Math. Program., Ser. A 88: 85–104 (2000)

Digital Object Identifier (DOI) 10.1007/s101070000153

Paul Tseng · Dimitri P. Bertsekas

An ϵ-relaxation method for separable convex cost generalized network flow problems*

Received: November 10, 1996 / Accepted: October 1999
Published online April 20, 2000 – © Springer-Verlag 2000

Abstract. We generalize the ϵ-relaxation method of [14] for the single commodity, linear or separable convex cost network flow problem to network flow problems with positive gains. The method maintains ϵ-complementary slackness at all iterations and adjusts the arc flows and the node prices so as to satisfy flow conservation upon termination. Each iteration of the method involves either a price change on a node or a flow change along an arc or a flow change along a simple cycle. Complexity bounds for the method are derived. For one implementation employing ϵ-scaling, the bound is polynomial in the number of nodes N, the number of arcs A, a certain constant Γ depending on the arc gains, and $\ln(\epsilon^0/\overline{\epsilon})$, where ϵ^0 and $\overline{\epsilon}$ denote, respectively, the initial and the final tolerance ϵ.

1. Introduction

Consider a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ with node set $\mathcal{N} = \{1, \ldots, N\}$ and arc set $\mathcal{A} \subset \mathcal{N} \times \mathcal{N}$. We denote by N the number of nodes and by A the number of arcs. (The implicit assumption that there exists at most one arc in each direction between any pair of nodes is made for notational convenience and can be dispensed with.) We are given, for each node $i \in \mathcal{N}$, a scalar s_i (*supply* of i) and, for each arc $(i, j) \in \mathcal{A}$, a positive scalar γ_{ij} (*gain* of (i, j)) and a convex, closed, proper function $f_{ij} : \Re \to \Re \cup \{\infty\}$ (*cost function* of (i, j)). The generalized separable convex cost network flow problem is

$$\text{minimize} \quad f(x) := \sum_{(i,j)\in\mathcal{A}} f_{ij}(x_{ij}) \tag{P}$$

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

where the real variable x_{ij} is referred to as the *flow* of the arc (i, j) and the vector $x = \{x_{ij} \mid (i, j) \in \mathcal{A}\}$ is referred to as the *flow vector*. We refer to (P) as the *primal* problem. We assume that each f_{ij} is co-finite in the sense that $\lim_{\zeta\to-\infty} f_{ij}^-(\zeta) = -\infty$

P. Tseng: Department of Mathematics, University of Washington, Seattle, Washington 98195, USA
e-mail: tseng@math.washington.edu

D.P. Bertsekas: Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Rm. 35-210, Cambridge, Massachusetts 02139, USA, e-mail: dimitrib@mit.edu

Mathematics Subject Classification (1991): 90C35, 90C25

* This research is supported by the National Science Foundation, Grant Nos. CCR-9311621 and DMI-9300494. The extended abstract of this article appeared in the proceedings of the 5th International IPCO Conference, Vancouver, June 1996.

and $\lim_{\zeta\to\infty} f_{ij}^+(\zeta) = \infty$, where $f_{ij}^-(\zeta)$ and $f_{ij}^+(\zeta)$ denote, respectively, the left and right derivative of f_{ij} at ζ [31, p. 116], [32, p. 315]. An important special case is the *linear cost* case, where

$$f_{ij}(x_{ij}) := \begin{cases} a_{ij}x_{ij} & \text{if } b_{ij} \le x_{ij} \le c_{ij} \\ \infty & \text{otherwise} \end{cases}, \tag{2}$$

for given scalars a_{ij}, b_{ij}, c_{ij}. This special case has been much studied (see [3, Chap. 15], [9, Chap. 8], [25], [28, Chap. 8], and references therein) and has applications in many areas including financial planning, logistics, hydroelectric power system control (see [2], [4], [20], [22] and references therein). Another important special case is the *ordinary network* case, where $\gamma_{ij} = 1$ for all $(i, j) \in \mathcal{A}$ (see [9], [32]). For the general case, a range of applications is surveyed in [20]. A flow vector x satisfying

$$f_{ij}(x_{ij}) < \infty, \qquad \forall\ (i, j) \in \mathcal{A},$$

as well as the conservation-of-flow constraint (1) is called *feasible*. A feasible x satisfying

$$f_{ij}^-(x_{ij}) < \infty \ \text{ and } \ f_{ij}^+(x_{ij}) > -\infty, \qquad \forall\ (i, j) \in \mathcal{A}, \tag{3}$$

is called *regularly feasible* (see [32, p. 329], [9, p. 418]). We will assume that *there exists at least one regularly feasible flow vector*. In the linear cost case of (2), the condition (3) reduces to the standard capacity constraint: $b_{ij} \le x_{ij} \le c_{ij}$ for all $(i, j) \in \mathcal{A}$.

There is a well-known duality framework for this problem (see [32] and also [9], [17], [31]), involving a Lagrange multiplier p_i for the ith conservation-of-flow constraint (1). We refer to p_i as the *price* of node i, and to the vector $p = \{p_i \mid i \in \mathcal{N}\}$ as the *price vector*. The *dual* problem is

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

where the dual cost function q is given by

$$q(p) := \sum_{(i,j)\in\mathcal{A}} q_{ij}(p_i - \gamma_{ij}p_j) - \sum_{i\in\mathcal{N}} s_i p_i,$$

and q_{ij} is derived from f_{ij} by the conjugacy relation

$$q_{ij}(t_{ij}) := \sup_{x_{ij}\in\Re} \{x_{ij}t_{ij} - f_{ij}(x_{ij})\}.$$

The co-finite assumption on f_{ij} implies that q_{ij} is real-valued and, together with the existence of a regularly feasible solution, guarantees that both the primal problem (P) and the dual problem (D) have optimal solutions and their optimal costs are the negatives of each other ([32, p. 360], [9, p. 452]).

Following [13] and [34], we say that a flow vector x and a price vector p satisfy the *ϵ-complementary slackness* (ϵ-CS for short) conditions, where ϵ is any positive scalar, if

$$\begin{aligned} &f_{ij}(x_{ij}) < \infty, \quad \text{and} \\ &f_{ij}^-(x_{ij}) - \epsilon \le p_i - \gamma_{ij}p_j \le f_{ij}^+(x_{ij}) + \epsilon, \qquad \forall\ (i, j) \in \mathcal{A}. \end{aligned} \tag{ϵ − CS}$$

It is known ([32, p. 360], [9, p. 452]) that a feasible flow vector x and a price vector p are primal and dual optimal, respectively, if and only if they satisfy 0-CS.

For the primal problem (P) and its dual (D), there are available a number of solution methods, such as nonlinear variants of the primal simplex method, dual simplex method [32, Sec. 11J], and relaxation method [34]. The primal simplex method iteratively improves the primal cost function and the other methods iteratively improve the dual cost function. For the ordinary network case (with convex cost), there have recently been proposed other solution methods, based on minimum mean cycle canceling [26], and on ϵ-relaxation [9, Chap. 9], [14], [15], [21], [29]. For the linear cost ordinary network case, further specialization and improvements of the preceding methods, as well as many other methods, have been proposed (see the books [3], [16], [28], [32] and references therein). If f is twice differentiable (not necessarily separable), the primal truncated-Newton method specialized to generalized network flow [2], [20] and the general-purpose reduced gradient method using quasi-Newton updates [27] can also be applied.

Here, we propose an extension of the ϵ-relaxation method studied in [14], [15], [21], [29] from the ordinary network case to the generalized network case of (P) and (D), and we present a (computational) complexity analysis for the method. Our interest in the ϵ-relaxation method stems from its good performance on linear/quadratic cost ordinary network flow problems, as reported in the above references, and its suitability for implementation on parallel computers [5], [6], [11]. However, the extension is highly nontrivial due to the presence of nonunity arc gains. In particular, flow augmentations along cycles of non-unity gain need to be considered and new techniques need to be developed to deal with the presence of directed cycles in the admissible graph. In fact, even for the linear cost case our method and the associated complexity bounds are new to our knowledge. Previous complexity bounds for the linear cost case (other than those obtained by specializing general linear programming complexity bounds [35]) are further restricted to either the case of all nodes being supply nodes (i.e., $s_i \geq 0$ for all $i \in \mathcal{N}$) or being demand nodes (i.e., $s_i \leq 0$ for all $i \in \mathcal{N}$), with zero lower capacity on all arcs [1], [18], or the case of a generalized circulation (i.e., maximizing the flow on a particular arc) [23], [30]. We also report some of our computational experience with the method. Our experience indicates that, as in the ordinary network case (for which extensive computational results are given in [14]), the method is not significantly affected by ill-conditioning in the cost function. Furthermore, on nonlinear problems, our method substantially outperforms a nonspecialized nonlinear programming code such as MINOS.

The remainder of this paper is organized as follows. In Sect. 2, we motivate and formally describe the ϵ-relaxation method for solving (P) and (D). In Sect. 3, we show that the method has finite termination for any fixed ϵ. In Sect. 4, we present one specific implementation of the method which has a particularly favorable complexity bound. In Sect. 5, we report some of our numerical experience with the method. In what follows, by a path P in $\mathcal{G}$, we mean a sequence of nodes $(i_1, i_2, ..., i_m)$ in $\mathcal{N}$ and an associated sequence of $(m-1)$ arcs in $\mathcal{A}$ such that, for each $k = 1, ..., m-1$, either (i_k, i_{k+1}) or (i_{k+1}, i_k) is an arc of the sequence. The set of forward arcs of P (those of the form (i_k, i_{k+1})) is denoted by P^+ and the set of backward arcs of P (those of the form

(i_{k+1}, i_k)) is denoted by P^-. We define the gain of the path P by

$$\gamma_P := \Big(\prod_{(i,j)\in P^+} \gamma_{ij}\Big) \Big/ \Big(\prod_{(i,j)\in P^-} \gamma_{ij}\Big), \tag{4}$$

with $\gamma_P := 1$ if P comprises a single node. We say that a path P is *forward* if $P^- = \emptyset$. A *cycle* is a path whose starting node equals the ending node. A path is said to be *simple* if it contains no repeated nodes except (in the case where the path is a cycle) for the starting and the ending nodes.

2. The ϵ-relaxation method

In this section we formally describe an ϵ-relaxation method, based on ϵ-CS, for solving (P) and (D). For a flow vector x, we define the *surplus* of node i to be the difference between the supply s_i and the net outflow from i:

$$g_i := s_i + \sum_{\{j|(j,i)\in\mathcal{A}\}} \gamma_{ji} x_{ji} - \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij}. \tag{5}$$

The idea of the ϵ-relaxation method is to alternately adjust the price of the nodes and the flow of the arcs so as to maintain ϵ-CS while decreasing the surplus of the nodes toward zero. Termination occurs when the surpluses of all the nodes are zero. We describe this in more detail below.

The method uses two fixed scalars $\epsilon > 0$ and $\theta \in (0, 1)$. For any flow-price vector pair (x, p) satisfying ϵ-CS, we say that an arc $(i, j) \in \mathcal{A}$ is *active* if

$$p_i - \gamma_{ij} p_j > f_{ij}^+(x_{ij}) + \theta\epsilon \tag{6a}$$

and is *inactive* if

$$p_i - \gamma_{ij} p_j < f_{ij}^-(x_{ij}) - \theta\epsilon. \tag{6b}$$

In the ordinary network case, an arc being active (respectively, inactive) is equivalent to, in the terminologies of [9], [14], [15], the arc being in the candidate/push list of its starting (respectively, ending) node. For an active (respectively, inactive) arc (i, j), the supremum of δ for which

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

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

Proposition 1. *The flow margins of all active and inactive arcs are positive.*

Proof. We argue by contradiction. Assume that for an arc $(i, j) \in \mathcal{A}$ we have

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

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

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

and thus (i, j) cannot be active. A similar argument shows that an arc $(i, j) \in \mathcal{A}$ such that

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

cannot be inactive.

□

The ϵ-relaxation method starts with a flow-price vector pair (x, p) satisfying ϵ-CS. The method comprises two phases. In the first phase, only "up" iterations (to be defined shortly) are performed so as to adjust (x, p) until no node with positive surplus remains. In the second phase, only "down" iterations (to be defined shortly) are performed so as to adjust (x, p) until no node with negative surplus remains.

ϵ-relaxation method – general form ($\epsilon > 0$)

Initialization: Choose any flow vector $x = \{x_{ij} \mid (i, j) \in \mathcal{A}\}$ and price vector $p = \{p_i \mid i \in \mathcal{N}\}$ satisfying ϵ-CS. Fix any scalar $\theta \in (0, 1)$.

First phase: Repeatedly choose a node i with positive surplus g_i and adjust (x, p) by doing an up iteration at node i, until all nodes have nonpositive surplus.

Second phase: Repeatedly choose a node i with negative surplus g_i and adjust (x, p) by doing a down iteration at node i, until all nodes have nonnegative surplus.

In an up iteration at a node i with positive surplus g_i, we perform one of the following three operations:

(a) A *price rise* on node i, which increases the price p_i by the maximum amount that maintains ϵ-CS, while leaving all arc flows and all other prices unchanged.

(b) A *flow push* (also called a *δ-flow push*) *along an arc* (i, j) [or along an arc (j, i)], which increases x_{ij} [or decreases x_{ji}] by a positive amount δ, while leaving all node prices and all other arc flows unchanged.

(c) A *flow push* (also called a *δ-flow push*) *along a cycle* C containing i, which increases x_{kl} [respectively, decreases x_{kl}] by a positive amount $\gamma_{C_k}\delta$ for all $(k, l) \in C^+$ [respectively, for all $(k, l) \in C^-$], while leaving all node prices and all other arc flows unchanged. [Here, C_k denotes the portion of C from i to k, and γ_{C_k} is given by (4).]

[The effect of operation (c) is to decrease the surplus of node i by the amount $\delta/(1-\gamma_C)$ (respectively, 0) if $\gamma_C \neq 1$ (respectively, if $\gamma_C = 1$), while leaving the surplus of all other nodes unchanged.] A δ-flow push along an arc (respectively, a cycle) is said to be *saturating* if it changes the flow margin of the arc (respectively, one of the arcs of the cycle) from positive to zero. For a fixed $\epsilon > 0$ and $\theta \in (0, 1)$, and a given flow-price vector pair (x, p) satisfying ϵ-CS, an up iteration updates (x, p) as follows:

An up iteration at a node i with $g_i > 0$

Step 1: If i has no active outgoing arc and no inactive incoming arc, go to Step 3. Otherwise, choose any active arc (i, j) or inactive arc (j, i). If this arc belongs to some cycle C of $\mathcal{G}$ whose forward arcs are all active and whose backward arcs are all inactive *and* if no price rise nor saturating flow push has been performed since the last up iteration at node j, go to Step 2b. Otherwise, go to Step 2a.[1]

Step 2a: Perform a δ-flow push along the chosen arc, where

$$\delta = \begin{cases} \min\{\text{flow margin of } (i, j),\ g_i\} & \text{if } (i, j) \text{ is the chosen arc} \\ \min\left\{\text{flow margin of } (j, i),\ g_i/\gamma_{ji}\right\} & \text{if } (j, i) \text{ is the chosen arc.} \end{cases}$$

Exit.

Step 2b: Perform a δ-flow push along C, where

$$\delta = \begin{cases} \min_{(k,l)\in C}\{(\text{flow margin of } (k, l))/\gamma_{C_k}\} & \text{if } \gamma_C \geq 1 \\ \min\{\min_{(k,l)\in C}\left\{(\text{flow margin of } (k, l))/\gamma_{C_k}\right\},\ g_i/(1-\gamma_C)\} & \text{if } \gamma_C < 1, \end{cases}$$

and C_k denotes the portion of C from i to k. Exit.

Step 3: Perform a price rise on i and exit.

In general, finding the cycle C in Step 1 is expensive. However, such a cycle can be found without excessive overhead by using special implementations of the first phase of the method. More precisely, consider the following implementation, which aims at performing a flow push along a cycle for as long as no price rise or no saturating flow push is performed.

(a) Select any node i_0 with positive surplus. If no such node exists, terminate the first phase of the method. Else let $k := 0$ and go to (b).

(b) If $i := i_k$ has no active outgoing arc and no inactive incoming arc, perform a price rise on node i_k and go to (a). Otherwise, choose any active arc (i, j) or inactive arc (j, i). If $j = i_l$ for some $l < k$, go to (c). Else perform a δ-flow push along this arc as in Step 2a, and if this saturates the arc or if the surplus of j remains nonpositive, go to (a); else let $i_{k+1} := j$, increment k by 1 and go to (b).

(c) A cycle whose forward arcs are all active and whose backward arcs are all inactive is $C : i_l, i_{l+1}, ..., i_k, i_l$. Perform a δ-flow push along C as in Step 2b, and go to (a).

[1] In variants of the method, instead of always going to Step 2a at this point, we go to Step 2b if we had encountered a cycle C containing this arc and whose forward arcs are all active and whose backward arcs are all inactive; and go to Step 2a if no such cycle was encountered. These variants have similar termination properties and admit the same complexity bound as the stated method.

A *down iteration at a node i with $g_i < 0$* is defined analogously to an up iteration, but with "active" and "inactive" switched and with "increase" and "decrease" switched. In addition, "up", "rise", "g_i" are replaced by, respectively, "down", "drop", "$-g_i$".

There is also an important modification of the above implementation, called the *auction/sequential-shortest-path algorithm* (see [8], [9], [15], [29] for special cases of this algorithm that apply to the ordinary network case), in which we refrain from performing a flow push until we encounter a node j with negative surplus, at which time we push flow from i_0 along the path $i_0, i_1, ..., i_k, j$ towards j by an amount that either saturates an arc or zeroes out the surplus of i_0 or j. With this modification, it is possible to mix up and down iterations without affecting the termination or the complexity of the method. Finally, we note that, in contrast to the ordinary network case (see [14]), we need to consider not only flow pushes along arcs, but also flow pushes along cycles. The intuition for this is that a cycle C with $\gamma_C < 1$ is "flow absorbing" when flow is pushed along C and thus C acts like a node with negative surplus; similarly, a cycle C with $\gamma_C > 1$ is "flow generating" when flow is pushed along C and thus C acts like a node with positive surplus.

We make the following observations about the up iterations in the ϵ-relaxation method. (Analogous observations can be made for the down iterations.)

1. The iterations preserve ϵ-CS and the prices are monotonically nondecreasing. This is evident from the initialization and Step 3 of the up iteration.
2. Once the surplus of a node becomes nonnegative, it remains nonnegative for all subsequent up iterations. The reason is that a flow push at a node i cannot make the surplus of i negative (cf. Steps 2a, 2b), and cannot decrease the surplus of any other node.
3. If at some iteration a node has negative surplus, then its price must be equal to its initial price. This is a consequence of observation 2 above and the fact that price rises occur only on nodes with positive surplus.

Notice that the surpluses of all nodes are nonpositive at the beginning of the second phase. Moreover, the surpluses remain nonpositive throughout the second phase (since a down iteration does not change the surplus of any node from nonpositive to positive). Therefore, it follows that, at the end of the second phase, the surplus of all nodes are zero, i.e., the flow vector x is feasible.

3. Termination of the ϵ-relaxation method

To prove the termination of the ϵ-relaxation method of Sect. 2, we first have the following proposition which bounds from below the price rise/drop increments.

Proposition 2. *Each price rise (respectively, price drop) increment in the ϵ-relaxation method is at least $(1-\theta)\epsilon/\bar{\gamma}$, where $\bar{\gamma} := \max\{1, \max_{(i,j)\in\mathcal{A}} \gamma_{ij}\}$.*

Proof. We note that a price rise on a node i occurs only when it has no active outgoing arc nor inactive incoming arc. Thus for every arc $(i, j) \in \mathcal{A}$ we have $p_i - \gamma_{ij} p_j \leq f_{ij}^+(x_{ij}) + \theta\epsilon$, and for every arc $(j, i) \in \mathcal{A}$ we have $p_j - \gamma_{ji} p_i \geq f_{ji}^-(x_{ji}) - \theta\epsilon$. This

implies that we can increase p_i by an amount of at least $(1-\theta)\epsilon/\bar{\gamma}$ and still maintain ϵ-CS. A similar argument applies to a price drop.

□

Next, we have the following technical lemma, obtained by specializing the Conformal Realization theorem in [32, Chap. 10] to circulations in a certain augmented generalized network.

Lemma 1. *Consider any flow vector $y = \{y_{ij} \mid (i,j) \in \mathcal{A}\}$ and let $h_i := \sum_{\{j|(j,i)\in\mathcal{A}\}} \gamma_{ji} y_{ji} - \sum_{\{j|(i,j)\in\mathcal{A}\}} y_{ij}$ for all $i \in \mathcal{N}$. Then, for any $s \in \mathcal{N}$ with $h_s < 0$, there exist a $t \in \mathcal{N}$ and a simple path H in $\mathcal{G}$ from s to t that conforms to y; that is, $y_{ij} > 0$ for all $(i,j) \in H^+$ and $y_{ij} < 0$ for all $(i,j) \in H^-$. Moreover, either $h_t > 0$ or t belongs to a simple cycle C in $\mathcal{G}$ that conforms to y and satisfies $\gamma_C < 1$.*

Proof. Let $\mathcal{S} := \{i \in \mathcal{N} \mid h_i < 0\}$ and $\mathcal{T} := \{i \in \mathcal{N} \mid h_i > 0\}$. Define the augmented directed graph $\mathcal{G}' = (\mathcal{N}', \mathcal{A}')$, where $\mathcal{N}' := \mathcal{N} \cup \{0\}$, $\mathcal{A}' := \mathcal{A} \cup (\{0\} \times \mathcal{S}) \cup (\mathcal{T} \times \{0\})$, and define the scalars:

$$\gamma'_{ij} := \begin{cases} \gamma_{ij} & \text{if } (i,j) \in \mathcal{A} \\ -\sum_{k\in\mathcal{S}} h_k / \sum_{k\in\mathcal{T}} h_k & \text{if } i \in \mathcal{T},\ j = 0\,, \\ 1 & \text{if } i = 0,\ j \in \mathcal{S} \end{cases}$$

$$y'_{ij} := \begin{cases} y_{ij} & \text{if } (i,j) \in \mathcal{A} \\ h_i & \text{if } i \in \mathcal{T},\ j = 0\,. \\ -h_j & \text{if } i = 0,\ j \in \mathcal{S} \end{cases}$$

Then, the vector $y' := \{y'_{ij} \mid (i,j) \in \mathcal{A}'\}$ is a circulation for the generalized network $\mathcal{G}'$ with arc gains γ'_{ij}, $(i,j) \in \mathcal{A}'$. For any $s \in \mathcal{S}$, since $y'_{0s} > 0$, we have from the Conformal Realization theorem [32, p. 456] and the characterization of elementary primal supports for generalized networks [32, p. 463] that there exist in $\mathcal{G}'$ either (i) a simple cycle C with $\gamma_C = 1$ or (ii) two disjoint simple cycles C_1 and C_2, with $\gamma_{C_1} > 1$ and $\gamma_{C_2} < 1$, and a simple path H from a node in C_1 to a node in C_2 or (iii) two simple cycles C_1 and C_2, with $\gamma_{C_1} > 1$ and $\gamma_{C_2} < 1$, that have a (single) joint portion or meet in exactly one node. Moreover, in case (i), C conforms to y' and uses $(0,s)$; in case (ii), C_1, C_2, H conform to y' and one of them uses $(0,s)$; in case (iii), C_1, C_2 conform to y' and one of them uses $(0,s)$. It can be verified that, in all cases, there exists in $\mathcal{G}$ a simple path that conforms to y and goes from s to either a node in $\mathcal{T}$ or a node in a simple cycle that conforms to y and whose gain is less than 1.

□

For a path P in $\mathcal{G}$, define

$$\Gamma_P := \sum_{i\in P} \gamma_{P_i}, \tag{7}$$

where, for each node $i \in P$, P_i denotes the portion of the path P from the starting node of P to i. By using Prop. 2 and Lemma 1, we obtain the following proposition which bounds the total number of price rises in terms of Γ_P and other network parameters. The proof is patterned in part after the proofs for the linear cost ordinary network case [7], [12], and for the convex cost ordinary network case [14], [29].

Proposition 3. *Let K be any nonnegative scalar such that the initial price vector p^0 for the ϵ-relaxation method satisfies $K\epsilon$-CS together with some feasible flow vector x^0. Then, the ϵ-relaxation method performs at most $(K+1)\Gamma/(1-\theta)$ price rises on each node and at most $(1+\Gamma)(K+1)\Gamma/(1-\theta)$ price drops on each node, where*

$$\Gamma := \bar{\gamma} \cdot \max_{H\text{: simple path}} \left\{ \gamma_H \left(\max_{C\text{: simple cycle with } \gamma_C < 1} \frac{\Gamma_C}{1-\gamma_C} \right) + \Gamma_H \right\},$$

and $\gamma_H, \gamma_C, \Gamma_C, \Gamma_H$ are given by Eqs. (4), (7) and $\bar{\gamma} := \max\{1, \max_{(i,j)\in\mathcal{A}} \gamma_{ij}\}$.

Proof. Consider the pair (x, p) at the beginning of an up iteration in the ϵ-relaxation method. Since the flow vector x^0 is feasible, we have upon applying Lemma 1 to $y := x^0 - x$ (for which $h_i = -g_i$) that, for each node s with $g_s > 0$, there exist a node t and a simple path H in $\mathcal{G}$ from s to t that conforms to $x^0 - x$, i.e.,

$$x_{ij} < x^0_{ij}, \qquad \forall\, (i,j) \in H^+, \tag{8a}$$

$$x_{ij} > x^0_{ij}, \qquad \forall\, (i,j) \in H^-. \tag{8b}$$

Moreover, either $g_t < 0$ or t belongs to a simple cycle C in $\mathcal{G}$ with $\gamma_C < 1$ and conforming to $x^0 - x$, i.e.,

$$x_{ij} < x^0_{ij}, \qquad \forall\, (i,j) \in C^+, \tag{9a}$$

$$x_{ij} > x^0_{ij}, \qquad \forall\, (i,j) \in C^-. \tag{9b}$$

From Eqs. (8a) and (9b), and the convexity of the functions f_{ij} for all $(i,j) \in \mathcal{A}$, we have

$$f^+_{ij}(x_{ij}) \le f^-_{ij}(x^0_{ij}), \qquad \forall\, (i,j) \in H^+, \tag{10a}$$

$$f^-_{ij}(x_{ij}) \ge f^+_{ij}(x^0_{ij}), \qquad \forall\, (i,j) \in H^-. \tag{10b}$$

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

$$p_i - \gamma_{ij} p_j \in \left[f^-_{ij}(x_{ij}) - \epsilon,\ f^+_{ij}(x_{ij}) + \epsilon \right], \qquad \forall\, (i,j) \in \mathcal{A}. \tag{11a}$$

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

$$p^0_i - \gamma_{ij} p^0_j \in \left[f^-_{ij}\left(x^0_{ij}\right) - K\epsilon,\ f^+_{ij}\left(x^0_{ij}\right) + K\epsilon \right], \qquad \forall\, (i,j) \in \mathcal{A}. \tag{11b}$$

Combining Eqs. (10a)–(11b), we obtain

$$p_i - \gamma_{ij} p_j \le p^0_i - \gamma_{ij} p^0_j + (K+1)\epsilon, \qquad \forall\, (i,j) \in H^+,$$

$$p_i - \gamma_{ij} p_j \ge p^0_i - \gamma_{ij} p^0_j - (K+1)\epsilon, \qquad \forall\, (i,j) \in H^-.$$

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

$$p_s - \gamma_H p_t \le p^0_s - \gamma_H p^0_t + (K+1)\left(\sum_{i\in H} \gamma_{H_i} \right)\epsilon, \tag{12}$$

where H_i denotes the portion of the path H from s to $i \in H$. We observed earlier that if a node has negative surplus at some time, then its price is unchanged from the beginning of the method until that time. Thus if $g_t < 0$, then

$$p_t = p_t^0. \tag{13}$$

On the other hand, if t belongs to some simple cycle C satisfying Eqs. (9a) and (9b), a similar argument shows that

$$p_i - \gamma_{ij} p_j \le p_i^0 - \gamma_{ij} p_j^0 + (K+1)\epsilon, \qquad \forall\, (i,j) \in C^+,$$

$$p_i - \gamma_{ij} p_j \ge p_i^0 - \gamma_{ij} p_j^0 - (K+1)\epsilon, \qquad \forall\, (i,j) \in C^-,$$

which when applied for all arcs of the cycle C yields

$$p_t - \gamma_C p_t \le p_t^0 - \gamma_C p_t^0 + (K+1)\Bigg(\sum_{i \in C} \gamma_{C_i}\Bigg)\epsilon,$$

where C_i denotes the portion of the cycle C from t to $i \in C$. Using $\gamma_C < 1$, we obtain

$$p_t \le p_t^0 + (K+1)\frac{\left(\sum_{i \in C} \gamma_{C_i}\right)}{1-\gamma_C}\epsilon. \tag{14}$$

Therefore, if $g_t < 0$, then Eqs. (12) and (13) yield

$$p_s \le p_s^0 + (K+1)\Bigg(\sum_{i \in H} \gamma_{H_i}\Bigg)\epsilon \le p_s^0 + (K+1)\Gamma\epsilon,$$

and if t belongs to some simple cycle C satisfying (9a) and (9b), then Eqs. (12) and (14) yield

$$p_s \le p_s^0 + (K+1)\gamma_H \frac{\left(\sum_{i \in C} \gamma_{C_i}\right)}{1-\gamma_C}\epsilon + (K+1)\Bigg(\sum_{i \in H} \gamma_{H_i}\Bigg)\epsilon \le p_s^0 + (K+1)\Gamma\epsilon/\bar{\gamma}, \tag{15}$$

where the second inequality follows from the definition of Γ. Since only nodes with positive surplus can increase their prices and, by Prop. 2, each price rise increment is at least $(1-\theta)\epsilon/\bar{\gamma}$, we conclude from Eq. (15) that the total number of price rises that can be performed for node s is at most $(K+1)\Gamma/(1-\theta)$.

Now we estimate the number of price drops on each node. Let $p^1 = \{p_i^1 \mid i \in \mathcal{N}\}$ denote the price vector at the end of the first phase of the ϵ-relaxation method. From Eq. (15) we see that $p_i^0 \le p_i^1 \le p_i^0 + (K+1)\Gamma\epsilon/\bar{\gamma}$ for all $i \in \mathcal{N}$, so that

$$p_i^0 - \gamma_{ij} p_j^0 - (K+1)\Gamma\epsilon \le p_i^1 - \gamma_{ij} p_j^1 \le p_i^0 - \gamma_{ij} p_j^0 + (K+1)\Gamma\epsilon, \quad \forall (i,j) \in \mathcal{A}.$$

Since (x^0, p^0) satisfies $K\epsilon$-CS, this implies that (x^0, p^1) satisfies $(K+(K+1)\Gamma)\epsilon$-CS. Since, by Prop. 2, each price drop increment is at least $(1-\theta)\epsilon/\bar{\gamma}$, an argument analogous to the one above, but with p^0 replaced by p^1 and with Lemma 1 applied to $y := x - x^0$ instead, yields that the number of price drops that can be performed on each node is at most $(K+(K+1)\Gamma+1)\Gamma/(1-\theta) = (1+\Gamma)(K+1)\Gamma/(1-\theta)$.

□

The preceding proposition shows that the bound on the number of price changes is independent of the cost functions, but depends only on the arc gains and the scalar K^0 given by

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

which is the minimum multiplicity of ϵ by which 0-CS is violated by the initial price together with some feasible flow vector. This result will be used later to prove a particularly favorable complexity bound for the ϵ-relaxation method. Note that K^0 is well defined for any p^0 because, for all K sufficiently large, $K\epsilon$-CS is satisfied by p^0 and any feasible flow vector x.

We will now derive a bound on the number of flow pushes required by the ϵ-relaxation method. By our choice of δ (see Steps 2a and 2b of the up iteration), a nonsaturating flow push always exhausts (i.e., sets to zero) the surplus of the node being iterated on. In what follows, for any $\epsilon > 0$ and $\theta \in (0, 1)$, and any flow-price vector pair (x, p) satisfying ϵ-CS, we define the arc set

$$\mathcal{A}^* := \{(i, j) \mid (i, j) \in \mathcal{A} \text{ is active}\} \cup \{(j, i) \mid (i, j) \in \mathcal{A} \text{ is inactive}\}$$

and the *admissible graph* $\mathcal{G}^* := (\mathcal{N}, \mathcal{A}^*)$. By analyzing changes in the admissible graph, we have the following proposition, which bounds the number of flow pushes between successive price rises in the first phase.

Proposition 4. *The number of flow pushes along arcs (respectively, cycles) between two successive price rises (not necessarily at the same node) performed by the ϵ-relaxation method is at most N^2A (respectively, NA).*

Proof. Consider the flow pushes between two successive price rises. First, we observe that the number of arcs in the admissible graph $\mathcal{G}^*$ is nonincreasing after a flow push, and is strictly decreasing after a saturating flow push. Thus, the number of saturating flow pushes is at most A.

Consider the flow pushes between changes in the admissible graph $\mathcal{G}^*$ (which must all be nonsaturating). Each flow push along an arc, being nonsaturating, does not increase the number of nodes with positive surplus, while each flow push along a cycle, being nonsaturating, decreases this number by one. (For a flow push from node i along a cycle C to be nonsaturating, we must have $\gamma_C < 1$ and the surplus of i must be set to zero, while the surplus of all other nodes must be left unchanged.) Thus, there can be at most N flow pushes along cycles. By the logic of an up iteration, a flow push along an arc (oriented in the direction of flow change) belonging to a forward cycle of $\mathcal{G}^*$ is performed only if the ending node of this arc has not been iterated upon in an earlier flow push. Thus, there can be at most N flow pushes along arcs belonging to forward cycles of $\mathcal{G}^*$. There remains to estimate the number of flow pushes along arcs (oriented in the direction of flow change) not belonging to any forward cycle of $\mathcal{G}^*$. These arcs form an acyclic directed graph, say $\mathcal{G}^{**}$. Moreover, a flow push can repeat at an arc, say (i, j), in $\mathcal{G}^{**}$ only if there is an arc in $\mathcal{G}^{**}$ pointing into i along which a flow push was performed earlier. Since any forward path in $\mathcal{G}^{**}$ has length at most $N - 1$ (so the surplus of a node can be propagated to successors along the arcs of $\mathcal{G}^{**}$ by at most $N - 1$

flow pushes) and originally at most N nodes have positive surplus, this implies that the total number of flow pushes along arcs in $\mathcal{G}^{**}$ is at most $(N-1)N$.

Thus, between two successive price rises, the admissible graph can change at most A times and, between successive changes in the admissible graph, there are at most N flow pushes along cycles and at most N^2 flow pushes along arcs.

□

It follows from Props. 3 and 4 that the first phase of the ϵ-relaxation method terminates after at most $O(NK\Gamma)$ price rises, and at most $O(N^3AK\Gamma)$ flow pushes along arcs, and at most $O(N^2AK\Gamma)$ flow pushes along cycles, where K is any nonnegative scalar such that the initial price vector satisfies $K\epsilon$-CS together with some feasible flow vector. A similar result can be shown for the second phase, though the bounds increase by a multiplicative factor of Γ (cf. the estimates of Prop. 3). In Sect. 4, a specific implementation of the method with sharper complexity bound will be presented. Upon termination of the ϵ-relaxation method, we have that the flow-price vector pair (x, p) satisfies ϵ-CS and that x is feasible since the surplus of all nodes is zero. The following result from [34, Props. 7 and 8] shows that this flow vector and price vector are within a factor that is essentially proportional to ϵ of being optimal for, respectively, the primal problem (P) and the dual problem (D).

Proposition 5. *For each $\epsilon > 0$, let $x(\epsilon)$ and $p(\epsilon)$ denote any flow and price vector pair satisfying ϵ-CS with $x(\epsilon)$ feasible and let $\xi(\epsilon)$ denote any flow vector satisfying 0-CS with $p(\epsilon)$ [$\xi(\epsilon)$ need not be feasible]. Then*

$$0 \le f\big(x(\epsilon)\big) + q\big(p(\epsilon)\big) \le \epsilon \sum_{(i,j)\in\mathcal{A}} \big|x_{ij}(\epsilon) - \xi_{ij}(\epsilon)\big|. \tag{16}$$

Furthermore, $f\big(x(\epsilon)\big) + q\big(p(\epsilon)\big) \to 0$ as $\epsilon \to 0$.

Proposition 5 does not give an a priori estimate of how small ϵ has to be in order to achieve a certain degree of approximate optimality, as measured by the duality gap. However, in the common case where finiteness of the arc cost functions f_{ij} imply lower and upper bounds on the arc flows:

$$-\infty < b_{ij} := \inf_{\xi}\{\xi \mid f_{ij}(\xi) < \infty\} \le \sup_{\xi}\{\xi \mid f_{ij}(\xi) < \infty\} =: c_{ij} < \infty,$$

as in the linear cost case of (2), the right-hand side of (16) is bounded above by $\epsilon \sum_{(i,j)\in\mathcal{A}} |c_{ij} - b_{ij}|$, which gives an a priori estimate of the duality gap between $x(\epsilon)$ and $p(\epsilon)$.

4. A sweep implementation of the ϵ-relaxation method

We say that a strongly connected component (abbreviated as SCC) of the admissible graph $\mathcal{G}^*$ is a *predecessor* of another SCC of $\mathcal{G}^*$ if there is a forward path in $\mathcal{G}^*$ from a node in the first SCC to a node in the second SCC (and we say that the second SCC is a successor of the first SCC). [An SCC of $\mathcal{G}^*$ is a subgraph $\mathcal{G}'$ of $\mathcal{G}^*$ with the properties

that (i) there is a forward path in $\mathcal{G}'$ from every node in $\mathcal{G}'$ to every other node in $\mathcal{G}'$ and (ii) $\mathcal{G}'$ is not properly contained in any other subgraph of $\mathcal{G}^*$ with property (i).] Observe that flow is pushed towards the successors of a SCC and that flow cannot be pushed from a SCC to any of its predecessor SCC. We say that an SCC is *positive* if it contains at least one node with positive surplus; otherwise the SCC is called *nonpositive*.

The *sweep implementation* of the ϵ-relaxation method, introduced in [7] and further analyzed in [12], [17], and [10] for the linear cost ordinary network case, selects a node for an up iteration as follows (an analogous rule holds for selecting a node for a down iteration): Let $\mathcal{G}^*$ denote the current admissible graph. Choose any positive SCC of $\mathcal{G}^*$ whose predecessor SCC are all non-positive. In Step 1 of an up iteration, select i to be any node in the chosen SCC with positive surplus. Also, in Step 1, always go to Step 2b when (i, j) belongs to some forward cycle C of $\mathcal{G}^*$.

For the sweep implementation, we can improve on Prop. 4 as shown in the proposition below. The intuition for this improvement is that an up iteration at a node i having a positive predecessor SCC may be wasteful since its surplus may be set to zero through a flow push and become positive again by a flow push at a node in the predecessor SCC. The sweep implementation avoids performing such an up iteration. Our proof follows the corresponding line of analysis for the ordinary network case in [14], [29].

Proposition 6. *For the sweep implementation of the ϵ-relaxation method, the number of nonsaturating flow pushes between two successive price rises (not necessarily at the same node) is at most $N + N\tilde{A}$, where $\tilde{A}$ denotes the maximum number of arcs contained in any SCC of the admissible graph.*

Proof. Consider the flow pushes between two successive price rises. Each nonsaturating flow push at a node i changes the surplus of i to zero. Since i, by selection, does not have any predecessor node in a different SCC with positive surplus, the surplus of i will remain at zero until the SCC containing i changes (due to the removal of a saturated arc from this SCC). Thus, the number of nonsaturating flow pushes between changes in the SCC of the admissible graph is at most N. Since at least one arc is removed from an SCC of the admissible graph each time the latter changes, the number of changes in the SCC of the admissible graph is at most $\tilde{A}$.

□

By using Props. 3 and 6, we obtain the following improved complexity bound for the sweep implementation of the ϵ-relaxation method.

Proposition 7. *Let K be any nonnegative scalar such that the initial price vector for the sweep implementation of the ϵ-relaxation method satisfies $K\epsilon$-CS together with some feasible flow vector. Then, the method requires $O(K\Gamma N)$ price rises and $O(K\Gamma N^2(1 + \tilde{A}))$ flow pushes in the first phase and $O(K\Gamma^2 N)$ price drops and $O(K\Gamma^2 N^2(1 + \tilde{A}))$ flow pushes in the second phase.*

Proof. It suffices to analyze the first phase of the ϵ-relaxation method, which involves up iterations only. According to Prop. 3, there are $O(K\Gamma)$ price rises on each node, so the number of price rises is $O(K\Gamma N)$. Furthermore, whenever a flow push is saturating, it takes at least one price rise on one of the end nodes before the flow on that arc can be

changed again. Thus the total number of saturating flow pushes is $O(K\Gamma A)$. Finally, by Prop. 6, the number of nonsaturating flow pushes between successive price rises is at most $N(1+\tilde{A})$, so the total number of nonsaturating flow pushes is $O(K\Gamma N^2(1+\tilde{A}))$. Since $A \leq N^2$, the result for the first phase follows. An analogous analysis applies for the second phase.

□

In the ordinary network case where $\Gamma = O(N)$ and the second phase is not needed, it was shown in [15, Prop. 5] (also see [14, Prop. 4] for the case of $\theta = 1/2$) that if the initial admissible graph is acyclic, then the admissible graph will remain acyclic at all iterations of the ϵ-relaxation method, in which case $\tilde{A} = 0$ (each SCC always comprises an individual node and hence contains no arc at all) and, since $\Gamma = O(N)$, Prop. 7 would yield a complexity bound of $O(KN^2)$ price changes and $O(KN^3)$ flow pushes. In general, we have $\tilde{A} \leq A$. We can further improve the complexity of the ϵ-relaxation method by using *ϵ-scaling* as is described in [14]: initially set $\epsilon = \epsilon^0$ for some ϵ^0, run the ϵ-relaxation method until it terminates with some (x^0, p^0), then decrease ϵ by a fixed fraction (e.g., a half) and rerun the ϵ-relaxation method with p^0 as the starting price vector, and so on, till ϵ reaches some target value $\overline{\epsilon}$. Assuming that ϵ^0 is chosen sufficiently large so that the initial price vector satisfies ϵ^0-CS together with some feasible flow vector, this yields an improved complexity bound in which K is replaced by $\ln(\epsilon^0/\overline{\epsilon})$, that is, a bound of $O(\ln(\epsilon^0/\overline{\epsilon})\Gamma N)$ on the number of price changes and a bound of $O(\ln(\epsilon^0/\overline{\epsilon})\Gamma N^2(1+\tilde{A}))$ on the number of flow pushes in the first phase. A similar bound, though higher by a multiplicative factor of Γ, holds for the second phase. To our knowledge, this is the first complexity result for the generalized network flow problem with nonlinear cost function.

5. Computational experimentation

We have developed an experimental Fortran code implementing the ϵ-relaxation method for the case of problems (P) and (D) with quadratic arc cost functions. The code, named QE-RELAXG ("Q" stands for quadratic and "G" stands for generalized), implements the version of the ϵ-relaxation method whereby the cycle C in Step 1 of each iteration is found using the technique described in Sect. 2 and ϵ is adjusted using the ϵ-scaling technique of Sect. 4. In this section, we report on our computational experience with the code on some test problems. (We have also implemented a version of the auction/sequential-shortest-path algorithm mentioned in Sect. 3. This work is still very preliminary, although the initial results are encouraging.)

First we describe the test problems. In these problems, the cost function of each arc (i, j) is quadratic of the form

$$f_{ij}(x_{ij}) = \begin{cases} a_{ij}x_{ij} + b_{ij}x_{ij}^2 & \text{if } 0 \leq x_{ij} \leq c_{ij}, \\ \infty & \text{otherwise,} \end{cases}$$

for some $a_{ij} \in \Re$ and $b_{ij} \in [0, \infty)$ and $c_{ij} \in [0, \infty)$. We call a_{ij}, b_{ij}, and c_{ij} the linear cost coefficient, the quadratic cost coefficient, and the capacity, respectively, of arc (i, j). The test problems are created using the public-domain Fortran problem

generator NETGENG [24], which is an extension of the popular generator NETGEN that creates linear-cost generalized assignment/transportation/transshipment problems having a certain random structure. (See Tables 1 and 2 for the NETGENG parameters that we used to create the test problems.) As NETGENG creates only linear-cost problems, we modified the created problems as in [13] and [14] so that, for a user-specified fraction (taken to be a half in our tests) of the arcs, the quadratic cost coefficient is randomly generated from a user-specified range of consecutive integers (taken to be $\{1, 2, 3, 4, 5\}$ in our tests) according to a uniform distribution, and, for the remaining arcs, the quadratic cost coefficient is set to a user-specified value b. When $b = 0$, the cost function f is mixed linear/quadratic. When $b > 0$, the cost function f is strictly convex quadratic and, as $b \to 0$, the dual problem (D) becomes increasingly more ill-conditioned in the traditional sense of unconstrained nonlinear programming.

Table 1. Solution times (in seconds) for QE-RELAXG. Problems are created using NETGENG with *SEED* = 13502460, No. sources = No. sinks = (No. nodes)/2, Supply = 500 × (No. nodes), and Linear Cost Coeff. ∈ [1, 1000]. Quadratic Cost Coeff. is randomly generated from {1, 2, 3, 4, 5} for half of the arcs and is equal to the value b shown in column six for the remaining arcs

		No. nodes	No. arcs	Gain range	Quad. coeff. b	Optimal cost	QE-RELAXG Soln. times
Symmetric	No.	400	2000	.5–1.5	0	153309205	1.21
Capacitated	nodes	400	6000	.5–1.5	0	65375508	2.12
Transhipment	fixed	400	8000	.5–1.5	0	62536366	3.14
cap∈ [500, 1000]		400	10000	.5–1.5	0	59481317	8.24
Symmetric	No.	400	2000	.5–1.5	0	84626835	.92
Uncapacitated	nodes	400	6000	.5–1.5	0	38725118	1.94
Transportation	fixed	400	8000	.5–1.5	0	38421522	5.52
		400	10000	.5–1.5	0	28760650	8.89
Symmetric	No.	400	7000	.5–1.5	0	65894687	4.10
Capacitated	arcs	600	7000	.5–1.5	0	123247677	5.37
Transhipment	fixed	800	7000	.5–1.5	0	220246167	4.31
cap∈ [500, 1000]		1200	7000	.5–1.5	0	404739277	5.42
Symmetric	No.	400	7000	.5–1.5	0	34137013	5.29
Uncapacitated	arcs	600	7000	.5–1.5	0	77835120	3.64
Transportation	fixed	800	7000	.5–1.5	0	121479983	4.52
		1200	7000	.5–1.5	0	245152297	3.95

Next, we describe the implementation details for QE-RELAXG. This code uses $\theta = 1/2$ and is initialized with $p_i = 0$ for all nodes i and with $\epsilon = \frac{1}{5}\max_{(i,j)\in\mathcal{A}}\{a_{ij}+2b_{ij}\hat{x}_{ij}\}$, where $\hat{x}_{ij} = \min\{c_{ij}, \max_{i\in\mathcal{N}}|s_i|\}$. The initial flow vector x is then chosen to satisfy ϵ-CS with the initial price vector. The code terminates when the node surpluses and the duality gap $f(x) - q(p)$ are below 10^{-5} times, respectively, $\max_{i\in\mathcal{N}}|s_i|$ and $|f(x)|$. A further enhancement, adapted from the ϵ-relaxation codes for the ordinary network case [14], is the use of a surplus threshold whereby only nodes whose surplus exceeds this threshold in magnitude are selected for up/down iterations. This threshold is initially set to $\frac{1}{4}\max_{i\in\mathcal{N}}|s_i|$ and is decreased at the rate $\frac{1}{4}$ each time ϵ is decreased, until this threshold reaches $10^{-5}\cdot\max_{i\in\mathcal{N}}|s_i|$. (We also experimented with a stricter termination criterion where 10^{-5} is replaced by 10^{-8}. The code did not change its qualitative behavior, though the solution times increased by a factor between 1 and 4.) After some

experimentation, we settled on $1/5$ as the fraction used in ϵ-scaling. Otherwise, we have not fine-tuned the parameters in the code.

Table 2. Solution times (in seconds) for QE-RELAXG. Problems are created using NETGENG with *SEED* = 13502460, No. sources = No. sinks = (No. nodes)/2, Supply = 500 × (No. nodes), and Linear Cost Coeff. ∈ [1, 1000]. Quadratic Cost Coeff. is randomly generated from {1, 2, 3, 4, 5} for half of the arcs and is equal to the value *b* shown in column six for the remaining arcs

		No. nodes	No. arcs	Gain range	Quad. coeff. *b*	Optimal cost	QE-RELAXG Soln. times
Symmetric	No.	1200	7000	1.0–1.0	0	394520079	4.37
Capacitated	nodes,	1200	7000	.9–1.1	0	399896552	52.03
Transhipment	arcs	1200	7000	.5–1.5	0	404739277	5.42
cap∈ [500, 1000]	fixed	1200	7000	.2–4.0	0	482554996	4.38
Symmetric	No.	400	10000	1.0–1.0	0	27910212	4.32
Uncapacitated	nodes,	400	10000	.9–1.1	0	29439130	12.42
Transportation	arcs	400	10000	.5–1.5	0	245152297	3.95
	fixed	400	10000	.2–4.0	0	38990680	2.14
Symmetric	No.	1200	7000	.5–1.5	1	727711866	7.47
Capacitated	nodes,	1200	7000	.5–1.5	.01	409727972	3.90
Transhipment	arcs	1200	7000	.5–1.5	.0001	404789495	4.14
cap∈ [500, 1000]	fixed	1200	7000	.5–1.5	0	404739277	5.42
Symmetric	No.	400	10000	.5–1.5	1	474088846	4.77
Uncapacitated	nodes,	400	10000	.5–1.5	.01	249634201	2.57
Transportation	arcs	400	10000	.5–1.5	.0001	245197678	3.21
	fixed	400	10000	.5–1.5	0	245152297	3.95

Now we describe our computational tests and experience. Our tests were designed to study the performance of the ϵ-relaxation method relative to the earlier relaxation methods, and the dependence of this performance on network topology, arc gains, and problem ill-conditioning. We experimented with three sets of test problems generated using NETGENG as described above: the first set comprises mixed linear/quadratic cost problems with varying topology and arc capacities (Table 1); the second set comprises mixed linear/quadratic cost problems with varying ranges of arc gains (top half of Table 2); the third set comprises strictly convex quadratic cost problems with varying degrees of ill-conditioning (bottom half of Table 2). The solution time for QE-RELAXG on these problems are shown in the last column of the tables. These times were obtained by compiling and running QE-RELAXG on a Sun Ultra-1 workstation and under the Solaris operating system, Version 2.5.1. The -O option was invoked when compiling. From the solution times we see that the performance of QE-RELAXG is not significantly affected by changes in the number of nodes (see bottom half of Table 1) or problem ill-conditioning (see bottom half of Table 2). A possible explanation for the latter is that, by its use of ϵ-CS, quadratic cost coefficients that are small are effectively treated as zeros by the ϵ-relaxation method. Thus, in contrast to the relaxation methods of [13] and [34], the ϵ-relaxation method is well suited to handle ill-conditioned problems (also see [14] for analogous observations in the ordinary network case). On the other hand, the performance of QE-RELAXG is adversely affected by increases in the number of arcs (see top half of Table 1) and, more significantly, by the presence of non-unity arc gains near 1 (see top half of Table 2). The reason for the latter is not well understood, though it seems to be related to the way in which NETGENG

generates problems with arc gains near 1, namely, the generated problems tend to be infeasible or nearly infeasible, with many flow generating cycles needed to meet the flow demands and many flow aborbing cycles needed to absorb the flow supplies. For these nearly infeasible problems, a large number of price rises/drops are required to direct flow from flow generating cycles to sinks and from sources to flow aborbing cycles. (The value of Γ does not appear to be a factor since, according to the proof of Prop. 3, Γ affects complexity of the ϵ-relaxation method only through upper and lower bounds on the prices generated by the method. In our tests, these bounds did not change significantly, nor did the average price increment.) Also, we observed that, on all runs, the computation was dominated by price rises/drops and flow pushes along arcs, with less than 0.1 percent of flow pushes being made along cycles (and the cycles were typically short). In other words, QE-RELAXG behaves much like its counterpart for the ordinary network case where flow pushes are made only along arcs.

To assess the efficiency of our coding, we compared QE-RELAXG with two specialized Fortran codes: the ϵ-relaxation code NE-RELAXF from [14] for mixed linear/quadratic cost ordinary network flow problems, and the primal-simplex code NET2 from [19] for linear cost generalized network flow problems (also see [33]). In our tests, we found QE-RELAXG to be slower than NE-RELAXF, though not beyond a factor of 1.5 in solution time. QE-RELAXG was typically slower than NET2, by a factor between 1.2 to 4. NET2 was also adversely affected by the presence of non-unity arc gains near 1, as well as by wider gain range. And on two problems with gain range of .2–4.0, NET2 was unable to find a feasible solution even with double-precision arithmetic (see Table 3). Thus, although QE-RELAXG is not as fast as the specialized codes, as might be expected, it can serve as a good all-around code, since it can handle arcs that involve gains as well as nonlinear cost.

Table 3. Solution times (in seconds) for QE-RELAXG, NET2 and MINOSL on a Dec Alpha–linear cost case. Problem parameters are as in Table 2, with the linear cost coefficients a_{ij}, capacities c_{ij}, gains γ_{ij}, and supplies s_i truncated to 3 decimal places. The quadratic cost coefficients b_{ij} are set to zero. [[1]NET2 exited with unsatisfied demand of -6.73 at node 507 and 12.65 at node 1170. [2]NET2 exited with unsatisfied demand of -2.13 at node 14 and 1.47 at node 304.]

	No. nodes	No. arcs	Gain range	Optimal cost	QE-RELAXG Soln. times	NET2 Soln. times	MINOSL Soln. times
Symmetric	1200	7000	1.0–1.0	206232923	2.26	1.00	25.19
Capacitated	1200	7000	.9–1.1	206357872	7.63	3.71	25.06
Transhipment	1200	7000	.5–1.5	205287384	2.74	3.23	18.19
cap$\in$ [500, 1000]	1200	7000	.2–4.0	227083686	1.42	infeas[1]	11.67
Symmetric	400	10000	1.0–1.0	15745545	1.31	.07	2.37
Uncapacitated	400	10000	.9–1.1	15778149	4.54	0.34	2.80
Transportation	400	10000	.5–1.5	16052702	1.35	0.30	2.38
	400	10000	.2–4.0	20672078	0.36	infeas[2]	1.61

While this paper was under review, the referees suggested that we compare QE-RELAXG with popular linear programming (LP) or nonlinear programming (NLP) codes such as MINOS or CPLEX. Although we did not have CPLEX, we did have MINOS 5.4 by Murtagh and Saunders [27], which we ran on a common set of test

problems as QE-RELAXG. The test results with MINOSL, a version MINOS adapted for LP, are tabulated in Table 3. The test results with MINOS, the NLP code, are tabulated in Table 4. The tests were run on a Dec Alpha as the Sun Ultra-1 was not easily accessible to us at this time. The test problems were generated using NETGENG with the same settings as in Table 2, although, to simplify input into MINOS, we truncated the real data a_{ij}, c_{ij}, γ_{ij}, and s_i to 3 decimal places. Due to the large problem size, some care was needed to set parameters in MINOS so that it has sufficient workspace. After some experimentation, we settled on nwcore $= 15000000$ and Superbasics limit $= \min\{3000, A+1\}$. The objective function and gradient were defined in the subroutine funobj, with the quadratic cost coefficients b_{ij} stored in a common block inside funobj. We consulted with Michael Saunders to ensure that these MINOS settings and data inputs were reasonable. To make a fair comparison of QE-RELAXG with MINOS, we changed 10^{-5} in the termination criterion for QE-RELAXG to 10^{-8} and 10^{-7}, respectively, for the first eight and the last eight problems in Table 4. This ensured that the accuracy of the solutions generated by QE-RELAXG, as measured by cost, is similar to that generated by MINOS. The solution times for MINOSL and MINOS, as reported under "Time for solving problem", do not include the problem input time. As can be seen from Tables 3 and 4, QE-RELAXG is significantly faster than MINOSL and MINOS on most of the test problems. The exceptions are the linear cost uncapacitated transportation problems in Table 3 and the two quadratic cost problems in Table 4 with non-unity arc gains near 1, for which QE-RELAXG is at most twice as fast as MINOSL or MINOS (and slower than MINOSL on one problem). Changing the termination criterion for MINOSL and MINOS does not appear to improve their solution times appreciably. Workspace allocation is also an issue for MINOS on the quadratic cost problems. For example, setting Superbasics limit too low (e.g., 2000) caused early exit, while setting it too high (e.g., 10000) resulted in workspace requirement exceeding the available disk space.

Table 4. Solution times (in seconds) for QE-RELAXG and MINOS on a Dec Alpha. Problem parameters are as in Table 2, with the linear cost coefficients a_{ij}, capacities c_{ij}, gains γ_{ij}, and supplies s_i truncated to 3 decimal places. [[1]The costs obtained by the two codes differ from the optimal cost only slightly in the least significant digit.]

	No. nodes	No. arcs	Gain range	Quad. coeff. b	Optimal cost[1]	QE-RELAXG Soln. times	MINOS Soln. times
Symmetric	1200	7000	1.0–1.0	0	399436018	3.85	212.11
Capacitated	1200	7000	.9–1.1	0	399896578	112.95	233.69
Transhipment	1200	7000	.5–1.5	0	404739276	6.08	223.72
cap$\in$ [500, 1000]	1200	7000	.2–4.0	0	482554994	5.83	104.72
Symmetric	400	10000	1.0–1.0	0	29359255	6.73	82.99
Uncapacitated	400	10000	.9–1.1	0	29439130	37.95	87.42
Transportation	400	10000	.5–1.5	0	28760650	17.99	92.45
	400	10000	.2–4.0	0	38990680	5.44	95.83
Symmetric	1200	7000	.5–1.5	1	727712182	19.35	945.69
Capacitated	1200	7000	.5–1.5	.01	409727971	6.79	254.10
Transhipment	1200	7000	.5–1.5	.0001	404789494	6.78	228.46
cap$\in$ [500, 1000]	1200	7000	.5–1.5	0	404739276	6.08	223.72
Symmetric	400	10000	.5–1.5	1	84149783	5.35	5185.96
Uncapacitated	400	10000	.5–1.5	.01	30373637	3.99	108.98
Transportation	400	10000	.5–1.5	.0001	28777090	12.99	103.53
	400	10000	.5–1.5	0	28760650	17.99	92.45

References

1. Adler, I., Cosares, S. (1991): A strongly polynomial algorithm for a special class of linear programs. Oper. Res. **39**, 955–960
2. Ahlfeld, D.P., Mulvey, J.M., Dembo, R.S., Zenios, S.A. (1987): Nonlinear programming on generalized networks. ACM Trans. Math. Software **13**, 350–367
3. Ahuja, R.K., Magnanti, T.L., Orlin, J.B. (1993): Network Flows. Prentice-Hall, Englewood Cliffs
4. Ahuja, R.K., Magnanti, T.L., Orlin, J.B., Reddy, M.R. (1995): Applications of network optimization. In: Network Models, Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser, G.L., eds., North-Holland, Amsterdam, 1–83
5. Beraldi, P., Guerriero, F. (1997): A parallel asynchronous implementation of the ϵ-relaxation method for the linear minimum cost flow problem. Parallel Comput. **23**, 1021–1044
6. Beraldi, P., Guerriero, F., Musmanno, R. (1997): Efficient parallel algorithms for the minimum cost flow problem. J. Optim. Theory Appl. **95**, 501–530
7. Bertsekas, D.P. (1986): Distributed asynchronous relaxation methods for linear network flow problems. Laboratory for Information and Decision Systems Report P–1606, M.I.T., Cambridge, November 1986
8. Bertsekas, D.P. (1992): An auction sequential shortest path algorithm for the minimum cost network flow problem. Laboratory for Information and Decision Systems Report P-2146, M.I.T., Cambridge, MA
9. Bertsekas, D.P. (1998): Network Optimization: Continuous and Discrete Models. Athena Scientific, Belmont, MA
10. Bertsekas, D.P., Castañon, D.A. (1993): A generic auction algorithm for the minimum cost network flow problem. Comput. Optim. Appl. **2**, 229–260
11. Bertsekas, D.P., Castañon, D., Eckstein, J., Zenios, S.A. (1995): Parallel computing in network optimization. In: Network Models, Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser, G.L., eds., North-Holland, Amsterdam, pp. 331–399
12. Bertsekas, D.P., Eckstein, J. (1988): Dual coordinate step methods for linear network flow problems. Math. Program. **42**, 203–243
13. Bertsekas, D.P., Hosein, P.A., Tseng, P. (1987): Relaxation methods for network flow problems with convex arc costs. SIAM J. Control Optim. **25**, 1219–1243
14. Bertsekas, D.P., Polymenakos, L.C., Tseng, P. (1997): An ϵ-relaxation method for separable convex cost network flow problems. SIAM J. Optim. **7**, 853–870
15. Bertsekas, D.P., Polymenakos, L.C., Tseng, P. (1997): ϵ-relaxation and auction methods for separable convex cost network flow problems. In: Network Optimization, Lecture Notes in Economics and Mathematical Systems 450, Pardalos, P.M., Hearn, D.W., Hager, W.W., eds., Springer-Verlag, Berlin, 103–126
16. Bertsekas, D.P., Tseng, P. (1988): Relaxation methods for minimum cost ordinary and generalized network flow problems. Oper. Res. **36**, 93–114
17. Bertsekas, D.P., Tsitsiklis, J.N. (1989): Parallel and Distributed Computation: Numerical Methods. Prentice-Hall, Englewood Cliffs
18. Cohen, E., Megiddo, N. (1994): New algorithms for generalized network flows. Math. Program. **64**, 325–336
19. Currin, D.C. (1983): A comparative evaluation of algorithms for generalized network problems. NRIMS Technical Report TWISK 289, Pretoria, South Africa
20. Dembo, R.S., Mulvey, J.M., Zenios, S.A. (1989): Large-scale nonlinear network models and their application. Oper. Res. **37**, 353–372
21. De Leone, R., Meyer, R.R., Zakarian, A. (1999): A partitioned ϵ-relaxation algorithm for separable convex network flow problems. Comput. Optim. Appl. **12**, 107–126
22. Glover, F., Hultz, J., Klingman, D., Stutz, J. (1978): Generalized networks: a fundamental computer-based planning tool. Manage. Sci. **24**, 1209–1220
23. Goldberg, A.V., Plotkin, S.A., Tardos, É. (1991): Combinatorial algorithms for generalized circulation problem. Math. Oper. Res. **16**, 351–381
24. Hultz, J. (1976): Algorithms and applications for generalized networks. Unpublished Dissertation, University of Texas, Austin, TX
25. Jewell, W.S. (1962): Optimal flow through networks with gains. Oper. Res. **10**, 476–499
26. Karzanov, A.V., McCormick, S.T. (1997): Polynomial methods for separable convex optimization in unimodular linear spaces with applications. SIAM J. Comput. **26**, 1245–1275
27. Murtagh, B.A., Saunders, M.A. (1993): MINOS 5.4 user's guide. Technical Report SOl 83-20R, Department of Operations Research, Stanford University, Stanford, CA, March 1993
28. Murty, K.G. (1992): Network Programming. Prentice-Hall, Englewood Cliffs
29. Polymenakos, L.C. (1995): ϵ-relaxation and auction algorithms for the convex cost network flow problem. Electrical Engineering and Computer Science Department Ph.D. Thesis, M.I.T., Cambridge, MA

30. Radzik, T. (1998): Faster algorithms for the generalized network flow problem. Math. Oper. Res. **23**, 69–100
31. Rockafellar, R.T. (1970): Convex Analysis, Princeton University Press, Princeton
32. Rockafellar, R.T. (1998): Network Flows and Monotropic Programming. Wiley-Interscience, New York, NY, 1984; republished by Athena Scientific, Belmont, MA
33. Tseng, P., Bertsekas, D.P. (1987): Relaxation methods for linear programs. Math. Oper. Res. **12**, 569–596
34. Tseng, P., Bertsekas, D.P. (1990): Relaxation methods for monotropic programs. Math. Program. **46**, 127–151
35. Vaidya, P.M. (1989): Speeding-up linear programming using fast matrix multiplication. In: Proceedings of the 30th IEEE Annual Symposium on Foundations of Computer Science, IEEE Press, New York, 332–337