PATH PROBLEMS IN SKEW-SYMMETRIC GRAPHS

ANDREW V. GOLDBERG

COMPUTER SCIENCE DEPARTMENT
STANFORD UNIVERSITY
STANFORD, CA 94305
GOLDBERG@CS.STANFORD.EDU

AND

ALEXANDER V. KARZANOV

INSTITUTE FOR SYSTEMS ANALYSIS
9, PROSPECT 60 LET OKTYABRYA
117312 MOSCOW, RUSSIA
KARZANOV@CS.VNIISI.MSK.SU

October 1993

Abstract. We study path problems in skew-symmetric graphs. These problems generalize the standard graph reachability and shortest paths problems. We develop duality theory for the skew-symmetric problems and use it to design efficient algorithms for the problems. The algorithms presented are competitive with the fastest algorithms for the standard problems.

The first author was supported in part by ONR Young Investigator Award N00014-91-J-1855, NSF Presidential Young Investigator Grant CCR-8858097 with matching funds from AT&T, DEC, and 3M, and a grant from Powell Foundation.
This research was done while the second author was visiting Stanford University Computer Science Department and supported by the above mentioned NSF and Powell Foundation Grants.

1. Introduction

A digraph $G = (V, E)$ is *skew-symmetric* if there is a permutation σ on V such that for every $v \in V$, $\sigma(v) \neq v$ and $\sigma(\sigma(v)) = v$, and $(v, w) \in E$ if and only if $(\sigma(w), \sigma(v)) \in E$. In this paper, we shall usually use the term *symmetric* instead of skew-symmetric. We say that a node $\sigma(v)$ is symmetric to v, and an arc $(\sigma(v), \sigma(w))$ is symmetric to (v, w). We extend σ to $V \cup E$ by defining $\sigma(v, w) = (\sigma(w), \sigma(v))$. Unless otherwise explicitly stated, we assume that the input graph G has no multiple arcs except in the following case. Suppose $v' = \sigma(v)$ and $(v, v') \in E$. Then there are exactly two copies of (v, v'), and they are, obviously, symmetric to each other.

Skew-symmetric graphs come up in a natural way when certain undirected objects, such as matchings, are reduced to directed objects. These can be used to get insight into combinatorial structure of the underlying problem and to design algorithms for it. The skew-symmetric graphs come up in other situations as well, *e.g.* in [21, 22] a maximum flow problem in a skew-symmetric graph appears in conjunction with a certain minimum-cost multicommodity flow problem.

A *regular path (r-path)* is a path in G that does not contain a pair of symmetric arcs. Given a *length function* $\ell : E \Rightarrow \mathbf{R}$, the length $\ell(P)$ of a path P is the sum of the lengths of the arcs of the path. We assume that the length function is symmetric: $\ell(v, w) = \ell(\sigma(v, w))$. Suppose we are given two symmetric nodes s and s'. The *r-reachability problem (RRP)* is to find an r-path from s to s' or a proof that there is none. The *shortest r-paths problem (SRPP)* is to find the shortest r-path from s to s' or a proof that there is none. Note that the latter can happen if s' is not reachable from s via an r-path or if there is an r-path from s to s' containing a negative length cycle.

The goal of this paper is to establish a solvability criterion for the RRP and an optimality criterion for the shortest r-paths problem, and to develop fast algorithms for these problems. Computationally, these r-path problems are at least as hard as the standard path and shortest paths problems. This is because given a graph G with two distinguished nodes x and y, we can make a disjoint copy G' of G with the opposite orientation of each arc and work with the skew-symmetric graph H formed by adding to $G \cup G'$ new nodes s and s' and arcs $(s, x), (y, s'), (s, y'), (x', s')$. Then any path from s to s' in H is regular and corresponds to a path from x to y in G. Note also that the problem on paths with forbidden pairs of arcs in an arbitrary digraph, which is related to the RRP, is NP-complete [14].

The RRP can be reduced to a certain matching problem in a way similar to that described in [29] for the node-regular path analog of RRP. Such a reduction can be used to describe a solvability criterion for the RRP via the classical theorems on perfect matchings due to Berge [4] and Tutte [27] (see also [23]), as well as to solve the RRP using a maximum matching algorithm, e.g. the classical algorithm in [8]. Similarly, the optimality criterion for the SRPP can be obtained from the weighted matching theory developed by Edmonds in [7] (see also [23]), and the problem can be solved in polynomial time using a minimum-cost matching algorithm. These reductions, however, considerably increase the graph size (if the input graph has n nodes and m arcs, the reductions produce a graph that has $m+2$ nodes and may have $\Omega(n^3)$ arcs), so these reductions are expensive from the algorithmic point of view. The relationship between problems on skew-symmetric graphs and matching problems is discussed in Section 5. The section also gives applications of RRP and SRPP to certain problems on matchings.

Our proofs of the solvability and optimality criteria are in terms of the input graph itself, which is simpler and more enlightening than the proofs obtained by translating the corresponding theorems on matchings into the skew-symmetric framework. These theorems demonstrate nice structural properties of the r-path problems and lead to natural approaches for solving them.

Running time bounds of the algorithms we develop in this paper are within a $\log n$ factor from those for the standard path problems. The algorithm for RRP runs in $O(m)$ (linear) time, where $n = |V|$ and $m = |E|$.[1] We call the SRPP with nonnegative arc lengths *NSRPP*. Our NSRPP algorithm runs in $O(m \log n)$ time or in $O(m\sqrt{\log C})$ time (the latter bound assumes that costs are integers in the interval $[0, \dots, C]$). The corresponding bounds in the standard paths case, achieved by the implementations of Dijkstra's algorithm [6] described in [1] and [11], are $O(m + n \log n)$ and $O(m+n\sqrt{\log C})$, respectively. Our SRPP algorithm runs in $O(nm(\min(\log n, \sqrt{\log C})))$ time. The corresponding bound in the standard paths case is $O(nm)$ [3, 10, 25].

In the sequel [19] to this paper we extend the structural and algorithmic results developed here to more general problems on skew-symmetric graphs, such as the maximum integral symmetric flow problem, the minimum-cost integral symmetric circulation problem, and their unit capacity variants. Being interesting in their own right, these problems also bridge flows and matchings. Using the algorithms developed in the present paper, we design algorithms for the skew-symmetric flow problems which are as efficient as the fastest known algorithms for the corresponding maximum flow and minimum-cost maximum flow problems.

The classical matching problems, such as maximum matchings, minimum-cost matchings, b-matchings and their capacitated versions, and even the minimum-cost bidirected flows [9], can be reduced to skew-symmetric flow problems without increasing the problem size significantly. As a result, the algorithms of [19] solve the matching problems as efficiently as the fastest matching algorithms [2, 13, 16, 17, 24, 28]. At the same time, the results we present are more general and more uniform, and seem to be simpler.

[1]To simplify the presentation, we assume that $m \geq n - 1 \geq 2$.

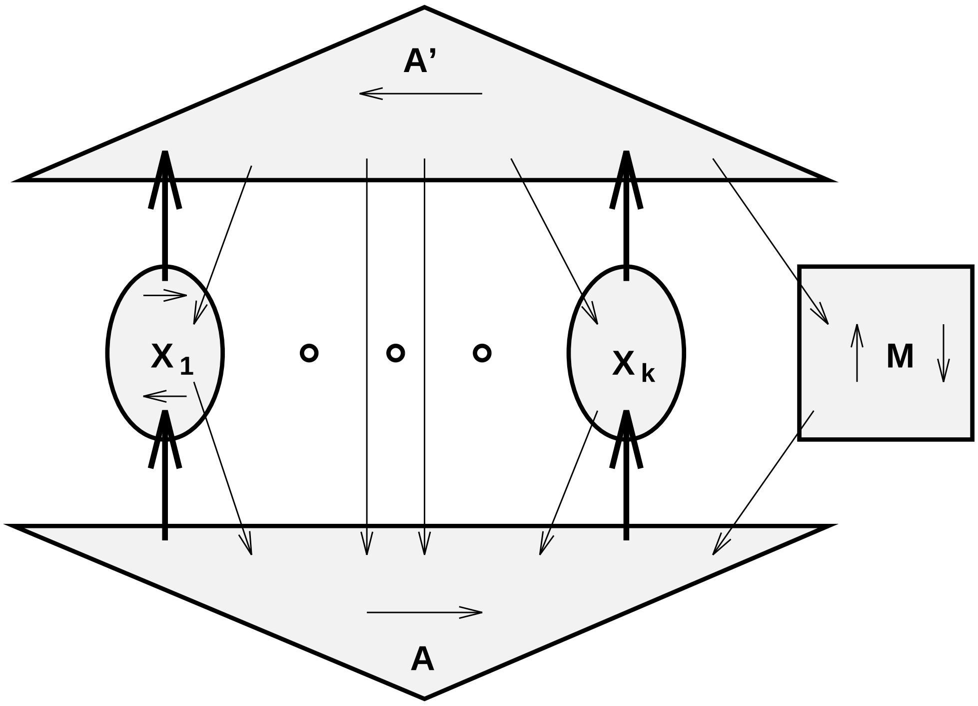

FIGURE 1. *A barrier.*

2. THE R-REACHABILITY PROBLEM

2.1. Barriers. Barriers provide infeasibility certificates for RRP. We say that

$$\mathcal{B} = (A; X_1, \ldots, X_k)$$

is a *barrier* if the following conditions hold.

(B1) $A, X_1, \ldots, X_k$ are pairwise disjoint subsets of V.

(B2) $s \in A$.

(B3) For $A' = \sigma(A)$, $A \cap A' = \emptyset$.

(B4) For $i = 1, \ldots, k$, X_i is symmetric, *i.e.*, $\sigma(X_i) = X_i$.

(B5) For $i = 1, \ldots, k$, there is a unique arc, a_i, from A to X_i.

(B6) For $i, j = 1, \ldots, k$ and $i \neq j$, no arc connects X_i to X_j.

(B7) For $M = V - (A \cup X_1 \cup \ldots \cup X_k)$ and $i = 1, \ldots, k$, no arc connects X_i to M.

(B8) No arc connects A and $A' \cup M$.

(Note that arcs from A' to A, from X_i to A, and from M to A are possible.) Figure 1 illustrates the definition.

If a node x of a graph G is reachable from s by an r-path, we say that x is *reachable* in G. Given a graph $G = (V, E)$ and a subset of nodes $X \subseteq V$, let $\langle X \rangle$ denote the subgraph of G induced by

X.

Suppose that no r-path from s to s' exists. Let Z be the set of reachable nodes in G. Define

$$Z' = \sigma(Z), \quad A = Z - Z', \quad A' = Z' - Z, \quad X = Z \cap Z', \quad \text{and } M = V - (Z \cup Z').$$

Let $K_1, \dots, K_k$ be the weakly connected components of $\langle X \rangle$, and let X_i (E_i) be the node set (arc set) of K_i. We call $\mathcal{B} = (A; X_1, \dots, X_k)$ the *canonical barrier*.

Lemma 2.1. *The canonical barrier is indeed a barrier.*

Proof. Let $\mathcal{B} = (A; X_1, \dots, X_k)$ be the canonical barrier. Properties (B1)–(B3) and (B6)–(B8) follow immediately from the definition of $\mathcal{B}$. We prove the remaining properties by induction on $|E|$. The base case $|E| = 0$ is trivial.

Let $e_1 = (v_1, w_1), \dots, e_h = (v_h, w_h)$ be the arcs from A to X, and let $v'_i = \sigma(v_i)$ and $w'_i = \sigma(w'_i)$, $1, \dots, h$. We may assume that $h \geq 2$ (otherwise (B4) and (B5) are trivial). Obviously, there exist distinct $i, j \in \{1, \dots, h\}$ such that at least one r-path from s to v_i does not meet the arc e_j; let for definiteness $i = 1$ and $j = h$.

Let $\overline{G} = (v, \overline{E})$ be the graph obtained by deleting e_h and e'_h from G. Clearly $\overline{G}$ has no r-path from s to s'. Let $\overline{Z}, \overline{Z}', \overline{A}, \overline{A}', \overline{X}, \overline{M}$ be corresponding sets in the definition of the canonical barrier for $\overline{G}$. The fact that every r-path in $\overline{G}$ is an r-path in G implies that $\overline{Z} \subseteq Z$, whence $\overline{Z}' \subseteq Z'$ and $\overline{X} \subseteq X$. Let $\overline{K}_i = (\overline{X}_i, \overline{E}_i)$, $i = 1, \dots, p$, be the weakly connected components of the subgraph $\langle \overline{X} \rangle$ of $\overline{G}$. By induction $\mathcal{B} = (\overline{A}; \overline{X}_1, \dots, \overline{X}_p)$ is a barrier. Next, from the above property of e_1 and e_h we observe that $v_1, v_h \in \overline{Z}$, whence $v_1, v_h \in \overline{A}$ (as $v_1, v_h \in A$ and $\overline{X} \subseteq X$). Also $w_1 \in \overline{Z}$. Three cases are possible.

Case 1. $w_h \in \overline{M}$. Let Q be the graph obtained by adding to $\langle \overline{M} \rangle$ the nodes v_h, v'_h and arcs e_h, e'_h. Then Q is skew-symmetric and contains less arcs than G does (as $h \geq 2$ and Q does not contain e_1). Furthermore, e_h is the only arc in Q that leaves v_h, therefore Q has no r-path from v_h to v'_h. By induction the canonical barrier $(B; Y_1, \dots, Y_q)$ for Q, v_h, v'_h is a barrier. Clearly $\widehat{\mathcal{B}} = (\overline{A} \cup B; \overline{X}_1, \dots, \overline{X}_p, Y_1, \dots, Y_p)$ is a barrier for G. Moreover, each node in $\widehat{Z} = \overline{A} \cup \overline{X} \cup B \cup Y$ is obviously reachable in G, where $Y = Y_1 \cup \ldots \cup Y_q$. Hence, $\widehat{Z} = Z$. The facts that $(\overline{A} \cup B) \cap \sigma(\overline{A} \cup B) = \emptyset$ and that all sets $\overline{X}_1, \dots, \overline{X}_p, Y_1, \dots, Y_p$ are symmetric imply that $\widehat{\mathcal{B}}$ coincides with $\mathcal{B}$.

Case 2. $w_h \in \overline{X} \cup \overline{A}'$. Then, by symmetry, $w'_h \in \overline{X} \cup \overline{A}$. Hence, there is an r-path from s to w'_h in $\overline{G}$. Adding to it the arc e'_h we obtain an r-path from s to v'_h in G. But v'_h is not reachable in G; a contradiction.

Case 3. $w_h \in \overline{A}$. Then $w'_h \in \overline{A}'$. Since $w'_h \in Z$, there is an r-path P from s to w'_h in G. Note that P passes through e_h (otherwise v'_h would be reachable in G). Let P' be the part of P from

w_h to w'_h. Then P' is an r-path in $\overline{G}$ going from $\overline{A}$ to $\overline{A}'$. This contradicts the fact that $\overline{\mathcal{B}}$ is a barrier in $\overline{G}$. ■

We are now ready to prove the duality theorem for RRP.

Theorem 2.2. *There is an r-path from s to s' if and only if there is no barrier.*

Proof. The fact that if there is a barrier then there is no r-path from s to s' is immediate from the definition of a barrier. If there is no r-path from s to s', then the canonical barrier is well-defined, and Lemma 2.1 completes the proof. ■

As mentioned in the introduction, Theorem 2.2 can also be obtained using matching theory.

2.2. Buds and Trimming Operation. A *bud* is a triple $\tau = (V_\tau, E_\tau, e_\tau = (v, w))$ such that

(i) $E_\tau \subset E$ is symmetric: $\sigma(E_\tau) = E_\tau$.
(ii) $e_\tau \in E$.
(iii) $V_\tau \subset V$ is the set of nodes incident to E_τ. Note that V_τ is symmetric.
(iv) $w \in V_\tau$.
(v) $v \notin V_\tau$.
(vi) $s \notin V_\tau$.
(vii) For every node $x \in V_\tau$ there is an r-path from w to x in (V_τ, E_τ) (and therefore an r-path from x to $\sigma(w)$).
(viii) There is an r-path P from s to v such that P is node-disjoint from V_τ.

A particular case of a bud arises when the graph (V_τ, E_τ) is the union of an r-path Γ from w to $\sigma(w')$ and the symmetric path $\sigma(\Gamma)$; such a bud is called *elementary.*

The node w is called the *base node* of the bud and the arc (v, w) is called the *base arc.* The node $\sigma(w)$ is called the *anti-base node* of the bud and the arc $\sigma(v, w)$ is called the *anti-base arc.* Given a collection of buds, a bud τ from this collection is called *maximal* if V_τ is maximal (with respect to inclusion).

Buds play important role in the algorithms described in this paper. They are similar, in a certain sense, to blossoms that come up in the classical matching algorithm [8].

Given a symmetric graph $G = (V, E)$ with a bud $(V_\tau, E_\tau, e_\tau = (v, w))$, let $v' = \sigma(v)$, $w' = \sigma(w)$, and $e'_\tau = \sigma(e_\tau)$. The *trimming operation* transforms G into $\overline{G}$ with the node set

$$\overline{V} = V - (V_\tau - \{w, w'\})$$

and arc set $\overline{E}$ constructed as follows.

(1) Each arc $a = (x, y) \in E$ such that either $x, y \in V - V_\tau$, or $a = e_\tau$, or $a = e'_\tau$ remains in $\overline{E}$.

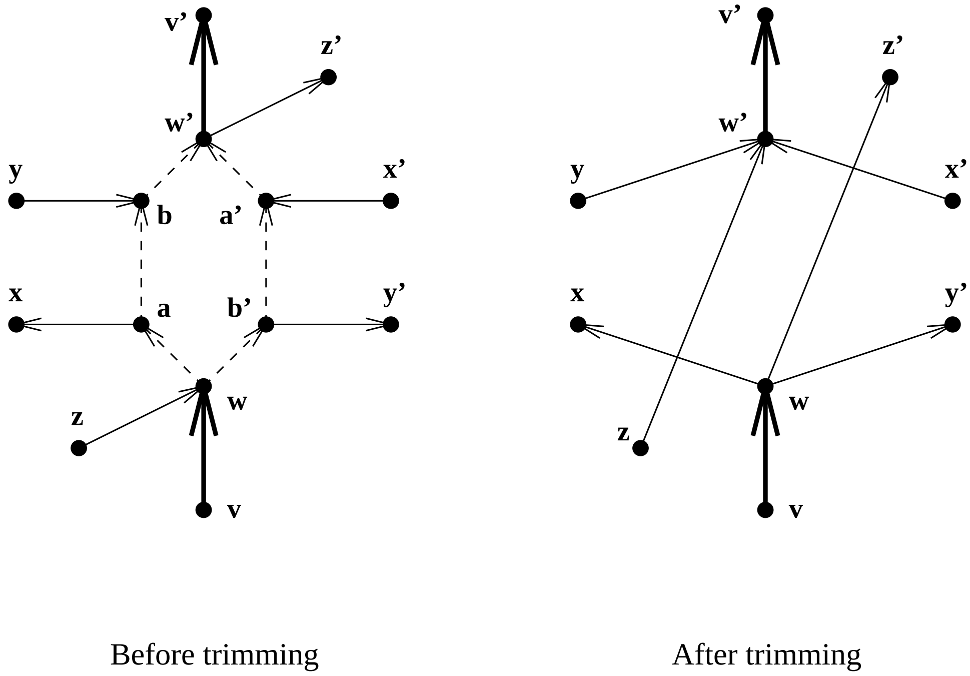

FIGURE 2. *Bud trimming example.*

(2) Each arc $a = (x, y) \in E$ such that $x \in V_\tau$, $y \in V - V_\tau$, and $a \neq e'_\tau$ is replaced by an arc from w to y.

(3) Each arc $a = (x, y) \in E$ such that $x \in V - V_\tau$, $y \in V_\tau$, and $a \neq e_\tau$ is replaced by an arc from x to w'.

The sets of arcs of $\overline{G}$ defined by (1)–(3) are denoted by E_1, E_2, and E_3, respectively; thus $\{E_1, E_2, E_3\}$ is a partition of $\overline{E}$. For convenience we identify the corresponding arcs in G and $\overline{G}$. See Figure 2 for an example of bud trimming.

The next lemma follows from the definition of $\overline{G}$.

Lemma 2.3. *The graph $\overline{G}$ is symmetric.*

Theorem 2.4. *There is an r-path from s to s' in G if and only if there is an r-path from s to s' in $\overline{G}$.*

Proof. Suppose that s' is not reachable in $\overline{G}$. Apply Theorem 2.2 and consider a barrier $\mathcal{B} = (A; X_1, \ldots, X_k)$ for $\overline{G}$. By the definition of a bud, there is an r-path P in G from s to w with the last arc e_τ. Since P is also an r-path in $\overline{G}$, either (i) $w \in X_i$ for some i, or (ii) $w \in A$. In case (i), add $V_\tau - \{w, \sigma(w)\}$ to X_i. In case (ii), let I be the set of indices $i \in \{1, \ldots, k\}$ such that the arc a_i from A to X_i has its tail at w. Remove w from A and replace the sets X_i, $i \in I$, in $\mathcal{B}$ by the only set X' that is the union of these X_i's and V_τ. It is easy to see that in both cases we obtain a barrier for G.

Next we prove the other direction of the theorem. Suppose that s' is not reachable in G. Consider a barrier $\mathcal{B} = (A; X_1, \ldots, X_k)$ for G. The r-reachability from s of all nodes in V_τ and the symmetry of V_τ imply that $V_\tau \subset X_i$ for some i. Then

$$(A, X_1, \ldots, X_{i-1}, X_i - (V_\tau - \{w, w'\}), X_{i+1}, \ldots, X_k)$$

is a barrier in $\overline{G}$. ∎

Like shrinking blossoms in matching algorithms, bud trimming lowers the size of the graph and gives an efficient approach to solving the r-reachability problem, refined in the algorithm that we describe in further sections. In essence, the algorithm is based on the following two lemmas, the first one is obvious, while the second one is easily proved by arguing as in the proof of the part "only if" in Theorem 2.4.

Lemma 2.5. *Let P be an r-path from s to s' in $\overline{G}$.*

(i) *If P meets neither e_τ nor e'_τ, then P is an r-path in G.*

(ii) *If P meets e_τ and u is the arc in P following e_τ, then, in G, P is split into two paths, P_1 and P_2. The former is an initial portion of P with the last arc e_τ and the latter is the final portion of P with the first arc u. Moreover, the concatenation of P_1, Q, and P_2 is an r-path from s to s' in G, where Q is an r-path in (V_τ, E_τ) from w to the tail of u.*

(If P meets e'_τ the statement is "symmetric" to that in (ii).)

For a node x in $\overline{G}$ let $\Omega(x) = \Omega_\tau(x)$ denote the set V_τ if x is the base or the anti-base of a trimmed bud τ, and the node x otherwise. We say that $\Omega(x)$ is the *preimage* of x in G. For $X \subseteq \overline{V}$ let $\Omega(X)$ stand for the union of preimages of elements of X.

Lemma 2.6. *Let $\overline{Y} \subset \overline{V}$ be a set such that $s' \notin \overline{Y}$ and $\overline{Y}$ contains all reachable nodes in $\overline{G}$. Let $Y = \Omega(\overline{Y})$. Then $s' \notin Y$ and Y contains all reachable nodes in G. If, in addition, each element of $\overline{Y}$ is reachable in $\overline{G}$ then the same is true for Y and G.*

In the algorithm we use the following corollaries from Lemmas 2.5 and 2.6. Suppose we have constructed a sequence $G = G_0, G_1, \ldots, G_r$ of graphs, where $G_i = (V_i, E_i)$ is obtained from G_{i-1} by trimming a bud $\tau_i = (V_{\tau_i}, E_{\tau_i}, e_{\tau_i} = (v_i, w_i))$ in it. Suppose also that for each i and $x \in V_{\tau_i}$ we have a procedure of finding an r-path from w_i to x in (V_{τ_i}, E_{τ_i}), (*e.g.* in the case when each τ_i is an elementary bud in G_{i-1}). If we succeed in finding an r-path $\widetilde{P}$ from s to s' in G_r then Lemma 2.5 provides an explicit method to construct an r-path P from s to s' in G, by going along the sequence backward and finding a corresponding path in G_i from the previous path in G_{i+1}.

On the other hand, suppose we discovered a set $\widetilde{Y}$ of nodes of G_r such that $s \in \widetilde{Y}$, $\widetilde{Y} \cap \sigma(\widetilde{Y}) = \emptyset$, and none of the arcs of G_r goes from $\widetilde{Y}$ to $V_r - \widetilde{Y}$. Then Lemma 2.6 shows that G has no r-path from s to s' and that the set $Y = \Omega_{\tau_1}(\ldots\Omega_{\tau_r}(\widetilde{Y}))$ contains all reachable nodes in G. Moreover, if each element of $\widetilde{Y}$ is reachable in G_r then Y is exactly the set Z of reachable nodes in G, and therefore, we have an explicit method to construct a canonical barrier in G.

In what follows we often need to distinguish the base node w (or $w' = \sigma(w)$) in the graphs G and $\overline{G}$. To do this, we denote w (w') in $\overline{G}$ by w_τ (respectively, w'_τ). Also, considering sequences $G = G_0, \ldots, G_r$ of graphs and $\tau_1, \ldots, \tau_r$ of buds as above, we denote by $\mathcal{V}_{\tau_i}$ the set of preimages in G of the node w_{τ_i} (or w_{τ_i}) of G_i, *i.e.*, $\mathcal{V}_{\tau_i} = \Omega_{\tau_1}(\ldots\Omega_{\tau_i}(w_{\tau_i}))$. We use notation $(\mathcal{V}_{\tau_i}, \mathcal{E}_{\tau_i}, e_{\tau_i})$ for the "preimage" in G of the bud τ_i; here $\mathcal{E}_{\tau_i}$ is the union of sets E_{τ_j} among all $j \le i$ such that $\mathcal{V}_{\tau_j} \subseteq \mathcal{V}_{\tau_i}$. It is easy to see the following result.

Lemma 2.7.

(i) *For $1 \le i, j \le r$, either $\mathcal{V}_{\tau_i} \cap \mathcal{V}_{\tau_j} = \emptyset$ or $\mathcal{V}_{\tau_i} \subseteq \mathcal{V}_{\tau_j}$; in particular, the sets $\mathcal{V}_{\tau_i}$, $i = 1, \ldots, r$, form a nested family.*

(ii) *The sets E_{τ_i}, $i = 1, \ldots, r$, are pairwise disjoint; in particular, $\sum(|E_{\tau_i}| : i = 1, \ldots, r) \le m$.*

(iii) *Each $(\mathcal{V}_{\tau_i}, \mathcal{E}_{\tau_i}, e_{\tau_i})$ is a bud in G.*

We usually keep the same notation τ_i for the bud $(\mathcal{V}_{\tau_i}, \mathcal{E}_{\tau_i}, e_{\tau_i})$ in G. A bud τ_i is called *maximal* if $\mathcal{V}_{\tau_i} \cap \mathcal{V}_{\tau_j} = \emptyset$ for $j = i+1, \ldots, r$.

2.3. High-Level Algorithm Description. Lemmas 2.5 and 2.6 suggest the following algorithm for RRP. The algorithm maintains a set A of nodes reachable from s by r-paths, with the r-paths represented by a spanning tree $T \subset E$ of A rooted at s. The algorithm also maintains $A' = \sigma(A)$ and $T' = \sigma(T)$. By symmetry, from every node $v' \in A'$ there is an r-path to s' in T'. The invariant $A \cap A' = \emptyset$ always holds. Initially $A = \{s\}$ and $T = \emptyset$.

At each step the algorithm scans an arc (v, w) with $v \in A$ and $w \notin A$. If $w \notin A'$, then w is added to A, (v, w) to T, $\sigma(w)$ to A', and $\sigma(v, w)$ to T'. Now suppose $w \in A'$ and let P be the concatenation of the s to v path P_1 in T, (v, w), and the w to s' path P_2 in T'. If P is regular, then the algorithm terminates and returns P (in Section 2.6 we explain how to efficiently transform P into a regular s to s' path in the original graph). If P is not regular, then there is an arc (x, y) on P_1 such that $\sigma(x, y)$ is on P_2. Let (x, y) be the closest to v arc on P_1 that has this property. Let E_τ be the set of arcs on P between y and $\sigma(y)$, and their symmetric arcs. Let V_τ be the set of nodes incident to E_τ in the current graph. Then $\tau = (V_\tau, E_\tau, (x, y))$ is an elementary bud. The algorithm trims τ, accordingly updating the current graph, the sets A, A', and the trees T, T'.

If there is no arc (v, w) with $v \in A$ and $w \notin A$, the algorithm concludes that s' is not reachable, in view of Lemma 2.6. If needed, we can find, in linear time, the canonical barrier in the original G.

Correctness of this algorithm follows from Lemma 2.3 and Theorem 2.4.

Note that at each step either A grows or at least two arcs are deleted from the graph by the trimming operation. This implies a polynomial bound on the algorithm complexity. To achieve a linear bound, we implement the bud trimming operation efficiently and pick the next arc to scan quickly.

2.4. Data Structures and Implementation of Bud Trimming. Efficient implementation of bud trimmings is crucial to the algorithm performance. We discuss the bud trimming implementation issues below. In this paper, we assume the adjacency list representation of the input graph in which each node has a list of outgoing arcs arranged in a doubly linked list.

Remark. For every node v of the current graph we maintain the list of outgoing arcs and do not maintain the list of incoming arcs. We can easily access the incoming arcs, however, because by symmetry every arc (u, v) is symmetric to the arc $(\sigma(v), \sigma(u))$ which is on the adjacency list of $\sigma(v)$.

A naive implementation of trimming a bud $\tau = (V_\tau, E_\tau, (v, w))$ is to delete nodes in V_τ and arcs adjacent to these nodes, add new nodes $w_\tau, {w_\tau}'$, and add arcs adjacent to these nodes as prescribed by the definition of bud trimming. By (ii) in Lemma 2.7, there can be $O(m)$ bud trimmings, and at most m arcs need to be processed during a bud trimming. It can be easily shown that processing of arcs dominates the running time bound and the naive implementation runs in $O(m^2)$ time.

To do better, an implementation of bud trimming should avoid looking at every arc involved in the trimming. This can be achieved by using the original arc lists. The arc list for w_τ is formed by deleting the arcs of E_τ and the arc $\sigma(v, w)$ from the appropriate lists, and then by concatenating the resulting arc lists of nodes in V_τ.

Next we address the following issue: if we are examining an arc (x, y), what is its head node in the current graph? If y does not belong to a bud, then the head node is y. If does, consider the maximal bud τ that contains y. If (x, y) is the base arc of τ, then the head node is the base of τ and otherwise, the upper base of τ.

For this implementation we need to maintain the sets $\mathcal{V}_\tau$ of the current maximal buds τ. These sets are disjoint, by Lemma 2.7. The set operations that we need are MAKE-SET(x), UNION(x, y), and FIND-SET(x). These operations create a single element set containing x, form the union of the sets containing x and y, and return the name of the set containing x, respectively. The operations are supported by the well-known *disjoint set union* data structure (see e.g. [5]). In the fastest known implementation of the disjoint set union data structure [26], a sequence of k operations on elements from a universe of size n takes $O(k\alpha(k,n)+n)$ time, where α is a functional inverse of Ackermann's function. As we shall see later, the set union operations we need fall into a special class and can be done in time $O(k+n)$.

Next we describe an efficient implementation of bud trimming. Since the sets we need to maintain are symmetric, it is convenient to deal with a *representative* $r(x)=r(x')$ to every pair x, x' of symmetric nodes. Initially each representative is included in a single element set. The sets are modified only by the trimming operation. If a node $x \in V$ belongs to a bud, then the set $\mathcal{V}_\tau$ of nodes of the maximal bud containing x is exactly the set of nodes with representatives in the same set as $r(x)$.

To trim a bud $(V_\tau, E_\tau, (x,y))$, we do the following.

(T1) Delete arcs in E_τ from the appropriate arc lists and from T and T', if applicable.

(T2) For every $z \in V_\tau$, concatenate the arc list of z to the arc list of y.

(T3) Form the union of sets containing representatives of nodes in V_τ.

To implement the last step, recall that the bud in the current graph found by the algorithm is elementary, i.e., V_τ contains exactly nodes on a path Γ and the symmetric path $\sigma(\Gamma)$. Thus, to find the sets which need to be joined it suffices to traverse once Γ and $\sigma(\Gamma)$.

As we shall see later, the algorithm does $O(m)$ set operations. Under the above mentioned implementation, the set operations take $O(m\alpha(m+n,n))$ time. However, we show below that, under a slightly different implementation, the set operations fall into the incremental tree set union case studied by Gabow and Tarjan [15]. Because of this, a sequence of k operations can be done in $O(k+n)$ time.

The *incremental tree set union* case [15] is as follows. Let R be a rooted tree whose nodes correspond to disjoint sets of elements. Initially R contains a single node corresponding to a single element set. The operations allowed on R are a contraction of an edge of R that results in merging the sets corresponding to the two end points, and addition of a leaf node corresponding to a single element set.

For the RRP algorithm, the tree R is just T (assuming that the nodes of T are the corresponding sets of representatives). If the algorithm adds a node to T, this node becomes a leaf of T and corresponds to a single element set. If the algorithm finds a bud $\tau = (V_\tau, E_\tau, (x,y))$ while scanning an arc (v,w), each node in V_τ occurs on a path Γ or $\sigma(\Gamma)$. Furthermore, Γ consists of a path Γ_1 from y to v in T, (v,w), and a path Γ_2 from w to $\sigma(y)$ in T'. Observe that $\sigma(\Gamma_2)$ is a path in T from y to $\sigma(w)$, and for every node $z \in V_\tau$, exactly one of the nodes $z, \sigma(z)$ appears on Γ_1 or $\sigma(\Gamma_2)$. Therefore we form the unions of only the sets which are connected by the edges of R, i.e., the incremental tree method is applicable in our case.

2.5. Linear-Time Implementation. Now we specify how to find an arc going out of A efficiently. We maintain the set Q of arcs $(v, w) \in E$ (of the current graph) such that $v \in A$ and $w \notin A$. The set Q can be represented by any data structure that allows constant time membership query, addition, extraction, and deletion (*e.g.* a queue implemented using a doubly linked list).

Since initially $A = \{s\}$, we initialize Q by adding all arcs $(s, x) \in E$ such that $x \neq s$ to Q. To select an arc to scan, we extract an arc (v, w) from Q and scan it. Every time we add a node x to A, we add (x, y) to Q for every y such that $(x, y) \in E$ and $y \notin A$, and delete (z, x) from Q for every $(z, x) \in E$ in Q.

Finally, we show how to find a bud quickly. Recall that a bud is formed when we scan an arc (v, w) connecting A to A' and find out that the path $P = (P_1, (v, w), P_2)$ is not regular. To find the desired bud (or a regular path from s to s'), we trace P_2 forward from w and P_1 backward from v, intermixing forward and backward steps. We stop as soon as we discover an arc (x, y) in P_i such that its symmetric arc has already been traced in P_{3-i}. The time spent on finding a bud using this techniques is proportional to the number of nodes in the discovered bud.

Next we show that the above algorithm terminates in linear time. In the next section we show how to compute an s to s' r-path in the original graph if the above algorithm found an s to s' r-path in the resulting graph.

Theorem 2.8. *The above algorithm solves the problem of determining whether or not there exists an r-path from s to s', in time $O(m)$.*

Proof. We show that the number of operations of the algorithm is $O(m)$. These operations include the set union operations spent; however, the whole work in performing the set union operations requires time $O(m)$ because the operations fall into the incremental tree case discussed above.

To account for the time to perform the bud trimmings, note that the cost of trimming a bud τ is proportional to $|E_\tau|$, and the number of arcs of the graph decreases by $|E_\tau|$ because of the trimming. Hence, the total cost of bud trimmings is $O(m)$.

Similarly, since the cost of finding a bud is proportional to the size of the bud, the total cost of finding the buds is $O(m)$. If the algorithm finds an r-path from s to s', it takes $O(n)$ time to verify this fact.

Each time the algorithm scans an arc going out of A, a node is added to A or a bud trimming occurs, so there are $O(m)$ scans. Each scan requires a constant number of operations (including selection of the arc to be scanned but not including those accounted for above or those needed to maintain the set Q).

Finally, each time a node of the current graph is added to A, we need to update Q. Since a node $v \in V$ is added to the preimage of A at most once and the cost of updating Q due to the node addition is $O(1 + \text{degree}(v))$, the total number of additions to Q is $O(n)$ and the total work involved is $O(m)$. The number of extractions from Q cannot exceed the number of additions. ■

2.6. Extracting the Path. Note that if the above algorithm finds an r-path, the path is in the resulting graph $\widetilde{G}$. The corresponding path in the original graph can be constructed in linear time by adding bookkeeping and postprocessing to the algorithm.

We say that a sequence $(a_1, \dots, a_k)$ of arcs is a *partial path* if there is a path $(b_1, \dots, b_r)$ and a sequence of indices $i_1 < i_2 < \dots < i_k$ such that $a_j = b_{i_j}$ for $1 \le j \le k$.

For the bookkeeping, we store information about trimmed buds in a way that allows us to undo the trimmings during the postprocessing. We maintain a rooted forest F that represents the nested structure of the buds appeared during the algorithm. Each vertex of F represents to a bud or a pair of symmetric nodes of G. Each leaf of F represents a pair of symmetric nodes; there are $n/2$ leaves. Each non-leaf vertex x represents to a bud τ, and the leaves of the subtree rooted at x is just the representatives of the symmetric pairs in $\mathcal{V}_\tau$. In the latter case we also say that the x represents the pair $\{w_\tau, w_\tau{}'\}$.

The forest F is maintained as follows. At the beginning of the algorithm F is a trivial forest with $n/2$ vertices and no edges. When a bud $\tau = (V_\tau, E_\tau, e_\tau)$ is trimmed, we add to F a new vertex x representing τ. For each pair $\{y, \sigma(y)\}$ of elements of V_τ we make the vertex of F representing this pair a child of x.

When a bud τ is trimmed, we also store the two paths that form E_τ. These paths are stored as the corresponding sequences of arcs of G, organized as doubly-linked lists. Because of (ii) in Lemma 2.7 the overall work in storing the paths is $O(m)$.

We say that a vertex of F which is not a child of any other vertex is *maximal*. The forest F is changed during the path restoring algorithm by removing (repeatedly) certain maximal vertices. In the process of restoring we use the following F-ROOT(v) operation: given a leaf x of F, find the maximal vertex r_x of the tree T of the current F that contains x, i.e., r_x is the root of T. To implement this operation, we maintain a stack of vertices of F, which either is empty or corresponds to the path from x to r_x. Initially each stack is empty. We say that a leaf is *active* if its stack is already non-empty. Suppose we want to find r_x for some leaf x. If x is active then r_x is just at the top of the stack, and we find r_x immediately. If x is not active, we traverse F up, starting from x and simultaneously pushing the traversed vertices into the stack. At the end of the traversal r_x occurs at the top of the stack, and x becomes active.

Let $\widetilde{P}$ be the (simple) path in $\widetilde{G}$ found by the algorithm. This path corresponds to a partial path P in G. Note that P may have consecutive arcs (a,b) and (c,d) such that $b \neq c$. In this case b and c must belong to the same maximal bud and therefore to the same tree T of F. Let $\overline{b}$ and $\overline{c}$ be the leaves in F representing the pairs $\{b,\sigma(b)\}$ and $\{c,\sigma(c)\}$, respectively. Using F-ROOT operation, we find the root x of T and work with the bud $\tau = (V_\tau, E_\tau, (v,w))$ represented by x as follows. Note that either (a,b) is the base arc (v,w) of τ, or (c,d) is its anti-base arc (w',v'). Assume the former; the other case is symmetric. We remove x from F, which makes the children of x maximal vertices in the new F. Accordingly, we delete the top element x from the stacks of $\overline{b}$ and $\overline{c}$; let y and z be the new elements in these stacks, respectively. Then y and z represent some pairs $\{f,\sigma(f)\}$ and $\{g,\sigma(g)\}$, respectively, of nodes of the graph G' from which $\widetilde{G}$ was created by trimming τ. Moreover, b is in the preimage of f or $\sigma(f)$, and c is in the preimage of g or $\sigma(g)$. By the above assumption, $\{f,\sigma(f)\}$ coincides with $\{w,w'\}$.

Our aim is to find a path $\Gamma' = (e_1,\dots,e_k)$ in (V_τ, E_τ) to be inserted in $\widetilde{P}$ between (a,b) and (c,d) so that the resulting sequence P' would be an s to s' r-path in G'. If $y = z$ then $\widetilde{P}$ is already the desired path. Otherwise we should take as Γ' either the path Γ_1 from w to g or the path Γ_2 from w to $\sigma(g)$ (these Γ_1 and Γ_2 are found by traversing the two paths forming E_τ). If z is a leaf then $c \in \{g,\sigma(g)\}$, and the task of choosing Γ' among Γ_1 and Γ_2 is trivial. And if z represents a bud τ' then we must take as Γ' that of Γ_1 and Γ_2 last arc of which is the base arc of τ'. Here we use the facts that (a,b) and (c,d) are not in E_τ and that P' must contain the base or anti-base arc of τ' to conclude that in our case the base and anti-base arcs of τ' are in E_τ.

We recursively apply the above procedure to the graph G' and path P', and so on. As a result, we eventually find the desired path in G.

Formally, at an iteration of the postprocessing algorithm we deal with a partial path $(a_1,\dots,a_k)$ in G and a pair (a_i,a_{i+1}) in it and decide whether the head h of a_i coincides the tail t of a_{i+1}. If $h \neq t$, we find a sequence $(e_1,\dots,e_k)$ in E_τ to insert between a_i and a_{i+1} as described above, where τ is the maximal bud (in the current collection of buds) such that $h,t \in \mathcal{V}_\tau$.

The scan of all the pairs (a_i,a_{i+1}) throughout the algorithm can be easily organized so that it takes linear time. Furthermore, determining the required sequence in E_τ takes time $O(|E_\tau|)$, therefore the whole work in constructing such sequences takes linear time. It remains to show that the total time η of the F-ROOT operation is linear. We observe that once the F-ROOT is applied to a leaf x of F, each repeated F-ROOT(x) operation takes $O(1)$ time and the stack of x decreases. Therefore, η is proportional to the total length of the stacks designed during the algorithm, or to the total number μ of traversals of edges in F when the stacks are created. Analysis of the algorithm easily shows that each edge of F is traversed at most twice (taking into account that while a bud τ is occurring in F, there can happen at most two arcs (a,b) and (c,d) in the current partial paths for which b and c belong to $\mathcal{V}_\tau$). Hence, μ is $O(n)$, and the result follows.

3. Shortest R-Paths under Nonnegative Lengths

In this section we study the problem of finding a shortest s to s' r-path under the nonnegative arc lengths. We refer to this problem as NSRPP.

3.1. Length Transformations. A *fragment* is a pair $\tau = (V_\tau, e_\tau = (v, w))$, where V_τ is a symmetric set of nodes and e_τ is an arc such that $v \notin V_\tau$, $w \in V_\tau$. Note that every bud corresponds to a fragment defined by its node set and base arc. The *characteristic function* χ_τ of τ is the function on E defined by

$$\chi_\tau(a) = \begin{cases} 1 & \text{if } a \in \{e_\tau, \sigma(e_\tau)\}, \\ -1 & \text{if } a \in \delta(V_\tau) - \{e_\tau, \sigma(e_\tau)\}, \\ 0 & \text{otherwise.} \end{cases}$$

Here $\delta(V_\tau)$ is the set of arcs with one end in V_τ and the other in $V - V_\tau$.

Let $\epsilon \in \mathbf{R}_+$. The (ϵ, τ)-*transformation* maps the length function ℓ to $\ell' = \ell + \epsilon\chi_\tau$. We can apply a sequence of (ϵ_i, τ_i)-transformations and obtain the length function $\ell' = \ell + \sum_i \epsilon_i \chi_{\tau_i}$. Observe that if length of a path P increases after an (ϵ, τ)-transformation, P must pass through both the base and the anti-base of τ, thus P is non-regular. Therefore the (ϵ, τ)-transformation does not increase the length of any r-path from s to s'.

Node *potentials* are given by a *potential function* $p : V \to \mathbf{R}$. Given a length function ℓ, we define the *reduced cost* function ℓ_p by $\ell_p(x, y) = \ell(x, y) + p(x) - p(y)$, $(x, y) \in E$. If the reduced costs are nonnegative, and a path P has zero reduced cost, then P is a shortest path for ℓ.

A *separator* is a set $S \subset V$ such that $s \in S$, $s' \notin S$. Given a separator S, a potential function p, and a real $\epsilon \in P$, the (ϵ, S)-*relabeling* operation modifies p as follows:

$$p(x) = \begin{cases} p(x) & \text{if } x \in S, \\ p(x) + \epsilon & \text{otherwise.} \end{cases}$$

Given a length function ℓ' and a potential function p such that $\ell'_p \geq 0$, we define the *zero-graph* $G_0 = (V, E_0)$ by defining $E_0 = \{a \in E | \ell'_p(a) = 0\}$.

3.2. Linear Programming Formulation. Next we describe a linear program (LP) and discuss its relationship to the shortest r-paths problem. Although we are now studying the nonnegative length case, the linear program remains the same for the arbitrary length case, which we study in the next section. Let $\mathcal{T}$ be the set of all fragments and let $\mathcal{P}$ be the set of all regular paths from s to s'. Given a path P, we define its characteristic function χ_P on E by

$$\chi_P(e) = \begin{cases} 1 & \text{if } e \in P, \\ 0 & \text{otherwise.} \end{cases}$$

Variables of the LP are of two types. A variable of the first type corresponds to a fragment $\tau \in \mathcal{T}$ and is denoted by ϵ_τ. A variable of the second type corresponds to a node $x \in V$ and is denoted by π_x. The LP is as follows.

$$(3.2\text{-}1) \qquad \max \pi_{s'} \quad \text{subject to} \quad \begin{cases} \epsilon_\tau \geq 0 & \forall \tau \in \mathcal{T} \\ \pi_s = 0 \\ \pi_y - \pi_x - \sum_{\mathcal{T}} \epsilon_\tau \chi_\tau(x,y) \leq \ell(x,y) & \forall (x,y) \in E. \end{cases}$$

Note that if $\ell \geq 0$, then $\epsilon = 0$ and $\pi = 0$ give a feasible solution to this LP.

It can be shown that the s to s' r-paths induce a subset of feasible dual solutions to this LP. We do not state the dual program explicitly but prove the facts we need directly, without an explicit use of the linear programming duality.

For any feasible solution to the LP, $\pi_{s'}$ gives a lower bound on the shortest s to s' r-path length. This is because any (ϵ, τ)-transformation can only decrease the length of an r-path from s to s' (as mentioned above), and the length of any s to s' path with respect to $\ell + \sum_{\mathcal{T}} \epsilon_\tau \chi_\tau$ is at least $\pi_{s'}$.

Let $P \in \mathcal{P}$ and let (ϵ, π) be a feasible solution to the LP. Let $\ell' = \ell + \sum_{\mathcal{T}} \epsilon_\tau \chi_\tau$. The following *"complementary slackness"* conditions give an optimality criterion for the problem, in view of Lemmas 3.1–3.2 and Theorem 3.3 below.

(CS1) $\epsilon_\tau > 0 \Rightarrow \chi_P \cdot \chi_\tau = 0$,
(CS2) $(x,y) \in P \Rightarrow \pi_y - \pi_x = \ell'(x,y)$.

Lemma 3.1. *If (CS1) and (CS2) hold, then P is a shortest r-path from s to s' with respect to ℓ and $\ell(P) = \pi_{s'}$.*

Proof. Condition (CS2) implies that P is a shortest path with respect to ℓ' and $\ell'(P) = \pi_{s'}$. Condition (CS1) implies that $\ell(P) = \ell'(P)$. The fact that for any r-path Γ, $\ell(\Gamma) \geq \ell'(\Gamma)$ completes the proof. ∎

This lemma is easy, while the next one is provided by the algorithm described in Section 3.3.

Lemma 3.2. *If P is a shortest r-path with respect to ℓ, then there exists a feasible solution (ϵ, π) to the LP such that (CS1) and (CS2) hold.*

Taken together, Lemmas 3.1 and 3.2 yield the following duality theorem.

Theorem 3.3. $\max \pi_{s'} = \min_{\mathcal{P}} \ell(P)$.

Proof. We know that $\max \pi_{s'} \leq \min_{\mathcal{P}} \ell(P)$. By Lemmas 3.1 and 3.2, there exist $P \in \mathcal{P}$ and a feasible solution (ϵ, π) such that $\ell(P) = \pi_{s'}$. ∎

3.3. Algorithm Description. The intuition for the algorithm of this section is as follows. Suppose we have a current length function ℓ' (obtained by a sequence of (ϵ, τ)-transformations) and a potential function p such that the reduced costs ℓ'_p are nonnegative. Then in G_0 either there is an r-path P from s to s' or a barrier. In the former case, if P satisfies (CS1) then P is a required path by Theorem 3.3. In the latter case, using the barrier, we modify the length and potential functions so that the length of the shortest (non-regular) paths increases and the reduced costs remain nonnegative. The search for the path in G_0 is similar to the RRP algorithm. Each bud $\tau = (V_\tau, E_\tau, e_\tau)$ found during the search induces the fragment (V_τ, e_τ) (also denoted by τ), and the function ϵ on $\mathcal{T}$ can take nonzero values only on these fragments.

The algorithm maintains a length function ℓ' and a potential function p such that ℓ'_p is nonnegative and symmetric. Initially ℓ' is the input length function ℓ and p is the zero function. The algorithm also maintains a set A of nodes reachable from s by r-paths in the current zero-graph $\overline{G}_0$, with the r-paths represented by a spanning tree $T \subset E$ of $\langle A \rangle$ rooted at s. By symmetry, for every node $v' \in A' = \sigma(A)$ there is an r-path to s' in $T' = \sigma(T)$. The invariant $A \cap A' = \emptyset$ always holds. Initially $A = \{s\}$ and $T = \emptyset$.

At each step the algorithm attempts to find an arc (v, w) such that $v \in A$, $w \notin A$, and $\ell'_p(v, w) = 0$. If such an arc exists, the algorithm proceeds as the RRP algorithm. The only difference is that the buds are discovered in $\overline{G}_0$ but trimmed in the current graph $\overline{G}$. The algorithm maintains the sets C_V and C_E of base nodes and arcs of the maximal fragments (in the current collection of fragments).

If no arc out of A with zero reduced cost exists, the algorithm chooses $\epsilon \in R_+$ as described below and modifies ℓ' and p by performing the following *adjustment procedure* consisting of the four consecutive steps (1)–(4) (in (1) and (2) ℓ' is being changed step by step, by treating the elements of C_E in succession). Define $X = A - C_V$.

(1) For every $(v, w) \in C_E$, set $\ell'(v, w)$ to $\ell'(v, w) + \epsilon$, and for all $(w, z) \in \overline{E}$ with $z \notin V_\tau$, set $\ell'(w, z)$ to $\ell'(w, z) - \epsilon$, where τ is the maximal fragment with $e_\tau = (v, w)$.

(2) For every $(v, w) \in C_E$, set $\ell'(\sigma(v, w))$ to $\ell'(\sigma(v, w)) + \epsilon$ and for all $\sigma(w, z) \in \overline{E}$ with $z \notin V_\tau$, set $\ell'(\sigma(w, z))$ to $\ell'(\sigma(w, z)) - \epsilon$, where τ is as in (1).

(3) Apply the (ϵ, X)-relabeling.

(4) Apply the $(\epsilon, V - \sigma(X))$-relabeling.

(The first two steps are equivalent to applying to ℓ' the (ϵ, τ)-transformations simultaneously for all maximal fragments τ.)

The value of the *adjustment parameter* ϵ is chosen to be as large as possible without creating negative reduced cost arcs. One can see that after the adjustment, the reduced cost of an arc is either preserved or changed by ϵ, 2ϵ, $-\epsilon$, or -2ϵ. The arcs whose reduced costs decrease by

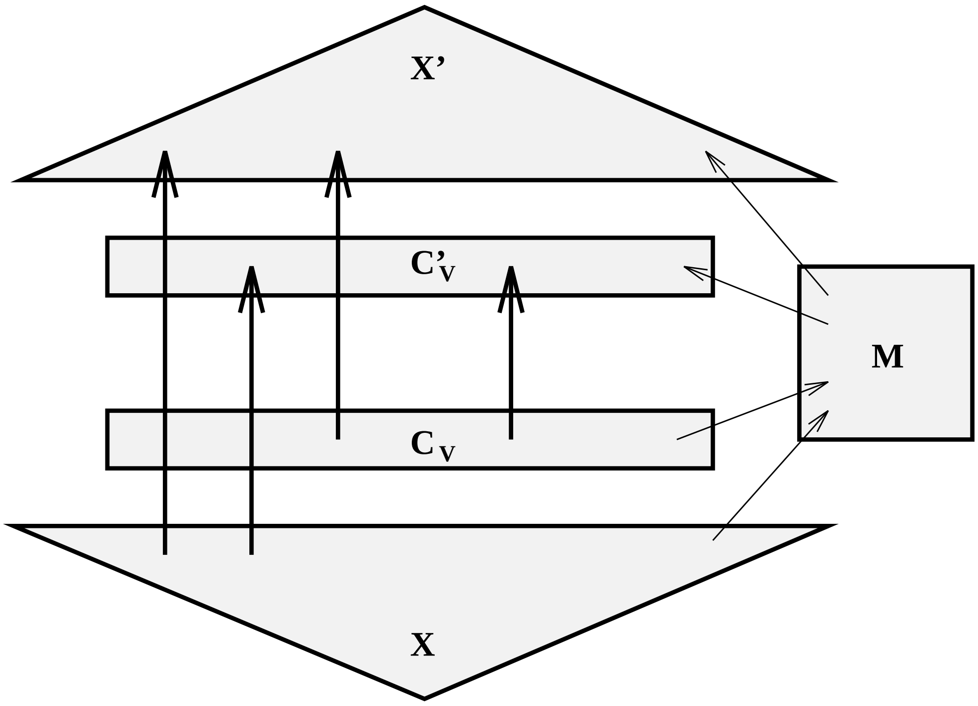

FIGURE 3. *The adjustment parameter choice. The length of light arcs decreases by ϵ, and the length of heavy arcs by 2ϵ.*

ϵ include the arcs from X to M and the symmetric arcs, and the arcs from C_V to M and the symmetric arcs. (M is defined in Section 2.1.) (See Figure 3.) We denote the set of these arcs by E_1^ϵ. One can see that the arcs whose reduced costs decrease by 2ϵ are of three types.

(1) The arcs from X to $\sigma(X) = X'$.
(2) The arcs from X to $\sigma(C_V) = C'_V$ and the symmetric arcs.
(3) The arcs of the form $(v_1, \sigma(v_2))$, where $v_1, v_2 \in C_V$ and $v_1 \neq v_2$.

We denote the set of these arcs by E_2^ϵ. The value of ϵ is the smallest of the minimum reduced cost over the arcs in E_1^ϵ and one half of the minimum reduced cost over the arcs in E_2^ϵ.

Clearly if no arc restricts the value of ϵ, then there is no path from s to s' in the current graph, so the problem is infeasible. Otherwise, at the next iteration of the algorithm the current $\overline{G}_0$ must contain an arc (v, w) with $v \in A$, $w \notin A$, so during this iteration either a bud trimming occurs and the graph $\overline{G}$ becomes smaller, or a node is added to A. Therefore any reasonable implementation of the algorithm runs in polynomial time. We discuss a particularly efficient implementation later in Section 3.5.

3.4. Correctness. We start the correctness proof with several lemmas. The proof of the first lemma is trivial.

Lemma 3.4. *The reduced cost function ℓ'_p maintained by the algorithm is symmetric; the graph $\overline{G}_0$ is also symmetric.*

Lemma 3.5. *T and T' contain only zero reduced cost arcs.*

Proof. The only case which is not completely straight-forward is the case of an arc $(x, y) \in T$ that is a base arc of a fragment (*i.e.,* $(x, y) \in C_E$). In this case $y \in C_V$ and x is either in C_V or in X. If $x \in C_V$ then both potentials $p(x)$ and $p(y)$ increase by ϵ and $\ell'(x, y)$ does not change. To see the latter, let τ_x be the maximal fragment with the base x and let τ_y be the maximal fragment with the base y. Note that (x, y) is the base arc of τ_y, whence the treatment of τ_y in the adjustment procedure increases $\ell'(x, y)$ by ϵ. But (x, y) is also an arc going out of V_{τ_x}, whence the treatment of τ_x in the adjustment procedure decreases $\ell'(x, y)$ by ϵ. Thus the whole adjustment procedure does not change $\ell'(x, y)$. If $x \in X$ then $p(x)$ is not changed and both $p(y)$ and $\ell'(x, y)$ are increased by ϵ. Hence, $\ell'_p(x, y)$ is not changed. ■

The adjustment parameter is chosen so that the following result holds.

Lemma 3.6. *The reduced costs maintained by the algorithm are nonnegative.*

Lemma 3.7. *Bud trimming preserves the minimum length of an r-path from s to s'.*

Proof. Let $\widetilde{G}$ be the graph obtained from the current graph $\overline{G}$ by trimming $\tau = (V_\tau, E_\tau, (v, w))$ and let ℓ'_p denote the reduced costs in $\overline{G}$. Then $\ell'_p \geq 0$, $\ell'_p(e) = 0$ for every $e \in E_\tau$, and trimming preserves the reduced costs of the arcs that remain in $\widetilde{G}$. Suppose that P' is an ℓ'_p-shortest r-path from s to s' in $\widetilde{G}$. Since (v, w) is the only arc in $\overline{G}$ entering w and $\sigma(v, w)$ is the only arc leaving $\sigma(w)$, P' passes through at most one node among $w, \sigma(w)$. This implies that there is an r-path P in $\overline{G}$ from s to s' consisting of all arcs of P' and, possibly, some arcs in E_τ. Then $\ell'_p(P) = \ell'_p(P')$, hence the minimum ℓ'_p-length of an r-path from s to s' in $\overline{G}$ is at most $\ell'_p(P')$.

To see the converse inequality, suppose that P is an ℓ'_p-shortest r-path in $\overline{G}$. Consider the subgraph $Q = (V_Q, E_Q)$ induced by the arc-sets of $P, \sigma(P), L, \sigma(L)$ and the set E_τ, where L is the path in T from s to w. By Theorem 2.4, after trimming τ in Q the resulting graph $\widetilde{Q}$ still contains an r-path P' from s to s'. Since the paths P' and $\sigma(P')$ are edge-disjoint and $\ell'_p(e) = 0$ for all $e \in E_\tau \cup L \cup \sigma(L)$, we have

$$2\ell'_p(P') = \ell'_p(P') + \ell'_p(\sigma(P')) \leq \ell'_p(E_Q)) = \ell'_p(P) + \ell'_p(\sigma(P)) = 2\ell'_p(P),$$

hence $\ell'_p(P') \leq \ell'_p(P)$. ■

Theorem 3.8. *If there is no r-path from s to s', the algorithm declares the problem infeasible, and otherwise the value of $p(s')$ at the end of the algorithm is equal to the length of the shortest r-path from s to s' with respect to ℓ.*

Proof. The first part of the theorem follows from Theorem 2.4 and the description of the algorithm.

For the second part of the theorem, note that the above lemmas imply that if the algorithm terminates with an r-path, this r-path has a zero reduced cost with respect to the final reduced cost function ℓ'_p and ℓ'_p is nonnegative. Lemmas 3.1 and 3.7 complete the proof. ∎

In particular, this proves Lemma 3.2 and Theorem 3.3.

3.5. Efficient Implementation. In this section we give details of an efficient implementation of the algorithm. This implementation runs in time which is close to linear. The only nonlinear term comes from $O(m)$ operations on the priority queue data structure.

An important part of the implementation is the way node potentials are maintained. The potentials of nodes in A are maintained explicitly (by A, A' we mean these sets themselves or their preimages in the original graph, depending on the context). We also maintain the number D such that $p(s') = 2D$. For $v \in A'$, $p(v) = 2D - p(\sigma(v))$. For a node $v \in M$, $p(v) = D$. When a node from M is added to A, its potential is computed and stored explicitly. Each time we trim a bud τ with base node w, we store the values D_τ and p_τ of D and $p(w)$, respectively, at the time of trimming. Then for a node $w \in C_V$ that is the base of a trimmed bud τ, the current $p(w)$ is given by $p(w) = p_\tau + (D - D_\tau)$. Accordingly, for a node $w \in C'_V$, $p(w) = 2D - p(\sigma(w))$. The adjustment procedure with the parameter value ϵ increases D by 2ϵ.

The function ℓ' is maintained explicitly on the arcs of $\overline{G}$ not incident to nodes in C_V or C'_V and implicitly on the remaining arcs of $\overline{G}$. (Note that the adjustment procedure does not change ℓ' on the former arcs.) Initially all values of $\ell' = \ell$ are explicit. We denote the last value of $\ell'(v,w)$ stored explicitly by $\beta(v,w)$. Consider an arc (v,w) with exactly one end incident to a maximal fragment, and let τ be the single minimal fragment incident to (v,w). We have the following possibilities.

(1) If $v \in C_V$, then the current value of $\ell'(v,w)$ is $\beta(v,w) - (D - D_\tau)$.
(2) If $w \in C_V$ then the current value of $\ell'(v,w)$ is $\beta(v,w) + (D - D_\tau)$.
(3) If $w \in C'_V$ then the current value of $\ell'(v,w)$ is $\beta(v,w) - (D - D_\tau)$.
(4) If $v \in C'_V$, then the current value of $\ell'(v,w)$ is $\beta(v,w) + (D - D_\tau)$.

Now consider an arc (v,w) adjacent to two maximal buds τ_1 and τ_2. Without loss of generality assume that v is the base of τ_1. If (v,w) is the base arc of τ_2, then the current value of $\ell'(v,w)$ is

$\beta(v,w) - (D_{\tau_1} - D_{\tau_2})$, and otherwise the value is $\beta(v,w) - (D - D_{\tau_1}) - (D - D_{\tau_2})$.

To select the next arc for scanning and to compute the adjustment parameter values efficiently, we represent the sets E_1^ϵ and E_2^ϵ by priority queues Q_1 and Q_2, respectively. (For a discussion of the priority queue data structure, see *e.g.* [5].) We use the standard priority queue operations: MEMBER, INSERT, MINIMUM, EXTRACT-MINIMUM, and DELETE. The key $k(v,w)$ of an element (v,w) on Q_1 is equal to $\beta_p(v,w) - D_w$, where D_w is the value of D at the last time $k(v,w)$ was computed and $\beta_p(x,y) = p(x) - p(y) + \beta(x,y)$. The key $k(v,w)$ of an element (v,w) on Q_2 is equal to $\beta_p(v,w) - 2D_w$, where D_w is the value of D at the time $k(v,w)$ was computed. Below we show how to maintain D so that for (v,w) on Q_1, $\ell'_p(v,w) = k(v,w) - D$, and for (v,w) on Q_2, $\ell'_p(v,w) = k(v,w) - 2D$.

Using the priority queues, the adjustment parameter value can be quickly computed because it is given by

(3.5-2) $$\epsilon = (\min(\text{MINIMUM}(Q_1), \frac{1}{2}\text{MINIMUM}(Q_2))) - D.$$

(We assume that if Q_i is empty, $\text{MINIMUM}(Q_i)$ returns infinity.)

The efficient implementation of the algorithm works as follows. Initially Q_2 is empty and Q_1 contains the arcs $(s,v) \in E$ with $v \neq s$. At each step, we compute the adjustment parameter ϵ as described above.

If $0 < \epsilon < \infty$, then we increase D by ϵ. Note that given our representation of p and ℓ', this implements the adjustment procedure.

If $\epsilon = 0$, let Q_i be the queue containing the arc for which the minimum in (3.5-2) is achieved. Then $\text{EXTRACT-MINIMUM}(Q_i)$ returns an arc (v,w) such that $v \in A$, $w \notin A$, and $\ell'_p(v,w) = 0$. We examine all $(w,x) \in E$. If (w,x) is not in Q_1 or Q_2, we compute $k(w,x)$, and execute $\text{INSERT}(Q_1,(w,x))$ if $x \in M$ and $\text{INSERT}(Q_2,(w,x))$ otherwise. Next we examine all $(z,w) \in E$ and delete (z,w) from Q_i if (z,w) is in the queue.

Then we scan (v,w) as follows. If $w \notin A'$, then w is added to A, (v,w) to T, w' to A', and $\sigma(v,w)$ to T' (as usual, $w' = \sigma(w)$). For each $(x,w) \in E$ that is in Q_1, we execute $\text{DELETE}(Q_1,(x,w))$. For each $(x,w') \in E$ that is in Q_1, we execute $\text{DELETE}(Q_1,(x,w'))$, set $D_{w'} = D$, $\beta(x,w') = \ell'(x,w')$, and $k(x,w') = \beta(x,w')) - 2D_{w'}$, and execute $\text{INSERT}(Q_2,(x,w'))$.

Now suppose $w \in A'$ and let P be the concatenation of the s to v path P_1 in T, (v,w), and the w to s' path P_2 in T'. If P is regular, then we return P and halt. If P is not regular, then we find and trim a bud $\tau = (V_\tau, E_\tau, (x,y))$ in $\overline{G}$ in the same way as in the RRP algorithm. If an arc a is deleted from the current graph by the trimming operation, we delete the arc from Q_1 and Q_2 if appropriate.

If $\epsilon = \infty$, we declare the problem infeasible.

The above analysis immediately implies the following theorem.

Theorem 3.9. *The shortest regular path problem with nonnegative arc lengths can be solved in $O(m)$ time plus $O(m)$ priority queue operations mentioned above.*

Since some priority queue operations require non-constant time, the running time of the NSRPP algorithm is dominated by $O(m)$ priority queue operations and time bound achieved by the algorithm depends on the priority queue implementation used. Using the simple binary heap data structure (see *e.g.* [5]), we obtain an $O(m \log n)$ bound. Using the R-heap implementation [1], we obtain an $O(m\sqrt{\log C})$ bound. Note that the latter bound implies that we can solve the problem with unit arc lengths in linear time.

3.6. Extracting Primal and Dual Solutions. Next we discuss how to modify the algorithm so that we can in linear time recover, upon termination, an r-path P in G, a potential function p, and length transformations $(\epsilon_1, \tau_1), \ldots, (\epsilon_k, \tau_k)$ that satisfy the complementary slackness conditions.

We start with the length transformation. To obtain the linear time bound, we use techniques for representing nested families similar to those for the RPP algorithm, including the data structures used for extracting the path at the end of the RPP algorithm.

To extract ϵ, recall that with each bud τ trimmed by the algorithm we also maintain the value D_τ. Let D_f be the final value of the variable D. Then for each bud τ, the corresponding ϵ_τ value is $D_f - D_\tau$.

Next we discuss how to extract the potential function p. Recall that we have already discussed how the potentials of nodes in $\overline{G}$ are maintained, and the explicit value of such a potential can be computed in constant time. Let v be a node deleted during trimming of a bud τ, and let $p_f(v)$ be the potential value at v at the time of trimming. Then $p(v) = p_f(v) + D_f - D_\tau$. Note that τ corresponds to the parent of v in the forest representing the buds and can be found in constant time.

Finally, the path P in G corresponding to the path in $\overline{G}$ found by the algorithm can be reconstructed in the same way as in the RRP algorithm. Because of the way the bud trimming operation works, the path P satisfies (CS1) and (CS2).

4. Shortest R-Paths under Arbitrary Lengths

Let $\ell : E \to \mathbf{R}$ be an arbitrary symmetric length function. We say that (G, ℓ) is r-*conservative* if there is no r-cycle C in G of negative length $\ell(C)$ (an r-*cycle* is a directed cycle containing no pair of symmetric arcs). Note that an r-conservative (G, ℓ) can have (non-regular) negative length cycles. The SRPP is the problem to decide whether (G, ℓ) is r-conservative, and if it is, to find a shortest r-path from s to s', given $s \in V$ and $s' = \sigma(s)$.

It is easy to see that for any nonnegative function ϵ on the set $\mathcal{T}$ of fragments in G, the length transformation $\ell \to \ell'$, defined in Section 3.1, does not increase the length of any r-cycle. The length of an r-cycle can decrease, however, so we have to assure that no negative r-cycles are introduced by the length transformations.

Consider the linear program (3.2-1).

Theorem 4.1.

(i) *(G, ℓ) is r-conservative if and only if LP* (3.2-1) *has a feasible solution, in other words, if and only if there exists a length transformation $\ell \to \ell'$ such that all cycles for (G, ℓ') are nonnegative.*

(ii) *Let (G, ℓ) be r-conservative. Then* $\max\ \pi_{s'} = \min(\ell(P) : P$ *is an r-path from s to s'*).

The proof of this theorem is provided by the algorithm to solve SRPP described next.

As a preprocessing step, we apply the following *node-splitting* transformation to construct an equivalent problem which is at most a constant factor bigger than the original problem and has $O(n)$ negative length arcs. We iterate over all nodes and do the following for every node x that has an outgoing arc of negative length. We replace x by two nodes, x_1 and x_2. Let μ be the minimum length of the arcs going out of x. We replace all arcs of the form (z, x) by the arcs (z, x_1) with $\ell(z, x_1) = \ell(z, x) - \mu$ and all arcs of the form (x, z) by the arcs (x_2, z) with $\ell(x_2, z) = \ell(x, z) - \mu$. We also add an arc (x_1, x_2) with $\ell(x_1, x_2) = 2\mu$ and call this arc the *x-arc*. Note that if x is split this way, then $x' = \sigma(x)$ is also split. It is easy to see that if we define $\sigma(x_1) = x'_2$ and $\sigma(x_2) = x'_1$, then the resulting graph is skew-symmetric and the new function ℓ is symmetric. Note that in the graph obtained by node-splitting, if (x_1, x_2) is an x-arc, then this is the only arc out of x_1 and the only arc into x_2.

The SRPP algorithm works as follows. Let $N = \{e \in E : \ell(e) < 0\}$, and let the pairs of symmetric arcs in N be numbered as $\{e_1, \sigma(e_1)\}, \{e_2, \sigma(e_2)\}, \dots, \{e_{|N|/2}, \sigma(e_{|N|/2})\}$. Denote by G^i the subgraph (V, E^i) in G, where E^i is obtained from G by deleting the arcs $e_j, \sigma(e_j)$ for $j = i+1, \dots, |N|/2$.

The algorithm consists of at most $|N|/2 + 1$ iterations. After executing $i-1$ iterations, we have the following current objects: a set $\mathcal{T}^i$ of fragments in G^{i-1}; a *positive* function ϵ^i on $\mathcal{T}^i$; a potential function p^i. In addition, the following conditions are satisfied.

(N1) The reduced costs $\ell^i(x, y) = \ell'_{p_i}(x, y)$ of the arcs (x, y) in G^{i-1} are symmetric and nonnegative.

(N2) $\mathcal{T}^i$ forms a *base compatible nested* family; this means that for any two distinct $\tau = (V_\tau, (v, w)), \tau' = (V_{\tau'}, (v', w'))$ in $\mathcal{T}^i$ either $V_\tau \cap V_{\tau'} = \emptyset$, $V_\tau \subset V_{\tau'}$, or $V_{\tau'} \subset V_\tau$; moreover, if $V_\tau \subset V_{\tau'}$ and $v \notin V_{\tau'}$ then $(v, w) = (v', w')$.

(N3) For every $\tau = (V_\tau, a_\tau = (v, w)) \in \mathcal{T}^i$ and every $x \in V_\tau$, in the subgraph $\langle V_\tau \rangle$ of G^{i-1} induced by V_τ there are r-paths from w to x and from x to $\sigma(w)$ such that all arcs on these paths have zero reduced costs.

Initially, $\mathcal{T}^1 = \emptyset$ and $p^1 = 0$. At i-th iteration we examine the arcs $e_i = (x_i, y_i)$ and $\sigma(e_i) = (y'_i, x'_i)$. If $\ell^i(e_i) \geq 0$, we finish the iteration by setting $\mathcal{T}^{i+1} = \mathcal{T}^i$, $\epsilon^{i+1} = \epsilon^i$, and $p^{i+1} = p^i$. Otherwise we form the auxiliary graph $H = (V_H, E_H)$ by adding to G^{i-1} new nodes t and t' and arcs $(t, y_i), (t, x'_i), (x_i, t'), (y'_i, t')$ and define ℓ' on these new arcs to be zero. We extend p^i to V_H by setting $p^i(v)$ for $v = t, t'$ in such a way that the reduced costs of the added arcs are nonnegative. We say that an r-path P in H from t to t' is *strong* if for any $\tau \in \mathcal{T}^i$, $\chi_\tau \cdot \chi_P = 0$. Let $\widetilde{\mathcal{P}}$ be the set of strong paths.

Note that because e_i is the only arc leaving x_i and the only arc entering y_i, none of x_i, y_i, x'_i, y'_i is in any fragment of $\mathcal{T}^i$, so we can construct the auxiliary graph without any complications even if we work with trimmed graphs.

Suppose we have a procedure for finding a shortest strong path P with respect to ℓ^i. Also suppose that, when constructing P in H, the procedure transforms $\mathcal{T}^i, \epsilon^i, p^i, \ell^i$ into $\mathcal{T}^{i+1}, \epsilon^{i+1}, p^{i+1}, \ell^{i+1}$ satisfying (N1)–(N3) for G^i except, possibly, the nonnegativity of ℓ^{i+1} on $e_i, \sigma(e_i)$. If the latter happens (*i.e.*, if for the part P' of the resulting P from y_i to x_i (or from x'_i to y'_i) the inequality $\ell(P') < |\ell(e_i)|$ occurs), then adding e_i (or $\sigma(e_i)$) to P' we get a negative r-cycle and conclude that (G, ℓ) is not r-conservative.

If after executing the iterations as above we found out that (G, ℓ) is r-conservative, then at the last, say j-th, iteration, we put H, t, t' to be G, s, s', respectively, thus finding a strong r-path from s to s' which satisfies (CS1) and (CS2), along with the current $\mathcal{T}^{j+1}, \epsilon^{j+1}, p^{j+1}$. Therefore P is the shortest r-path from s to s'.

We show below that a shortest strong path can be found in $O(m \log n)$ time. This implies the following bound on the algorithm.

Theorem 4.2. *The SRPP problem can be solved in* $O((|N| + 1)m \log n)$*, or* $O(nm \log n)$*, time.*

If $\mathcal{T}^i$ is empty, the procedure for finding a shortest strong path is the same as the NSRPP algorithm of Section 3. Otherwise, the procedure is similar but somewhat more complicated because for some fragments $\tau \in \mathcal{T}^i$ the value of ϵ^i_τ may decrease and, if ϵ^i_τ reaches zero, τ may be deleted from $\mathcal{T}^i$.

As before, the fragments used by the algorithm are induced by the buds. We use the same bud trimming operation as above, but we also need the "inverse" *bud expansion* operation.

4.1. Data Structures and Bud Expansion. Let $\tau = (V_\tau, E_\tau, a_\tau = (v, w))$ be a maximal bud. Recall the efficient implementation of the bud trimming operation described in Section 2.4. To expand τ, we need to undo the steps (T2) and (T3) (see Section 2.4). We can undo (T2) if for every $z \in V_\tau$ we store the pointers to the beginning and the end of the arc list of z when we concatenate the list to the arc list of y.

To undo (T3), we need a set-union data structure that allows the SPLIT(r) operation in addition to the MAKE-SET(x), UNION(x, y), and FIND-SET(x) operations. The SPLIT(r) operation applied to the set containing r formed by UNION(x, y) partitions this set into the original two sets which were combined by the UNION operation. The *union by rank* variant of the disjoint set union data structure (see *e.g.* [5]) implements the desired operations so that each operation takes $O(\log n)$ time (where n is the maximum set size).

In the union by rank implementation, a collection of disjoint sets is represented by a rooted forest on the base elements. Each tree corresponds to a set represented by its root. The MAKE-SET(x) operation creates a single element set $\{x\}$. The FIND-SET(x) operation starts at x, goes up the tree containing x, and returns the root of the tree. The UNION(x, y) operation compares the sizes of the sets containing x and y, and makes the root of the tree representing the smaller set into a child of the root of the tree representing the bigger set. The children of a tree node are maintained as a stack. The SPLIT(r) operation finds the root z of the tree containing r, pops the stack containing the children of z, and makes the popped vertex a tree root. It is easy to see that each operation takes $O(\log n)$ time if n is the largest set size.

4.2. Finding a Shortest Strong Path. Now we describe the procedure for finding a shortest strong path.

In addition to the set $\mathcal{T}'$ of fragments and a function ϵ' on $\mathcal{T}'$, the procedure maintains a length function ℓ' and a potential function p such that ℓ'_p is nonnegative and symmetric. Initially $\mathcal{T}' = \mathcal{T}^i$, $\epsilon' = \epsilon^i$, $p = p^i$, and $\ell' = \ell + \sum_{\mathcal{T}} \epsilon'_\tau \chi_\tau$. When the procedure terminates, we set $\mathcal{T}^{i+1} = \mathcal{T}'$, $\epsilon^{i+1} = \epsilon'$, $p^{i+1} = p$.

The procedure maintains a set A of nodes reachable in G_0 from t by r-paths, with the r-paths represented by a spanning tree $T \subset E$ of A rooted at t. By symmetry, for every node $v' \in A' = \sigma(A)$ there is an r-path to t' in $T' = \sigma(T)$. The invariant $A \cap A' = \emptyset$ always holds. Initially $A = \{t\}$ and $T = \emptyset$. In addition, the algorithm maintains the set $\mathcal{T}'$ (equal to $\mathcal{T}^i$ initially), and the sets C_V, C_E, D_V, D_E defined as follows.

- For every maximal fragment $\tau = (V_\tau, a_\tau = (v, w))$ such that a_τ is in T, $w \in C_V$ and $a_\tau \in C_E$. We call τ with the base node in C_V a *growing fragment*.
- For every maximal fragment $\tau = (V_\tau, a_\tau = (v, w))$ with anti-base $w' = \sigma(w)$ such that there is an arc (z, w') in T, then $w' \in D_V$, and $(v, w') \in D_E$. We call τ with the anti-base node in D_V a *shrinking fragment*.

Initially the sets C_V, C_E, D_V, and D_E are empty.

Note that a fragment τ cannot be growing and shrinking simultaneously. This is because if τ is growing than its anti-base node must be in A', if τ is shrinking than its base node must be in A, and $A \cap A' = \emptyset$.

At each step the algorithm attempts to find an arc (v, w) such that $v \in A$, $w \notin A$, and $\ell'_p(v, w) = 0$. If such an arc exists, the algorithm scans the arc as in the NSRPP algorithm with the following modifications. If (v, w) is added to T and (v, w) is the base arc of a maximal fragment in $\mathcal{T}'$ then we add w to C_V and (v, w) to C_E. If (v, w) is added to T and w is the anti-base node of a maximal fragment in $\mathcal{T}'$, we add w to D_V and (v, w) to D_E.

If no arc out of A with zero reduced cost exists, the algorithm chooses ϵ as described below and modifies ℓ' and p by performing the *adjustment procedure* that is somewhat different from the NSRPP case: step (3) and (4) are added, compared with that described in Section 3.3. These steps are equivalent to applying the $(-\epsilon, \tau)$-transformation to every shrinking bud τ. Define $X = A - (C_V \cup D_V)$.

(1) For every $(v, w) \in C_E$, set $\ell'(v, w)$ to $\ell'(v, w) + \epsilon$, and for all $(w, z) \in \overline{E}$ with $z \notin V_\tau$, set $\ell'(w, z)$ to $\ell'(w, z) - \epsilon$.

(2) For every $(v, w) \in C_E$, set $\ell'(\sigma(v, w))$ to $\ell'(\sigma(v, w)) + \epsilon$ and for all $(\sigma(w, z)) \in \overline{E}$ with $z \notin V_\tau$, set $\ell'(\sigma(w, z))$ to $\ell'(\sigma(w, z)) - \epsilon$.

(3) For every $w' \in D_V$, let (v, w) be the base arc of the maximal fragment τ with the anti-base node w'. Set $\ell'(v, w)$ to $\ell'(v, w) - \epsilon$, and for all $(w, z) \in \overline{E}$ with $z \notin V_\tau$, set $\ell'(w, z)$ to $\ell'(w, z) + \epsilon$.

(4) For every $w' \in D_V$, let (w', v') be the anti-base arc of the maximal fragment τ with the anti-base node w'. Set $\ell'(w', v'))$ to $\ell'((w', v')) - \epsilon$, and for all $(z, w') \in \overline{E}$ with $z \notin V_\tau$, set $\ell'(\sigma(z, w'))$ to $\ell'(\sigma(z, w')) + \epsilon$.

(5) Apply the (ϵ, X)-relabeling.

(6) Apply the $(\epsilon, V - \sigma(X))$-relabeling.

The value of the *adjustment parameter* ϵ is chosen to be as large as possible without creating negative reduced cost arcs or making ϵ' negative on some τ. After the adjustment, the reduced cost of an arc is either preserved or changed by ϵ, 2ϵ, $-\epsilon$, or -2ϵ. The sets E_1^ϵ and E_2^ϵ are defined in the same way as in Section 3.3 (note that in the description of types of elements in these sets one should replace C_V by $C_V \cup D_V$ and C'_V by $C'_V \cup D'_V$). The value of ϵ is the smaller of the minimum reduced cost over the arcs in E_1^ϵ, one half of the minimum reduced cost over the arcs in E_2^ϵ, and the minimum value of ϵ'_τ over shrinking fragments τ.

If the value of ϵ is restricted by a shrinking fragment τ, then we expand the bud corresponding to τ.

It is easy to verify that the invariants (N1)–(N3) are maintained.

One can easily implement the procedure for finding a shortest strong path using the techniques developed in Sections 2 and 3. The only additional data structure needed is the third priority queue containing arcs in D_E with keys equal to the value of ϵ' on the corresponding fragments. The resulting implementation runs in $O(m)$ time plus $O(m)$ set operations and $O(m)$ priority queue operations. The analysis is the same as in the NSRPP case with the additional observation that only the fragments which are in $\mathcal{T}'$ at the beginning of the procedure are expanded, so there can be at most $|\mathcal{T}'|$, or $O(n)$, bud expansions. Using the data structures discussed in Sections 3 and 4.1, the set and priority queue operations take $O(m \log n)$ time. We can extract the primal and dual solutions in the same way as in Section 3.6. Thus we have the following result.

Lemma 4.3. *A shortest strong path can be found in $O(m \log n)$ time.*

4.3. Dual Half Integrality. In conclusion we show that if the length function ℓ is integral, the linear program (3.2-1) always has a half-integral optimal solution. (This can be also derived from a theorem on dual half-integrality for the weighted perfect matching problem.) In fact, we prove that the dual solution found by the algorithm is half-integral. In particular, this result is important for the analysis of blocking flow method for symmetric flows [19].

Lemma 4.4. *If the current functions $\epsilon : \mathcal{T} \to \boldsymbol{R}$ and $p : V \to \boldsymbol{R}$ are half-integral and $p(s')$ is integral before the adjustment procedure, then the adjustment parameter ϵ selected for the procedure is half-integral.*

Proof. Recall that the adjustment parameter ϵ is equal to either one or one-half of the reduced cost of some arc (v, w) or to ϵ'_τ for some bud τ. In the first case ϵ is half-integral because the reduced costs are half-integral.

In the second case we have two subcases: either $v \in X$, $w \in X'$ or v is a base of a current bud and w is an anti-base of another bud. We show that in this case $\ell'_p(v, w)$ is integral and so ϵ must be half-integral. Note that in both subcases $v \in A$ and $w \in A'$, so there are paths Γ_1 and Γ_2 of zero reduced cost from s to v and from w to s', respectively. Since $p(s) = 0$, we have

$$p(s') = p(s') - p(s) = \ell'_p(\Gamma_1) + \ell'_p(v, w) + \ell'_p(\Gamma_2) = \ell'_p(v, w).$$

Thus $\ell'_p(v, w)$ is integral.

In the third case ϵ_τ is half-integral and therefore the adjustment parameter is half-integral. ∎

Since the adjustment procedure increases $p(s')$ by 2ϵ, this lemma immediately implies the following theorem.

Theorem 4.5. *If the length function ℓ is integral and LP* (3.2-1) *is feasible, then the optimal solution to* (3.2-1) *found by the algorithm is half-integral.*

5. Relationship to Matching Theory

As mentioned in the introduction, the RRP can be reduced to a certain matching problem in a way similar to that described in [29]. Given a skew-symmetric graph $G = (V, E)$ with distinguished symmetric nodes s and s', we form an undirected graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ as follows. The node set $\widetilde{V}$ consists of nodes v_a corresponding to arcs $a \in E$ and two additional nodes t and t'. The edge set $\widetilde{E}$ is $M \cup Y \cup W \cup W'$, where

- M consists of edges $\{v_a, v_{\sigma(a)}\}$ for $a \in E$;
- Y consists of edges $\{v_a, v_{\sigma(b)}\}$ for $a, b \in E$ such that the head of a coincides with the tail of b;
- W consists of edges $\{t, v_{\sigma(a)}\}$ for $a \in E$ such that s is the tail of a;
- W consists of edges $\{v_a, t'\}$ for $a \in E$ such that s' is the head of a.

Note that because of the symmetry there are two copies of every edge in M and Y; we identify these edges.

Clearly M is a matching in $\widetilde{G}$ covering all nodes except t and t'. A regular path $P = (x_0, a_1, x_1, \ldots, a_k, x_k)$ from $s = x_0$ to $t = x_k$ in G, corresponds to the path $\Psi(P)$ from t to t' in $\widetilde{G}$ given by the sequence $(t, v_{\sigma(a_1)}, v_{a_1}, v_{\sigma(a_2)}, \ldots, v_{\sigma(a_k)} v_{a_k}, t)$ and $\Psi(P)$ is an alternating path (*i.e.,* for each two consecutive edges oh $\Psi(P)$, exactly one is in M). It is easy to see that Ψ gives a one-to-one correspondence between the set of simple r-paths from s to s' in G and the set of alternating paths from t to t' in $\widetilde{G}$. Therefore RRP is equivalent to the problem of finding a (simple) alternating path in $\widetilde{G}$ from t to t'. ¿From the algorithmic viewpoint, however, this reduction is expensive since $|\widetilde{V}| = |E| + 2$ and $|\widetilde{E}|$ can be significantly larger than $|E|$.

Berge's theorem [4] implies that the alternating path exists if and only if $\widetilde{G}$ has a perfect matching M'. Suppose that no perfect matching in $\widetilde{G}$ exists. Then by Tutte's theorem [27], there exists a subset $S \subseteq \widetilde{V}$ such that removing S from $\widetilde{G}$ disconnects the graph and the number of odd components, r, exceeds $|S|$. (An *odd (even) component* is a connected component with an odd (even) number of nodes.) Let $K_1, \ldots, K_r$ be the odd components of the resulting graph, let $K_{r+1}, \ldots, K_q$ be the even components, and let V_i be the nodes set of K_i. Since in our case $\widetilde{G}$ contains the matching M with $|\widetilde{V}|/2 - 1$ edges, one can easily see that

(1) $r = |S| + 2$;
(2) t and t' are contained in two different odd components;
(3) each component K_i contains $\lfloor (|V_i| - 1)/2 \rfloor$ edges of M.

Using (1)–(3), we can show (although arguments are not straightforward) that $K_1, \ldots, K_r$ enables us to define subsets $A, A', X_1, \ldots, X_{r-2}$ such that $\mathcal{B} = (A; X_1, \ldots, X_{r-2})$ is a barrier in G. This implies Theorem 2.2. We can also show that if S is chosen according to Edmonds-Gallai theorem (see *e.g.* [23]), then $\mathcal{B}$ is the canonical barrier; in this case the set Z of nodes reachable by r-paths from s corresponds, in a sense, to the set of nodes in $\widetilde{G}$ reachable by simple alternating paths from t.

Next we extend the above reduction to SRPP. First assume that the length function ℓ is nonnegative. For $e \in \widetilde{E}$, define $\widetilde{\ell}(e)$ to be zero if $e \in M$, $(\ell(a) + \ell(b))/2$ if $e = \{v_a, v_{\sigma(b)}\} \in Y$, and $\ell(a)/2$ if $e = \{t, v_{\sigma(a)}\} \in W$ or $e = \{v_a, t'\} \in W'$. Then for any r-cycle or any t to t' path P in G, $\ell(P)$ is equal to $\widetilde{\ell}(\Psi(P))$ of the "image" $\Psi(P)$ of P in $\widetilde{G}$. To solve SRPP in G, we find the minimum weight perfect matching M' in $\widetilde{G}$. Clearly the symmetric difference $M \ominus M'$ consists of a simple path Q containing t and t', and a (possibly empty) collection of pairwise disjoint cycles $C_1, \ldots, C_k$ of zero $\widetilde{\ell}$-length. Thus, for $P = \Psi^{-1}(Q)$, we have $\ell(P) = \widetilde{\ell}(Q) = \widetilde{\ell}(M')$, and P is a minimum ℓ-length r-path in G.

Now suppose ℓ is arbitrary. We first determine whether (G, ℓ) is r-conservative. To do this, we connect t and t' by an edge e_0 with a negative length of large absolute value, obtaining $\widetilde{G}'$ and $\widetilde{\ell}'$. Next we find a minimum weight perfect matching M' for $(\widetilde{G}', \widetilde{\ell}')$. By the choice of $\widetilde{\ell}'(e_0)$, M' must contain e_0. One can easily see that (G, ℓ) is r-conservative if and only if $M \ominus M'$ has no negative length cycles. If (G, ℓ) is r-conservative, we proceed in the same way as for nonnegative ℓ.

Note that the above reductions give polynomial-time algorithms for RRP and SRPP, but the running times of these algorithms are significantly worse than those of the corresponding algorithms of our paper. Also, the direct proofs of our analogs of the Berge's and Tutte's theorems seem simpler than the translations of the classical proofs of these theorems to the skew-symmetric graph domain.

Next we give applications of RRP and SRPP to matching problems.

First we consider an augmenting path problem. Suppose we are given an undirected graph $H = (V_H, E_H)$ and a matching M in it, and let $Z \subseteq V_H$ be the set of the unmatched nodes. We are interested in finding an alternating path connecting two distinct nodes in Z. Such a problem is reduced to RRP for $G = (V, E)$, s, s' as follows. The node set V is obtained by splitting each $v \in V_H$ into two nodes v_1 and v_2, and adding two more nodes, s and s'. The arc set E contains the arcs (u_1, v_2) and (v_1, u_2) for each edge $\{u, v\} \in V_H - M$, the arcs (u_2, v_1) and (v_2, u_1) for each edge $\{u, v\} \in M$, and the arcs (s, v_1) and (v_2, s') for each $v \in Z$. See Figure 4 for an example. One can see that there is a natural one-to-one correspondence between the set of alternating paths in H connecting distinct nodes of Z and the set of r-paths from s to s' in G. Note also

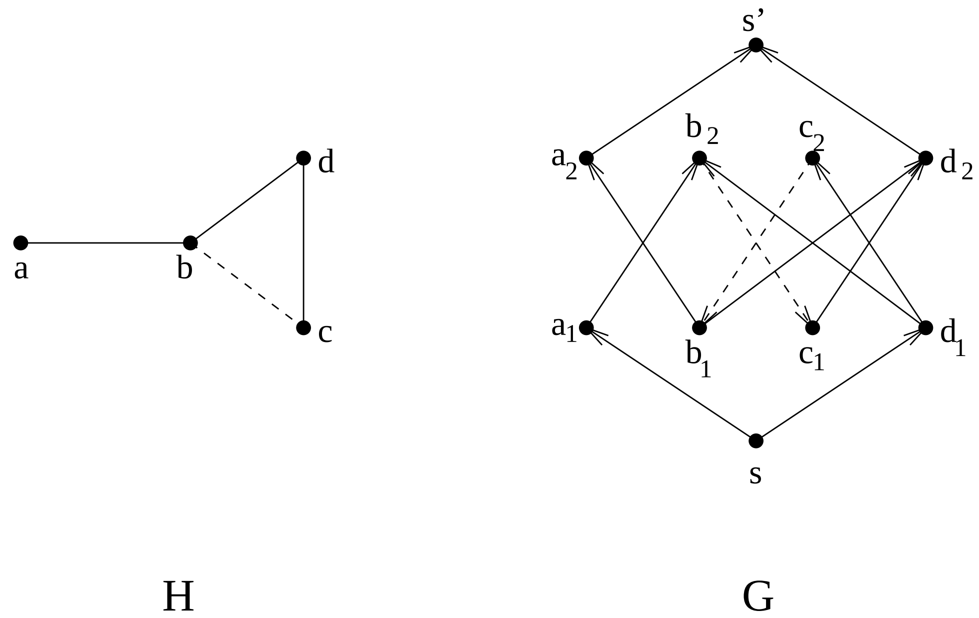

FIGURE 4. *An example of the augmenting path problem reduction.*

that $|V| = O(|V_H|)$ and $|E| = O(E_H)$, so the algorithm of Section 2 solves the augmenting path problem in linear time.

Next we consider a weighted version of the augmenting path problem. In addition to H and M, we are given a length function $\ell_H : E_H \to R$, and we wish to find an alternating path Q connecting a pair of distinct nodes of Z such that the length $\ell_H(Q)$ is minimized, or show that there is an alternating cycle C of negative ℓ_H-length. To solve this problem, we construct G as before. For each arc $a \in E$ obtained from an edge $e \in E_H$, define $\ell(a) = \ell_H(e)$, and for all other arcs $a \in E$ define $\ell(a) = 0$. Then the problem is reduced to SRPP for G, s, s', ℓ.

The weighted augmenting path problem can be used to find a shortest odd cycle and a shortest even cycle in an undirected graph. See *e.g.* [20], Chapter 8. Thus in the case of nonnegative lengths the algorithm of Section 3 can be used to solve these problems in $O(m \log n)$ and $O(m^2 \log n)$ time, respectively, where m in the number of edges and n is the number of nodes in the graph. Similarly, we can use the weighted augmenting path problem to find a shortest even (odd) path between two prescribed nodes.

In [19] we discuss the reductions of matching and bidirected flow problems to skew-symmetric flow problems. The algorithms developed in [19] use the algorithms of the present paper as subroutines.

6. Concluding Remarks

In conclusion we would like to discuss some differences in the best known bounds for the shortest paths problems and our bounds for the skew-symmetric versions of these problems.

One technical difference of our NSRPP algorithm from Dijkstra's shortest paths algorithm is that our algorithm removes some arcs from Q_1 and Q_2 by an operation other than EXTRACT-MINIMUM. Because of this difference, some faster estimates known for the standard shortest paths problem do not seem to apply in the NSRPP case.

Using a RAM model of computation that allows certain constant-time operations on words, Fredman and Willard [12] give an $O(m+n\log n/\log\log n)$ implementation of Dijkstra's algorithm. It is possible that their techniques can be used to improve our algorithms.

A scaling algorithm of Goldberg [18] solves the shortest paths problem with integral arc lengths in $O(\sqrt{n}m\log N)$ time, where $-N$ is a lower bound on the arc lengths. An interesting open problem is to extend this result to the skew-symmetric case.

References

1. R. K. Ahuja, K. Mehlhorn, J. B. Orlin, and R. E. Tarjan. Faster Algorithms for the Shortest Path Problem. Technical Report CS-TR-154-88, Department of Computer Science, Princeton University, 1988.
2. R.P. Anstee. A Polynomial Algorithm for b-Matchings: an Alternative Approach. *Information Processing Let.*, 24:153–157, 1987.
3. R. E. Bellman. On a Routing Problem. *Quart. Appl. Math.*, 16:87–90, 1958.
4. C. Berge. Two Theorems in Graph Theory. *Proc. Nat. Acad. Sci. U.S.A.*, 43:842–844, 1957.
5. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.
6. E. W. Dijkstra. A Note on Two Problems in Connection with Graphs. *Numer. Math.*, 1:269–271, 1959.
7. J. Edmonds. Maximum Matchings and a Polyhedron with 0, 1-vertices. *J. Res. Nat. Bur. Stand.*, 96B:125–130, 1965.
8. J. Edmonds. Paths, Trees and Flowers. *Canada J. Math.*, 17:449–467, 1965.
9. J. Edmonds and E. L. Johnson. Matching, a Well-Solved Class of Integer Linear Programs. In R. Guy, H. Haneni, and J. Schönhein, editors, *Combinatorial Structures and Their Applications*, pages 89–92. Gordon and Breach, NY, 1970.
10. L. R. Ford, Jr. and D. R. Fulkerson. *Flows in Networks.* Princeton Univ. Press, Princeton, NJ, 1962.
11. M. L. Fredman and R. E. Tarjan. Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *J. Assoc. Comput. Mach.*, 34:596–615, 1987.
12. M. L. Fredman and D. E. Willard. Trans-dichotomous Algorithms for Minimum Spanning Trees and Shortest Paths. In *Proc. 31st IEEE Annual Symposium on Foundations of Computer Science*, pages 719–725, 1990.
13. H. N. Gabow. Data Structures for Weighted Matching and Nearest Common Ansestors with Linking. In *Proc. 1st ACM-SIAM Symposium on Discrete Algorithms*, pages 434–443, 1990.
14. H. N. Gabow, S.N. Maheshwari, and L. Osterweil. On Two Problems in the Generation of Program Test Paths. *IEEE Trans. on Software Eng.*, SE-2:227–231, 1976.
15. H. N. Gabow and R. E. Tarjan. A Linear-Time Algorithm for a Special Case of Disjoint Set Union. *J. Comp. and Syst. Sci.*, 30:209–221, 1985.

16. H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for general graph-matching problems. *J. Assoc. Comput. Mach.*, 38:815–853, 1991.
17. Z. Galil, S. Micali, and H.N. Gabow. Priority Queues with Variable Priority and an $O(EV \log V)$ Algorithm for Finding a Maximal Weighted Matching in General Graph. In *Proc. 23th IEEE Annual Symposium on Foundations of Computer Science*, pages 255–261, 1982.
18. A. V. Goldberg. Scaling Algorithms for the Shortest Paths Problem. In *Proc. 4nd ACM-SIAM Symposium on Discrete Algorithms*, pages 222–231, 1993.
19. A. V. Goldberg and A. V. Karzanov. Skew-Symmetric Flows. Manuscript, Stanford University, 1993.
20. M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer Verlag, 1988.
21. A. V. Karzanov. A Minimum Cost Maximum Multiflow Problem. In *Combinatorial Methods for Flow Problems*, volume 3, pages 138–156. Inst. for Systems Studies, Moscow, 1979. In Russian.
22. A. V. Karzanov. Minimum Cost Multiflow in Undirected Networks. Technical Report RR 849-M, IMAG Artemis, Grenoble, 1991.
23. L. Lovás and M. D. Plummer. *Matching Theory.* Akadémiai Kiadó, Budapest, 1986.
24. S. Micali and V. V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *Proc. 21th IEEE Annual Symposium on Foundations of Computer Science*, pages 17–27, 1980.
25. E. F. Moore. The Shortest Path Through a Maze. In *Proc. of the Int. Symp. on the Theory of Switching*, pages 285–292. Harvard University Press, 1959.
26. R. E. Tarjan. Efficiency of a Good but not Linear Set Union Algorithm. *J. Assoc. Comput. Mach.*, 22:1975, 1975.
27. W.T. Tutte. The Factorization of Linear Graphs. *J. London Math. Soc.*, 22:107–111, 1947.
28. V. V. Vazirani. A Theory of Alternating Paths and Blossoms for Proving Correctness of the $O(\sqrt{V}E)$ General Graph Maximum Matching Algorithm. *Combinatorica*, to appear.
29. H. Yinnone. On Paths Avoiding Forbidden Pairs of Vertices in a Graph. Manuscript, 1991.