalized Auction Algorithm)

Consider the shortest path problem, and assume that all cycles have positive length and that there is at least one path from each node to each other node. Let p be a price vector satisfying the CS condition

$$p_i \leq a_{ij} + p_j, \qquad \forall\ (i,j) \in \mathcal{A} \tag{3.22}$$

and let d_{mn} be the shortest distance from m to n. For each node m define the *chain of* m to be the subset of nodes

$$T_m(p) = \{m\} \cup \{n \mid p_m - p_n = d_{mn}\}.$$

(a) Show that $n \in T_m(p)$ if and only if either $n = m$ or else for every shortest path P from m to n we have

$$p_i = a_{ij} + p_j, \qquad \text{for all pairs of successive nodes } i \text{ and } j \text{ of } P.$$

Hint: Think in terms of the balls-and-strings model of Fig. 3.1.

(b) Define a *price rise* of node m to be the operation that increases the prices of the nodes in $T_m(p)$ by the increment

$$\gamma = \min\big\{a_{ij} + p_j - p_i \mid (i,j) \in \mathcal{A}, i \in T_m(p), j \notin T_m(p)\big\}.$$

Show that $\gamma > 0$ and that a price rise maintains the CS condition (3.22). Interpret a price rise in terms of the balls-and-strings model of Fig. 3.1.

(c) Let 1 be the origin node and let t be the destination node. Consider an algorithm that starts with a price vector satisfying Eq. (3.22), performs price rises of nodes m such that $t \notin T_m(p)$ (in any order), and terminates when $t \in T_1(p)$. Show that the algorithm terminates and that upon termination, $p_1 - p_t$ is the shortest distance from 1 to t.

(d) Show that the (forward) shortest path algorithm of this section is a special case of the algorithm of part (c).

(e) Adapt the algorithm of part (c) for the all origins/single destination problem, and discuss its potential for parallel computation. *Hint*: Note that if p^1 and p^2 are two price vectors satisfying

$$p_i^1 \le a_{ij} + p_j^1, \qquad p_i^2 \le a_{ij} + p_j^2, \qquad \forall\ (i,j) \in \mathcal{A},$$

then

$$\max\big\{p_i^1, p_i^2\big\} \le a_{ij} + \max\big\{p_j^1, p_j^2\big\}, \qquad \forall\ (i,j) \in \mathcal{A}.$$

(f) Develop an algorithm similar to the one of part (c) but involving price decreases in place of price increases. Develop also an algorithm involving both price increases and price decreases, which contains the forward/reverse algorithm of this section as a special case.

4.4 A GENERIC AUCTION ALGORITHM FOR THE MINIMUM COST FLOW PROBLEM

We will now generalize the auction idea and apply it to the minimum cost flow problem

$$\text{minimize} \quad \sum_{(i,j)\in\mathcal{A}} a_{ij}x_{ij} \tag{MCF}$$

subject to

$$\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} = s_i, \qquad \forall\ i \in \mathcal{N}, \tag{4.1}$$

$$b_{ij} \le x_{ij} \le c_{ij}, \qquad \forall\ (i,j) \in \mathcal{A}, \tag{4.2}$$

where a_{ij}, b_{ij}, c_{ij}, and s_i are given integers. For a given flow vector x, the surplus of each node i is denoted by

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

The algorithm to be described shortly maintains at all times a capacity-feasible flow vector x and a price vector p satisfying the ϵ-CS condition

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

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

(see Fig. 4.1). The usefulness of ϵ-CS is due in large measure to the following proposition.

Proposition 4.1: 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 minimum cost flow problem (MCF).

Proof: If x is not optimal, then by Prop. 2.1 in Section 1.2, there exists a simple cycle Y that has negative cost, i.e.,

$$\sum_{(i,j)\in Y^+} a_{ij} - \sum_{(i,j)\in Y^-} a_{ij} < 0, \tag{4.4}$$

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

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

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

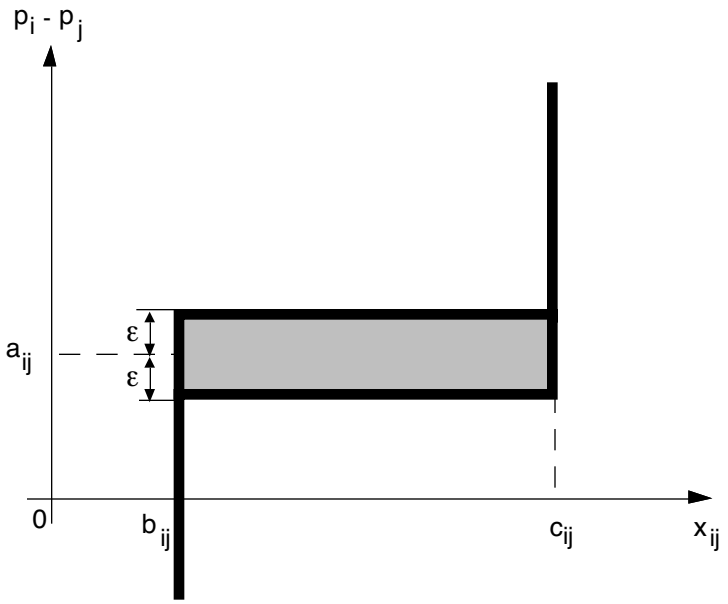


Figure 4.1 Illustration of ϵ-CS. All pairs of arc flows x_{ij} and price differences $p_i - p_j$ should either lie on the thick lines or in the shaded area between the thick lines.

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

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

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

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

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

Since the arc costs a_{ij} are integer, we obtain a contradiction of Eq. (4.4). **Q.E.D.**

Exercises 4.5-4.7 provide various improvements of the tolerance $\epsilon < 1/N$ in some specific contexts.

Some Basic Algorithmic Operations

We now define some terminology and computational operations that can be used as building blocks in various algorithms. Each of these definitions assumes that (x, p) is a flow–price vector pair satisfying ϵ-CS, and will be used only in that context.

Definition 4.1: An arc (i, j) is said to be ϵ^+-*unblocked* if

$$p_i = p_j + a_{ij} + \epsilon \qquad \text{and} \qquad x_{ij} < c_{ij}. \tag{4.5}$$

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

$$p_i = p_j - a_{ji} + \epsilon \qquad \text{and} \qquad b_{ji} < x_{ji}. \tag{4.6}$$

The *push 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.

In the algorithms of this chapter, flow is allowed to increase only along ϵ^+-unblocked arcs and is allowed to decrease only along ϵ^--unblocked arcs. The next two definitions specify the type of flow changes considered.

Definition 4.2: For an arc (i, j) [or arc (j, i)] of the push list of node i, let δ be a scalar such that $0 < \delta \leq c_{ij} - x_{ij}$ ($0 < \delta \leq x_{ji} - b_{ji}$, respectively). A *$\delta$-push at node i on arc (i, j)* [(j, i), respectively] consists of increasing the flow x_{ij} by δ (decreasing the flow x_{ji} by δ, respectively), while leaving all other flows, as well as the price vector unchanged.

In the context of the auction algorithm, a δ-push (with $\delta = 1$) corresponds to assigning an unassigned person to an object; this results in an increase of the flow on the corresponding arc from 0 to 1. The next operation consists of raising the prices of a subset of nodes by the maximum common increment γ that will not violate ϵ-CS.

Definition 4.3: A *price rise* of a nonempty, strict subset of nodes I (i.e., $I \neq \emptyset$, $I \neq \mathcal{N}$) consists of leaving the flow vector x and the prices of nodes not belonging to I unchanged, and increasing the prices of the nodes in I by the amount γ given by

$$\gamma = \begin{cases} \min\{S^+, S^-\}, & \text{if } S^+ \cup S^- \neq \emptyset \\ 0, & \text{if } S^+ \cup S^- = \emptyset, \end{cases} \tag{4.7}$$

where S^+ and S^- are the sets of scalars given by

$$S^+ = \{p_j + a_{ij} + \epsilon - p_i \mid (i, j) \in \mathcal{A} \text{ such that } i \in I, j \notin I, x_{ij} < c_{ij}\}, \tag{4.8}$$

$$S^- = \{p_j - a_{ji} + \epsilon - p_i \mid (j, i) \in \mathcal{A} \text{ such that } i \in I, j \notin I, b_{ji} < x_{ji}\}. \tag{4.9}$$

In the case where the subset I consists of a single node i, a price rise of the singleton set $\{i\}$ is also referred to as a *price rise of node* i. If the price increment γ of Eq. (4.7) is positive, the price rise is said to be *substantive*; if $\gamma = 0$, the price rise is said to be *trivial*. (A trivial price rise changes nothing; it is introduced in order to facilitate the statement of some of the algorithms given below.)

Note that every scalar in the sets S^+ and S^- of Eqs. (4.8) and (4.9) is nonnegative by the ϵ-CS conditions (4.3a) and (4.3b), respectively, so we have $\gamma \geq 0$, and we are indeed dealing with price rises.

The generic algorithm to be described shortly consists of a sequence of δ-push and price rise operations. The following proposition lists some properties of these operations that are important in the context of this algorithm.

Proposition 4.2: Let (x, p) be a flow–price vector pair satisfying ϵ-CS.

(a) The flow–price vector pair obtained after a δ-push or a price rise satisfies ϵ-CS.

(b) Let I be a subset of nodes such that $\sum_{i \in I} g_i > 0$. Then if the sets of scalars S^+ and S^- of Eqs. (4.8) and (4.9) are empty, the problem is infeasible.

Proof: (a) By the definition of ϵ-CS, the flow of an ϵ^+-unblocked and an ϵ^--unblocked arc can have any value within the feasible flow range. Since a δ-push only changes the flow of an ϵ^+-unblocked or ϵ^--unblocked arc, it cannot result in violation of ϵ-CS. Let p and p' be the price vectors before and after a price rise of a set I, respectively. For arcs (i, j) with $i \in I$, and $j \in I$, or with $i \notin I$ and $j \notin I$, the ϵ-CS condition (4.3) is satisfied by (x, p'), since it is satisfied by (x, p) and we have $p_i - p_j = p'_i - p'_j$. For arcs (i, j) with $i \in I$, $j \notin I$ and $x_{ij} < c_{ij}$ we have, using Eqs. (4.7) and (4.8),

$$p'_i - p'_j = p_i - p_j + \gamma \leq p_i - p_j + (p_j + a_{ij} + \epsilon - p_i) = a_{ij} + \epsilon, \tag{4.10}$$

so the ϵ-CS condition (4.3a) is satisfied. For arcs (j, i) with $i \in I$, $j \notin I$ and $x_{ji} > b_{ji}$ the ϵ-CS condition (4.3b) is similarly satisfied.

(b) Since $S^+ \cup S^-$ is empty,

$$x_{ij} = c_{ij}, \qquad \text{for all } (i, j) \in \mathcal{A} \text{ with } i \in I, j \notin I, \tag{4.11}$$

$$x_{ji} = b_{ji}, \qquad \text{for all } (j, i) \in \mathcal{A} \text{ with } i \in I, j \notin I. \tag{4.12}$$

We have

$$0 < \sum_{i \in I} g_i = \sum_{i \in I} s_i - \sum_{\{(i,j) \in \mathcal{A} \mid i \in I,\, j \notin I\}} x_{ij} + \sum_{\{(j,i) \in \mathcal{A} \mid i \in I,\, j \notin I\}} x_{ji}, \tag{4.13}$$

and by combining Eqs. (4.11)-(4.13), it follows that

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

For any feasible vector, the above relation implies that the sum of the divergences of nodes in I exceeds the capacity of the cut $[I, \mathcal{N} - I]$, which is a contradiction. Therefore, the problem is infeasible. **Q.E.D.**

The Generic Algorithm

Suppose that the minimum cost flow problem (MCF) is feasible, and consider a pair (x, p) satisfying ϵ-CS. Suppose that for some node i we have $g_i > 0$. There are two possibilities:

(a) The push list of i is nonempty, in which case a δ-push at node i is possible.

(b) The push list of i is empty, in which case the set $S^+ \cup S^-$ corresponding to the set $I = \{i\}$ [cf. Eqs. (4.8) and (4.9)] is nonempty, since the problem is feasible [cf. Prop. 4.2(b)]. Therefore, from Eqs. (4.7)-(4.9), a price rise of node i will be substantive.

Thus, *if $g_i > 0$ for some i and the problem is feasible, then either a δ-push or a substantive price rise is possible at node i.*

The preceding observations motivate a method, called *generic algorithm*, which starts with a pair (x, p) satisfying ϵ-CS and performs a sequence of δ-pushes and substantive price rises. The algorithm keeps ϵ at a fixed positive value and terminates when $g_i \leq 0$ for all nodes i.

Typical Iteration of the Generic Algorithm

> Perform in sequence and in any order a finite number of δ-pushes and substantive price rises; there should be at least one δ-push but not necessarily at least one price rise. Each δ-push should be performed at some node i with $g_i > 0$, and the flow increment δ must satisfy $\delta \leq g_i$. Furthermore, each price rise should be performed on a set I with $g_i \geq 0$ for all $i \in I$.

The following proposition establishes the validity of the generic algorithm.

Proposition 4.3: Assume that the minimum cost flow problem (MCF) is feasible. If the increment δ of each δ-push is integer, then the generic algorithm 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 the following observations.

(a) The algorithm preserves ϵ-CS; this is a consequence of Prop. 4.2.

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

(c) Once a node has nonnegative surplus, its surplus stays nonnegative thereafter. The reason is that a δ-push at a node i cannot drive the surplus of i below zero (since $\delta \leq g_i$), and cannot decrease the surplus of neighboring nodes.

(d) If at some time a node has negative surplus, 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 surplus can be involved in a price rise.

Suppose, to arrive at a contradiction, that the algorithm does not terminate. Then, since there is at least one δ-push per iteration, an infinite number of δ-pushes must be performed at some node i on some arc (i,j). Since for each δ-push, δ is integer, an infinite number of δ-pushes 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 δ-pushes.

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,

$$x_{ij} = c_{ij} \qquad \text{for all } (i,j) \in \mathcal{A} \text{ with } i \in \mathcal{N}^\infty, j \notin \mathcal{N}^\infty, \tag{4.14}$$

$$x_{ji} = b_{ji} \qquad \text{for all } (j,i) \in \mathcal{A} \text{ with } i \in \mathcal{N}^\infty, j \notin \mathcal{N}^\infty. \tag{4.15}$$

After some iteration, by (d) above, every node in $\mathcal{N}^\infty$ must have nonnegative surplus, so the sum of surpluses of the nodes in $\mathcal{N}^\infty$ must be positive at the start of the δ-pushes where either $g_i > 0$ or $g_j > 0$. It follows using the argument of the proof of Prop. 4.2(b) [cf. Eqs. (4.11)-(4.13)] 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 divergences 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 ternination 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. 4.1, if $\epsilon < 1/N$, x is optimal. **Q.E.D.**

The example of Fig. 4.2 shows how the generic algorithm may never terminate even for a feasible problem, if we do not require that it performs at least one δ-push per iteration.

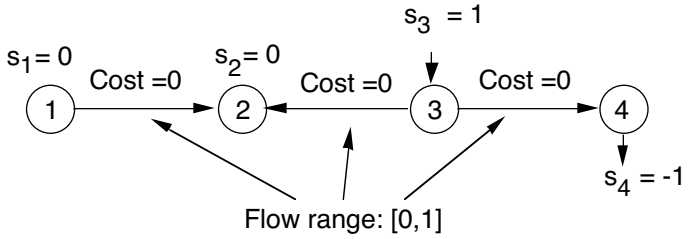


Figure 4.2 Example of a feasible problem where the generic algorithm does not terminate, if it does not perform at least one δ-push per iteration. Initially, all flows and prices are zero. Here, the first iteration raises the price of node 1 by ϵ. Subsequent iterations consist of a price rise of node 2 by an increment of 2ϵ followed by a price rise of node 1 by an increment of 2ϵ.

Consider now what happens when the problem is infeasible. Let us assume that the generic algorithm is operated so that for each δ-push, δ is integer. Then either the algorithm will 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 δ-pushes. In the latter case, from the proof of Prop. 4.3 it can be seen that the prices of the nodes involved in an infinite number of δ-pushes will diverge to infinity. This, together with a bound on the total price change of a node given in Exercise 4.9, can be used to detect infeasibility. It may also be possible to detect infeasibility by discovering in the course of the algorithm a subset of nodes I such that $\sum_{i \in I} g_i > 0$, and the sets of scalars S^+ and S^- of Eqs. (4.8) and (4.9) are empty [cf. Prop. 4.2(b)]. There is no guarantee, however, that such a set will be encountered during the algorithm's execution.

The generic algorithm can be applied in different ways to a variety of problems with special structure, yielding a variety of specific algorithms. In particular, it yields as a special case the auction algorithm for the symmetric assignment problem (see Exercise 4.1). The next section discusses an algorithm for the general minimum cost flow problem. We give here an example for an important class of transportation problems. Several related possibilities are explored in Exercises 4.1-4.4.

Example 4.1. Transportation Problems with Unit Sources

Consider a transportation problem where all the sources have unit supply. It has the form

$$\begin{aligned}
&\text{minimize} \quad \sum_{(i,j)\in\mathcal{A}} a_{ij}x_{ij} \\
&\text{subject to} \\
&\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} = 1, \qquad \forall\ i = 1,\ldots,m, \\
&\sum_{\{i|(i,j)\in\mathcal{A}\}} x_{ij} = \beta_j, \qquad \forall\ j = 1,\ldots,n, \\
&0 \le x_{ij} \le 1, \qquad \forall\ (i,j) \in \mathcal{A}.
\end{aligned}$$

Here a_{ij} are integers, and β_j are positive integers satisfying $\sum_{j=1}^{n} \beta_j = m$.

The following algorithm is a special case of the generic algorithm. (With a little thought it can also be seen to be equivalent to the auction algorithm with similar objects, given in Exercise 4.2.) At the start of each iteration, we have a pair (x, p) satisfying ϵ-CS and also the following two properties:

(a) $x_{ij} = 0$ or $x_{ij} = 1$ for all arcs (i, j).

(b) $g_i = 0$ or $g_i = 1$ for all sources i, and $g_j \le 0$ for all sinks j.

During the typical iteration, we do the following.

Step 1: Select a source i with $g_i = 1$ and an arc (i, j_i) with $p_{j_i} + a_{ij_i} = \min_{(i,j)\in\mathcal{A}}\{p_j + a_{ij}\}$.

Step 2: Perform a price rise of i (to the level $p_{j_i} + a_{ij_i} + \epsilon$), then a 1-push operation at node i along the arc (i, j_i), then another price rise of i (to the level $\min_{(i,j)\in\mathcal{A},\, j\neq j_i}\{p_j + a_{ij}\} + \epsilon$).

Step 3: Let m_i be such that

$$m_i = \arg \min_{\{m|(m,j_i)\in\mathcal{A},\, x_{mj_i}=1\}} \{p_m - a_{mj_i}\},$$

perform a price rise of j_i (to the level $p_{m_i} - a_{m_i j_i} + \epsilon$); if $g_{j_i} = 1$ (after the 1-push operation of Step 2) perform a 1-push operation at node j_i along arc (m_i, j_i), and then perform a price rise of j_i.

It can be seen that the properties (a) and (b) mentioned above, as well ϵ-CS, are preserved by the iteration. Furthermore, each iteration qualifies as an iteration of the generic algorithm, because a finite number of 1-pushes and price rises are performed, while at least one 1-push is performed. Therefore, Prop. 4.3 applies and asserts termination, if the problem is feasible. The flow

vector obtained upon termination will be optimal if $\epsilon < 1/(m+n)$. (Actually, for optimality it is sufficient that $\epsilon < 1/2n$; see Exercise 4.6.)

It is possible to derive auction algorithms for other types of transportation problems similar to the one just given. For example, a generalization for the case where the supplies of the sources can be greater than 1 is given in Exercise 4.8. Other generalizations, based on the reverse auction ideas of Section 4.2, can be used to solve various transportation problems involving inequality constraints. Finally, algorithms for problems with unit sinks are possible (see [BeC90a] and Exercise 4.3), as well as algorithms for the general transportation problem (see [BeC90a] and Exercise 4.4).

E X E R C I S E S

Exercise 4.1 (Relation to the Auction Algorithm for Assignment)

Describe how the auction algorithm for the symmetric assignment problem is a special case of the generic algorithm of this section. *Hint:* Introduce a price variable for each person. Show that a bid by a person i can be described as a price rise of i, followed by a 1-push operation along the arc (i, j) corresponding to the person's preferred object j, followed by another price rise of i, followed by a 1-push operation along arc (i', j) (if j is already assigned to i'), followed by a price rise of j.

Exercise 4.2 (Auction Algorithm with Similar Objects [BeC89a])

Given a symmetric assignment problem, we say that two objects j and j' are *similar*, and write $j \sim j'$, if for all persons $i = 1, \ldots, n$ we have

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

For each object j, the set of all objects similar to j is called the *similarity class* of j and is denoted $M(j)$. Consider a variation of the auction algorithm that is the same as the one of Section 4.1 except for one difference: in the bidding phase, w_i is defined now as

$$w_i = \max_{j \in A(i),\, j \notin M(j_i)} \{a_{ij} - p_j\}$$

(instead of $w_i = \max_{j \in A(i),\, j \neq j_i} \{a_{ij} - p_j\}$).

(a) Show that if the initial assignment S satisfies ϵ-CS together with the initial vector $\hat{p}$ defined by

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

that is,

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

the same is true of the assignment and the vector $\hat{p}$ obtained at the end of each assignment phase.

(b) Show also that the algorithm is equivalent to the algorithm of Example 4.1, and that for integer problem data it terminates with an optimal assignment if $\epsilon < 1/n$. (Actually, it is sufficient that $\epsilon < 1/m$, where m is the number of similarity classes, but proving this requires an argument of the type given in the proof of Prop. 4.1; see also the subsequent Exercise 4.6.)

Exercise 4.3

Derive an algorithm similar to the one of Example 4.1 for the transportation problem, where all sinks have unit demand. *Hint:* At the start of each iteration we must have $x_{ij} = 0$ or $x_{ij} = 1$ for all arcs (i,j), $g_i \geq 0$ for all sources i, and $g_j = 0$ or $g_j = -1$ for all sinks j.

Exercise 4.4 (Auction for Transportation Problems [BeC89a])

Consider the symmetric assignment problem. We say that two persons i and i' are *similar*, and write $i \sim i'$, if for all objects $j = 1, \ldots, N$ we have

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

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

(a) Generalize the auction algorithm with similar objects given in Exercise 4.2 so that it takes into account both similar persons and similar objects. *Hint:* Consider simultaneous bids by all persons in the same similarity class.

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

Exercise 4.5 (Improved Optimality Condition [BeE87b])

Show that if x is feasible, and x and p satisfy ϵ-CS, then x is optimal for the minimum cost flow problem, provided

$$\epsilon < \min_{\text{All simple cycles } Y} \left\{ -\frac{\text{Cost of } Y}{\text{Number of arcs of } Y} \,\middle|\, \text{Cost of } Y < 0 \right\},$$

where

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

Show that this is true even if the problem data are not integer.

Exercise 4.6 (Termination Tolerance for Transportation Problems)

Consider a transportation problem with m sources and n sinks and integer data. Show that in order for a feasible x to be optimal it is sufficient that it satisfies ϵ-CS together with some p and

$$\epsilon < \frac{1}{2\min\{m,n\}}$$

[instead of $\epsilon < 1/(m+n)$]. *Hint*: Modify the proof of Prop. 4.1 or use the result of Exercise 4.5.

Exercise 4.7 (Termination Tolerance for Multiassignment)

Consider the multiassignment problem of Section 4.2.2, and assume that the problem data are integer. Show that in order for the modified reverse auction algorithm to yield an optimal multiassignment it is sufficient that $\epsilon < 1/m$ (instead of $\epsilon < 1/n$). *Hint*: Observe the similarity with Exercises 4.5 and 4.6.

Exercise 4.8 (Auction for Capacitated Transportation Problems)

Consider the transportation problem

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

where the problem data are all integer, and $\alpha_i > 0$, $\beta_j > 0$ for all i and j, respectively. The following algorithm starts with a flow–price vector pair (x,p) such that ϵ-CS is satisfied, each x_{ij} is either 0 or 1, and

$$\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} \le \alpha_i, \qquad \forall\ i,$$

$$\sum_{\{i|(i,j)\in\mathcal{A}\}} x_{ij} \leq \beta_j, \qquad \forall\ j.$$

In the typical iteration, we select a source i with $\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} < \alpha_i$ (if no such source can be found the algorithm terminates). Then, we find

$$\hat{p}_i = \min\{z \mid \text{the number of sinks } j \text{ with } z \geq a_{ij} + p_j + \epsilon \text{ is greater than } \alpha_i\},$$

$$\tilde{p}_i = \min\{z \mid \text{the number of sinks } j \text{ with } z \geq a_{ij} + p_j + \epsilon \text{ is no less than } \alpha_i\}.$$

We also consider sinks j with $x_{ij} = 0$ and $\tilde{p}_i \geq a_{ij} + p_j + \epsilon$, and we find a subset T, which consists of $\alpha_i - \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij}$ such sinks and includes all sinks j with $\tilde{p}_i > a_{ij} + p_j + \epsilon$. We then set $p_i = \hat{p}_i$ and $x_{ij} = 1$ for all $j \in T$. After these changes, for each $j \in T$ with $\sum_{\{k|(k,j)\in\mathcal{A}\}} x_{kj} \geq \beta_j$, we find

$$\tilde{p}_j = \min_{\{k|x_{kj}=1\}} \{p_k - a_{kj} + \epsilon\},$$

and a source $\tilde{k}$ that attains the above minimum. If $\sum_{\{k|(k,j)\in\mathcal{A}\}} x_{kj} = \beta_j$, we set $p_j = \tilde{p}_j$; otherwise, we also find

$$\hat{p}_j = \min_{\{k|x_{kj}=1,\, k\neq\tilde{k}\}} \{p_k - a_{kj} + \epsilon\},$$

and we set $p_j = \hat{p}_j$ and $x_{\tilde{k}j} = 0$.

(a) Show that the algorithm is a special case of the generic algorithm, and for a feasible problem, it terminates with a pair (x, p) satisfying ϵ-CS. Show also that when $\alpha_i = 1$ and $\beta_j = 1$ for all i and j, respectively, the algorithm reduces to the (forward) auction algorithm for symmetric assignment problems.

(b) Derive a reverse and a combined forward/reverse version of the algorithm.

(c) Consider an asymmetric version of the problem where the equality constraints $\sum_{\{i|(i,j)\in\mathcal{A}\}} x_{ij} = \beta_j$ are replaced by the inequality constraints

$$\sum_{\{i|(i,j)\in\mathcal{A}\}} x_{ij} \leq \beta_j.$$

Derive a forward/reverse auction algorithm along the lines of the asymmetric assignment algorithm of Section 4.2.

Exercise 4.9 (Dealing with Infeasibility)

Consider the generic algorithm applied to a feasible minimum cost flow problem with initial prices p_i^0.

(a) Show that the total price increase $(p_i - p_i^0)$ of any node i prior to termination of the algorithm satisfies

$$p_i - p_i^0 \leq (N-1)(C+\epsilon) + \max_{j \in \mathcal{N}} p_j^0 - \min_{j \in \mathcal{N}} p_j^0,$$

where $C = \max_{(i,j) \in \mathcal{A}} |a_{ij}|$. *Hint:* Let x^0 be a feasible flow vector and let (x, p) be the flow–price vector pair generated by the algorithm prior to its termination. Show that there exist nodes t and s such that $g_t > 0$ and $g_s < 0$, and a simple path H starting at s and ending at t such that $x_{ij} - x_{ij}^0 > 0$ for all $(i,j) \in H^+$ and $x_{ij} - x_{ij}^0 < 0$ for all $(i,j) \in H^-$. Now use ϵ-CS to assert that

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

Add these conditions along H to obtain

$$p_t - p_s \leq (N-1)(C+\epsilon).$$

Use the fact $p_s = p_s^0$ to conclude that

$$p_t - p_t^0 \leq (N-1)(C+\epsilon) + p_s - p_s^0 \leq (N-1)(C+\epsilon) + \max_{j \in \mathcal{N}} p_j^0 - \min_{j \in \mathcal{N}} p_j^0.$$

(b) Discuss how the result of part (a) can be used to detect infeasibility.

(c) Suppose we introduce some artificial arcs to guarantee that the problem is feasible. Discuss how to select the cost coefficients of the artificial arcs so that optimal solutions are not affected in the case where the original problem is feasible; cf. Exercise 1.6 in Section 4.1.

Exercise 4.10 (Suboptimality of a Feasible Flow Satisfying ϵ-CS)

Let x^* be an optimal flow vector for the minimum cost flow problem and let x be a feasible flow vector satisfying ϵ-CS together with a price vector p.

(a) Show that the cost of x is within $\epsilon \sum_{(i,j) \in \mathcal{A}} |x_{ij} - x_{ij}^*|$ from the optimal. *Hint*: Show that $(x - x^*)$ satisfies CS together with p for a minimum cost flow problem with arcs (i,j) having flow range $[b_{ij} - x_{ij}^*, c_{ij} - x_{ij}^*]$ and arc cost $\hat{a}_{ij}$ that differs from a_{ij} by no more than ϵ.

(b) Show by example that the suboptimality bound $\epsilon \sum_{(i,j) \in \mathcal{A}} |c_{ij} - b_{ij}|$ deduced from part (a) is tight. *Hint*: Consider a graph with two nodes and multiple arcs connecting these nodes. All the arcs have cost ϵ except for one that has cost $-\epsilon$.

4.5 THE ϵ-RELAXATION METHOD

We now describe the ϵ-relaxation method, which is a special case of the generic algorithm of the previous section, where, at each iteration, all δ-pushes and price rises involve a single node. The ϵ-relaxation method may also be viewed as a mathematically equivalent method to the auction algorithm for the assignment problem of Section 4.1. Indeed the auction algorithm can be obtained as a special case of ϵ-relaxation (see Exercise 5.3). Conversely, we can convert the minimum cost flow problem to a transportation problem (see Example 1.3 in Section 1.1), and then convert the latter problem to an assignment problem (by creating enough duplicate persons and objects). The reader can verify that when the auction algorithm is applied to this assignment problem, and the computation is appropriately streamlined, one obtains the ϵ-relaxation method.

We assume that the problem is feasible. In practice, the method could be supplemented with additional mechanisms to detect infeasibility, as discussed in the preceding section (see also Exercise 4.9).

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; see e.g. [BeT89]. This can be easily enforced, although it does not seem to be very important in practice.)

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. During the iteration we perform several δ-push and price rise operations of the type described in the previous section involving node i.

Typical Iteration of the ϵ-Relaxation Method

Step 1: If the push list of node i is empty, go to Step 3; else select an arc a from the push list of i and go to Step 2.

Step 2: Let j be the end-node of arc a, which is opposite to i. Let

$$\delta = \begin{cases} \min\{g_i, c_{ij} - x_{ij}\} & \text{if } a = (i, j) \\ \min\{g_i, x_{ji} - b_{ji}\} & \text{if } a = (j, i). \end{cases} \tag{5.1}$$

Perform a δ-push of i on arc a. If as a result of this operation we obtain $g_i = 0$, go to Step 3; else go to Step 1.

Step 3: Perform a price rise of node i. If $g_i = 0$, go to the next iteration; else go to Step 1.

Some insight into the ϵ-relaxation iteration can be obtained by noting that in the limit as $\epsilon \to 0$ it yields the single node relaxation iteration of Section 3.3. Figure 5.1 illustrates the sequence of price rises in an ϵ-relaxation iteration; this figure should be compared with the corresponding Fig. 3.2 in Section 3.3 for the single node relaxation iteration. As Fig. 5.1 illustrates, the ϵ-relaxation iteration 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. 1.1 in Section 4.1.

We now establish the validity of the ϵ-relaxation method by using the analysis of the preceding section. In particular, we claim that the above iteration consists of a finite (but positive) number of δ-pushes with δ integer, and a finite (possibly zero) number of price rises at nodes with nonnegative surplus. Indeed, since the starting arc flows, the node supplies, and the arc flow bounds are integer, the flow increments δ of all δ-pushes will be positive integers throughout the algorithm. Furthermore, from Eq. (5.1) it is seen that the condition $\delta \leq g_i$ of the generic algorithm is satisfied. We also note that at most one δ-push per incident arc of node i is performed at each iteration because from Eq. (5.1) it is seen that a δ-push on arc a in Step 2 either sets the arc flow to the corresponding flow bound, which causes arc a to drop out of the push list of i through the end of the iteration, or else results in $g_i = 0$, which leads the iteration to branch to Step 3 and subsequently stop. Therefore, the number of δ-pushes 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 δ-push must occur before an iteration can stop.

Regarding price rises, it is seen that Step 3 can be reached under two conditions:

(a) The push list of i is empty and $g_i > 0$, in which case the price rise in Step 3 will be substantive [in view of the assumption that the problem is feasible and Prop. 4.2(b)], and the iteration will branch to Step 1 with the push list of i having at least one new arc, or

(b) $g_i = 0$, in which case the iteration will stop after a (possibly trivial) price rise in Step 3.

Thus, all price rises involve a node with nonnegative surplus as required in the generic algorithm. Since after each substantive price rise with $g_i > 0$ at least one δ-push must be performed, it follows that the number of substantive price rises per iteration is finite.

From the preceding observations it is seen that, if the problem is feasible, the ϵ-relaxation method is a special case of the generic algorithm and satisfies the assumptions of Prop. 4.3. Therefore, it must terminate with a feasible flow vector, which is optimal if $\epsilon < 1/N$.

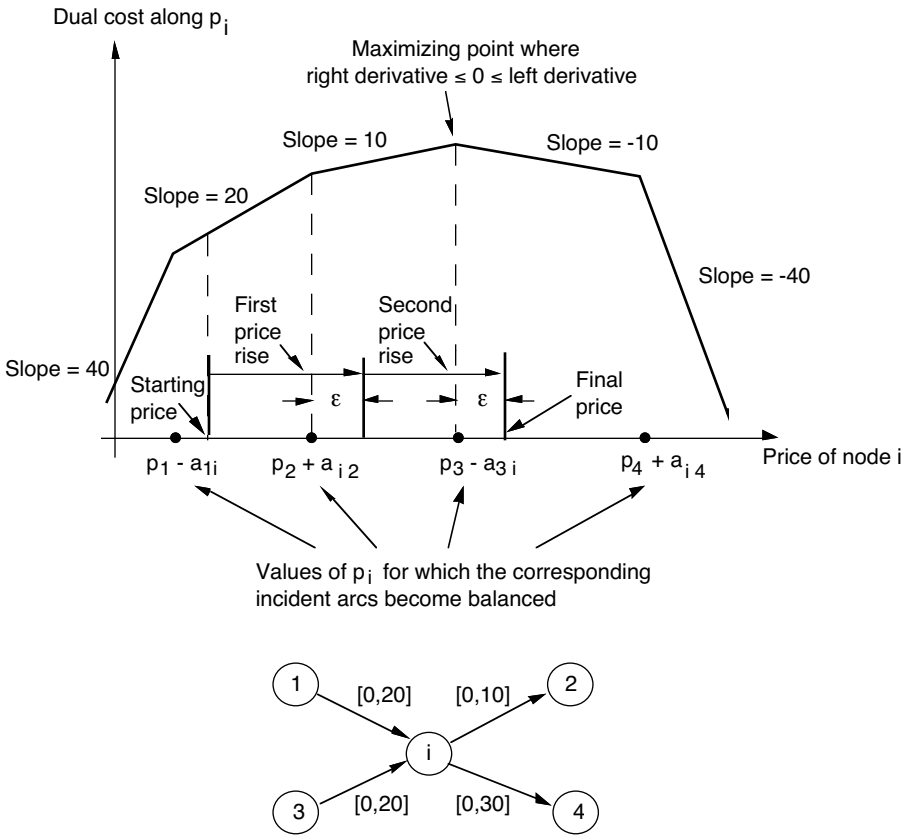


Figure 5.1 Illustration of the price rises of the ϵ-relaxation iteration. Here, node i has four incident arcs $(1,i)$, $(3,i)$, $(i,2)$, and $(i,4)$ with flow ranges $[0,20]$, $[0,20]$, $[0,10]$, and $[0,30]$, respectively, and supply $s_i = 0$. The arc costs and current prices are such that

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

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.

Scaling

The ϵ-relaxation method, with the use of some fairly simple data structures (the so called sweep implementation), but without the use of scaling, can be shown to have an

$$O\big(N^3 + N^2 L/\epsilon\big) \tag{5.2}$$

worst-case running time, where L is the maximum over the lengths of all simple paths, with the length of each arc (i,j) being the absolute reduced cost value $|p_j + a_{ij} - p_i|$, and p being the initial price vector. (The sweep implementation together with the above estimate were first given in [Ber86a]; see [BeE88] and [BeT89] for a detailed description and analysis.) Thus, the amount of work to solve the problem can depend strongly on the values of ϵ and L.

The ϵ-scaling technique discussed for the auction algorithm in Section 5.1 is also important in the context of the ϵ-relaxation method, and improves both the practical and the theoretical worst-case performance of the method. Although ϵ-scaling was first proposed in [Ber79] in connection with the auction algorithm, its first analysis was given in [Gol87] and [GoT90]. These references provided an $O\big(NA\log(N)\log(NC)\big)$ running time estimate for a scaled version of ϵ-relaxation that uses some complicated data structures called *dynamic trees*. By using ϵ-scaling and the sweep implementation referred to earlier, the worst-case running time of the algorithm for integer data can be shown to be $O\big(N^3\log(NC)\big)$, where $C = \max_{(i,j)\in\mathcal{A}} |a_{ij}|$; see [BeE87b], [BeE88], [BeT89]. These references actually analyze an alternative form of scaling, known as *cost scaling*, which is based on successively representing the cost coefficients by an increasing number of bits. Cost scaling and ϵ-scaling admit a very similar complexity analysis. Their practical performance is roughly comparable, although ϵ-scaling is somewhat easier to implement. For this reason, ϵ-scaling was used in the codes of Appendix A.4 and Appendix A.7.

Surplus Scaling

When applying the ϵ-scaling technique, except for the last scaling phase, it is not essential to reduce the surpluses 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 with nodes whose surplus 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 surplus and decreases (rather than increases) the

price of that node. Furthermore, one can mix positive surplus and negative surplus 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 [BeT89], 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.

Application to the Max-Flow Problem

The ϵ-relaxation method can be applied to the max-flow problem, once this problem is converted to the minimum cost flow format, involving a feedback arc connecting the sink with the source, and having cost -1 (see Example 1.2 in Section 1.2). Since all other arc costs are zero, the maximum path length L used in Eq. (5.2) is equal to 1, assuming a zero initial price vector. Therefore, the complexity estimate of Eq. (5.2) becomes

$$O\big(N^3 + N^2/\epsilon\big). \tag{5.3}$$

One can solve the problem without using scaling by taking $\epsilon = 1/(N+1)$, so in this case the preceding estimate yields an $O(N^3)$ worst-case running time. With the use of more sophisticated data structures, this running time can be considerably improved; see the references at the end of the chapter.

In practice, the ϵ-relaxation method initialized with zero flow and zero price vectors often finds a minimum cut very quickly. It may then work quite a bit more to set the remaining positive surpluses to zero. Thus, if one is interested in just a minimum cut or just the value of the maximum flow, it is worth testing periodically to see whether a minimum cut can be determined before the algorithm terminates. A method for detecting whether a minimum cut has been found is outlined in Exercise 5.4 and is used in the code of Appendix A.6. Given a minimum cut, one may find a maximum flow by continuing the algorithm until all node surpluses are zero, or by employing a version of the Ford-Fulkerson algorithm to return the positive surpluses to the source (see Exercise 5.4).

E X E R C I S E S

Exercise 5.1

Apply the ϵ-relaxation method to the problem of Fig. 2.3 of Section 3.2 with $\epsilon = 1$. Comment on the optimality of the solution obtained.

Exercise 5.2 (Degenerate Price Rises)

In this exercise, we consider a variation of the ϵ-relaxation method that involves *degenerate price rises*. A degenerate price rise changes the price of a node that currently has zero surplus to the maximum possible value that does not violate ϵ-CS with respect to the current flow vector (compare with degenerate price increases in the context of the single-node relaxation iteration where $\epsilon = 0$, as illustrated in Fig. 3.3 of Section 3.3). One example of such a price rise occurs when Step 3 of the ϵ-relaxation iteration is executed with $g_i = 0$.

Consider a variation of the ϵ-relaxation method where there are two types of iterations: (1) *regular* iterations, which are of the form described in the present section, and (2) *degenerate* iterations, which consist of a single degenerate price rise.

(a) Show that if the problem is feasible and the number of degenerate iterations is bounded by a constant times the number of regular iterations, then the method terminates with a pair (x, p) satisfying ϵ-CS.

(b) Show that the assumption of part (a) is essential for the validity of the method. *Hint*: Consider the example of Fig. 4.2.

Exercise 5.3 (Deriving Auction from ϵ-Relaxation)

Consider the assignment problem formulated as a minimum cost flow problem (see Example 1.1 in Section 1.1). We say that source i is assigned to sink j if (i, j) has positive flow. We consider a version of the ϵ–relaxation algorithm in which ϵ-relaxation iterations are organized as follows: between iterations (and also at initialization), only source nodes i can have positive surplus. Each iteration finds any unassigned source i (i.e., one with positive surplus), and performs an ϵ-relaxation iteration at i, and then takes the sink j to which i was consequently assigned and performs an ϵ-relaxation iteration at j, even if j has zero surplus. (If j has zero surplus, such an iteration will consist of just a degenerate price rise; see Exercise 5.2.)

More specifically, an iteration by an unassigned source i works as follows:

(1) Source node i sets its price to $p_j + a_{ij} + \epsilon$, where j minimizes $p_k + a_{ik} + \epsilon$ over all k for which $(i, k) \in \mathcal{A}$. It then sets $x_{ij} = 1$, assigning itself to j.

(2) Node i then raises its price to $p_{j'} + a_{ij'} + \epsilon$, where j' minimizes $p_k + a_{ik} + \epsilon$ for $k \neq j$, $(i, k) \in \mathcal{A}$.

(3) If sink j had a previous assignment $x_{i'j} = 1$, it breaks the assignment by setting $x_{i'j} := 0$. (One can show inductively that if this occurs, $p_j = p_{i'} - a_{i'j} + \epsilon$.)

(4) Sink j then raises its price p_j to

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

Show that the corresponding algorithm is equivalent to the Gauss-Seidel version of the auction algorithm.

Exercise 5.4 (Detecting a Minimum Cut Using ϵ-Relaxation)

Consider the max-flow problem with zero lower arc flow bounds. Suppose that we have a pair (x, p) satisfying the 1-CS condition

$$p_i - p_j \leq 1 \qquad \text{for all } (i,j) \in \mathcal{A} \text{ with } x_{ij} < c_{ij},$$

$$p_i - p_j \geq -1 \qquad \text{for all } (i,j) \in \mathcal{A} \text{ with } 0 < x_{ij},$$

$$p_s = p_t + N + 2.$$

[These are the 1-CS conditions for the equivalent minimum cost flow problem obtained by introducing an artificial arc (t, s) with cost $-(N+1)$.] Suppose also that we have a cut $Q = [\mathcal{S}, \mathcal{N} - \mathcal{S}]$ with $s \in \mathcal{S}$ and $t \notin \mathcal{S}$, which is saturated with respect to x. Finally, suppose that the surplus of all nodes in $\mathcal{S}$ except s is nonnegative, while the surplus of all nodes in $\mathcal{N} - \mathcal{S}$ except t is nonpositive.

(a) Show that Q is a minimum cut. *Hint*: Apply the ϵ-relaxation method with $\epsilon = 1$ starting with (x, p). Argue that the flux across Q will not change.

(b) Given Q, show that a maximum flow can be found by solving two feasibility problems (which are in turn equivalent to some other max-flow problems; cf. Exercise 2.5 in Section 1.2). One feasibility problem should involve just the nodes of $\mathcal{S}$ and the other should involve just the nodes not in $\mathcal{S}$.

(c) Construct algorithms based on augmentations that solve the feasibility problems in part (b), thereby yielding a maximum flow.

4.6 IMPLEMENTATION ISSUES

The main operations of auction algorithms involve scanning the incident arcs of nodes; this is a shared feature with dual ascent methods. For this reason the data structures and implementation ideas discussed in connection with

dual ascent methods, also apply to auction algorithms. In particular, for the max-flow and the minimum cost flow problems, using the *FIRST_IN*, *FIRST_OUT*, *NEXT_IN*, and *NEXT_OUT* arrays, described in Section 3.5, is convenient. In addition, a similar set of arrays can be used to store the arcs of the push lists in the ϵ-relaxation method; see the code given in Appendix A.7.

4.7 NOTES AND SOURCES

4.1. The auction algorithm, and the notions of ϵ-complementary slackness and ϵ-scaling were first proposed by the author in [Ber79] (see also [Ber85] and [Ber88]). Reference [BeE88] surveys coordinate step methods based on ϵ-complementary slackness and derives the worst-case complexity of the auction algorithm; see also [BeT89]. The parallel implementation aspects of the auction algorithm have been explored by several authors; see [BeC89c], [KKZ89], [PhZ88], [WeZ90], and [Zak90]. Exercise 1.3 that deals with the average complexity of the auction algorithm was inspired by [Sch90], which derives related results for the Jacobi version of the algorithm and its potential for parallelism.

4.2. The reverse auction algorithm and its application in inequality constrained assignment problems is due to [BCT91], which discusses additional related algorithms and gives computational results. Note that aside from faster convergence, the combined forward/reverse algorithm has potential for greater concurrency in a parallel machine than the forward algorithm.

4.3. The auction algorithm for shortest paths is due to the author [Ber90]. This reference also discusses an arc length scaling technique that can be used to make the algorithm polynomial. In practice, this scaling technique does not seem to be very useful, primarily because pseudopolynomial practical behavior appears to be unlikely, particularly for the forward/reverse version of the algorithm. The MS thesis [Pol91] discusses the parallelization aspects of the method. The interpretation of the forward/reverse version of Dijkstra's algorithm of Exercise 3.5 and the generalized auction algorithm of Exercise 3.6 are new.

4.4. The generic auction algorithm for minimum cost flow problems is due to [BeC89b]. Reference [BeC89a] describes various special cases involving bipartite graphs; see also Exercises 4.2 and 4.4.

4.5. The ϵ-relaxation method is due to the author; it was first published in [Ber86a] and [Ber86b], although it was known much earlier (since the development of the equivalent auction algorithm). Various implementations of the method aimed at improved worst-case complexity can be found in [BeE87b], [BeE88], [BeT89], [Gol87], and [GoT90]. The worst-case complexity of ϵ-

scaling was first analyzed in [Gol87] in connection with various implementations of the ϵ-relaxation method. Computational experience suggests, however, that the complexity analysis of the ϵ-relaxation method is not very useful in predicting practical behavior. In particular, despite its excellent polynomial complexity, the method is typically outperformed by the relaxation method of the previous chapter, which can be shown to have pseudopolynomial complexity. However, the ϵ-relaxation method is better suited for parallel computation than the other minimum cost flow methods described in this book; see [BeT89] for a discussion of related issues.

When the ϵ-relaxation method is applied to the max-flow problem, it bears a close resemblance with an $O(N^3)$ max-flow algorithm first proposed in [Gol85b]; see also [GoT86], which describes a similar max-flow algorithm that achieves a more favorable worst-case complexity using sophisticated data structures. These max-flow algorithms were derived from a different point of view that is unrelated to duality or ϵ-CS. They use node "labels," which in the context of the ϵ-relaxation approach can be viewed as prices. The max-flow version of the ϵ-relaxation method, first given in [Ber86a], is simpler than the algorithms of [Gol85] and [GoT86] in that it obtains a maximum flow in one phase rather than two. It can also be initialized with arbitrary prices, whereas in the max-flow algorithms of [Gol85], [GoT86] the initial prices must satisfy $p_i \leq p_j + 1$ for all arcs (i, j). Related max-flow algorithms are discussed in [AhO86], [ChM87], and [AMO89].