May 1992 **LIDS - P - 2108**

Auction Algorithms for Network Flow Problems: A Tutorial Introduction [1]

by

Dimitri P. Bertsekas[2]

Abstract

This paper surveys a new and comprehensive class of algorithms for solving the classical linear network flow problem and its various special cases such as shortest path, max-flow, assignment, transportation, and transhipment problems. The prototype method, from which the other algorithms can be derived, is the auction algorithm for the assignment problem. This is an intuitive method that operates like a real auction where persons compete for objects by raising their prices through competitive bidding; the prices can be viewed as dual variables. Conceptually, auction algorithms represent a significant departure from the cost improvement idea that underlies primal simplex and dual ascent methods; at any one iteration, they may deteriorate both the primal and the dual cost. Auction algorithms perform very well for several important types of problems, both in theory and in practice, and they are also well suited for parallel computation.

[1] Research supported by NSF under Grant No. DDM-8903385 and Grant CCR-9103804, and by the ARO under Grant DAAL03-86-K-0171. To be published in the Journal of Computational Optimization and its Applications.

[2] Laboratory for Information and Decision Systems, M.I.T, Cambridge, Mass. 02139.

Contents

1. INTRODUCTION

The classical algorithms for solving linear network flow problems are primal cost improvement methods, including simplex methods, which iteratively improve the primal cost by moving flow around simple cycles, and dual ascent methods, which iteratively improve the dual cost by changing the prices of a subset of nodes by equal amounts.

Auction algorithms, the subject of this tutorial paper, are fundamentally different; they depart from the cost improvement idea and at any one iteration, they may deteriorate both the primal and the dual cost, although in the end they find an optimal primal solution. Their origin lies in nondifferentiable optimization (where nonmonotonic subgradient-based algorithms are common), and in the ϵ-subgradient method of Bertsekas and Mitter [BeM73] in particular (where the progress of the method depends on gradually reducing an ϵ parameter to an acceptable tolerance level). Auction algorithms are also highly intuitive and easy to understand; they can be explained in terms of economic competition concepts, which are couched on everyday experience.

Auction algorithms originated in 1979 with an algorithm proposed by the author for the classical assignment problem [Ber79]. This algorithm was further developed in [Ber85], [Ber88], and [BeE88]. The motivation was to solve the problem by using parallelism in a natural way. It turned out, however, that the resulting method was very fast in a serial environment as well. Subsequent work extended the auction algorithm to other linear network flow problems. In particular, an extension to the minimum cost problem, the ϵ-relaxation method, was given by the author in [Ber86a] and [Ber86b]. An auction algorithm for transportation problems was given by the author in collaboration with D. Castañon in [BeC89a]. An auction algorithm for shortest paths was given by the author in [Ber91b]. Finally, a max-flow algorithm, developed by Goldberg [Gol85] from an entirely different point of view, fundamentally involves auction ideas and can be viewed as the natural extension of the auction algorithm to the max-flow problem.

All of the algorithms just mentioned can be derived from the original 1979 auction algorithm for the assignment problem. The derivation is based on well-known transformations of the general linear minimum cost flow problem and its various special cases (e.g., shortest path, max-flow, etc.) to equivalent assignment problems. Once the auction algorithm is applied to the corresponding equivalent assignment problem and the computations are streamlined, one obtains an auction algorithm for the original problem.

The purpose of this paper is to provide a tutorial introduction to auction algorithms. For textbook presentations, we refer the reader to [BeT89] and [Ber91a]. We will initially focus on the basic algorithm for the assignment problem, and then discuss various extensions to other problems. In Sections 2 and 3, we develop the auction algorithm for the symmetric assignment problem. In particular, we discuss a number of issues that are common to all types of auction algorithms, including ϵ-scaling, and dealing with infeasibility. We also discuss an alternative form of the auction algorithm, called *reverse auction*; while in regular auction persons bid for objects by raising their prices, in reverse auction objects compete for persons by essentially offering discounts. In Section 4 we develop auction algorithms for shortest path problems. In Section 5, we discuss the extension of the auction algorithm to transportation problems, and in Section 6 we give a generic auction algorithm for the minimum cost network flow problem. In

Section 7, we develop the ϵ-relaxation method for the minimum cost flow problem as a special case of the generic algorithm. The ϵ-relaxation method is specialized to the max-flow problem in Section 8, yielding algorithms similar to those of Goldberg and Tarjan [Gol85], [GoT86]. In Section 9 we discuss specialized auction algorithms for inequality constrained problems such as the asymmetric assignment problem. Finally, in Section 10 we discuss the computational aspects of auction algorithms, including issues of parallel and asynchronous implementation.

2. ASSIGNMENT BY NAIVE AUCTION

In the classical *symmetric assignment problem* there are n persons and n objects that we have to match on a one-to-one basis. There is a 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. We are given a set $\mathcal{A}$ of pairs (i, j) that can be matched. For each person i, we denote by $A(i)$ the set of objects that can be matched with i

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

and for each object j, we denote by $B(j)$ the set of persons that can be matched with j

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

By an *assignment* we mean a set S of person-object pairs (i, j) such that each person i and each object j is involved in at most one pair from S. If the number of pairs in S is n, so that every person is assigned to a distinct object, we say that S is *feasible*; otherwise S is said to be *infeasible*. If a feasible assignment exists the problem is said to be feasible, and otherwise it is said to be infeasible. We seek a feasible assignment [a set of person-object pairs $(1, j_1), \ldots, (n, j_n)$ from $\mathcal{A}$, such that the objects $j_1, \ldots, j_n$ are all distinct], which is optimal in the sense that it maximizes the total benefit $\sum_{i=1}^{n} a_{ij_i}$.

The symmetric assignment problem should be distinguished from its *asymmetric* version where the number of objects is larger than the number of persons, and while there is a requirement to assign every person, there is no such requirement for the objects. Auction algorithms for asymmetric and other related assignmment problems will be discussed in Section 9.

The assignment problem is important in many practical contexts, but it is also of great theoretical importance. Despite its simplicity, it embodies a fundamental linear programming structure. One of the most important type of linear programming problem, the minimum cost network flow problem, can be reduced to the assignment problem by means of a simple reformulation (see e.g. [BeT89], p. 335, [Ber91a], p. 17, [PaS82], p. 149, and Section 6). Thus, any method for solving the assignment problem can be generalized to solve the minimum cost flow problem. For this reason, the assignment problem has served as a convenient starting point for important algorithmic ideas in linear programming. For example, the primal-dual method ([FoF62], [Min60]), was motivated and developed through Kuhn's Hungarian method [Kuh55], the first specialized method for the assignment problem.

To develop an intuitive understanding of the auction algorithm, it is helpful to introduce an economic equilibrium problem that turns out to be equivalent to the assignment problem.

Consider the possibility of matching the n objects with the n persons through a market mechanism, viewing each person as an economic agent acting in his/her own best interest. Suppose that object j has a price p_j and that the person who receives the object must pay the price p_j. Then, the (net) value of object j for person i is $a_{ij} - p_j$ and each person i would logically want to be assigned to an object j_i with maximal value, that is, with

$$a_{ij_i} - p_{j_i} = \max_{j \in A(i)} \{a_{ij} - p_j\}. \tag{1}$$

The economic system would then be at equilibrium, in the sense that no person would have an incentive to act unilaterally, seeking another object.

Equilibrium assignments and prices are naturally of great interest to economists, but there is also a fundamental relation with the assignment problem; it turns out that *an equilibrium assignment offers maximum total benefit (and thus solves the assignment problem), while the corresponding set of prices solves an associated dual problem*. This is a consequence of the duality theorem of linear programming (see e.g. [Dan63], [PaS82], [Lue84]). In the terminology of linear programming, relation (1) is known as *complementary slackness* (CS for short). A simple, first principles proof of the relation of equilibria to optimal assignments and dual optimization is also developed in Appendix 1.

The Naive Auction Algorithm

Let us consider a natural process for finding an equilibrium assignment and price vector. We will call this process the *naive* auction algorithm, because it has a serious flaw, as will be seen shortly. Nonetheless, this flaw will help motivate a more sophisticated and correct algorithm.

The naive auction algorithm proceeds in iterations and generates a sequence of price vectors and assignments. At the beginning of each iteration, the CS condition

$$a_{ij_i} - p_{j_i} = \max_{j \in A(i)} \{a_{ij} - p_j\} \tag{2}$$

is satisfied for all pairs (i, j_i) of the assignment. If all persons are assigned, the algorithm terminates. Otherwise a nonempty subset I of persons i that are unassigned is selected and the following computations are performed.

Typical Iteration of Naive Auction Algorithm

Let I be a nonempty subset of persons that are unassigned.

Bidding Phase: Each person $i \in I$ finds an object j_i which offers maximal value, that is,

$$j_i \in \arg\max_{j \in A(i)} \{a_{ij} - p_j\}, \tag{3}$$

and computes a bidding increment

$$\gamma_i = v_i - w_i, \tag{4}$$

where v_i is the best object value,

$$v_i = \max_{j \in A(i)} \{a_{ij} - p_j\}, \tag{5}$$

and w_i is the second best object value

$$w_i = \max_{j \in A(i), j \neq j_i} \{a_{ij} - p_j\}. \tag{6}$$

[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.]

Assignment Phase: Each object j that is selected as best object by a nonempty subset $P(j)$ of persons in I, determines the highest bidder

$$i_j = \arg\max_{i \in P(j)} \gamma_i, \tag{7}$$

raises its prices by the highest bidding increment $\max_{i \in P(j)} \gamma_i$, and gets assigned to the highest bidder i_j; the person that was assigned to j at the beginning of the iteration (if any) becomes unassigned.

The algorithm continues with a sequence of iterations until all persons have an assigned object.

Note that γ_i cannot be negative since $v_i \geq w_i$ [compare Eqs. (5) and (6)], so the object prices tend to increase. In fact, when i is the only bidder, γ_i is the largest bidding increment for which CS is maintained following the assignment of i to his/her preferred object. Just as in a real auction, bidding increments and price increases spur competition by making the bidder's own preferred object less attractive to other potential bidders.

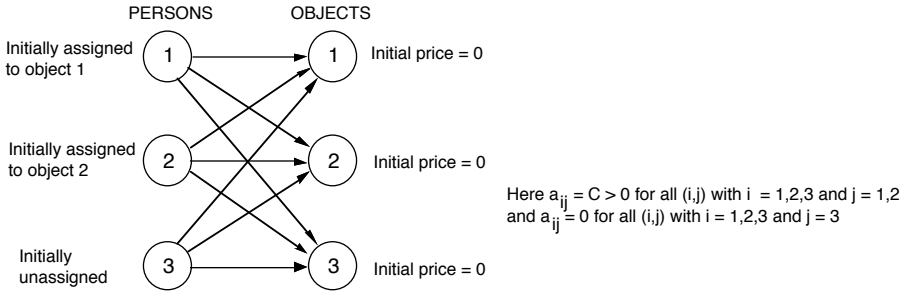
Note also that there is some freedom in choosing the subset of persons I that bid during an iteration. One possibility is to let I consist of a single unassigned person. This version, known as the *Gauss-Seidel version* because of its similarity with Gauss-Seidel methods for solving systems of nonlinear 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.

3. ϵ-COMPLEMENTARY SLACKNESS AND THE AUCTION ALGORITHM

Unfortunately, the naive auction algorithm does not always work (although it is an excellent initialization procedure for other methods that are based on price adjustment, e.g. primal-dual or relaxation). The difficulty is that the bidding increment γ_i is zero when more than one object offers maximum value for the bidder i. As a result, a situation may be created where several persons contest a smaller number of equally desirable objects without raising their prices, thereby creating a never ending cycle; see Fig. 1.

To break such cycles, we introduce a perturbation mechanism, motivated by real auctions where each bid for an object must raise the object's price by a minimum positive increment, and bidders must on occasion take risks to win their preferred objects. In particular, let us fix a positive scalar ϵ and say that an assignment and a price vector p satisfy *ϵ-complementary slackness (or ϵ-CS for short)* if

$$a_{ij_i} - p_{j_i} \geq \max_{j \in A(i)} \{a_{ij} - p_j\} - \epsilon, \tag{8}$$



At Start of Iteration #	Object Prices	Assigned Pairs	Bidder	Preferred Object	Bidding Increment
1	0,0,0	(1,1), (2,2)	3	2	0
2	0,0,0	(1,1), (3,2)	2	2	0
3	0,0,0	(1,1), (2,2)	3	2	0

Figure 1: Illustration of how the naive auction algorithm may never terminate for a three person and three object problem. Here objects 1 and 2 offer benefit $C > 0$ to all persons, and object 3 offers benefit 0 to all persons. The algorithm cycles as persons 2 and 3 alternately bid for object 2 without changing its price because they prefer equally object 1 and object 2 ($\gamma_i = 0$).

for all assigned pairs (i, j_i). In words, to satisfy ϵ-CS, all assigned persons must be assigned to objects that are within ϵ of being best.

The Auction Algorithm

We now reformulate the previous auction process so that the bidding increment is always at least equal to ϵ. The resulting method, the *auction algorithm*, is the same as the naive auction algorithm, except that the bidding increment γ_i is

$$\gamma_i = v_i - w_i + \epsilon, \tag{9}$$

[rather than $\gamma_i = v_i - w_i$ as in Eq. (4)]. With this choice, the ϵ-CS condition is satisfied. The particular increment $\gamma_i = v_i - w_i + \epsilon$ used in the auction algorithm is the maximum amount with this property. Smaller increments γ_i would also work as long as $\gamma_i \geq \epsilon$, but using the largest possible increment accelerates the algorithm. This is consistent with experience from real auctions, which tend to terminate faster when the bidding is aggressive.

It can be shown that this reformulated auction process terminates in a finite number of iterations, necessarily with a feasible assignment and a set of prices that satisfy ϵ-CS. To see this for the case of a fully dense problem ($\mathcal{A}$ consists of all person-object pairs), note that if an object receives a bid in k

iterations, its price must exceed its initial price by at least $k\epsilon$. Thus, for sufficiently large k, the object will become "expensive" enough to be judged "inferior" to some object that has not received a bid so far. It follows an object can receive a bid in a limited number of iterations while some other object still has not yet received any bid. On the other hand, once all objects receive at least one bid, the auction terminates. Thus, the auction algorithm must terminate, and in fact the preceding argument shows that, for the case of zero initial prices, the total number of iterations in which an object receives a bid is no more than

$$\frac{\max_{(i,j)} |a_{ij}|}{\epsilon}.$$

If each iteration involves a bid by a single person, the total number of iterations is no more than n times the preceding quantity, and since each bid requires $O(n)$ operations, the running time of the algorithm is $O\big(n^2 \max_{(i,j)} |a_{ij}|/\epsilon\big)$.

This proof can be generalized for the case of a sparse problem (one where the set of person-object pairs that can be assigned is limited), as long as the problem is feasible; see Appendix 2. Figure 2 shows how the auction algorithm, based on the bidding increment $\gamma_i = v_i - w_i + \epsilon$ [cf. Eq. (9)], overcomes the cycling problem of the example of Fig. 1.

When the auction algorithm terminates, we have an assignment satisfying ϵ-CS, but is this assignment optimal? The answer here depends strongly on the size of ϵ. In a real auction, a prudent bidder would not place an excessively high bid for fear that he/she might win the object at an unnecessarily high price. Consistent with this intuition, we can show that if ϵ is small, then the final assignment will be "almost optimal." In particular, the following proposition (proved in Appendix 1) shows that *the total cost of the final assignment is within $n\epsilon$ of being optimal*. The idea is that when a feasible assignment and a set of prices satisfy ϵ-CS, they also satisfy CS (and are therefore primal and dual optimal, respectively) for a *slightly perturbed* problem where all costs a_{ij} are the same as before, except for the costs of the n assigned pairs, which are modified by an amount no more than ϵ.

Proposition 1: A feasible assignment satisfying ϵ-complementary slackness together with some price vector is within $n\epsilon$ of being optimal.

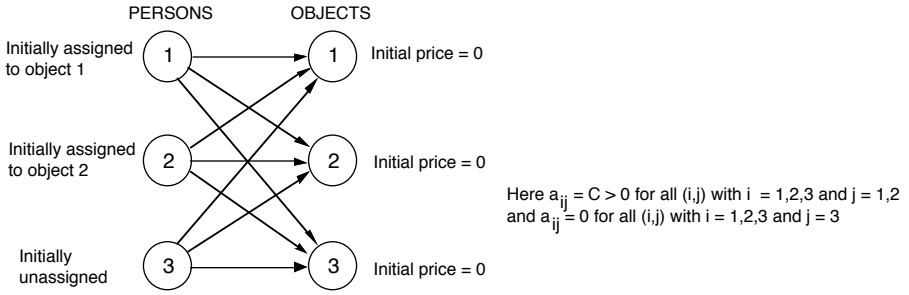
Suppose now that the costs a_{ij} are all integer, which is the typical practical case (if a_{ij} are rational numbers, they can be scaled up to integer by multiplication with a suitable common number). Then, the total benefit of any assignment is integer, so if $n\epsilon < 1$, any complete assignment that is within $n\epsilon$ of being optimal must be optimal. It follows, that if $\epsilon < 1/n$, and the benefits a_{ij} are all integer, then the assignment obtained upon termination of the auction algorithm is optimal. We state this result as a proposition; the proof is given in Appendix 2.

Proposition 2: Consider a feasible assignment problem with integer benefits a_{ij}. If

$$\epsilon < \frac{1}{n},$$

the auction algorithm terminates in a finite number of iterations with an optimal assignment.

Figure 3 shows the sequence of generated object prices for the example of Figs. 1 and 2 in relation to the contours of the dual cost function of the assignment problem, which is given in Appendix 1. It can



At Start of Iteration #	Object Prices	Assigned Pairs	Bidder	Preferred Object	Bidding Increment
1	0,0,0	(1,1), (2,2)	3	2	ϵ
2	0,ϵ,0	(1,1), (3,2)	2	1	2ϵ
3	2ϵ,ϵ,0	(2,3), (3,1)	1	2	2ϵ
4	2ϵ,3ϵ,0	(1,2), (2,1)	3	1	2ϵ
5	4ϵ,3ϵ,0	(1,3), (3,2)	2	2	2ϵ
6	...	...	...	...	...

Figure 2: Illustration of how the auction algorithm overcomes the cycling problem for the example of Fig. 1, by making the bidding increment at least equal to ϵ. The table shows one possible sequence of bids and assignments generated by the auction algorithm, starting with all prices equal to 0. At each iteration except the last, the person assigned to object 3 bids for either object 1 or 2, increasing its price by ϵ in the first iteration and by 2ϵ in each subsequent iteration. In the last iteration, after the prices of 1 and 2 rise at or above C, object 3 receives a bid and the auction terminates.

be seen from this figure that with each bid, the dual cost is approximately minimized (within ϵ) with respect to the price of the object receiving the bid. This observation can be established in generality; see [Ber88] or [Ber91a]. Successive minimization of a cost function along single coordinates is a central feature of coordinate descent and relaxation methods, which are popular for unconstrained minimization of smooth functions and for solving systems of smooth equations. Thus, the auction algorithm can be interpreted as an approximate coordinate descent method.

ϵ-Scaling

The amount of work needed for the auction algorithm to terminate can depend strongly on the value of ϵ and on the maximum absolute object benefit C given by

$$C = \max_{(i,j)\in\mathcal{A}} |a_{ij}|. \tag{10}$$

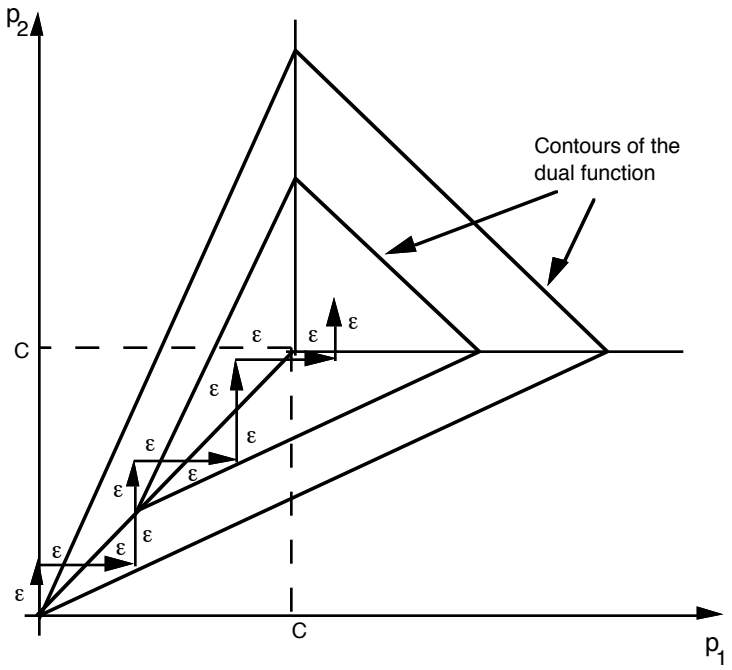


Figure 3: A sequence of prices p_1 and p_2 generated by the auction algorithm for the example of Figs. 1 and 2. The figure shows the equal dual cost surfaces in the space of p_1 and p_2, with p_3 fixed at 0.

Basically, for many types of problems, the number of iterations up to termination tends to be proportional to C/ϵ as argued earlier for fully dense problems. This can also be seen from the example of Fig. 3, where the number of iterations up to termination is roughly C/ϵ, starting from zero initial prices. For small ϵ, the method is susceptible to "price wars", that is, protracted sequences of small price rises resulting from groups of persons competing for a smaller number of roughly equally desirable objects.

Note also that there is a dependence on the initial prices; if these prices are "near optimal," we expect that the number of iterations to solve the problem will be relatively small. This can be seen from the example of Fig. 3; if the initial prices satisfy $p_1 \approx p_3 + C$ and $p_2 \approx p_3 + C$, the number of iterations up to termination is quite small.

The preceding observations suggest the idea of 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 some critical value (for example, $1/n$, when the benefits a_{ij} are integer). Typical ϵ-reduction factors after each scaling phase are of the order of 4 to 10. Each application of the algorithm provides good initial prices for the next application. ϵ-scaling was suggested in the original proposal of the auction algorithm [Ber79], based on extensive experimentation, which established its effectiveness for many types of assignment problems. In particular, ϵ-scaling is typically beneficial for sparse problems. The cost structure of the problem is also important in determining whether ϵ-scaling is needed.

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]. Based on experiments,

the running time of the algorithm for randomly generated problems seems to grow proportionally to something like $A \log n$ or $A \log n \log(nC)$. This is also supported by an approximate analysis in [Sch90]. The practical computational aspects of the auction algorithm will be discussed in Section 10.

Dealing with Infeasibility

Since termination can only occur with a feasible assignment, when the problem is infeasible, the auction algorithm will keep on iterating, as the user is wondering whether the problem is infeasible or just hard to solve. Thus for problems where existence of a feasible assignment is not known a priori, one must supplement the auction algorithm with a mechanism to detect infeasibility. There are several such mechanisms, which we will now discuss.

A basic result about (symmetric and asymmetric) assignment problems is that given an infeasible assignment S, there are two mutually exclusive possibilities:

(a) S has maximal cardinality, that is, there is no assignment having more assigned persons than S.

(b) There exists some unassigned person i and some unassigned object j and an *augmenting path* with respect to S that starts at i and ends at j, that is, a sequence of the form

$$(i, j_1, i_1, \ldots, j_m, i_m, j)$$

such that $j_1 \in A(i)$, $j \in A(i_m)$, and j_k is assigned to i_k under S for $k = 1, \ldots, m$.

This result can be proved in a number of ways. For example, by introducing in the assignment problem graph a supersource node s connected to all the person nodes and a supersink node t connected to all the object nodes, we can view the problem of finding an assignment of maximal cardinality as the problem of finding a maximum flow from s to t. The result then follows by using the max-flow/min-cut theorem and the related analysis (eg., [Ber91a], Chapter 1, Props. 2.3 and 2.4).

A corollary of the preceding result is that given a feasible assignment problem and an infeasible assignment S, for every person i that is unassigned under S, there exists an augmenting path with respect to S that starts at i. (To prove this, modify S by assigning every person that is unassigned under S, except i, to a fictitious object, and then apply the preceding result.)

One criterion that can be used to detect infeasibility is based on the maximum values

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

It turns out that in the course of the auction algorithm, all of these values will be bounded from below by a precomputable bound when the problem is feasible, but some of these values will be eventually reduced below this bound if the problem is infeasible. In particular, suppose that the auction algorithm is applied with initial object prices $\{p_j^0\}$. Then it can be shown that for any i, if person i is unassigned with respect to the current assignment S, and there is an augmenting path with respect to S that starts at i, we have

$$v_i \geq -(2n-1)C - (n-1)\epsilon - \max_j \{p_j^0\}, \tag{11}$$

where $C = \max_{(i,j)\in\mathcal{A}} |a_{ij}|$. The proof is obtained by adding the ϵ-CS condition along the augmenting path. If the problem is feasible, then as discussed earlier, there exists an augmenting path starting at each unassigned person at all times, so the lower bound (11) on v_i will hold for all unassigned persons i throughout the auction algorithm. On the other hand, if the problem is infeasible, some persons i will be submitting bids infinitely often, and the corresponding values v_i will be decreasing towards $-\infty$. Thus, we can apply the auction algorithm and keep track of the values v_i as they decrease. Once some v_i gets below its lower bound, we know that the problem is infeasible.

Unfortunately, it may take many iterations for some v_i to reach its lower bound. An alternative method to detect infeasibility is to convert the problem to a feasible problem by adding a set of artificial pairs $\overline{\mathcal{A}}$ to the original set $\mathcal{A}$. The benefits of these pairs should be very small, so that none of them participates in an optimal assignment unless the problem is infeasible. In particular, it can be shown that if the original problem was feasible, no pair $(i,j) \in \overline{\mathcal{A}}$ will participate in the optimal assignment, provided that

$$a_{ij} < -(2n-1)C, \qquad \forall\ (i,j) \in \overline{\mathcal{A}}, \tag{12}$$

where $C = \max_{(i,j)\in\mathcal{A}} |a_{ij}|$. To prove this by contradiction, assume that by adding to the set $\mathcal{A}$ the set of artificial pairs $\overline{\mathcal{A}}$ we create an optimal assignment S^* that contains a nonempty subset $\overline{S}$ of artificial pairs. Then, for every assignment S consisting exclusively of pairs from the original set $\mathcal{A}$ we must have

$$\sum_{(i,j)\in\overline{S}} a_{ij} + \sum_{(i,j)\in S^*-\overline{S}} a_{ij} \geq \sum_{(i,j)\in S} a_{ij},$$

from which

$$\sum_{(i,j)\in\overline{S}} a_{ij} \geq \sum_{(i,j)\in S} a_{ij} - \sum_{(i,j)\in S^*-\overline{S}} a_{ij} \geq -(2n-1)C.$$

This contradicts Eq. (12). Note that if $a_{ij} \geq 0$ for all $(i,j) \in \mathcal{A}$, the preceding argument can be modified to show that it is sufficient to have $a_{ij} < -(n-1)C$ for all artificial pairs (i,j).

On the other hand, the addition of artificial pairs with benefit $-(2n-1)C$ as per Eq. (12) expands the cost range of the problem by a factor of $(2n-1)$. In the context of ϵ-scaling, this necessitates a much larger starting value for ϵ and correspondingly large number of ϵ-scaling phases. If the problem is feasible these extra scaling phases are wasted. Thus for problems which are normally expected to be feasible, it may be better to introduce artificial pairs with benefits that are of the order of $-C$, and then gradually scale downward these benefits towards the $-(2n-1)C$ threshold if artificial pairs persist in the assignments obtained by the auction algorithm. This procedure of scaling downward the benefits of the artificial pairs can be embedded in a number of ways within the ϵ-scaling procedure.

Still another method to detect infeasibility is based on the following property: even when the problem is infeasible, the auction algorithm will find an assignment of maximal cardinality in a finite number of iterations, provided that unassigned persons are selected for bidding in an order that is either cyclic, or else ensures that each person will get a chance to submit a bid at least once within some fixed number of iterations. The proof of this is based on the lower bound (11) and the property of assignments of maximal cardinality stated earlier. In particular, if the current assignment never reached maximal cardinality, there would always exist an unassigned person i and a path that starts at i and is augmenting with respect to

the current assignment. By adding the ϵ-CS condition along this path, we see that v_i will always satisfy the lower bound (11), which is a contradiction because v_i will tend to $-\infty$ for all i that submit a bid infinitely often.

Suppose now that we periodically interrupt the auction algorithm and check whether there exists an augmenting path from some unassigned person to some unassigned object (a simple search of the breadth-first type can be used for this; see, e.g., [Ber91a], p. 26). Then once the cardinality of the current assignment becomes maximal but is less than n, this check will establish that the problem is infeasible. Thus this modified auction algorithm is guaranteed to either find a feasible assignment and a set of prices satisfying ϵ-CS, or to establish that the problem is infeasible and simultaneously obtain an assignment of maximal cardinality. In the latter case, it can be shown that the original (symmetric) problem will separate (through a saturated cut) into two components corresponding to (asymmetric) assignment problems. One may then use auction algorithms for asymmetric problems (see Section 9) to optimize the assignment within each component and obtain an optimal assignment within the class of all assignments with maximal cardinality.

Profits and Reverse Auction

Since the assignment problem is symmetric, it is possible to exchange the roles of persons and objects. This leads to the idea of *reverse auction* where the objects compete for persons by essentially offering discounts (lowering their prices). Roughly, given a price vector p, we can view the net value of the best object for person i

$$\max_{(i,j)\in A(i)} \{a_{ij} - p_j\} \tag{13}$$

as a *profit* for person i. When objects lower their prices they tend to increase the profits of the persons. Thus, profits play for persons a role analogous to the role prices play for objects. We can thus describe reverse auction in two different ways; one where unassigned objects lower their prices as much as possible to attract a person without violating ϵ-CS, and another where unassigned objects select a best person and raise his/her profit as much as possible without violating ϵ-CS. The second description is analytically more convenient, since with this description, forward and reverse auctions will be seen to be mathematically equivalent.

Let us introduce a profit variable π_i for each person i, and consider the following ϵ-CS condition for an assignment S and a profit vector π:

$$a_{ij} - \pi_i \geq \max_{k\in B(j)} \{a_{kj} - \pi_k\} - \epsilon, \qquad \forall\ (i,j) \in S, \tag{14}$$

where $B(j) = \{i \mid (i,j) \in \mathcal{A}\}$ is the set of persons that can be assigned to object j (assumed nonempty). The relation between the profit variable π_i and the expression $\max_{(i,j)\in A(i)}\{a_{ij} - p_j\}$ [cf. Eq. (13)] will become apparent later when we discuss a somewhat different ϵ-CS condition, which involves both prices and profits; see the following Eqs. (17a), (17b), and (20). Note the symmetry of the ϵ-CS condition (14) for profits with the corresponding one for prices; cf. Eq. (8).

The reverse auction algorithm maintains at the beginning of each iteration an assignment S and a profit vector π satisfying the ϵ-CS condition (14). It terminates when the assignment is feasible.

Typical Iteration of Reverse Auction

Let J be a nonempty subset of objects j that are unassigned.

Bidding Phase: For each object $j \in J$, find a "best" person i_j such that

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

and the corresponding value

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

and find

$$\omega_j = \max_{i \in B(j), i \neq i_j} \{a_{ij} - \pi_i\}. \tag{15}$$

[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.]

Assignment Phase: Each person i that is selected as best person by a nonempty subset $P(i)$ of objects in J, determines the highest bidder

$$j_i = \arg\max_{j \in P(i)} \{\beta_j - \omega_j + \epsilon\}, \tag{16}$$

increases π_i by the highest bidding increment $\max_{j \in P(i)} \{\beta_j - \omega_j + \epsilon\}$ and gets assigned to the highest bidder j_i; the object that was assigned to i at the beginning of the iteration (if any) becomes unassigned.

Note that reverse auction is identical to (forward) auction with the roles of persons and objects as well as profits and prices interchanged. Thus, by using the corresponding (forward) auction results (cf. Props. 1 and 2), we have:

Proposition 3: If at least one feasible assignment exists, the reverse auction algorithm terminates in a finite number of iterations. The feasible assignment obtained upon termination 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 (8) and a profit vector π satisfying the ϵ-CS condition (14). 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.

We say that an assignment S and a pair (π, p) satisfy ϵ-CS if

$$\pi_i + p_j \geq a_{ij} - \epsilon, \qquad \forall\ (i,j) \in \mathcal{A}, \tag{17a}$$

$$\pi_i + p_j = a_{ij}, \qquad \forall\ (i,j) \in S. \tag{17b}$$

We have the following proposition.

Proposition 4: 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

$$a_{ij} - \pi_i \geq \max_{k \in B(j)} \{a_{kj} - \pi_k\} - \epsilon, \qquad \forall\ (i,j) \in S. \tag{18}$$

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

$$a_{ij} - p_j \geq \max_{k \in A(i)} \{a_{ik} - p_k\} - \epsilon, \qquad \forall\ (i,j) \in S. \tag{19}$$

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

Proof: (a) In view of Eq. (17b), for all $(i,j) \in S$, we have $p_j = a_{ij} - \pi_i$, so Eq. (17a) implies that $a_{ij} - \pi_i \geq a_{kj} - \pi_k - \epsilon$ for all $k \in B(j)$. This shows Eq. (18).
(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 (19) is satisfied, by Prop. 1, S is within $n\epsilon$ of being optimal. **Q.E.D.**

We now introduce a combined forward/reverse algorithm. The algorithm maintains an assignment S and a profit-price pair (π, p) satisfying the ϵ-CS conditions (17). It terminates when the assignment is feasible. A common way to initialize the algorithm so that the ϵ-CS conditions are satisfied is to take S to be empty, to choose p arbitrarily, and to select π as a function of p via the relation $\pi_i = \max_{k \in A(i)}\{a_{ik} - p_k\}$ for each person i.

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

$$\pi_i = a_{ij_i} - p_{j_i}, \tag{20}$$

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

$$p_j = a_{i_j j} - \pi_{i_j}, \tag{21}$$

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 (20) or (21) is required per iteration for each object or person that received a bid during the iteration. An important property is that the updates of Eqs. (20) and (21) maintain the ϵ-CS conditions (17) for the pair (π, p), and therefore, by Prop. 4, maintain the required ϵ-CS conditions (18) and (19) for π and p, respectively. This is stated in the following proposition, which was proved in [BCT91]; see also [Ber91a].

Proposition 5: If the assignment and profit-price pair available at the start of an iteration of either the forward or the reverse auction algorithm satisfy the ϵ-CS conditions (17), the same is true for the assignment and profit-price pair obtained at the end of the iteration, provided Eq. (20) is used to update π (in the case of forward auction), and Eq. (21) is used to update p (in the case of forward auction).

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 used for forward auction (see Appendix 2) 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 finitely 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 at least one more person-object pair has been added to the assignment. In this way there can be a switch at most $(n-1)$ times between the forward and reverse steps of the algorithm. For a feasible problem, forward and reverse auction by themselves have guaranteed finite termination, so the final step will terminate with a feasible assignment satisfying ϵ-CS.

The combined forward/reverse auction algorithm typically works substantially (and often dramatically) faster than the forward version, as shown experimentally in the original paper [BCT91], and in the extensive computational study by D. Castañon [Cas92]. It seems to be affected less by the "price war" phenomenon that motivated ϵ-scaling. Price wars can still occur in the combined algorithm, but 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. One consequence of this is that starting with $\epsilon = 1/(n+1)$ and bypassing ϵ-scaling is often the best choice. Another consequence is that a larger ϵ-reduction factor can typically be used with no price war effects in ϵ-scaled forward/reverse auction than in ϵ-scaled forward auction. As a result, fewer ϵ-scaling phases are typically needed in forward/reverse auction to deal effectively with price wars.

4. AUCTION ALGORITHMS FOR SHORTEST PATH PROBLEMS

We now turn our attention to other types of network flow problems. Our approach for constructing auction algorithms for such problems is to convert them to assignment problems, and then to suitably apply the auction algorithm and streamline the computations. We start with the classical shortest path problem.

Suppose that we are given a graph with node set $\mathcal{N}$, arc set $\mathcal{A}$, and a length a_{ij} for each arc (i,j). In this section, by a *path* we mean a sequence of nodes $(i_1, i_2, \ldots, i_k)$ such that (i_m, i_{m+1}) is an arc for all $m = 1, \ldots, k-1$. If in addition the nodes $i_1, i_2, \ldots, i_k$ are distinct, the sequence $(i_1, i_2, \ldots, i_k)$ is called a *simple path*. The length of a path is defined to be the sum of its arc lengths. Assuming that all cycles have positive length, we want to find a path of minimum length over all paths that start at a given origin (node 1) and end at a given destination (node t).

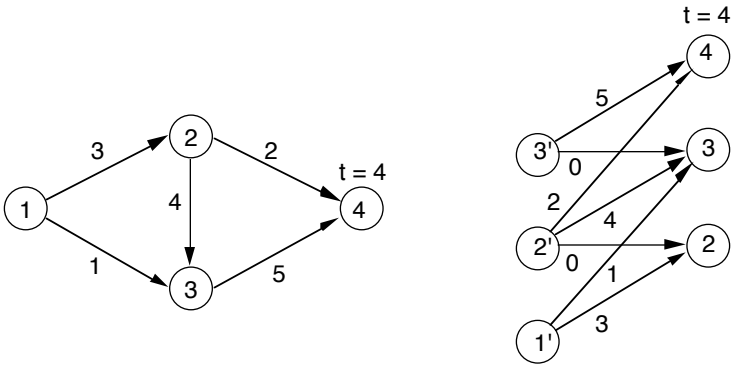


Figure 4: A shortest path problem (the origin is 1, the destination is $t = 4$) and its corresponding assignment problem. The arc lengths and the assignment costs are shown next to the arcs. A shortest path can be associated with an optimal augmenting path that starts at $1'$ and ends at $t = 4$. Generally, for each node $i \neq 1$ we introduce an "object" node i, and for each node $i \neq t$ we introduce a "person" node i'. For every arc (i, j) of the shortest path problem with $i \neq t$ and $j \neq 1$, we introduce the arc (i', j) with cost a_{ij} in the assignment problem. We also introduce the zero cost arc (i', i) for each $i \neq 1, t$. Given the assignment that assigns object i to person i' for $i \neq 1, t$, but leaves person 1' and object t unassigned, paths from 1 to t can be associated with augmenting paths that start at $1'$ and end at t. A shortest path from 1 to t corresponds to a shortest augmenting path from the unassigned person $1'$ to the unassigned object t. It can be shown that an augmentation along such a shortest augmenting path gives an optimal assignment.

There is a well-known transformation that converts this problem to a particular type of assignment problem as shown in Fig. 4. We can apply the auction algorithm to solve the equivalent problem, but it turns out that the structure of this problem is such that the *naive* auction algorithm ($\epsilon = 0$) works. Assuming that all arc lengths are nonnegative, we can start the naive auction algorithm with the zero price vector and with the assignment that assigns, for $i \neq 1, t$, each person i' to object i; this assignment-price pair satisfies CS [since all arc lengths are nonnegative and the assigned arcs (i', i) have zero cost], but is not feasible because it leaves person $1'$ and object t unassigned. It can be seen by using induction or by tracing the steps of the naive auction algorithm that each assignment generated consists of a (possibly empty) 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 $P = (1, i_1, \ldots, i_k)$. 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. If $i_k = t$, a feasible assignment results, in which case the naive auction algorithm terminates. Otherwise the unassigned person i'_k submits a bid and there are two possibilities:

(a) The best object is i_k, in which case i'_{k-1} becomes unassigned and the path $(1, i_1, \ldots, i_{k-1})$ corresponding to the new assignment is obtained from the previous path $(i, i_1, i_2, \ldots, i_k)$ via "contraction" by one node.

(b) The best object is $i_{k+1} \neq i_k$, in which case the path $(1, i_1, \ldots, i_k, i_{k+1})$ corresponding to the new assignment obtained from the previous path $(1, i_1, \ldots, i_k)$ via "extension" by one node.

We will now describe the naive auction algorithm directly in terms of the original shortest path problem, properly translating (a) the preceding operations of one-node contraction or extension of a path, and (b) the use of prices and the associated price changes of the terminal node of the path, while maintaining a CS condition.

The Auction/Shortest Path Algorithm

The auction algorithm for shortest paths maintains at all times a simple path $P = (1, i_1, i_2, \ldots, i_k)$. 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 together with P the following property

$$p_i \leq a_{ij} + p_j, \qquad \forall\ (i,j) \in \mathcal{A}, \tag{22a}$$

$$p_i = a_{ij} + p_j, \qquad \text{for all pairs of successive nodes } i \text{ and } j \text{ of } P, \tag{22b}$$

which is referred to as *complementary slackness* (CS for short). This condition can be related to the CS condition for the equivalent assignment problem as well as to CS conditions for a formulation of the shortest path problem as a minimum cost flow problem (see [Ber91a], Section 1.3).

It can be shown that if a pair (P, p) satisfies the CS conditions, then the portion of P between node 1 and any node $i \in P$ is a shortest path from 1 to i, while $p_1 - p_i$ is the corresponding shortest distance. To see this, note that by Eq. (22b), $p_i - p_k$ 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. (22a) along the arcs of the path]. 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

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

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_s being strictly increased. The iteration is as follows.

Typical Iteration of the Auction/Shortest Path Algorithm

Let i be the terminal node of P. If

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

go to Step 1; else go to Step 2.

Step 1 (Contract path): Set

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

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

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

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

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.

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 for the sake of the interpretation.) 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. 5(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. 5(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 well-known min path/max tension theorem; see e.g. [Roc84], [Ber91a].)

The auction/shortest path algorithm can also be interpreted in terms of the balls-and-strings model; it can be viewed as a process whereby nodes are raised in stages as illustrated in Fig. 6. Initially all nodes are resting 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.

When there is a single origin and multiple destinations, the algorithm can be applied with virtually no change. We simply stop the algorithm when all destinations have become the terminal node of the path P at least once. We also note that the algorithm can be similarly applied to a problem with multiple origins and a single destination, by first reversing the roles of origins and destinations, and the direction of each arc.

The Reverse Algorithm

There are a number of ways to speed up the basic algorithm, which are described in detail in [Ber91a], [Ber91b], and [BPS92]; see also Section 10. The most significant of these relates to a two-sided version of the algorithm that maintains, in addition to the path P, another path R that *ends at the destination*. To understand this version, we first note that in shortest path problems, one can exchange the role of origins and destinations by reversing the direction 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

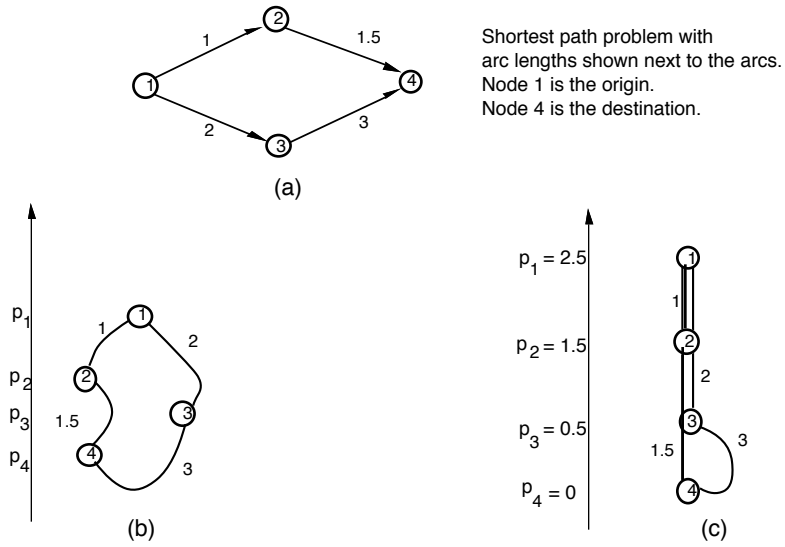


Figure 5: Illustration of the CS conditions for the shortest path problem. If each node is a ball, and for every arc $(i,j) \in \mathcal{A}$, nodes i and j are connected with a string of length a_{ij}, the vertical coordinates p_i of the nodes satisfy $p_i - p_j \leq a_{ij}$, as shown in (b) for the 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).

each iteration by means of a contraction or an extension. This algorithm, called the *reverse algorithm*, is mathematically equivalent to the earlier forward algorithm, and parallels the reverse auction algorithm for the assignment problem discussed in the previous section.

Initially, in the reverse algorithm, R is any path ending at the destination, and p is any price vector satisfying the CS conditions (22) 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

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

go to Step 1; else go to Step 2.

Step 1: (Contract path) Set

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

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

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

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

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

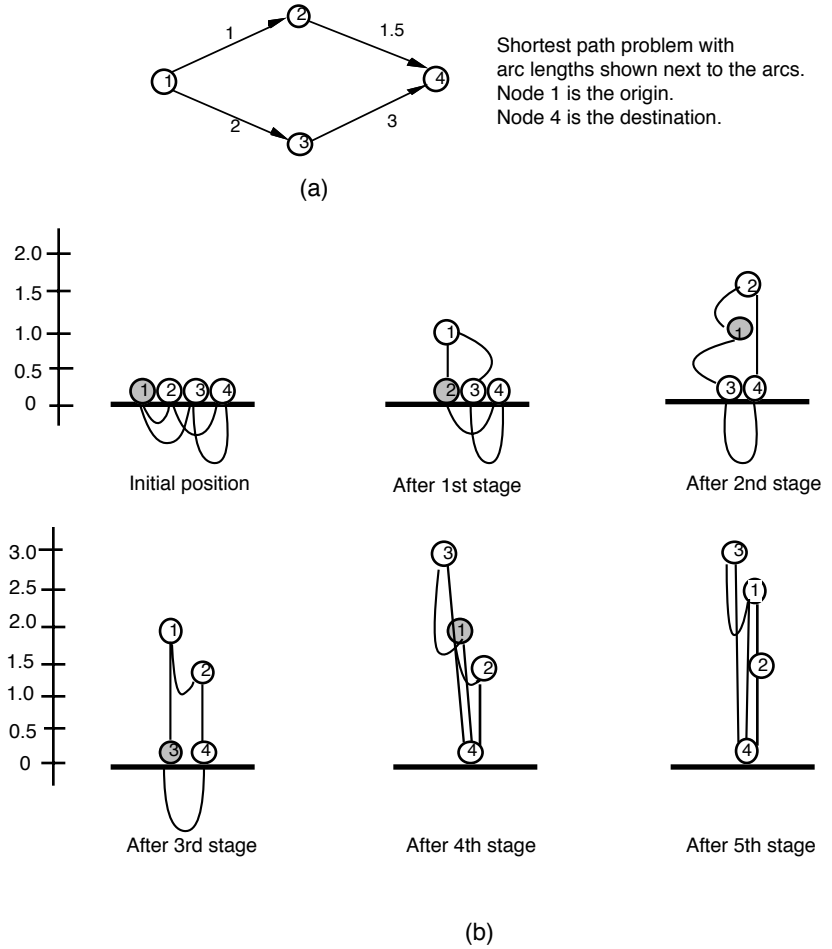


Figure 6: Illustration of the auction/shortest path algorithm in terms of the balls-and-strings model for the problem shown in (a). The model initially rests on a flat surface, and various balls are then raised in stages. At each stage we raise a single ball $i \neq t$ (marked by gray), which is at a lower level than the origin 1 and can be reached from 1 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 1. (If 1 does not have any tight string connecting it to another ball, which is at a lower level, we use $i = 1$.) 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.

The reverse algorithm is most helpful when it is combined with the forward algorithm. In a combined algorithm, initially we 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. Both P and R satisfy CS together with p throughout the algorithm, so when P and R meet, say at node i, the composite path consisting of the portion of P from 1 to i followed by the portion of R from i to t will be shortest.

Combined Forward/Reverse Auction/Shortest Path 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.

The combined forward/reverse algorithm can also be interpreted in terms of the balls-and-strings model of Fig. 5. Again, all nodes are resting initially on a flat surface. When the forward part of the algorithm is used, we raise nodes in stages as illustrated in Fig. 6. When the reverse part of the algorithm is used, we *lower* nodes in stages; at each stage, we lower the *top* node in a tight chain that ends at the destination to the level at which at least one more string becomes tight.

Note that the case of multiple destinations can be handled by using a separate reverse path for each destination. One then alternates between a forward step and a reverse step as in the preceding algorithm, while taking up cyclically different destinations in different reverse steps.

Based on experiments with randomly generated problems on a serial machine [Ber91b], the combined forward/reverse auction/shortest path algorithm outperforms substantially its closest competitors for problems with few destinations and a single origin (the computation time is reduced often by an order of magnitude or more). For the case of many (or all) destinations, the algorithm apparently runs slower than state-of-the-art label setting and label correcting methods (typical slowdown factors are of the order of two or three). On the other hand, for the case of multiple destinations, the algorithm is better suited for parallel computation than other shortest path algorithms. Also there is a variation of the algorithm and has substantially improved performance for problems with many destinations. This variation is described next.

The Auction Algorithm with Graph Reduction

Despite its excellent practical performance for problems with few destinations, the auction algorithm has pseudopolynomial complexity; for an example see [Ber91a] and [Ber91b]. Weakly polynomial versions of the algorithm were developed in [Ber91b] using the idea of scaling the arc lengths, but these versions did not prove effective in practice. Strongly polynomial versions of the algorithm were obtained by Pallottino and Scutella' [PaS91] by adding to the extension and contraction operations a *reduction* operation. Here, each time a node becomes the terminal node of the path for the first time, all its incoming arcs except the one of the path are deleted, since they cannot be used to improve the distance to the node. The auction algorithm thus obtained can be shown to have an $O(m^2)$ running time, where m is the number of arcs. By using the idea of presorting the outgoing arcs of each node in order of increasing length, the running time is reduced further to $O(mn)$, where n is the number of nodes.

An additional advantage of graph reduction is that it allows the relaxation of the positivity assumption on all cycle lengths to nonnegativity. The reason for requiring positive cycle lengths was to ensure that no cycle could be formed through the process of path extension. On the other hand, with graph reduction, every node already visited by the path has a unique (undeleted) incoming arc except for node 1, which has no incoming arc at all. With a little thought, it can be seen that this precludes the extension of the path to a node that is already on the path.

In subsequent work by Bertsekas, Pallottino, and Scutella' [BPS92] the graph reduction idea was strengthened by using certain upper bounds to the node shortest distances to delete arcs more effectively. Two algorithms were developed. The first maintains the basic simplicity of the auction algorithm given earlier, and has $O\big(n \min\{m, n \log n\}\big)$ running time. The second algorithm is somewhat more complex but has an $O(n^2)$ running time. These theoretical improvements, in conjunction with efficient implementation techniques, have resulted in substantially faster practical performance for single origin/all destination problems. In particular, for fully dense randomly generated problems, the auction algorithm with graph reduction has outpeformed its closest competitors [BPS92].

k Shortest Path Problems

Consider the generalization of the single origin/single destination shortest path problem, where instead of a single path, we seek the k best paths. In other words, we want to find the shortest path, say P_1, between an origin s and a destination t, then find the shortest path P_2 between s and t subject to the condition that it be different by at least one arc from P_1, then find the shortest path P_3 subject to the condition that it be different than P_1 and P_2, and so on up to the kth shortest path P_k. There are two versions of this problem depending on whether the paths P_i are allowed to contain cycles or not [Law76].

Important algorithmic ideas for solving k shortest path problems are based on solving a sequence of shortest path problems involving graphs that differ slightly from each other by a few arcs and/or nodes [Dre69], [Law76], [Mar84], [ACM91], [AMM91]. The auction/shortest path algorithm is particularly well suited for solving these shortest path problems for two reasons: First, it can conveniently transfer favorable initial price information from one shortest path problem to the next, so that the solution of each subproblem after the first is greatly expedited. Second, it requires the solution of single origin/single destination problems, which it can solve (in forward/reverse mode) much faster than the single origin/all destination problem that some of the existing methods must solve [ACM91], [AMM91]. At present, there are no computational studies exploring the use of auction algorithms for k shortest path problems; this is an interesting subject for further investigation.

k Node-Disjoint Shortest Path Problems

Let us consider another generalization of the single origin/single destination shortest path problem, where instead of a single path, we seek k node-disjoint paths with minimum sum of lengths. Once a supersourse node s and a supersink node t are added, the $n \times n$ assignment problem can be viewed as a special case of this problem, where $k = n$. Another special case is the separable version of the three-dimensional assignment problem, involving the optimal choice of k disjoint ordered triplets, where the cost of a triplet (i, j, m) is of the separable form $a_{ij} + a_{jm}$.

An auction algorithm for the k node-disjoint shortest path problem that bears similarity to the preceding shortest path algorithm has been developed in [BeC91]. In particular, the algorithm maintains a price vector p together with k node-disjoint paths starting at the origin, that satisfy an ϵ-CS condition. If all the paths end at the destination, the algorithm stops. Otherwise one of the paths, say P, that does not end at the destination is either extended or contracted at its terminal node. If as a result of an

extension, P joins another path P' at a node j, P' is truncated to the portion up to the node preceding j; the portion following j is added to P. Favorable computational results for this auction algorithm are given in [BeC91]. Interestingly, when this algorithm is specialized to the $n \times n$ assignment problem, it becomes identical to the original auction algorithm of Section 4, thereby illustrating vividly the relation of the bidding mechanism of the assignment auction algorithm and the extension/contraction mechanism of the shortest path auction algorithm.

5. EXTENSION TO TRANSPORTATION PROBLEMS

We now consider the extension of the auction algorithm to the uncapacitated transportation problem. Here we have a bipartite graph with m sources and n sinks. The problem is the same as the assignment problem except that the node supplies need not be 1 or -1. It has the form

$$\begin{aligned} &\text{maximize} \quad \sum_{(i,j)\in\mathcal{A}} a_{ij}x_{ij} \\ &\text{subject to} \\ &\sum_{j\in A(i)} x_{ij} = \alpha_i, \qquad \forall\ i = 1,\ldots,m, \\ &\sum_{i\in B(j)} x_{ij} = \beta_j, \qquad \forall\ j = 1,\ldots,n, \\ &0 \le x_{ij}, \qquad \forall\ (i,j) \in \mathcal{A}. \end{aligned} \tag{23}$$

Here α_i and β_j are positive integers, which for feasibility must satisfy

$$\sum_{i=1}^{m} \alpha_i = \sum_{j=1}^{n} \beta_j.$$

We can easily convert this problem to an assignment problem by replacing each source (or sink) node into a collection of "duplicate" persons (or objects, respectively) nodes. In particular, a source node i with supply α_i is replaced by α_i persons, and a sink node j with demand β_j is replaced by β_j objects. Furthermore, for each arc (i,j) we must create an arc of benefit a_{ij} connecting each person corresponding to i with each object corresponding to j. An example is given in Fig. 7.

It is possible to solve the equivalent assignment problem by straightforward application of the auction algorithm but there are two difficulties:

(a) The dimension of the problem is greatly increased (both the number of persons and the number of objects become $\sum_{i=1}^{m} \alpha_i$).

(b) The structure of the equivalent assignment problem is such that protracted price wars are inevitable due to the duplicate objects and persons created by the transformation. Figure 8 provides an example.

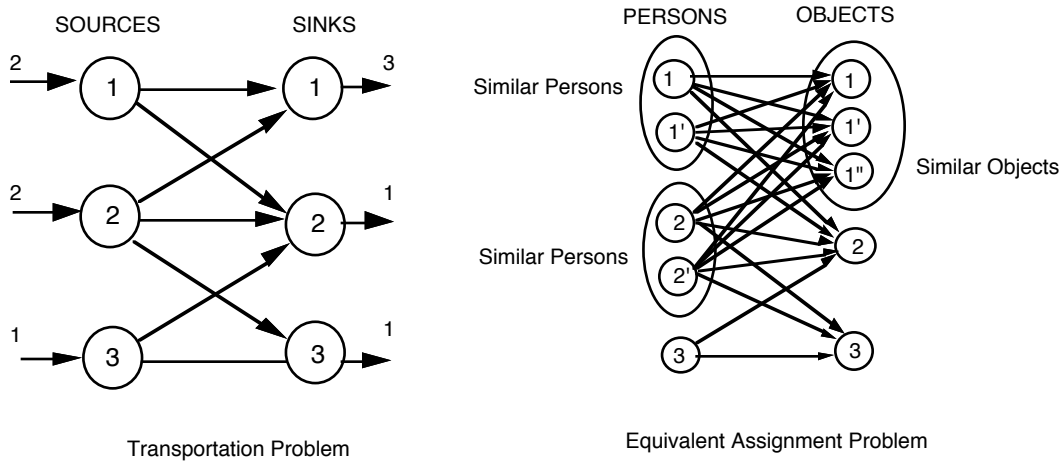


Figure 7: Illustration of the conversion of a transportation problem into an assignment problem. Each source i is transformed into α_i persons (for example source 1 is transformed into persons 1 and $1'$), and each sink j is transformed into β_j objects (for example sink 1 is transformed into objects 1, $1'$, and $1''$). If (i,j) is an arc of benefit a_{ij} in the transportation problem, there is an assignment arc of benefit a_{ij} connecting each of the α_i persons corresponding to i with each of the β_j objects corresponding to j.

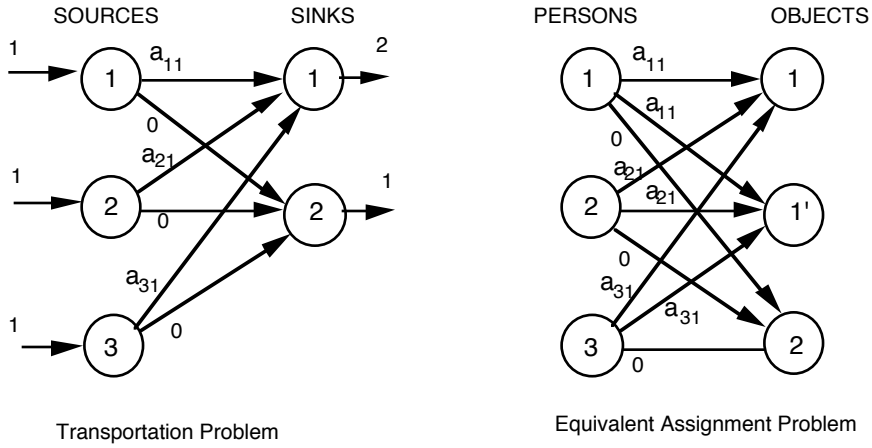


Figure 8: Typical example where a transportation problem is converted into an equivalent assignment problem and the duplicate objects induce price wars in the auction algorithm. Here a_{11}, a_{21}, and a_{31} are much larger than zero. The persons 1, 2, and 3 will keep on bidding up the prices of objects 1 and $1'$ by 2ϵ increments until one of the prices p_1 or $p_{1'}$ reaches or exceeds the minimum of a_{11}, a_{21}, and a_{31}. (For $a_{11} = a_{21} = a_{31} = C$, this example is the same as the one of Figs. 1-3.)

It is possible to address both of these difficulties by modifying the auction algorithm so that it takes advantage of the special structure of assignment problems created by duplicate objects and persons [BeC89a]. In particular, given the assignment problem, we say that two objects j and j' are *similar*, and write $j \sim j'$, if they can be matched with the same persons and at equal values, that is,

$$B(j) = B(j'), \qquad \text{and} \qquad a_{ij} = a_{ij'} \quad \forall\, i \in B(j).$$

We say that two persons i and i' are *similar*, and write $i \sim i'$, if

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

For each object j, the set of all objects similar to j is called the *similarity class* of j and is denoted $M(j)$. For each person i, the set of all persons similar to i is called the similarity class of i and is denoted $M(i)$.

To get a sense of how one can deal with similarity classes, consider the auction algorithm of Section 3 applied to the assignment problem of Fig. 8. Here, person i ($i = 1, 2, 3$) will continue bidding for the similar objects 1 and $1'$ until both prices p_1 and $p_{1'}$ become at least equal to a_{i1}, and object 2 becomes no less attractive for person i than objects 1 and $1'$. Thus a_{i1} may be viewed as a *contention threshold* to be reached by the prices of *all* the similar objects before person i becomes interested in an object outside their similarity class. The contention thresholds of an object in some similarity class can of course be much higher than the price of the object (which by the rules of the auction algorithm, is constrained to be within ϵ of the price of any other object of the same class). Suppose now that when a person bids for an object of a similarity class he/she not only updates its price, but also updates the corresponding contention threshold. Suppose further that the contention thresholds of *all* the objects of a similarity class rise above the (essentially common) price of these objects. Then it can be seen that, without violating ϵ-CS, we can raise simultaneously the prices of all the objects of the class to the minimum contention threshold, thereby resolving the corresponding price war.

The Auction Algorithm with Similar Objects

The preceding idea can be implemented very simply in an algorithm that we call *auction algorithm with similar objects*. For each object j, we maintain a contention threshold $\hat{p}_j$. If j is unassigned, the contention threshold $\hat{p}_j$ is equal to the initial price p_j; each time j is assigned to a new person i, the contention threshold $\hat{p}_j$ is set to ϵ plus the minimum level to which p_j should rise before i finds an object from a *different similarity class* equally attractive to j. The prices are determined by the contention thresholds. In particular, the price of an object j is the minimum contention threshold over the objects of the similarity class $M(j)$ of j

$$p_j = \min_{k \in M(j)} \hat{p}_k. \tag{24}$$

Thus, *all objects in a similarity class have the same price but possibly different contention thresholds*.

More formally, the auction algorithm with similar objects is the same as the auction algorithm of Section 4 except for one difference: in the bidding phase, an unassigned person i finds his/her best object j_i and corresponding best value on the basis of the contention thresholds,

$$j_i = \arg\max_{j \in A(i)} \{a_{ij} - \hat{p}_j\}, \qquad v_i = \max_{j \in A(i)} \{a_{ij} - \hat{p}_j\}. \tag{25}$$

Then i sets the contention threshold $\hat{p}_{j_i}$ to p_{j_i} plus a bidding increment $v_i - w_i + \epsilon$, which is based on the second best object from a different similarity class, that is, w_i is defined now

$$w_i = \max_{j \in A(i),\, j \notin M(j_i)} \{a_{ij} - \hat{p}_j\} \tag{26}$$

(instead of $w_i = \max_{j \in A(i),\, j \neq j_i}\{a_{ij} - p_j\}$). When the contention thresholds of all objects of a similarity class have been raised, the price of the class is also raised to the minimum contention threshold according to Eq. (24). Note that because of the relation $p_j = \min_{k \in M(j)} \hat{p}_k$ between prices and contention thresholds, $\hat{p}_j$ can be replaced by p_j in Eqs. (25) and (26). Thus, an unassigned person bids for the "best" object of the "best" similarity class, but the bidding increment is now based on the contention thresholds and can be significantly higher than the bidding increment based on prices; see Fig. 9.

Example:

For the problem of Fig. 8, if initially the object prices and the contention thresholds are zero, in the first iteration the bidding person 1 will bid the contention threshold of object 1 to $\hat{p}_1 = a_{11} + \epsilon$ rather than ϵ; in the second iteration the bidding person 2 will bid the contention threshold of object $1'$ to $\hat{p}_{1'} = a_{21} + \epsilon$, so the price of the similarity class $M(1)$ will increase to $\min\{a_{11}, a_{21}\} + \epsilon$; in the third iteration, if $a_{31} \leq \min\{a_{11}, a_{21}\} + \epsilon$, the bidding person 3 will bid for object 2 and the auction will terminate; otherwise person 3 will bid for either object 1 or $1'$ depending on which one has the smallest contention threshold, and the person that is left unassigned will bid for object 2 at the next iteration, thereby terminating the auction. It can thus be seen that the auction algorithm with similar objects resolves very quickly the price wars due to the duplicate objects corresponding to the sinks of the transportation problem (23).

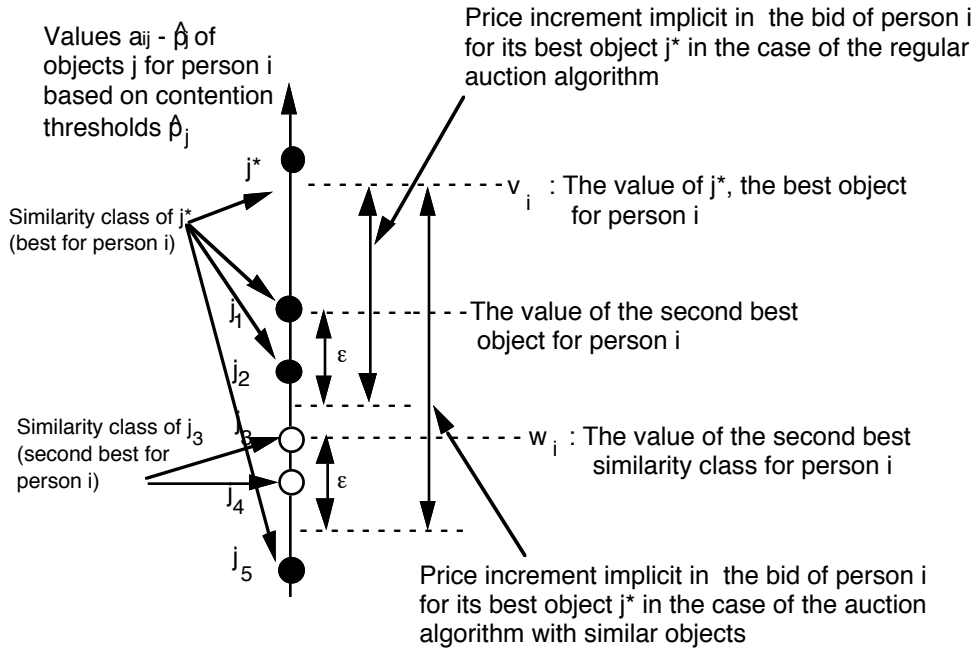


Figure 9: Illustration of the bid of person i in the auction algorithm with similar objects. The object j^* offers the best value $a_{ij} - \hat{p}_j$ for person i, based on the contention thresholds $\hat{p}_j$. However, here w_i is the value offered by the second best similarity class, rather than the value of the second best object as in the regular auction algorithm. When the second best object belongs to the similarity class of j^* (as in this figure), the bid of person i will be higher in the auction algorithm with similar objects that in the regular auction algorithm.

Regarding the validity of the algorithm, it can be shown that the object prices still satisfy ϵ-CS, and by essentially repeating the proof of Prop. 2, it follows that the algorithm will terminate for a feasible problem. However, it can be shown that for integer a_{ij}, the final assignment will be optimal if $\epsilon < 1/m$, where m *is the number of object similarity classes* rather than $\epsilon < 1/n$ as is the case for the auction algorithm of Section 3; see [BeC89a], or [Ber91a], pp. 230-232, or use Prop. 10, which is proved in Appendix 1. Thus the increased dimensionality due to the duplicate objects created by transforming the transportation problem (23) to an assignment problem is not reflected in a corresponding reduction of the threshold value of ϵ to obtain an optimal solution.

The Auction Algorithm with Similar Persons

One can further enhance the performance of the auction algorithm by taking into account the presence of similar persons. The idea is to submit a *common bid* for all persons in a similarity class if at least one person in the class is unassigned. As a result, persons in the same class do not "compete" against each other for the same objects, and the bids submitted are higher than they would otherwise be. In effect, the persons of the same similarity class cooperate to "compress" a price war and resolve it within one iteration. We illustrate this idea and the corresponding auction algorithm in Fig. 10.

The idea of a common bid for a person similarity class can be combined with the previously discussed idea of contention thresholds and the corresponding auction algorithm for similar objects. In particular, the cooperative bid of a person similarity class is based on the contention thresholds of the objects. With proper streamlining of the computations, one obtains an algorithm for the transportation problem (23), which is in effect the auction algorithm for the corresponding equivalent assignment problem (cf. Fig. 7) but with similar persons and objects properly taken into account. In this auction/transportation algorithm, each sink j has a price p_j, and each arc (i,j) that carries positive flow x_{ij} has a contention threshold y_{ij} associated with it. The price p_j is equal to the original price of the sink j if the demand of the sink has not yet been exhausted ($\sum_{i\in B(j)} x_{ij} < \beta_j$), and is equal to the minimum contention threshold of the incoming arcs that carry positive flow ($\min_{i\in B(j),\, x_{ij}>0} y_{ij}$) otherwise. The sources bid for sinks by increasing the contention thresholds of the corresponding arcs, and a sink price increases once all of its demand is exhausted and the associated contention thresholds have increased. As the sink prices increase, the corresponding profits of the sources defined by

$$\pi_i = \max_{j\in A(i)} \{a_{ij} - p_j\} \tag{27a}$$

decrease, while the ϵ-CS condition

$$\pi_i + p_j \leq a_{ij} + \epsilon, \qquad \forall\ (i,j) \in \mathcal{A} \text{ with } x_{ij} > 0 \tag{27b}$$

is maintained. The ϵ-CS conditions (27a) and (27b) extend the corresponding condition (8) for the assignment problem. It is shown in [BeC89a] that for an integer problem, a feasible flow vector is optimal provided that there exists a price vector p and corresponding profit vector π satisfying ϵ-CS with $\epsilon < 1/\min\{m,n\}$.

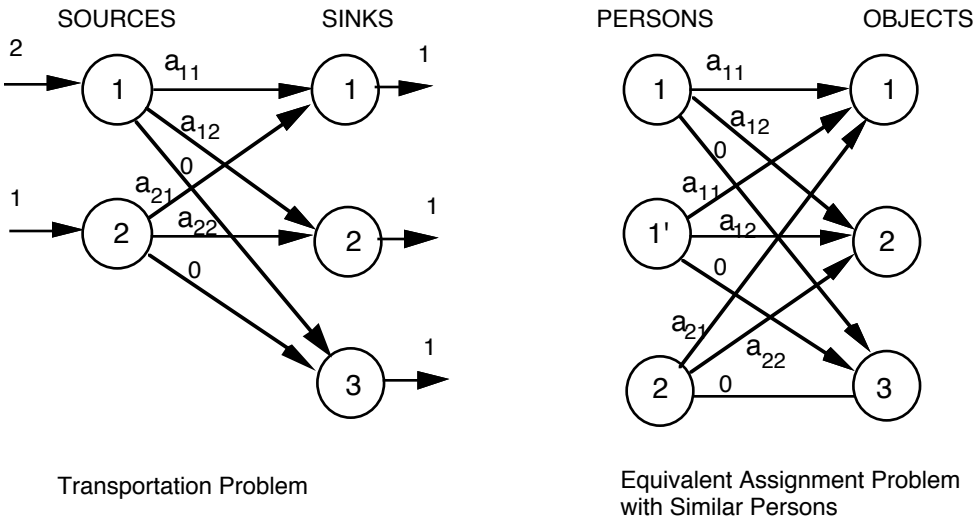


Figure 10: Illustration of the auction algorithm with similar persons. In this algorithm, all the persons in a similarity class with k persons submit bids for the k best objects for the class. The levels of the bids are such that once the bids are accepted, the profit of each person in the class is set at ϵ below the profit offered by the $(k+1)$st best object. In the example of the figure the similar persons result from the conversion of the transportation source 1 into the two similar persons 1 and $1'$. Consider first the case where the initial assignment is empty, all initial prices are 0, and $a_{11} = a_{12} = a_{21} = a_{12} = C > 0$. Then the regular auction algorithm will operate as in Fig. 2, involving a long price war if ϵ is small relative to C. In the first iteration of the auction algorithm with similar persons, persons 1 and $1'$ will submit a common bid of $C + \epsilon$ for the two best objects 1 and 2. Then person 2 will submit a bid for object 3 at the next iteration and the algorithm will terminate, thus avoiding the price war. If instead we have $a_{11} > a_{12} > 0$, at the initial iteration persons 1 and $1'$ will submit a bid of $a_{11} + \epsilon$ for object 1 and a bid of $a_{12} + \epsilon$ for object 2, bringing the net value of these objects to the value of the third object minus ϵ. If $a_{12} - \epsilon > a_{22} > a_{21} > 0$, person 2 will then submit a bid for object 3 at the next iteration and the algorithm will terminate. If $a_{22} > a_{12} - \epsilon$ and $a_{22} > a_{21} > 0$, person 3 will submit a bid for object 2 at the second iteration, raising its price to $a_{22} + \epsilon$; persons 1 and $1'$ will then submit another common bid for objects 1 and 3 at the third iteration, and the algorithm will terminate.

6. A GENERIC AUCTION ALGORITHM FOR MINIMUM COST FLOW PROBLEMS

We now discuss a general algorithm of the auction type for linear minimum cost flow problems, given by D. Castañon and the author in [BeC89b] and [BeC91]; it includes as special cases the auction algorithm for assignment and transportation problems, as well as the ϵ-relaxation method to be discussed in Section 7. By specializing this general algorithm to network flow problems with special structure, one may obtain efficient methods that exploit the structure; for example, the general algorithm is the basis for the k node-disjoint shortest path algorithm briefly described in Section 4.

We are given a directed graph with set of nodes $\mathcal{N}$ and set of arcs $\mathcal{A}$. The number of nodes is denoted by N and the number of arcs is denoted by A. For each arc (i,j) there an optimization variable x_{ij}, called the *flow of arc* (i,j). We denote by x the flow vector $\{x_{ij} \mid (i,j) \in \mathcal{A}\}$. The minimum cost flow problem that we consider is

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

$$\text{subject to}$$

$$\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} = s_i, \qquad \forall\ i \in \mathcal{N}, \tag{28}$$

$$b_{ij} \le x_{ij} \le c_{ij}, \qquad \forall\ (i,j) \in \mathcal{A}, \tag{29}$$

where a_{ij}, b_{ij}, c_{ij}, and s_i are given integers. The problem is said to be feasible if there exists a flow vector x satisfying the constraints (28) and (29); otherwise it is said to be infeasible.

The problem can be converted into an equivalent transportation problem as shown in Fig. 11. It is thus possible to use this equivalence to transcribe the auction/transportation ideas of the previous section into the minimum cost flow context. In particular, let us first derive the appropriate form of ϵ-CS.

In reference to the equivalent transportation problem of Fig. 11, let us denote by λ_{ij} the price of the sink corresponding to arc (i,j), and let us denote by p_i the profit of the source corresponding to node i. By Eq. (27a) (with a_{ij} replaced by $-a_{ij}$ to account for the change from maximization to minimization), we have

$$p_i + \lambda_{ij} \ge 0, \qquad p_j + \lambda_{ij} \ge -a_{ij}, \qquad \text{for all } (i,j) \in \mathcal{A}. \tag{30}$$

Let us assume that throughout the auction the total incoming flow to the sink corresponding to arc (i,j) along the two transportation arcs $\big(j,(i,j)\big)$ and $\big(i,(i,j)\big)$ is equal to the demand $c_{ij} - b_{ij}$; for any initial set of p_i, it is possible to enforce this condition while preserving ϵ-CS by selecting the initial λ_{ij} to be $\lambda_{ij} = \min\{-p_i, -p_j - a_{ij}\}$. Then, by Eq. (27b) we also have

$$p_i + \lambda_{ij} \le \epsilon \qquad \text{for all } (i,j) \in \mathcal{A} \text{ with } x_{ij} < c_{ij}, \tag{31a}$$

$$p_j + \lambda_{ij} \le -a_{ij} + \epsilon \qquad \text{for all } (i,j) \in \mathcal{A} \text{ with } b_{ij} < x_{ij}. \tag{31b}$$

From Eqs. (30) and (31a), we obtain $p_j + a_{ij} \ge -\lambda_{ij} \ge p_i - \epsilon$ if $x_{ij} < c_{ij}$, so that

$$p_i - p_j \le a_{ij} + \epsilon \qquad \text{for all } (i,j) \in \mathcal{A} \text{ with } x_{ij} < c_{ij}. \tag{32a}$$

Also from Eqs. (30) and (31b), we obtain $p_i \ge -\lambda_{ij} \ge a_{ij} + p_j - \epsilon$ if $b_{ij} < x_{ij}$, so that

$$p_i - p_j \ge a_{ij} - \epsilon \qquad \text{for all } (i,j) \in \mathcal{A} \text{ with } b_{ij} < x_{ij}. \tag{32b}$$

We say that a flow-price pair (x,p) satisfies *ϵ-complementary slackness* (ϵ-CS for short) if x satisfies the arc flow bounds (29), and Eqs. (32a) and (32b) hold; see Fig. 12. The following proposition parallels Prop. 1 for the assignment problem and is proved in more general form in Appendix 1 (cf. Prop. 10).

Proposition 6: If x is feasible, and for some price vector p, the pair (x,p) satisfies ϵ-CS with $\epsilon < 1/N$ (N is the number of nodes), then x is optimal for the minimum cost flow problem (MCF).

To develop auction algorithms for the minimum cost flow problem, we draw motivation from auction algorithms for the equivalent transportation problem, and the associated assignment problem with similar objects and persons. The notion of a bid by a single source and the associated increase of flow along the best outgoing arc can be transcribed in the context of the minimum cost problem as a flow increase by a single node along the "best" outgoing arcs or a flow reduction by a node along the "best" incoming

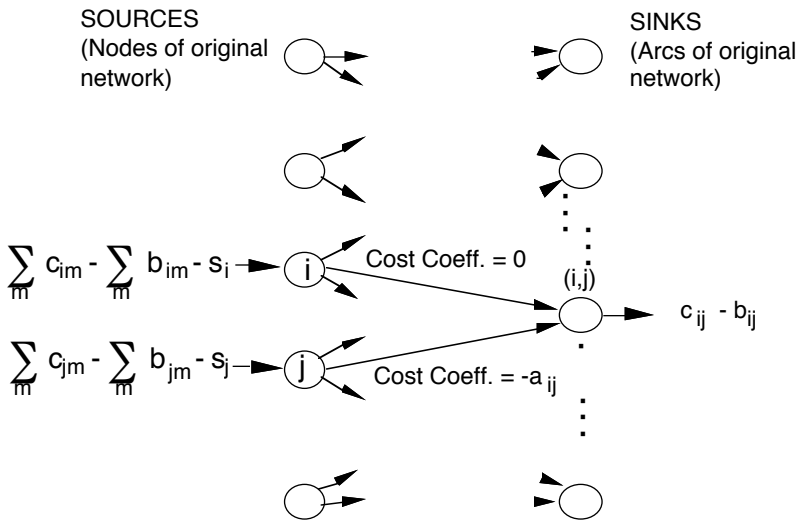


Figure 11: Transformation of a minimum cost flow problem into a transportation problem of the form (23). We take as sinks of the transportation problem the arcs of the original network, and as sources of the transportation problem the nodes of the original network. Each transportation problem sink has two incoming arcs with cost coefficients as shown. The demand of each transportation problem sink is the feasible flow range length of the corresponding original network arc. The demand of each transportation problem source is the sum of the feasible flow range lengths of the outgoing arcs from the corresponding original network node minus the supply of that node, as shown. An arc flow x_{ij} in (MCF) corresponds to flows equal to $x_{ij} - b_{ij}$ and $c_{ij} - x_{ij}$ on the transportation problem arcs $\big(j,(i,j)\big)$ and $\big(i,(i,j)\big)$, respectively.

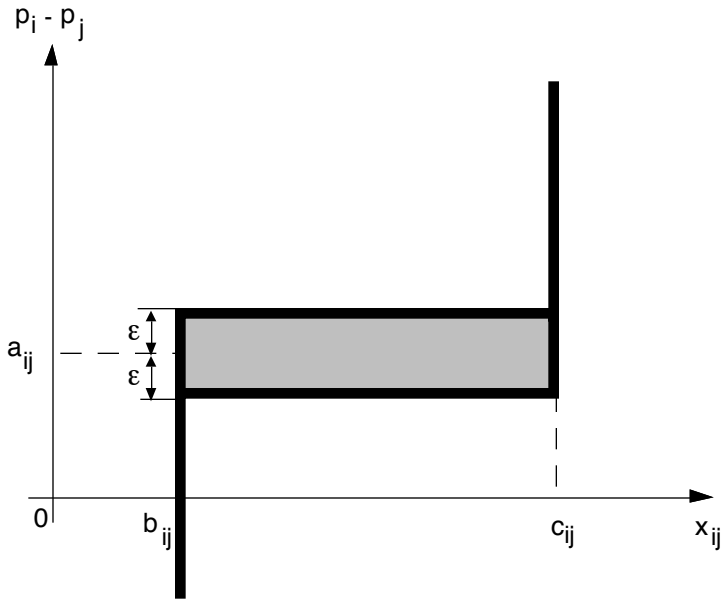


Figure 12: Illustration of ϵ-CS. All pairs of arc flows x_{ij} and price differences $p_i - p_j$ should either lie on the thick lines or in the shaded area between the thick lines.

arcs (cf. Fig. 11). These flow modifications could also be followed by an increase of the node price, while maintaining the ϵ-CS condition. Since in the context of the transportation algorithm we used notions of cooperative bids by several nodes, it also makes sense in the corresponding minimum cost flow context to consider simultaneous price increases by several nodes.

Motivated by these associations, we now introduce some computational operations that will constitute the building blocks for a general auction algorithm for the minimum cost flow problem. Each of the following definitions assumes that (x, p) is a flow-price vector pair satisfying ϵ-CS.

Definition 1: An arc (i, j) is said to be ϵ^+-*unblocked* if

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

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

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

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

In the subsequent algorithms, flow is allowed to increase only along ϵ^+-unblocked arcs and is allowed to decrease only along ϵ^--unblocked arcs. The next definition specifies the type of flow changes considered.

Definition 2: For an arc (i, j) [or arc (j, i)] of the push list P_i 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.

It is evident from the definitions that a δ-push preserves ϵ-CS. The next operation consists of raising the prices of a subset of nodes by the maximum common increment γ for which ϵ-CS will not be violated.

Definition 3: A *price rise* of a nonempty, strict subset of nodes I (i.e., $I \neq \emptyset$, $I \neq \mathcal{N}$), consists of leaving unchanged the flow vector x and the prices of nodes not belonging to I, and of increasing the prices of the nodes in I by the amount γ given by

$$\gamma = \begin{cases} \min\{S^+ \cup S^-\}, & \text{if } S^+ \cup S^- \neq \emptyset, \\ 0, & \text{if } S^+ \cup S^- = \emptyset, \end{cases} \tag{33}$$

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

$$S^+ = \{p_j + a_{ij} + \epsilon - p_i \mid (i, j) \in \mathcal{A} \text{ such that } i \in I, j \notin I, x_{ij} < c_{ij}\}, \tag{34}$$

$$S^- = \{p_j - a_{ji} + \epsilon - p_i \mid (j, i) \in \mathcal{A} \text{ such that } i \in I, j \notin I, x_{ji} > b_{ji}\}. \tag{35}$$

It can be verified using the definitions that a price rise maintains ϵ-CS. Note that every scalar in the sets S^+ and S^- of Eqs. (34) and (35) is nonnegative by the ϵ-CS conditions (32a) and (32b), respectively, so the scalar γ is nonnegative. If $\gamma > 0$, the price rise is said to be *substantive* and if $\gamma = 0$, the price

rise is said to be *trivial*. (A trivial price rise changes neither the flow vector nor the price vector; it is introduced to facilitate the presentation.)

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. A price rise of a single node i is substantive if and only if the set $S^+ \cup S^-$ is nonempty but the push list of i is empty. It can be shown that, for a feasible problem, if the push list of a node i is empty, then the set $S^+ \cup S^-$ must be nonempty.

The Generic Algorithm

The generic algorithm to be described shortly consists of a sequence of δ-push and price rise operations. The order of execution of these operations is subject to very mild restrictions, thus allowing a great deal of flexibility to exploit the structure of the problem at hand.

Suppose that the minimum cost flow problem (MCF) is feasible, and consider a pair (x, p) satisfying ϵ-CS. For a given flow vector x, let us define the *surplus* g_i *of node* i as the difference of the total flow coming into i and the total flow coming out of i, that is,

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

Consider a node i with $g_i > 0$. 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, as mentioned earlier, the 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. Furthermore, since following a price rise at a node i, the push list of i will be nonempty [see Eqs. (33)-(35)], for a feasible problem a δ-push is always possible at a node i with $g_i > 0$, possibly following a price rise at i.

The preceding observations form the basis for a method, called *generic algorithm*, which uses a fixed positive value of ϵ, and starts with a pair (x, p) satisfying ϵ-CS. The algorithm terminates when $g_i \leq 0$ for all nodes i; otherwise it continues to perform iterations. Each iteration consists of a sequence of δ-pushes and price rises, including at least one δ-push, as described below.

Typical Iteration of the Generic Algorithm

Perform in sequence and in any order a finite number of δ-pushes and price rises; there should be at least one δ-push but not necessarily at least one price rise. Furthermore:

(1) Each δ-push should be performed at some node i with $g_i > 0$, and the flow increment δ must satisfy $\delta \leq g_i$.

(2) Each price rise should be performed on a set I with $g_i \geq 0$ for all $i \in I$.

The following proposition, proved in [BeC89b], [BeC91], and [Ber91a], establishes the validity of the generic algorithm.

Proposition 7: 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 $\epsilon < 1/N$.

The idea of the proof is that δ-pushes cannot be indefinitely performed without some intermediate substantive price rises. Thus if the algorithm does not terminate, the prices of some nodes with positive surplus must increase to infinity, while the prices of some other nodes with negative surplus remain at their initial level. With some thought, it can be seen that this implies that for a feasible problem, ϵ-CS must be violated, leading to a contradiction.

If the problem is infeasible, the algorithm may not terminate. One way to deal with infeasibility is to convert the problem to one that is guaranteed to be feasible by introducing artificial high-cost arcs. The appropriate level of cost can be quantified similar to the case of assignment problems; see Eqs. (11) and (12), and [Ber91a].

7. THE ϵ-RELAXATION METHOD

The price rise operations of the generic algorithm may involve several nodes. When we require that only one node is involved in each price rise, we obtain the ϵ-relaxation method first proposed by the author in [Ber86a] and [Ber86b], and described in this section.

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; the algorithm preserves the integrality of the arc flows. 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-pushes and prise rises involving node i.

Typical Iteration of the ϵ-Relaxation Method

Select a node i with $g_i > 0$. If no such node exists, terminate the algorithm; else go to Step 1.

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

$$\delta = \begin{cases} \min\{g_i, c_{ij} - x_{ij}\} & \text{if } a = (i, j), \\ \min\{g_i, x_{ji} - b_{ji}\} & \text{if } a = (j, i). \end{cases} \tag{36}$$

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 $g_i = 0$ stop; else go to Step 1.

It is straightforward to verify that the ϵ-relaxation method is a special case of the generic algorithm of Section 6 to which Prop. 7 applies. As a result, for a feasible problem, the algorithm terminates with a pair (x, p) satisfying ϵ-CS; the flow x is optimal if $\epsilon < 1/N$. Similar to assignment problems, it is possible to use ϵ-scaling in conjunction with the method, and this is frequently essential for good practical performance.

The computational complexity of the unscaled version of the ϵ-relaxation method was first studied in [Ber86a]. The complexity of the scaled version was first studied in [Gol87], where particularly favorable polynomial running time estimates were derived; see [BeE87], [BeE88], [GoT90] for additional results along these lines.

8. APPLICATION OF THE ϵ-RELAXATION METHOD TO MAX-FLOW PROBLEMS

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 as shown in Fig. 13. Because of the small cost range one can often forego ϵ-scaling here, while still maintaining polynomial complexity [$O(N^3)$ or even better with appropriate implementation]. In practice one can also typically forego ϵ-scaling, but some additional computational tricks are needed to nullify the effects of price wars; see Section 10.5.

If ϵ–scaling is not used, one can change the cost of the feedback arc (t, s) to $a_{ts} = -(N + 1)$ and use $\epsilon = 1$ throughout. When this is done, the ϵ-relaxation method bears a close resemblance with a max-flow algorithm first proposed in [Gol85] and [GoT86], and often called "push-relabel" algorithm. This latter max-flow algorithm was derived from a different point of view that is unrelated to duality or ϵ-CS. It uses 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], [Ber86b], is simpler than the algorithm of [Gol85] and [GoT86] in that it obtains a maximum flow in one phase rather than two (single phase versions were also given later by several other authors; see [AMO89]). It can also be initialized with arbitrary prices, whereas in the max-flow algorithm of [Gol85], [GoT86] the initial prices must satisfy $p_i \leq p_j + 1$ for all arcs (i, j). This can be significant, for example if one wants to use scaling or more generally, in a reoptimization setting where one wants to capitalize on the results of the solution of a slightly different max-flow problem. Related max-flow algorithms and their computational complexity are discussed in [AhO86], [AMO89], [ChM89], and [MPS91].

9. EXTENSION TO ASYMMETRIC ASSIGNMENT PROBLEMS

A slight variation of the auction algorithm can be used for asymmetric assignment problems where the number of persons is m and the number of objects is n with $m < n$, while there is a requirement that each person be assigned to some object. To solve this problem, the auction algorithm need only be modified in the choice of initial conditions; it is sufficient to require that all initial prices be zero, as was noted in the original paper [Ber79].

Unfortunately, this approach to the asymmetric assignment problems has a deficiency: since the initial object prices must be zero, the use of ϵ-scaling is impossible. As a result the method is susceptible to "price wars" and the associated long running times. To address this difficulty, one may convert the asymmetric problem to a symmetric one by adding $n - m$ artificial persons that can be assigned to any

Figure 13: The max-flow problem and its minimum cost flow representation. Here there are two special nodes: the *source* (s) and the *sink* (t). The objective is to push as much flow as possible from s into t while observing the arc capacity constraints. We introduce an artificial arc (t, s) with cost -1 and very large upper flow bound and we assign cost zero to all other arcs. At the optimum, the flow x_{ts} equals the maximum flow that can be sent from s to t through the original graph.

object at zero cost. This introduces an undesirable increase in the problem's dimension. One could use the auction algorithm with similar persons of Section 5 to allow ϵ-scaling while taking advantage of the structure induced by the artificial persons. Nonetheless, the indirect approach of converting an asymmetric assignment problem into a symmetric one has not seen much use. We now discuss a direct and probably more effective auction approach for asymmetric assignment problems.

Reverse Auction for Asymmetric Assignment Problems

It is possible to use reverse auction in conjunction with forward auction to provide asymmetric assignment algorithms that use ϵ-scaling. We first introduce the proper form of the ϵ-CS condition for an assignment S and a pair (π, p). The following proposition, proved in [BCT91], parallels Prop. 1 for the symmetric assignment problem.

Proposition 8: If a feasible assignment S and a pair (π, p) satisfy the conditions

$$\pi_i + p_j \geq a_{ij} - \epsilon, \qquad \forall\ (i,j) \in \mathcal{A}, \tag{37a}$$

$$\pi_i + p_j = a_{ij}, \qquad \forall\ (i,j) \in S, \tag{37b}$$

$$p_j \leq \min_{k:\ \text{assigned under } S} p_k, \qquad \forall\ j:\ \text{unassigned under } S, \tag{37c}$$

then S is within $m\epsilon$ of being optimal for the asymmetric assignment problem.

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 (37a) and (37b), 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, it can be seen that this will yield, in a finite number of iterations, a feasible assignment S satisfying the first two conditions (37a) and (37b). This assignment may not be optimal because the prices of the unassigned objects may not be minimal, that is, they may not satisfy the third ϵ-CS condition (37c). Roughly, what is happening here is that forward auction cannot resolve whether the objects that were left unassigned upon termination are intrinsically "undesirable" because they offer relatively low benefit to the persons, or whether they were left unassigned because their initial prices were high relative to the initial prices of the assigned objects.

To resolve this dilemma, we use a modified form of reverse auction to lower the prices of the objects that were left unassigned upon termination of the forward auction. After several reverse auction iterations in which persons may be reassigned to other objects, the third condition (37c) is satisfied. The assignment thus obtained can be shown to satisfy all the ϵ-CS conditions (37a)-(37c) and by Prop. 8, is optimal within $m\epsilon$ (and strictly optimal if the problem data are integer and $\epsilon < 1/m$).

The modified reverse auction starts with a feasible assignment S and with a pair (π, p) satisfying the first two ϵ-CS conditions (37a) and (37b). (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,

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

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. (37a) and (37b), 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 (37c) will be satisfied as well. The scalar λ will be 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

$$\beta_j = \max_{i \in B(j)} \{a_{ij} - \pi_i\}, \tag{38}$$

and find

$$\omega_j = \max_{i \in B(j), i \neq i_j} \{a_{ij} - \pi_i\}. \tag{39}$$

[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

$$\delta = \min\{\beta_j - \lambda, \beta_j - \omega_j + \epsilon\}. \tag{40}$$

Set

$$p_j := \beta_j - \delta, \tag{41}$$

$$\pi_{i_j} := \pi_{i_j} + \delta, \tag{42}$$

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 (42) for the bidding increment δ is such that the object j enters the assignment at a price which is no less than λ [and is equal to λ if and only if the minimum in Eq. (42) is attained by the first term]. Furthermore, we have $\delta \geq \epsilon$ (when δ is calculated, that is, when $\lambda < \beta_j - \epsilon$), so it can be seen from Eqs. (41) and (42) that throughout the algorithm, prices are monotonically decreasing and profits are monotonically increasing. The following proposition, first proved in [BCT91] (see also [Ber91a]) asserts the validity of the method.

Proposition 9: The modified reverse auction algorithm for the asymmetric assignment problem terminates in a finite number of iterations and the assignment obtained is within $m\epsilon$ of being optimal.

As mentioned earlier, forward auction followed by modified reverse auction can start with arbitrary initial prices. As a result, one can use ϵ-scaling, performing a sequence of auctions with decreasing values of ϵ. Out of several possible variations of the method, the one found most effective in [BCT91] is to use the modified reverse auction only in the last ϵ-scaling phase. In all other ϵ-scaling phases, forward auction is used exclusively. There are also alternative algorithms for the asymmetric assignment problems that switch between the forward and the reverse methods within each ϵ-scaling phase; see [BeC92].

Forward/Reverse Auction for Other Types of Inequality Constrained Problems

Reverse auction can also be used to solve the variation of the *two-sided inequality constrained assignment problem*, where persons (as well as objects) need not be assigned if this degrades the assignment's total benefit. This problem can be converted to an asymmetric assignment problem where all persons must be assigned by introducing for each person i an artificial object i' and a zero benefit pair (i, i'). One can then use the algorithm given earlier to solve this problem. The algorithm can be streamlined so that the calculations involving the artificial objects and pairs are handled efficiently.

Finally, a class of interesting assignment-like problems, called *multiassignment problems*, can be solved efficiently by using combined forward/reverse auction ideas; see [BCT91]. In these problems, persons can be assigned to more than one object and objects can be assigned to more than one person simultaneously (the number of assignments for each person and object may or may not be subject to an upper bound).

10. PRACTICAL COMPUTATIONAL ASPECTS OF AUCTION ALGORITHMS

In this section we provide an overview of the computational aspects of auction algorithms concentrating primarily on the case of the assignment problem. Generally, at present, it appears that for simple problems such as assignment, shortest path, and max-flow, auction algorithms are at least competitive and often far superior to the classical methods. For problems with more complex structure, auction algorithms

have yet to match the performance of their closest competitors such as primal simplex, primal-dual, and relaxation methods. Generally, however, auction algorithms are better suited for parallel computation.

10.1. Assignment Problems

In practice, the auction algorithm has proved very effective for assignment problems. Computational studies [BCT91], [BeE88] have shown that it is often substantially faster than its closest competitors, particularly for sparse problems. The speedup factor typically increases with the problem size. As the problem density increases the speedup factor tends to decrease and for fully dense problems the auction algorithm is roughly competitive to methods that combine the primal-dual (or sequential shortest path method) with an extensive initialization procedure based on the naive auction algorithm of Section 2 [Ber81], [JoV87]. A recent extensive computational study [Cas92] explores the behavior of various auction algorithms for randomly generated problems with a broad variety of different structures.

Adaptive ϵ-Scaling

There are a number of variations of the ϵ-scaling technique, which are aimed at improving computational efficiency. There are many types of problems, where ϵ-scaling is essential because of the high likelihood of price wars. There are also other types of problems for which price wars are highly unlikely and it is better to either forego ϵ-scaling altogether or implement it "aggressively," that is, reduce ϵ very quickly to its ultimate value. This motivates scaling techniques, known collectively as *adaptive scaling*, where ϵ is reduced "aggressively" or "conservatively" based on the method's progress.

One type of adaptive scaling uses two values of ϵ within each scaling phase, the *ceiling value* denoted by $\overline{\epsilon}$, and the *current value* denoted by ϵ. The ceiling value $\overline{\epsilon}$ is kept constant within each scaling phase and is reduced at the end of the phase [up to an ultimate value of $1/(n+1)$]. The current value ϵ is the one used in the bidding increment $\gamma_i = w_i - v_i + \epsilon$ [cf. Eq. (9)], and may change within the range $[1/(n+1), \overline{\epsilon}]$. If the current value coincides with the ceiling value, the standard form of (nonadaptive) scaling is obtained. Scaling becomes truly adaptive if ϵ is started at a relatively small value [such as $1/(n+1)$] and is gradually increased according to various criteria towards the ceiling value $\overline{\epsilon}$. For example, ϵ may be kept constant as long as a new pair is added to the assignment within a fixed number of iterations, and may be gradually increased otherwise.

Adaptive scaling may also be useful when high-cost artificial pairs are introduced to guarantee that the problem is feasible as described earlier. Because of these pairs, the cost range of the problem is greatly expanded, thereby potentially increasing the number of scaling phases. On the other hand, the high-cost pairs are superfluous when the problem is feasible, and one may use a form of adaptive scaling to recognize this case and accordingly reduce ϵ to the appropriate levels.

Generally, the effective use of adaptive scaling requires some ingenuity and trial-and-error, as well as a fair amount of intuition into the structure of the problem at hand.

The Problem of Integer Overflow

If the benefits a_{ij} are integer, the auction algorithm can in most cases be executed using integer

arithmetic. The most common way to do this is to use integer starting prices, to multiply a_{ij} with $(n+1)$, and then to use integer values of ϵ and a final value of $\epsilon = 1$. The generated prices and other quantities will then be integer. There is a potential difficulty however: some prices may become large enough to exceed the integer range of the computer. In particular, from Eq. (10) we see that an upper bound for p_j is of the order of $n^2 \max_{(i,j)\in\mathcal{A}} |a_{ij}|$ [since a_{ij} has been multiplied by $(n+1)$ to work with integer arithmetic], which is large enough to lie outside the integer range of many computers even for moderate values of n. To remedy this situation one may store the prices in floating point format (preferably double precision) and use floating point arithmetic to execute the price update calculations, while using integer arithmetic for all other computations. In some experiments reported in [Cas92] it was observed that this involves a slowdown of the practical running time of the algorithm by a factor of no more than two. Thus it appears that the difficulty of integer overflow can be effectively addressed at a relatively small cost.

The "Third Best" Implementation

Frequently in the auction algorithm the two best objects for a given person do not change between two successive bids of that person. This motivates an implementation idea (developed by the author in collaboration with H. Tsaknakis – unpublished) 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 j is unnecessary.

Consider an auction algorithm iteration when we calculate the bid of the person i on the basis of a price vector p. Suppose that in addition to the best value $v_i = \max_{j\in A(i)}\{a_{ij} - p_j\}$, the best object $j_1 = \arg\max_{j\in A(i)}\{a_{ij} - p_j\}$, and the second best value $w_i = \max_{j\in A(i),\, j\neq j_1}\{a_{ij} - p_j\}$, we compute:

(a) The second best object

$$j_2 = \arg \max_{j\in A(i),\, j\neq j_1} \{a_{ij} - p_j\}.$$

(b) The *third best value*

$$y_i = \max_{j\in A(i),\, j\neq j_1,\, j\neq j_2} \{a_{ij} - p_j\}.$$

Consider now a subsequent iteration where person i bids based on an updated price vector $\overline{p} \geq p$. If we have

$$a_{ij_1} - \overline{p}_{j_1} \geq y_i \tag{43}$$

and

$$a_{ij_2} - \overline{p}_{j_2} \geq y_i, \tag{44}$$

it can be seen that j_1 and j_2 continue to be the two best objects for i. The reason is that the object prices cannot decrease in the course of the algorithm, implying that $y_i \geq a_{ij} - \overline{p}_j$ of all j other than j_1 and j_2. As a result, if the tests (43) and (44) are passed, we can forego the calculation of the values $a_{ij} - p_j$ for the objects j other than j_1 and j_2. If on the other hand one of the tests (43) and (44) is violated, we cannot make any inference regarding the best and second best objects; we must then exhaustively compare the values of all objects $j \in A(i)$, and compute the best, second best, and third best values. Computational

experience has shown that the savings in the calculation of object values whenever the tests (43) and (44) are passed far outweigh the overhead for maintaining the third best values y_i, and for performing the tests (43) and (44). This is particularly so when the problem is fairly dense, so that the set $A(i)$ has many objects.

Parallel and Asynchronous Implementation

Both the bidding and the assignment phases of the auction algorithm are highly parallelizable. In particular, the bids can be computed simultaneously and in parallel for all persons participating in the auction. Similarly, the subsequent awards to the highest bidders can be computed in parallel by all objects that received a bid. Furthermore, the bid of a single person can be computed cooperatively by several processors in parallel. There have been several implementations of the auction algorithm in parallel shared memory machines [BeC89c], [KKZ89], [Zak90], and in SIMD machines [PhZ88], [CSW89], [WeZ90], [WeZ91]. If there is strict temporal separation between the bid calculation phase and the highest bidder award phase, the implementation is said to be *synchronous*. In such an implementation, there are two basic methods to parallelize the bidding phase for the set I of unassigned persons that submit a bid, and a third method which is a combination of the other two. Let p be the number of processors:

(a) *Parallelization across bids (or Jacobi parallelization):* Here the calculations involved in the bid of each person $i \in I$ are carried out by a single processor. If the number of persons in I, call it $|I|$, exceeds the number of processors p, some processors will execute the calculations involved in more than one bid. If $|I| < p$, then $p - |I|$ processors will be idle during the bidding phase, thereby reducing efficiency. (This will typically happen in the late stages of the algorithm.)

(b) *Parallelization within a bid (or Gauss-Seidel parallelization):* Here the set I consists of a single person i. The calculations involved in the bid of person i are shared by the p processors of the system. Thus the set of admissible objects $A(i)$ is divided in p groups of objects $A_1(i), A_2(i), \ldots, A_p(i)$ [assuming the number of processors is less than the number of objects in $A(i)$; otherwise some of the processors will be left idle]. The best object, best value, and second best value are calculated within each group in parallel by a separate processor. After these calculations are completed (a synchronization of the processors is required to check this) the results are "merged" by one of the processors who finds the best value over all best group values, while simultaneously computing the corresponding best object and bid increment. (It is possible to do the merging in parallel using several processors, but in the absence of special SIMD-type hardware, this may be inefficient, particularly when the number of processors is small, because of the extra synchronization and other overhead involved.) The drawback of this method over the preceding one is that it typically requires a larger number of iterations, since each iteration involves a single person. This is significant because even though each Gauss-Seidel iteration may take less time because it is executed by multiple processors in parallel, the synchronization overhead is roughly proportional to the number of iterations.

(c) *Hybrid approach (or block Gauss-Seidel parallelization):* In this approach, the bid calculations of each person are parallelized as in the preceding method, but the number of processors used per bid

is s, where $1 < s < p$. Thus we can compute the bids of roughly p/s persons in parallel, assuming enough unassigned persons are available for the iteration. With proper choice of s, this method combines the best features and alleviates the drawbacks of the preceding two.

Once the bidding phase of an iteration is completed (a synchronization point), the assignment phase is executed. This phase may be carried out by a single processor. It is also possible to consider using multiple processors to execute the assignment phase in parallel, but the potential gain from parallelization may be modest while the associated overhead may more than offset this gain, as suggested by one computational study [BeC89c] that used a shared memory machine.

There are also *totally asynchronous* implementations of the auction algorithm, which are interesting because they are quite flexible and may also result in faster solution. To our knowledge, all parallel asynchronous implementations of the auction algorithm todate have used a shared memory machine. In one such implementation [BeC89c], the bidding and merging calculations of the Gauss-Seidel method are divided in tasks, which are organized in a first in – first out queue. When a processor becomes free it starts executing the top task of the queue, if the queue is nonempty, and otherwise it checks whether a termination condition is satisfied. The algorithm stops when all processors encounter the termination condition.

Similar to the synchronous block Gauss-Seidel implementation, each set of admissible objects $A(i)$ is divided in s groups of objects $A_1(i), \ldots, A_s(i)$. The calculation of the bid of a person i is divided in s tasks. The first $s-1$ tasks are search tasks involving the groups of objects $A_1(i), \ldots, A_{s-1}(i)$. To perform one of these tasks, a processor must calculate and store in memory the best value, second best value, and best object within the corresponding object group. The sth task starts with a search and memory storage of the best value, second best value, and best object within the group $A_s(i)$, and following this, it completes the bid of person i by merging the individual group search results, that is, by finding the best object and bid for person i based on the currently stored group results. The sth task also includes raising the price of the best object and changing the assignment of the object (assuming the calculated bid is larger than the best object's price by at least ϵ).

There are two sources of asynchronism here. First, it is possible for some prices to be changed between the time a search task is completed and the time the results of that task are used to calculate a person bid. Second, it is possible that the merging task of a person's bid is carried out before some of the search tasks associated with that bid are completed. In both cases the bid may reflect out-of-date price information and may prove ineffective in that it yields a bid that does not exceed the corresponding best object's price by at least ϵ (if this occurs, one should simply cancel the bid and forego the corresponding update of the object's price and assignment).

The advantage of the asynchronous implementation is that processors do not remain idle waiting to get synchronized with other processors or waiting for merging tasks to be completed. A careful formulation of the totally asynchronous model, and a proof of its validity is given in [BeC89c], which includes also extensive computational results on a shared memory machine, confirming the advantage of asynchronous over synchronous implementations.

Computational experience so far suggests that generally, because of the typically prolonged "endgame" of the auction algorithm, where only a small percentage of the persons remains unassigned, the speedup

that can be obtained by Jacobi parallelization is relatively modest (in the order of three to six). The attainable speedup for Gauss-Seidel parallelization or hybrid schemes is potentially higher, particularly when the problem is dense, and also when special hardware with vector processing capabilities are used.

10.2. Shortest Path Problems

Aside from combining the forward auction/shortest path algorithm with its reverse counterpart, there are a number of useful implementation ideas.

The main computational bottleneck of the algorithm is the calculation of

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

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

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

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

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

together with an arc (i,j_i) such that

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

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 $\min_{(i,j)\in\mathcal{A}}\big\{a_{ij}+p_j\big\}$. 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. For some theoretical substantiation of this point, see [BPS92].

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

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

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

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

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

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

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,"

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

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 $\min_{(i,j)\in\mathcal{A}}\big\{a_{ij} + p_j\big\}$ by a factor that is typically in the range from 3 to 5, thereby dramatically reducing the total computation time. Both of the above devices, together with graph reduction, are used in the auction/shortest path codes that are presently the fastest for single origin/all destination randomly generated problems; see [BPS92].

Parallel Implementation

When there is a single destination and multiple origins, several interesting parallel computation possibilities arise. The idea is to maintain a different path P^i for each origin i, and possibly, a reverse path R for the destination. Different paths may be handled by different processors, and price information can be shared by the processors in some way. There are several possible implementations of this idea. We will describe a shared memory implementation, and we refer to [Ber91b], and the MS thesis [Pol91] for discussions of message passing implementations. For simplicity, we will not consider the possibility of using the reverse path R; the thesis [Pol91] and the paper [PoB92] discuss parallel two-sided algorithms.

Here, there is a common price vector p stored in the shared memory that is accessible by all processors. For each origin i, there is a path P^i satisfying CS together with p. In a synchronous implementation of the algorithm, an iteration is executed simultaneously for some origins (possibly all origins, depending on the availability of processors). At the end of an iteration, the results corresponding to the different origins are coordinated. To this end, we note that if a node is the terminal node of the path of several origins, the result of the iteration will be the same for all these origins, i.e., a path extension or a path contraction and corresponding price change will occur simultaneously for all these origins. The only potential conflict arises when a node i is the terminal path node for some origin and the path of a different origin is extended by i as a result of the iteration. Then, if p_i is increased due to a path contraction for the former origin, the path extension of the latter origin is cancelled. An additional important detail is that an origin i can stop its computation once the terminal node of its path P^i is an origin that has already found its shortest

path to the destination. Thus, the processor handling this origin may be diverted to handle the path of another origin.

It is reasonable to speculate that the parallel time to solve the multiple origins problem is closer to the smallest time over all origins to find a single-origin shortest path, rather than to the longest time. However, this conjecture needs to be tested experimentally on a shared memory machine.

The parallel implementation outlined above is synchronous, that is, all origins iterate simultaneously, and the results are communicated and coordinated at the end of the iteration to the extent necessary for the next iteration. An asynchronous implementation is also possible principally because of the monotonicity of the mapping

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

see [BeT89]. We refer to [Pol91] and [PoB92] for a discussion of such an asynchronous implementation.

10.3. Transportation Problems

A serial version of the auction/transportation algorithm of Section 6 has been implemented and tested in [BeC89a]. The algorithm uses adaptive ϵ-scaling and integer arithmetic; a version using floating point arithmetic, which would be useful for a large cost range, has not been tested. Computational results show that this auction algorithm is considerably faster than its chief competitors for important classes of transportation problems. Generally these problems are characterized by two properties, *homogeneity* and *asymmetry*. A homogeneous problem is one for which there are only few levels of supply and demand. An asymmetric problem is one for which the number of sources is much larger than the number of sinks. For other types of transportation problems, computational experimentation suggests that the auction algorithm is outperformed by state-of-the-art codes based on relaxation methods; see [BeC89a], [Ber91a].

We know also of unpublished studies of parallel and asynchronous implementations of the transportation/auction algorithm by D. A. Castañon (unpublished). These studies indicate that with Jacobi parallelization, a modest speedup (of the order of about five) is possible on shared memory machines.

10.4. Minimum Cost Flow Problems

Serial implementations of the ϵ-relaxation method for general minimum cost flow problems are not yet competitive in practice with implementations of other methods, although they are not overwhelmingly worse; see [Ber91a]. However, the ϵ-relaxation method is well suited for parallel computation, so in some parallel computing environments it may be faster than its closest competitors. The method admits a totally asynchronous implementation, as shown in the original paper [Ber86a]; see also [BeE88], [BeT89]. A synchronous massively parallel implementation of the ϵ-relaxation method is presented in [LiZ91].

The ϵ-relaxation method can also be implemented in a reverse version where a node price is *decreased*, while flow is "pulled" along incident arcs towards the node rather than "pushed" away from the node. The idea here is fairly similar to the one of reverse auction. In computational tests ([Ber91a], [LiZ91]), a combined forward/reverse version of ϵ-relaxation seems to perform better than the forward version, but the evidence is not conclusive. The forward/reverse version also has greater parallelism potential

since nodes with positive as well as negative surplus can iterate simultaneously (with some safeguards to preserve desirable termination properties). Parallel implementations of this type have not yet been tested.

10.5. Max-Flow Problems

As mentioned in Section 8, the max-flow version of the ϵ-relaxation method can be implemented efficiently without using ϵ-scaling. It is important to understand, however, that intense price wars can still occur, particularly for very sparse problems, despite the small cost range and the polynomial complexity of the method. To alleviate the detrimental effects of price wars some computational tricks are essential. In particular, it has been observed that for problems where price wars occur, the ϵ-relaxation method finds a minimum cut very quickly and may then spend a great deal of additional time to resolve subsequent price wars. 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 has already been obtained. One method for doing this is based on breadth-first search and is described in [Ber91a] (Exercise 5.4); it is used in a max-flow code given in Appendix A.6 of [Ber91a]. A maximum flow can still be obtained after a minimum cut is found by using a variant of the Ford-Fulkerson method, and this can typically be done very quickly. Related procedures that aim at ameliorating the effects of price wars have been discussed in [Gol87] and [MPS91].

It should be noted, however, that despite the practical success of schemes such as the one described above, worst-case max-flow examples have been constructed where the unscaled version of the ϵ-relaxation method can take a very large number of iterations (proportional to N^2); see [BeT89], p. 387. These examples are quite artificial, and it is unknown whether the method can excibit such behavior for some important types of practical problems.

11. CONCLUSIONS

Much progress has been made in the last few years to extend and apply auction algorithms to a variety of linear network flow problems, and to place them on an equal footing with the classical primal cost and dual cost improvement methods. Still, auction algorithms are new and not yet fully developed. With further research, they should become more broadly applicable and more competitive with the classical methods.

The auction algorithms discussed in this paper have been implemented in computer codes. Several of these codes appear in the author's textbook [Ber91a]; their latest versions are available from the author.

REFERENCES

[ACM91] Ajevedo, J. A., Costa, M. E. S., Madeira, J. J. S., and Martins, E. V., "An Algorithm for the Ranking of Shortest Paths," Working paper, Universidade de Coimbra, Coimbra, Portugal, 1991.

[AMM91] Ajevedo, J. A., Madeira, J. J. S., Martins, E. V., and Pires, F. M., "A Computational Improvement for a Shortest Paths Ranking Algorithm," Working paper, Universidade de Coimbra, Coimbra, Portugal, 1991.

[AMO89] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B., "Network Flows," Sloan W. P. No. 2059-88, M.I.T., Cambridge, MA, 1989, (also in Handbooks in Operations Research and Management Science, Vol. 1, Optimization, G. L. Nemhauser, A. H. G. Rinnooy-Kan, and M. J. Todd (eds.), North-Holland, Amsterdam, 1989, pp. 211-369).

[AhO86] Ahuja, R. K., and Orlin, J. B., "A Fast and Simple Algorithm for the Maximum Flow Problem," Working paper, M.I.T., Cambridge, MA, 1986, (also in Operations Research, Vol. 37, 1989, pp. 748-759).

[BCT91] Bertsekas, D. P., Castañon, D. A., and Tsaknakis, H., 1991. "Reverse Auction and the Solution of Inequality Constrained Assignment Problems," Unpublished Report, 1991.

[BPS92] Bertsekas, D. P., Pallottino, S., and Scutella', M. G., "Polynomial Auction Algorithms for Shortest Paths," submitted for publication.

[BeC89a] Bertsekas, D. P., and Castañon, D. A., "The Auction Algorithm for Transportation Problems," Annals of Operations Research, Vol. 20, pp. 67-96.

[BeC89b] Bertsekas, D. P., and Castañon, D. A., "The Auction Algorithm for the Minimum Cost Network Flow Problem," Laboratory for Information and Decision Systems Report LIDS-P-1925, M.I.T., Cambridge, MA, November 1989.

[BeC89c] Bertsekas, D. P., and Castañon, D. A., "Parallel Synchronous and Asynchronous Implementations of the Auction Algorithm," Alphatech Report, Burlington, MA, Nov. 1989; also Parallel Computing, Vol. 17, 1991, pp. 707-732.

[BeC91] Bertsekas, D. P., and Castañon, D. A., "A Generic Auction Algorithm for the Minimum Cost Network Flow Problem," Alphatech Report, Burlington, MA, Sept. 1991.

[BeC92] Bertsekas, D. P., and Castañon, D. A., "A Forward/Reverse Auction Algorithm for the Asymmetric Assignment Problem," Alphatech Report, Burlington, MA, April 1992.

[BeE87] Bertsekas, D. P., and Eckstein, J., "Distributed Asynchronous Relaxation Methods for Linear Network Flow Problems," Proc. of IFAC '87, Munich, Germany, July 1987.

[BeE88] Bertsekas, D. P., and Eckstein, J., "Dual Coordinate Step Methods for Linear Network Flow Problems," Math. Progr., Series B, Vol. 42, 1988, pp. 203-243.

[BeM73] Bertsekas, D. P., and Mitter, S. K., "Descent Numerical Methods for Optimization Problems with Nondifferentiable Cost Functions," SIAM Journal on Control, Vol. 11, pp. 637-652.

[BeT89] Bertsekas, D. P., and Tsitsiklis, J. N., Parallel and Distributed Computation: Numerical Methods, Prentice-Hall, Englewood Cliffs, N. J., 1989.

[Ber79] Bertsekas, D. P., "A Distributed Algorithm for the Assignment Problem," Lab. for Information and Decision Systems Working Paper, M.I.T., March 1979.

[Ber81] Bertsekas, D. P., "A New Algorithm for the Assignment Problem," Math. Programming, Vol. 21, 1981, pp. 152-171.

[Ber85] Bertsekas, D. P., "A Distributed Asynchronous Relaxation Algorithm for the Assignment Problem," Proc. 24th IEEE Conf. Dec. & Contr., 1985, pp. 1703-1704.

[Ber86a] Bertsekas, D. P., "Distributed Asynchronous Relaxation Methods for Linear Network Flow Problems," Lab. for Information and Decision Systems Report P-1606, M.I.T., November 1986.

[Ber86b] Bertsekas, D. P., "Distributed Relaxation Methods for Linear Network Flow Problems," Proceedings of 25th IEEE Conference on Decision and Control, 1986, pp. 2101-2106.

[Ber88] Bertsekas, D. P., "The Auction Algorithm: A Distributed Relaxation Method for the Assignment Problem," Annals of Operations Research, Vol. 14, 1988, pp. 105-123.

[Ber91a] Bertsekas, D. P., Linear Network Optimization: Algorithms and Codes, M.I.T. Press, Cambridge, Mass., 1991

[Ber91b] Bertsekas, D. P., "The Auction Algorithm for Shortest Paths," SIAM J. on Optimization, Vol. 1, 1991, pp. 425-447.

[CSW89] Castañon, D., Smith, B., and Wilson, A., "Performance of Parallel Assignment Algorithms on Different Multiprocessor Architectures", Alphatech inc. Report, Burlington, Mass., 1989.

[Cas92] Castañon, D. A., "Reverse Auction Algorithms for Assignment Problems," DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1992.

[ChM89] Cheriyan, J., and Maheshwari, S. N., "Analysis of Preflow Push Algorithms for Maximum Network Flow," SIAM J. Comput., Vol. 18, 1989, pp. 1057-1086.

[Dan63] Dantzig, G. B., Linear Programming and Extensions, Princeton Univ. Press, Princeton, N. J, 1963.

[Dre69] Dreyfus, S. E., "An Appraisal of Some Shortest-Path Algorithms," Operations Research, Vol. 17, 1969, pp. 395-412.

[FoF62] Ford, L. R., Jr., and Fulkerson, D. R., Flows in Networks, Princeton Univ. Press, Princeton, N. J, 1962.

[GoT86] Goldberg, A. V., and Tarjan, R. E., "A New Approach to the Maximum Flow Problem," Proc. 18th ACM STOC, 1986, pp. 136-146.

[GoT90] Goldberg, A. V., and Tarjan, R. E., "Solving Minimum Cost Flow Problems by Successive Approximation," Math. of Operations Research, Vol. 15, 1990, pp. 430-466.

[Gol85] Goldberg, A. V., "A New Max-Flow Algorithm," Tech. Mem. MIT/LCS/TM-291, Laboratory for Computer Science, M.I.T., Cambridge, MA., 1985.

[Gol87] Goldberg, A. V., "Efficient Graph Algorithms for Sequential and Parallel Computers," Tech. Report TR-374, Laboratory for Computer Science, M.I.T., Cambridge, MA., 1987.

[JoV87] Jonker, R., and Volegnant, A., "A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems," Computing, Vol. 38, 1987, pp. 325-340.

[KKZ89] Kempa, D., Kennington, J., and Zaki, H., "Performance Characteristics of the Jacobi and Gauss-Seidel Versions of the Auction Algorithm on the Alliant FX/8," Report OR-89-008, Dept. of Mech. and Ind. Eng., Univ. of Illinois, Champaign-Urbana, 1989.

[Kuh55] Kuhn, H. W., "The Hungarian Method for the Assignment Problem," Naval Research Logistics Quarterly, Vol. 2, 1955, pp. 83-97.

[Law76] Lawler, E., Combinatorial Optimization: Networks and Matroids, Holt, Reinhart, and Winston, N. Y., 1976.

[LiZ91] Li, X., and Zenios, S. A., "Data Parallel Solutions of Min-Cost Network Flow Problems Using ϵ-Relaxations," Report 1991-05-20, Dept. of Decision Sciences, The Wharton School, Univ. of Pennsylvania, Phil., Penn., 1991.

[Lue84] Luenberger, D. G., Linear and Nonlinear Programming, Addison-Wesley, Reading, MA, 1984.

[Mar84] Martins, E. V., "An Algorithm for Ranking Paths that May Contain Cycles," European J. of Operations Research, Vol. 18, 1984, pp. 123-130.

[MPS91] Mazzoni, G., Pallotino, S., and Scutella', M. G., "The Maximum Flow Problem: A Max-Preflow Approach," European J. of Operational Research, Vol. 53, 1991.

[Min57] Minty, G. J., "A Comment on the Shortest Route Problem," Operations Research, Vol. 5, 1957, p. 724.

[PaS82] Papadimitriou, C. H., and Steiglitz, K., Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, N. J., 1982.

[PaS91] Pallottino, S., and Scutella', M. G., "Strongly Polynomial Algorithms for Shortest Paths," Dipartimento di Informatica Report TR-19/91, University of Pisa, Italy, 1991.

[PhZ88] Phillips, C., and Zenios, S. A., "Experiences with Large Scale Network Optimization on the Connection Machine," Report 88-11-05, Dept. of Decision Sciences, The Wharton School, Univ. of Pennsylvania, Phil., Penn., Nov. 1988.

[PoB92] Polymenakos, L., and Bertsekas, D. P., "Parallel Shortest Path Auction Algorithms," Lab. for Information and Decision Systems Report, M.I.T., April 1992.

[Pol91] Polymenakos, L., "Analysis of Parallel Asynchronous Schemes for the Auction Shortest Path Algorithm," MS Thesis, EECS Dept., M.I.T., Cambridge, MA., Jan. 1991.

[Roc84] Rockafellar, R. T., Network Flows and Monotropic Programming, Wiley-Interscience, N. Y., 1984.

[Sch90] Schwartz, B. L., "A Computational Analysis of the Auction Algorithm," Unpublished Manuscript.

[WeZ90] Wein, J., and Zenios, S. A., "Massively Parallel Auction Algorithms for the Assignment Problem," Proc. of 3rd Symposium on the Frontiers of Massively Parallel Computation, Md., 1990.

[WeZ91] Wein, J., and Zenios, S. A., "On the Massively Parallel Solution of the Assignment Problem," J. of Parallel and Distributed Computing, Vol. 13, 1991, pp. 228-236.

[Zak90] Zaki, H., "A Comparison of Two Algorithms for the Assignment Problem," Report ORL 90-002, Dept. of Mechanical and Industrial Engineering, Univ. of Illinois, Urbana, Ill., 1990.

APPENDIX 1: E-CS, PRIMAL OPTIMALITY, AND DUAL OPTIMALITY

Assignment Problems

Let us fix $\epsilon \geq 0$. We show that given a feasible assignment $\{(i, j_i) \mid i = 1, \ldots, n\}$ and a set of prices $\{\bar{p}_j \mid j = 1, \ldots, n\}$, which satisfy ϵ-complementary slackness (if $\epsilon > 0$) or complementary slackness (if $\epsilon = 0$), the assignment is within $n\epsilon$ of maximizing the total benefit, and is optimal if $\epsilon = 0$. Furthermore, the set of prices is within $n\epsilon$ of minimizing a certain dual cost function.

Let $\epsilon > 0$. We first note that the total benefit of *any* feasible assignment $\{(i, k_i) \mid i = 1, \ldots, n\}$ satisfies

$$\sum_{i=1}^{n} a_{ik_i} \leq \sum_{j=1}^{n} p_j + \sum_{i=1}^{n} \max_j \{a_{ij} - p_j\},$$

for any set of prices $\{p_j \mid j = 1, \ldots, n\}$, since the second term of the right-hand side is no less than

$$\sum_{i=1}^{n} \left(a_{ik_i} - p_{k_i}\right),$$

while the first term is equal to $\sum_{i=1}^{n} p_{k_i}$. Therefore,

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

where A^* is the optimal total assignment benefit

$$A^* = \max_{\substack{k_i \in A(i),\, i=1,\ldots,n \\ k_i \neq k_m \text{ if } i \neq m}} \sum_{i=1}^{n} a_{ik_i}$$

and

$$D^* = \min_{\substack{p_j \\ j=1,\ldots,n}} \left\{ \sum_{j=1}^{n} p_j + \sum_{i=1}^{n} \max_{j \in A(i)} \left\{a_{ij} - p_j\right\} \right\}.$$

On the other hand, since the given assignment $\{(i, j_i) \mid i = 1, \ldots, n\}$ satisfies ϵ-CS together with the set of prices $\{\bar{p}_j \mid j = 1, \ldots, n\}$, we have

$$\max_{j \in A(i)} \left\{a_{ij} - \bar{p}_j\right\} - \epsilon \leq a_{ij_i} - \bar{p}_{j_i},$$

and by adding this relation over all i, we see that

$$D^* \leq \sum_{i=1}^{n} \left(\bar{p}_{j_i} + \max_{j \in A(i)} \{a_{ij} - \bar{p}_j\} \right) \leq \sum_{i=1}^{n} a_{ij_i} + n\epsilon \leq A^* + n\epsilon.$$

Since we showed earlier that $A^* \leq D^*$, it follows that the total assignment benefit $\sum_{i=1}^{n} a_{ij_i}$ is within $n\epsilon$ of the optimal value A^*.

Note that the function

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

appearing in the definition of D^*, may be viewed as a *dual function* of the price variables p_j, and its minimization may be viewed as a *dual problem* in the standard linear programming duality context; see [Ber91a], [BeT89], [Roc84], and [PaS82]. It is seen from the preceding analysis, that the prices $\bar{p}_j$ attain within $n\epsilon$ the dual optimal value D^*.

If we let $\epsilon = 0$ in the preceding argument, we see that $A^* = D^*$, and that an assignment and a set of prices that are at equilibrium, maximize the total benefit and minimize the dual function, respectively.

Minimum Cost Flow Problems

We now consider the minimum cost flow problem and we prove the following generalized version of Prop. 6, which holds even if the problem data are not integer.

Proposition 10: Let the flow-price pair (x, p) satisfy ϵ-CS, and suppose that x is feasible. Then x is optimal for the minimum cost flow problem (MCF), 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

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

In particular, x is optimal if the problem data are integer and $\epsilon < 1/N$.

Proof: If x is not optimal, then (see e.g., [Ber91a], p. 24) there exists a simple cycle Y that has negative cost, i.e.,

$$\sum_{(i,j) \in Y^+} a_{ij} - \sum_{(i,j) \in Y^-} a_{ij} < 0, \tag{45}$$

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

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

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

By ϵ-CS [cf. Eq. (32)], 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. (45). **Q.E.D.**

APPENDIX 2: FINITE TERMINATION OF THE AUCTION ALGORITHM

In this appendix we show that for a feasible problem and for any positive value of ϵ, 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 $\epsilon < 1/n$). This shows in particular Prop. 2.

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 Eq. (9)]. 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 best object value v_i defined by

$$v_i = \max_{j\in A(i)} \{a_{ij} - p_j\} \tag{46}$$

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. (46). 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 $-\infty$.

We now 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 $i \in I^\infty$. Therefore, $a_{ij} - p_j$ tends to $-\infty$ for all $j \in A(i)$, implying that

$$A(i) \subset J^\infty, \qquad \forall\ i \in I^\infty. \tag{47}$$

The ϵ-CS condition (8) 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. (47), 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 (since the algorithm preserves ϵ-CS throughout), so by Prop. 1, this assignment is within $n\epsilon$ of being optimal.