November 1994

LIDS-P-2276

RELAX-IV: A Faster Version of the RELAX Code for Solving Minimum Cost Flow Problems [1]

by

Dimitri P. Bertsekas [2] **and Paul Tseng** [3]

Abstract

The structure of dual ascent methods is particularly well-suited for taking advantage of good initial dual solutions of minimum cost flow problems. For this reason, these methods are extremely efficient for reoptimization and sensitivity analysis. In the absence of prior knowledge of a good initial dual solution, one may attempt to find such a solution by means of a heuristic initialization. RELAX-IV is a minimum cost flow code that combines the RELAX code of [BeT88a], [BeT88b] with an initialization based on a recently proposed auction/sequential shortest path algorithm. This initialization is shown to be extremely helpful in speeding up the solution of difficult problems, involving for example long augmenting paths, for which the relaxation method has been known to be slow. On the other hand, this initialization procedure does not significantly deteriorate the performance of the relaxation method for the types of problems where it has been known to be very fast.

To obtain RELAX-IV by anonymous FTP, ftp to LIDS.MIT.EDU with username ANONYMOUS, enter password as directed, and type cd /pub/bertsekas/RELAX to go to the RELAX subdirectory, which in addition to the RELAX-IV code contains documentation and random problem generation and conversion codes.

[1] Research supported by NSF under Grant CCR-9103804 and Grant CCR-9311621.
[2] Dept. of Electrical Engineering and Computer Science, M.I.T., Cambridge, Mass., 02139.
[3] Dept. of Mathematics, Univ. of Washington, Seattle, Wash., 98195.

1. INTRODUCTION

This paper provides a brief description of a new version of the RELAX code for solving the classical minimum cost flow problem with integer data. In the problem (abbreviated as (MCF)), we are given a directed graph comprising node set $\mathcal{N}$ and arc set $\mathcal{A} \subset \mathcal{N} \times \mathcal{N}$. We are also given, for each $(i,j) \in \mathcal{A}$, an integer a_{ij} and a positive integer c_{ij}, which may be viewed as, respectively, the cost and the capacity of arc (i,j), and, for each $i \in \mathcal{N}$, an integer s_i, which may be viewed as the exogenous supply of node i. The aim is to find an arc flow x_{ij}, for all $(i,j) \in \mathcal{A}$, that minimize

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

subject to satisfying the flow conservation and the capacity constraints:

$$\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} = s_i, \qquad \forall\, i \in \mathcal{N}, \tag{1}$$

$$0 \le x_{ij} \le c_{ij}, \qquad \forall\, (i,j) \in \mathcal{A}. \tag{2}$$

We denote by x the flow vector consisting of the arc flows x_{ij}, $(i,j) \in \mathcal{A}$. A flow vector x is called *feasible* if it satisfies the constraints (1) and (2) and is called *optimal* if it solves (MCF).

We introduce a price p_i for each node i, which may be viewed as a dual variable associated to the flow conservation constraint at i. We denote by p the vector consisting of p_i, $i \in \mathcal{N}$. A price vector p, and a flow vector x are said to satisfy *complementary slackness* (CS for short) if x satisfies the capacity constraints (2) and

$$x_{ij} < c_{ij} \quad \Rightarrow \quad p_i \le a_{ij} + p_j \quad \forall\, (i,j) \in \mathcal{A},$$

$$0 < x_{ji} \quad \Rightarrow \quad p_i \le p_j - a_{ji} \quad \forall\, (j,i) \in \mathcal{A}.$$

It is well-known that if x is feasible, and (x,p) satisfies CS, then x is optimal for (MCF) and p is optimal for a corresponding dual problem.

Dual ascent methods generate a sequence of price vectors, each with an improved value of dual cost. Many of these methods also generate flow vectors satisfying CS together with the price vectors, and modify these flow vectors through the use of augmentations. Historically, the first dual ascent method is the primal-dual method of Ford and Fulkerson [FoF57], [FoF62]. The relaxation method originally proposed in [Ber95] and further developed in [Tse86] and [BeT88a] is another dual ascent method, which differs from the primal-dual method in the choice of ascent direction. While the primal-dual method aims for an ascent direction with as "steep" a slope as possible at the expense of considerable computational overhead, the relaxation method aims at finding an ascent direction quickly, and often selects a coordinate direction that corresponds to a single price. Thus the relaxation method resembles a coordinate ascent method, although the directions it uses are not always coordinate directions.

The relaxation method has proved particularly effective in practice. Its implementation in the RELAX code described in [BeT88b], has resulted in very fast solution times relative to its main competitors for many types of problems, particularly those involving graphs with relatively small diameter, or more generally, relatively short augmenting paths.

Several minimum cost flow algorithms, including the primal-simplex, the primal-dual and the relaxation method tend to be slow when faced with problems involving graphs with large diameter and long augmenting paths, such as grid graphs. This phenomenon cannot be explained by the presently existing worst-case complexity analysis, but has been consistently observed in practice, and can be understood through a closer examination of the calculations involved in a typical iteration of each method. The adverse effect of long augmenting paths is particularly strong for the relaxation method. In particular, there are difficult problems for which earlier versions of RELAX can be very slow relative to its competitors.

An important advantage of dual ascent methods over the primal simplex method is that they are very well-suited for reoptimization and sensitivity analysis. The reason is that the optimal prices obtained from solution of some problem, are feasible and very likely excellent starting prices for solving a slighly different problem. The primal simplex method does not afford this flexibility because the optimal flows obtained from solution of a problem, may not be feasible for a slighly different problem, and they may not be easily used to obtain a good initial basic solution.

This advantage of dual ascent methods can also be exploited when solving a new problem by trying to obtain good initial prices with some heuristic method. The purpose of the new version of the RELAX code, called RELAX-IV, is to improve the performance of the relaxation method for difficult problems by providing a procedure to obtain good sets of starting prices and flows. We describe this initialization procedure below.

3. INITIALIZATION USING THE AUCTION ALGORITHM

The auction algorithm for the minimum cost flow problem was introduced in [Ber92] and is also described in the Appendix. It relies on a sequential shortest path augmentation approach, where the length of each arc is equal to its reduced cost. Each shortest path is constructed by means of the recently proposed auction/shortest path algorithm [Ber91a], [Ber91b]. However, the naive implementation of this approach fails because of the presence of zero cost cycles. This difficulty is overcome by using as arc lengths ϵ-perturbations of reduced costs and by using ϵ-complementary slackness conditions in place of the usual complementary slackness conditions. For good practical performance, ϵ-scaling is also important here: the normal way to operate the method is to start with a relatively large value of ϵ in order to obtain good starting prices for applying the method for smaller values of ϵ.

The auction algorithm is not adversely affected by long augmenting paths to the extent that the relaxation method is, particularly for relatively large values of ϵ. As a result, it is a good candidate for initialization of the relaxation method. In particular, the auction initialization of RELAX-IV uses one or two scaling phases of the auction algorithm with relatively high values of ϵ. The number of scaling phases and values of ϵ can be adjusted by the user. The default initialization uses one scaling phase with $\epsilon = C/8$, where C is the cost range (the difference between maximum and minimum arc cost).

3. SOME COMPUTATIONAL RESULTS

In this section we report on the performance of RELAX-IV on various test problems, including NETGEN problems and grid-type problems, and compare the performance to those of RELAXT-III [BeT88b], an earlier version of RELAX that does not have the auction initialization, and of NETFLO, a very efficient Fortran implementation of the network primal-simplex method written by Kennington and Helgason [KeH80]. All three Fortran codes were compiled and ran on a DECstation 3100 under the operating system Ultrix 4.2. The test problems were generated by the following five problem generators: (i) NETGEN, by D. Klingman, A. Napier and J. Stutz [KNS74], which generates assignment/transportation/transshipment problems with random structure; (ii) MESH and GOTO, by A. V. Goldberg (see [ReV91]), which generate transshipment problems with a grid structure (GOTO problems differ from MESH problems mainly in that all grid nodes are pure transshipment nodes, rather than sources/sinks, and a super source and a super sink are added and joined to some of the grid nodes); (iii) GRIDGRAPH, by M. G. C. Resende and G. Veiga [ReV91], which generates grid-type transshipment problems much like GOTO, but with the supply at the super source set to the maximum possible; (iv) GRIDGEN, by D. P. Bertsekas [Ber91a, Appendix A.1], which generates grid-type transshipment problems with multiple sources/sinks and transshipment nodes. To better compare our results with those existing, we generated the same NETGEN problems as in [BeT88a], the same MESH, GOTO, GRIDGRAPH problems as in [ReV91], and we generated GRIDGEN problems with the same number of nodes, number of arcs, total supply, cost range, and capacity range as the GOTO problems in [ReV91]. In addition, a number of the GTE-BAD problems [LSS91], which are 49-node-520-arc shortest-path-type problems arising from the solution of a certain multicommodity flow problem, were used in our tests. The GOTO, GRIDGRAPH and GTE-BAD problems are considered to be difficult for the relaxation method, so they in some sense provide the stiffest test for RELAX-IV.

The test results are summarized in Tables 1 to 6. We make the following observations: First, RELAX-IV with option 1 (using auction initialization) is faster than RELAX-IV with option 0 (no auction initialization) on the GOTO and GRIDGRAPH problems, but is either comparable to or slower than the latter on other problems. This confirms the benefit of using auction initialization on difficult grid-type problems and points to the following optimal setting for RELAX-IV: use option 1 on difficult grid-type problems and use option 0 on all other problems. Second, compared to RELAXT-III, RELAX-IV with the optimal setting is either faster or comparable on all test problems. Compared to NETFLO, RELAX-IV with the optimal setting is faster on NETGEN and MESH problems, is somewhat faster on GRIDGEN problems, is comparable on GOTO problems (faster on some and slower on others), and is slower on the GTE-BAD and GRIDGRAPH problems. (The results on the GTE-BAD problems are difficult to judge due to the small size and special structure of the problems. The results on the GRIDGRAPH problems can be partially explained by noting that these problems, with the supply at the super source set to the maximum possible, are nearly primal infeasible. Such problems are known to be difficult for dual ascent methods since the corresponding dual problems are nearly unbounded.) Thus, RELAX-IV is efficient not only on problems for which the relaxation method is known to be very fast (such as NETGEN and MESH problems) but also on problems for which the relaxation method has been very slow (such as GOTO and GTE-BAD problems). On GRIDGRAPH problems, RELAX-IV shows clear improvement over its predecessor RELAXT-III, but is still slower than NETFLO.

SIZE		RELAX-IV (opt=0)	RELAX-IV (opt=1)	RELAXT-III	NETFLO
$\|\mathcal{N}\|$	$\|\mathcal{A}\|$	time	time	time	time
5000	23000	5.5	5.8	6.1	34.1
3000	35000	3.3	4.3	3.2	13.8
5000	15000	5.9	5.3	4.8	26.8
3000	23000	2.8	2.9	2.5	9.9
2000	7000	2.4	2.7	2.7	8.4
400	15000	1.1	2.0	1.4	1.7
1600	7000	2.1	2.8	3.4	6.7
400	15000	2.0	2.5	2.3	1.4
2000	7000	2.2	2.8	2.3	11.3
400	15000	0.9	1.3	0.9	3.4
1600	7000	2.1	2.3	1.8	10.5
400	15000	2.0	2.4	1.9	5.8
3000	12000	1.8	5.6	2.1	25.4
6000	24000	9.4	10.6	11.2	84.0
6000	24000	9.1	10.9	9.4	97.5
3000	30000	3.4	4.4	3.3	25.4

Table 1. Solution time (in seconds) for RELAX-IV, RELAXT-III and NETFLO on NETGEN problems. The first four problems are problems 37–40 in [KNS74, Table 1] (also see [BeT88a, Table I]); the next four problems are problems 5, 10, 15, 20 in [BeT88a, Table III]; the next four problems are problems 5, 10, 15, 20 in [BeT88a, Table IV]; the last four problems are problems 5, 10, 15, 20 in [BeT88a, Table VI].

SIZE		RELAX-IV (opt=0)	RELAX-IV (opt=1)	RELAXT-III	NETFLO
$\|\mathcal{N}\|$	$\|\mathcal{A}\|$	time	time	time	time
256	1040	0.2	0.3	0.2	0.6
1024	4096	2.2	2.6	2.3	14.3
4096	16384	16.6	27.5	22.9	305.2
256	2048	0.3	0.4	0.4	1.5
1024	8192	2.6	2.7	2.8	40.6
4096	32768	35.4	32.4	40.1	846.5

Table 2. Solution time (in seconds) for RELAX-IV, RELAXT-III and NETFLO on MESH problems. The first three problems are problems 1–3 in [ReV91, Table 20]; the last three problems are problems 1–3 in [ReV91, Table 22].

SIZE		RELAX-IV (opt=0)	RELAX-IV (opt=1)	RELAXT-III	NETFLO
$\|\mathcal{N}\|$	$\|\mathcal{A}\|$	time	time	time	time
256	2048	12.7	2.2	28.2	1.6
512	4096	57.2	7.1	299.5	8.6
1024	8192	168.9	32.2	461.5	37.5
2048	16384	1056.2	117.6	2087.9	124.5
256	4096	61.9	7.1	110.9	6.2
512	8192	309.1	21.2	539.1	21.0
1024	16384	1282.9	98.7	1874.8	80.6
2048	32768	5246.8	488.4	6422.2	317.9

Table 3. Solution time (in seconds) for RELAX-IV, RELAXT-III and NETFLO on GOTO problems. The first four problems are problems 1–4 in [ReV91, Table 4]; the last four problems are problems 1–4 in [ReV91, Table 6].

SIZE		RELAX-IV (opt=0)	RELAX-IV (opt=1)	RELAXT-III	NETFLO
$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{A}\rvert$	time	time	time	time
514	1040	1.9	0.5	1.5	0.2
1026	2096	6.4	2.1	6.5	0.9
2050	4208	27.5	6.8	29.2	3.8
4098	8432	130.1	21.7	150.8	14.2
514	1008	1.4	0.6	0.9	0.1
1026	2000	5.1	2.1	3.2	0.4
2050	3984	14.0	6.8	7.9	1.0
4098	7952	57.5	30.5	22.7	2.2

Table 4. Solution time (in seconds) for RELAX-IV, RELAXT-III and NETFLO on GRID-GRAPH problems. The first four problems are problems 1–4 in [ReV91, Table 14]; the last four problems are problems 1–4 in [ReV91, Table 16].

SIZE		RELAX-IV (opt=0)	RELAX-IV (opt=1)	RELAXT-III	NETFLO
$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{A}\rvert$	time	time	time	time
512	4096	0.2	0.3	0.4	0.4
1024	8192	1.0	1.6	0.9	1.6
2048	16384	2.4	2.1	4.8	4.0
4096	32768	7.9	6.8	4.9	9.7
512	8192	1.3	1.0	1.2	1.1
1024	16384	4.6	2.9	5.9	3.2
2048	32768	8.3	7.3	6.6	8.9
4096	65536	12.3	17.9	15.8	28.3

Table 5. Solution time (in seconds) for RELAX-IV, RELAXT-III and NETFLO on GRIDGEN problems. For all problems, the number of sources, the number of sinks, and the number of nodes in dimension 1 are fixed at 16; the cost range and capacity range for the added arcs are fixed at [0,4096] and [0,16384], respectively. Total supply for the eight problems are, respectively, 85055, 153113, 173074, 213302, 210731, 260384, 315135, 382723.

PROBLEM	RELAX-IV (opt=0)	RELAX-IV (opt=1)	RELAXT-III	NETFLO
NAME	time	time	time	time
298300	0.1	0.2	0.7	0.03
451760	0.04	0.2	117.9	0.02
469010	0.04	2.1	124.0	0.02
508829	0.05	0.2	169.4	0.02

Table 6. Solution time (in seconds) for RELAX-IV, RELAXT-III and NETFLO on GTE-BAD problems.

REFERENCES

[Ber86] Bertsekas, D. P., "Distributed Asynchronous Relaxation Methods for Linear Network Flow Problems," Laboratory for Information and Decision Systems Report P-1606, M.I.T., Cambridge, Nov. 1986.

[Ber91a] Bertsekas, D. P., Linear Network Optimization: Algorithms and Codes, MIT Press, Cambridge, MA, 1991.

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

[Ber92] Bertsekas, D. P., "An Auction/Sequential Shortest Path Algorithm for the Min Cost Flow Problem," Laboratory for Information and Decision Systems Report P-2146, M.I.T., Cambridge, MA, 1992.

[Ber93] Bertsekas, D. P., "An Auction Algorithm for the Max-Flow Problem," Report Laboratory for Information and Decision Systems Report P-2193, M.I.T., Cambridge, Aug. 1993; Journal of Optimization Theory and Applications, to appear.

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

[BPS92] Bertsekas, D. P., Pallottino, S., and Scutella', M. G., "Polynomial Auction Algorithms for Shortest Paths," Laboratory for Information and Decision Systems Report P-2107, M.I.T., Cambridge, May 1992; Computational Optimization and Applications, to appear.

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

[BeT88a] Bertsekas, D. P., and Tseng, P., "Relaxation Methods for Minimum Cost Ordinary and Generalized Network Flow Problems," Operations Research, Vol. 36, 1988, pp. 93-114.

[BeT88b] Bertsekas, D. P., and Tseng, P., "RELAX: A Computer Code for Minimum Cost Network Flow Problems," Annals of Operations Research, Vol. 13, 1988, pp. 127-190.

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

[CDP92] Cerulli, R., De Leone, R., and Piacente, G., "A Modified Auction Algorithm for the Shortest Path Problem," University of Salerno Report, Preprint, 1992.

[FoF57] Ford, L. R., Jr., and Fulkerson, D. R., "A Primal-Dual Algorithm for the Capacitated Hitchcock Problem," Naval Research Logistics Quarterly, Vol. 4, 1957, pp. 47-54.

[FoF62] Ford, L. R., Jr., and Fulkerson, D. R., Flows in Networks, Princeton University Press, Princeton, NJ, 1962.

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

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

[KeH80] Kennington, J. L., and Helgason, R. V., Algorithms for Network Programming, John Wiley and Sons, New York, NY, 1980.

[KNS74] Klingman, D., Napier, A., and Stutz, J., "NETGEN - A Program for Generating Large Scale (Un) Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems," Management Science, Vol. 20, 1974, pp. 814-822.

[LSS91] Leong, T., Shor, P. W., and Stein, C., "Implementation of a Combinatorial Multicommodity Flow Algorithm," in Network Flow and Matching: First DIMACS Implementation Challenge, Edited by D. Johnson and C. McGeoch, American Mathematical Society, Providence, RI, 1991, pp. 387-406.

[MPS91] Mazzoni, G., Pallotino, S., and Scutella', M. G., "The Maximum Flow Problem: A Max-Preflow Approach," European Journal of Operational Research, Vol. 53, 1991, pp. 257-278.

[PaS91] Pallottino, S., and Scutella', M. G., "Strongly Polynomial Algorithms for Shortest Paths," Ricerca Operativa, Vol. 60, 1991, pp. 33-53.

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

[ReV91] Resende, M. G. C., and Veiga, G., "An Efficient Implementation of a Network Interior Point Method," in Network Flow and Matching: First DIMACS Implementation Challenge, Edited by D. Johnson and C. McGeoch, American Mathematical Society, Providence, RI, 1991, pp. 299-348.

[Roc84] Rockafellar, R. T., Network Flows and Monotropic Programming, Wiley-Interscience, New York, NY, 1984.

[Tse86] Tseng, P., "Relaxation Methods for Monotropic Programming," Ph.D. Thesis, Department of Electrical Engineering and Operations Research Center, M.I.T., Cambridge, MA, 1986.

APPENDIX: THE AUCTION ALGORITHM FOR MINIMUM COST FLOW

For a given flow vector x, the *surplus* of node i is defined as the difference between the supply of i and the net outflow from i,

$$g_i = s_i + \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} - \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij}. \tag{3}$$

Given a scalar $\epsilon \geq 0$, a flow-price vector pair (x, p) is said to satisfy *ϵ-complementary slackness* (ϵ-CS for short) if x satisfies the capacity constraints (2) and

$$x_{ij} < c_{ij} \quad \Rightarrow \quad p_i \leq a_{ij} + p_j + \epsilon \qquad \forall\ (i,j) \in \mathcal{A}, \tag{4}$$

$$0 < x_{ji} \quad \Rightarrow \quad p_i \leq p_j - a_{ji} + \epsilon \qquad \forall\ (j,i) \in \mathcal{A}. \tag{5}$$

ϵ-CS was introduced in [Ber86] in the context of the ϵ-relaxation method, and its utility is due in large measure to the following proposition, which relies on the integrality of the problem data (see e.g. [BeT89] or [Ber91a] for a proof).

Proposition 1: If $\epsilon < 1/N$, x is feasible, and (x, p) satisfies ϵ-CS, then x is optimal for (MCF).

Note that when $\epsilon = 0$ the ϵ-CS conditions (4) and (5) reduce to the usual complementary slackness conditions. A standard duality result (see e.g., [BeT89], [Ber91a], [PaS82], [Roc84]) states that if x is feasible and together with some p satisfies these complementary slackness conditions, then x is optimal and p is optimal for an associated dual problem.

The classical primal-dual method maintains a pair (x, p) satisfying complementary slackness ($\epsilon = 0$), and at each iteration constructs a shortest path from some node with positive surplus to the set of nodes with negative surplus, along which it performs an augmentation of the current flow vector. The shortest path computation is performed in the *reduced graph* $G_R = (\mathcal{N}, \mathcal{A}_\mathcal{R})$ whose arc set $\mathcal{A}_\mathcal{R}$ consists of an arc (i,j) for each arc $(i,j) \in \mathcal{A}$ with $x_{ij} < c_{ij}$, and an arc (j,i) for each arc $(i,j) \in \mathcal{A}$ with $0 < x_{ij}$. The arc lengths are $a_{ij} + p_j - p_i$ for the arcs $(i,j) \in \mathcal{A}$ with $x_{ij} < c_{ij}$, and $p_i - a_{ij} - p_j$ for the arcs (j,i) corresponding to arcs $(i,j) \in \mathcal{A}$ with $0 < x_{ij}$. It is in principle possible to solve the shortest path problem by any shortest path method that requires nonnegative arc lengths, such as a Dijkstra-like method. The recently proposed auction algorithm for shortest paths (see [Ber91a], [Ber91b]) offers some advantages in this respect because of its ability to transfer information from one shortest path computation to the next, but requires that all cycles have strictly positive length. This method maintains a path, which is extended or contracted by a single arc at each iteration. Unfortunately, however, the method cannot be used conveniently in the context of the sequential shortest path method because the reduced graph has cycles with zero length [each arc (i,j) with $0 < x_{ij} < c_{ij}$ gives rise to the zero length arcs (i,j) and (j,i) in the reduced graph], and the path maintained by the auction/shortest path method can "double up on itself" and close a cycle.

To overcome this difficulty, it is possible to use auction/shortest path algorithms with graph reduction, as proposed in [PaS91] and [BPS92]. However, this requires considerable overhead and a separation of the shortest path construction process from the price change operations of the primal-dual algorithm. We use instead an alternative approach, where the auction/shortest path construction process is blended harmoniously with the remainder of the algorithm. In this approach, we use ϵ-perturbations of the arc lengths, which ensure that the path generated by the auction/shortest path method does not close a cycle through an extension. We first introduce some terminology.

Given a flow-price pair (x,p) satisfying ϵ-CS, an arc (i,j) is said to be ϵ^+*-unblocked* if

$$p_i = p_j + a_{ij} + \epsilon \qquad \text{and} \qquad x_{ij} < c_{ij}, \tag{6}$$

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

$$p_i = p_j - a_{ji} + \epsilon \qquad \text{and} \qquad 0 < x_{ji}. \tag{7}$$

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

A path P is a sequence of nodes $(n_1, n_2, \ldots, n_k)$ and a corresponding sequence of $k-1$ arcs such that the ith arc in the sequence is either (n_i, n_{i+1}) (in which case it is called a *forward* arc) or (n_{i+1}, n_i) (in which case it is called a *backward* arc). For any path P, we denote by $s(P)$ and $t(P)$ the start and terminal nodes of P, respectively, and by P^+ and P^- the sets of forward and backward arcs of P, respectively. We say that the path is *simple* if it has no repeated nodes. The path P is said to be ϵ*-unblocked* if all arcs of P^+ are ϵ^+-unblocked, and all arcs of P^- are ϵ^--unblocked. If P is ϵ-unblocked and the start node $s(P)$ has positive surplus and the terminal node $t(P)$ has negative surplus, we say that P is an *augmenting path*. An *augmentation* along such a path consists of increasing the flow of all arcs in P^+ and reducing the flow of all arcs in P^- by the common increment

$$\delta = \min\left\{ g_{s(P)},\ -g_{t(P)},\ \min_{(i,j)\in P^+}\{c_{ij} - x_{ij}\},\ \min_{(i,j)\in P^-}\{x_{ij}\} \right\}. \tag{8}$$

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

The algorithm of this appendix, first proposed in [Ber92], uses a fixed $\epsilon > 0$, and maintains a flow-price pair (x,p) satisfying ϵ-CS and also a simple path P (possibly consisting of a single node). It terminates when all nodes have nonnegative surplus; then either all nodes have zero surplus and x is feasible, or else some node has negative surplus showing that the problem is infeasible. Throughout the algorithm, x is integer, and (x,p) and P satisfy the following:

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

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

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

$$x_{ij} = \begin{cases} c_{ij} & \text{if } p_i \geq a_{ij} + p_j, \\ 0 & \text{if } p_i < a_{ij} + p_j, \end{cases} \tag{9}$$

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

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

Typical Iteration of the Auction/Sequential Shortest Path Algorithm

Let i be the terminal node of P. If

$$p_i < \min\left\{ \min_{\{(i,j)\in\mathcal{A} | x_{ij} < c_{ij}\}} \{a_{ij} + p_j + \epsilon\}, \min_{\{(j,i)\in\mathcal{A} | 0 < x_{ji}\}} \{p_j - a_{ji} + \epsilon\} \right\}, \tag{10}$$

go to Step 1; else go to Step 2.

Step 1 (Contract path): Set

$$p_i := \min\left\{ \min_{\{(i,j)\in\mathcal{A} | x_{ij} < c_{ij}\}} \{a_{ij} + p_j + \epsilon\}, \min_{\{(j,i)\in\mathcal{A} | 0 < x_{ji}\}} \{p_j - a_{ji} + \epsilon\} \right\}, \tag{11}$$

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

Step 2 (Extend path): Extend P by an arc (i, j_i) or an arc (j_i, i) that attains the minimum in Eq. (10). If the surplus of j_i is negative go to Step 3; otherwise, go to the next iteration.

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

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

Proposition 2: Suppose that at the start of an iteration:

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

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

Then the same is true at the start of the next iteration.

Proof: Suppose the iteration involves a contraction. Then it can be seen that the price increase (11) preserves the ϵ-CS conditions (4) and (5). Furthermore, since only the price of node i changes and no arc flow changes, the admissible graph remains unchanged except for the incident arcs of node i. In particular, all the incident arcs of i in the admissible graph at the start of the iteration are deleted and the arcs of the admissible graph corresponding to the arcs (i,j) and (j,i) that attain the minimum in Eq. (11) are added. Since all these arcs are outgoing from i in the admissible graph, a cycle cannot be closed. Finally, following a contraction, P does not contain the terminal node i, so it belongs to the admissible graph that we had before the iteration. Thus P consists of arcs that belong to the admissible graph that we obtain after the iteration.

Suppose the iteration involves an extension. Then by the ϵ-CS conditions (4) and (5), we must have

$$p_i = \min\left\{ \min_{\{(i,j)\in\mathcal{A} | x_{ij} < c_{ij}\}} \{a_{ij} + p_j + \epsilon\}, \min_{\{(j,i)\in\mathcal{A} | 0 < x_{ji}\}} \{p_j - a_{ji} + \epsilon\} \right\}, \tag{12}$$

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

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

A sequence of iterations between two successive augmentations (or the sequence of iterations up to the first augmentation) will be called an *augmentation cycle*. Let us fix an augmentation cycle and let $\overline{p}$ be the price vector at the start of the cycle. The reduced graph $G_R = (\mathcal{N}, \mathcal{A}_\mathcal{R})$ defined earlier will not change in the course of this augmentation cycle, since no arc flow will change during the cycle, except for the augmentation at the end. Suppose that we take as arc lengths of the reduced graph the reduced costs at the start of the cycle plus ϵ. In particular, during the cycle, the arc set $\mathcal{A}_\mathcal{R}$ consists of an arc (i, j) with length $a_{ij} + \overline{p}_j - \overline{p}_i + \epsilon$ for each arc $(i, j) \in \mathcal{A}$ with $x_{ij} < c_{ij}$, and an arc (j, i) with length $\overline{p}_i - a_{ij} - \overline{p}_j + \epsilon$ for each arc $(i, j) \in \mathcal{A}$ with $0 < x_{ij}$. Note that, because $(x, \overline{p})$ satisfies ϵ-CS, the arc lengths of the reduced graph are nonnegative. However, the reduced graph does not contain zero length cycles, since any such cycle must belong to the admissible graph, which is acyclic.

Using these observations, it can now be seen that the augmentation cycle is just the auction/shortest path algorithm of [Ber91a], [Ber91b] applied to the problem of finding a shortest path from the starting node $s(P)$ to some node with negative surplus in the reduced graph G_R, using the preceding ϵ-perturbed arc lengths. To understand this, one should view $p_i - \overline{p}_i$ during the augmentation cycle as the price of node i that is maintained by the auction/shortest path algorithm. The price increments $p_i - \overline{p}_i$ obtained by the auction/shortest path algorithm are added in effect to the starting prices $\overline{p}_i$ at the end of the augmentation cycle to form the new prices that will be used for the shortest path construction of the next augmentation cycle.

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

Proposition 3: Assume that the minimum cost flow problem (MCF) is feasible. Then the auction/sequential shortest path algorithm terminates with a pair (x, p) satisfying ϵ-CS. The flow vector x is feasible and is optimal if $\epsilon < 1/N$.

ϵ-Scaling

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

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

can be proved for the resulting method. The unscaled version of the method, where ϵ is kept fixed at $1/(N+1)$, is pseudopolynomial. These complexity bounds can be derived using well-known lines of analysis [Ber86a], [BeE87], [Gol87], [BeE88], [BeT89], [GoT90], and will not be proved here.

We now describe a number of variations of the algorithms of the preceding section, which we have empirically found to improve performance. Some of these variations are similar to corresponding variations of a related max-flow algorithm [Ber93].

Saving the Best Candidate

A number of implementation ideas have been shown to greatly accelerate the termination of the auction/shortest path algorithm [Ber91a], [Ber91b]. Some of these ideas are directly transferable to the minimum cost flow context, and are potentially very useful. In particular, the main computational bottleneck of the algorithm is the calculation of the best candidate arc for extension in Eq. (10), which is done every time node i becomes the terminal node of the path. We can reduce the number of these calculations by using the ϵ-CS condition

$$p_i \leq \min\left\{ \min_{\{(i,j)\in\mathcal{A}\mid x_{ij}<c_{ij}\}} \{a_{ij} + p_j + \epsilon\},\ \min_{\{(j,i)\in\mathcal{A}\mid 0<x_{ji}\}} \{p_j - a_{ji} + \epsilon\} \right\}, \tag{13}$$

and the following observation: if some arc (i, j_i) satisfies

$$p_i = a_{ij_i} + p_{j_i} + \epsilon \qquad \text{and} \qquad x_{ij_i} < c_{ij_i} \tag{14}$$

it follows that

$$p_i = \min\left\{ \min_{\{(i,j)\in\mathcal{A} | x_{ij} < c_{ij}\}} \{a_{ij} + p_j + \epsilon\}, \min_{\{(j,i)\in\mathcal{A} | 0 < x_{ji}\}} \{p_j - a_{ji} + \epsilon\} \right\},$$

so if i is the terminal node, the path can be extended by j_i. The same is true if some arc (j_i, i) satisfies

$$p_i = a_{j_i i} - p_{j_i} + \epsilon \qquad \text{and} \qquad 0 < x_{j_i i}. \tag{15}$$

This suggests the following implementation strategy: each time the minimum in Eq. (13) is calculated, we store an arc (i, j_i) such that Eq. (14) holds or an arc (j_i, i) such that Eq. (15) holds. At the next time node i becomes the terminal node of the path, we check whether Eq. (14) or (15), respectively, is still satisfied, and if it is, we extend the path by node j_i without going through the calculation of the minimum in Eq. (13). In practice this device is very effective.

Using a Second Best Candidate

Suppose that each time the minimum in Eq. (13) is calculated, we store an arc (i, j_i) such that Eq. (14) holds or an arc (j_i, i) such that Eq. (15) holds. Assume further that for the terminal node i of the current path P we have available a lower bound β_i on the value of the minimum in Eq. (13) over nodes j other than $j = j_i$, that is,

$$\min\left\{ \min_{\{(i,j)\in\mathcal{A} | x_{ij} < c_{ij},\, j\neq j_i\}} \{a_{ij} + p_j + \epsilon\}, \min_{\{(j,i)\in\mathcal{A} | 0 < x_{ji},\, j\neq j_i\}} \{p_j - a_{ji} + \epsilon\} \right\} \geq \beta_i. \tag{16}$$

Suppose also that the test for an extension is failed, that is, Eq. (13) holds with strict inequality. Then if the current "best" arc is (i, j_i), we can check to see whether we have

$$a_{ij_i} + p_{j_i} + \epsilon \leq \beta_i \qquad \text{and} \qquad x_{ij_i} < c_{ij_i} \tag{17}$$

and if this is so we know that (i, j_i) is still the best arc, thus making the computation of the minimum of Eq. (13) unnecessary. An analogous statement holds if the current "best" arc is (j_i, i) and we have

$$a_{j_i i} - p_{j_i} + \epsilon \leq \beta_i \qquad \text{and} \qquad 0 < x_{j_i i}. \tag{18}$$

A lower bound β_i can be obtained by calculating, together with the "best" arc, a "second best" arc (i, j_i') [or (j_i', i)] in the minimization of Eq. (13), and a corresponding value $\beta_i = a_{ij_i'} + p_{j_i'} + \epsilon$ (or $\beta_i = a_{j_i' i} - p_{j_i'} + \epsilon$, respectively) out of those entering the minimization in Eq. (13). Then, because node prices are monotonically nondecreasing throughout the algorithm, as long as no new arc incident to i becomes admissible, we can use β_i as a suitable lower bound (if a new arc incident to i enters the admissible graph due to an augmentation, we must suitably modify β_i and the corresponding "second best" arc). Furthermore, if the test of Eqs. (17) and (18) is failed, we can check to see whether the second best arc (i, j_i') [or (j_i', i)] is still admissible and whether $a_{ij_i'} + p_{j_i'} + \epsilon = \beta_i$ (or $a_{j_i'i} - p_{j_i'} + \epsilon = \beta_i$, respectively). If this is so the "second best arc" becomes the "best" arc, thereby obviating again the calculation of the minimum in Eq. (13).

The idea of using a "second best" arc has been shown to be very effective in auction algorithms for the assignment problem ([Ber91a], p. 176), the shortest path problem [Ber91b], [CDP92], and max-flow problems [Ber93]. It similarly improves substantially the performance of the algorithm of this paper.

Saving Path Fragments

Suppose that following an augmentation that starts at a node s, a portion of P starting at s and ending at some node i is still unblocked, while the surplus of s is still positive. Then we can start the next iteration with the same node s, move directly to the terminal node i, and continue the search for an augmenting path from there. This variation, which was also discussed in a different context in [MPS91], saved a modest amount of computation time in our experiments.

Early Flow Augmentations

We have found empirically that the total number of price changes is reduced if the length of the current path (the number of arcs of the path) is not allowed to become too long. Under some circumstances, this can lead to the path P becoming alternatively short and long many times before an augmentation can occur. We have thus employed the heuristic of performing an augmentation along the current path, whenever a contraction occurs with an attendant price change of 2ϵ or less, and furthermore the number of arcs of the path is more than two.

Optimistic Extensions

In practice, it appears that the effectiveness of the algorithm is enhanced significantly if an extension is performed not just when

$$p_i = \min\left\{ \min_{\{(i,j)\in\mathcal{A} \mid x_{ij}<c_{ij}\}} \{a_{ij} + p_j + \epsilon\},\ \min_{\{(j,i)\in\mathcal{A} \mid 0<x_{ji}\}} \{p_j - a_{ji} + \epsilon\} \right\}, \tag{19}$$

but also when the weaker condition

$$p_i \geq \min\left\{ \min_{\{(i,j)\in\mathcal{A} \mid x_{ij}<c_{ij}\}} \{a_{ij} + p_j\},\ \min_{\{(j,i)\in\mathcal{A} \mid 0<x_{ji}\}} \{p_j - a_{ji}\} \right\} \tag{20}$$

holds. This maintains ϵ-CS, and allows the path to "extend faster" towards a negative surplus node, but introduces a difficulty: a cycle may be closed by extending the path, that is, the extension node j_i may already belong to P. One can bypass this difficulty by keeping track of which nodes belong to P and by checking for a potential cycle formation. Whenever a cycle is about to be closed by extension, a "retreat" operation is performed, which backtracks along the path and sets the price of each successive terminal node i to

$$\min\left\{\min_{\{(i,j)\in\mathcal{A}\mid x_{ij}<c_{ij}\}}\{a_{ij}+p_j+\epsilon\},\ \min_{\{(j,i)\in\mathcal{A}\mid 0<x_{ji}\}}\{p_j-a_{ji}+\epsilon\}\right\} \tag{21}$$

up to the point where p_i strictly increases. Despite the overhead introduced by retreat operations, in our experiments, optimistic extensions resulted in considerable net saving in computation time.

A particularly interesting fact is that in the case of a max-flow problem, the retreat operations are unnecessary, that is, the path never closes a cycle even if the weaker criterion (20) is used for an extension. This is shown in [Ber93], where the corresponding path construction algorithm is studied in more detail and is embedded within the Ford-Fulkerson augmentation approach.