SIAM J. COMPUT.
Vol. 24, No. 3, pp. 494–504, June 1995

006

SCALING ALGORITHMS FOR THE SHORTEST PATHS PROBLEM*

ANDREW V. GOLDBERG†

Abstract. We describe a new method for designing scaling algorithms for the single-source shortest paths problem and use this method to obtain an $O(\sqrt{nm}\log N)$ algorithm for the problem. (Here n and m are the number of nodes and arcs in the input network and N is essentially the absolute value of the most negative arc length; arc lengths are assumed to be integral.) This improves previous bounds for the problem. The method extends to related problems.

Key words. shortest paths problem, graph theory, networks, scaling

AMS subject classifications. 68Q20, 68Q25, 68R10, 05C70

1. Introduction. In this paper we study the shortest paths problem where arc lengths can be both positive and negative. This is a fundamental combinatorial optimization problem that often comes up in applications and as a subproblem in algorithms for many network problems. We assume that the length function is integral, as is the case in most applications.

We describe a framework for designing scaling algorithms for the shortest paths problem and derive several algorithms within this framework. Our fastest algorithm runs in $O(\sqrt{nm}\log N)$ time, where n and m are the number of nodes and arcs of the input network, respectively, and the arc costs are at least $-N$.[1] Our approach is related to the cost-scaling approach to the minimum-cost flow problem [2], [14], [18], [21].

Previously known algorithms for the problem are as follows. The classical Bellman–Ford algorithm [1], [8] runs in $O(nm)$ time. Our bound is better than this bound for $N = o(2^{\sqrt{n}})$. Scaling algorithms of Gabow [12] and Gabow and Tarjan [13] are dominated by an assignment subroutine. The former algorithm runs in $O(n^{3/4}m\log N)$ time; the latter algorithm runs in $O(\sqrt{nm}\log(nN))$ time.[2] Our bound dominates these bounds. The fastest shortest paths algorithm currently known for planar graphs [9], [19] runs in $O(n^{1.5})$ time. Our algorithm runs in $O(n^{1.5}\log N)$ time on planar graphs and is competitive for small values of N.

Our framework is very flexible. In §§8 and 9 we describe two variations of the $O(\sqrt{nm}\log N)$ algorithm. The first variation seems more practical and the second variation shows the relationship between our method and Dijkstra's shortest path algorithm [6]. The flexibility of our method may lead to better running time bounds.

The shortest paths problem is closely related to other problems, such as the minimum-cost flow, assignment, and minimum-mean length cycle problems. Our method for the shortest paths problem extends to these problems. In §10 we sketch extensions to the minimum-cost flow and assignment problems. McCormick [20] shows an extension to the minimum-mean cycle problem. The resulting algorithms achieve bounds that are competitive with those of the fastest known algorithms, but are somewhat simpler.

2. Preliminaries. The input to the single-source shortest paths problem is (G, s, l), where $G = (V, E)$ is a directed graph, $l : E \to \mathbf{R}$ is a length function, and $s \in V$ is the

*Received by the editors May 28, 1992; accepted for publication (in revised form) November 22, 1993.

†Computer Science Department, Stanford University, Stanford, California 94305. This research was supported in part by the Office of Naval Research Young Investigator award N00014-91-J-1855; National Science Foundation Presidential Young Investigator grant CCR-8858097 with matching funds from AT&T, DEC, and 3M; Powell Foundation grant; and a Mitsubishi Electric Laboratories grant. Part of this work was done while the author was visiting IBM Almaden Research Center and supported by Office of Naval Research contract N00014-91-C-0026.

[1] We assume that $N \geq 2$ so that $\log N > 0$.

[2] In [12], [13] these bounds are stated in terms of C, the maximum absolute value of arc costs. As noted by an anonymous referee, it is easy to see that C can be replaced by N.

source node (see, e.g., [4], [23]). The goal is to find shortest paths distances from s to all other nodes of G or to find a negative length cycle in G. If G has a negative length cycle, we say that the problem is *infeasible*. We assume that the length function is integral. We also assume, without loss of generality, that all nodes are reachable from s in G and that G has no multiple arcs. The latter assumption allows us to refer to an arc by its endpoints without ambiguity.

We denote $|V|$ by n and $|E|$ by m. Let M be the smallest arc length. Define $N = -M$ if $M < -1$ and $N = 2$ otherwise. Note that $N \geq 2$ and $l(a) \geq -N$ for all $a \in E$.

A *price function* is a real-valued function on nodes. Given a price function p, we define a *reduced cost function* $l_p : E \to \mathbf{R}$ by

$$l_p(v, w) = l(v, w) + p(v) - p(w).$$

We say that a price function p is *feasible* if

$$l_p(a) \geq 0 \quad \forall a \in E. \tag{1}$$

For an $\epsilon \geq 0$, we say that a price function is *ϵ-feasible* if

$$l_p(a) > -\epsilon \quad \forall a \in E. \tag{2}$$

Given a price function p, we say that an arc a is *admissible* if $l_p(a) \leq 0$, and denote the set of admissible arcs by E_p. The *admissible graph* is defined by $G_p = (V, E_p)$.

If the length function is nonnegative, the shortest paths problem can be solved in $O(m + n \log n)$ time [10], or in $O(m + n \log n / \log\log n)$ time [11] in a random access machine computation model that allows certain word operations. We call such a problem *Dijkstra's shortest paths problem* [6]. Given a feasible price function p, the shortest paths problem can be solved as follows. Let d be a solution to the Dijkstra's shortest paths problem (G, s, l_p). Then the distance function d' defined by $d'(v) = d(v) + p(v) - p(s)$ is the solution to the input problem.

We restrict our attention to the problem of computing a feasible price function or finding a negative length cycle in G.

3. Successive approximation and bit scaling frameworks. Our method computes a sequence of ϵ-feasible price functions with ϵ decreasing by a factor of two at each iteration. Initially, all the prices are zero and ϵ is the smallest power of two that is greater than N. The method maintains integral prices. At each iteration, the method halves ϵ and applies the REFINE subroutine, which takes as input a (2ϵ)-feasible price function and returns an ϵ-feasible price function or discovers a negative length cycle. In the latter case, the computation halts.

LEMMA 3.1. *Suppose a price function p is integral and 1-feasible. Then for every $a \in E$, $l_p(a) \geq 0$.*

Proof. The lemma follows from the fact that $l_p(a)$ is integral and $l_p(a) > -1$. □

Bit scaling, first applied to the shortest paths problem by Gabow [12], can be used instead of successive approximation in all algorithms described in this paper. The bit scaling version of our method rounds lengths *up* to a certain precision, initially the smallest power of two that is greater than N. The lengths and prices are expressed in the units determined by the precision. Note that since the lengths are rounded up, a negative cycle with respect to the rounded lengths is also negative with respect to the input lengths.

Each iteration of the algorithm starts with a price function that is feasible with respect to the current (rounded) lengths. Note that this is true initially because of the choice of the initial unit. At the beginning of an iteration, the lengths and prices are multiplied by two, and one is subtracted from the arc lengths as appropriate to obtain the higher precision. The resulting price function is 1-feasible with respect to the current length function; the feasibility

is restored using REFINE. The method terminates when the precision unit becomes 1, which happens in $O(\log N)$ iterations. Note that the basic problem solved at each iteration of the bit scaling method is a special version of the shortest paths problem where the arc lengths are integers greater or equal to -1.

The following lemma is obvious.

LEMMA 3.2. *Both the successive approximation and the bit scaling methods terminate in* $O(\log N)$ *iterations.*

Note that if the current unit in the bit scaling method is U and the current price function is feasible with respect to the rounded length function, then the price function is U-feasible with respect to the input length function. Thus bit scaling can be viewed as a special case of successive approximation. The work on the minimum-cost flow problem [18] shows that successive approximation is more general than bit scaling; in particular, the former can be easily used to obtain strongly polynomial algorithms.

We describe bit scaling version in the algorithms. This allows us to avoid certain technical details and slightly simplifies the presentation. However, all algorithms can be restated in the successive approximation framework in a straightforward way.

When describing bit scaling implementations of REFINE, we denote the current rounded length function by l. We also use the following definitions. We call an arc (v, w) *improvable* if $l_p(v, w) = -1$, and we call a node w *improvable* if there is an improvable arc entering w.

4. Dealing with admissible cycles. Suppose that G_p has a cycle Γ. Since the reduced cost of a cycle is equal to the length of the cycle, $l(\Gamma) \leq 0$.

If $l(\Gamma) < 0$, or $l(\Gamma) = 0$ and there is an arc (v, w) such that $l_p(v, w) < 0$ and both v and w are on Γ, then the input problem is infeasible and the method terminates. Otherwise, we contract Γ and remove self-loops adjacent to the contracted node. A feasible price function on the contracted graph extends to a feasible price function on the original graph in a straightforward way.

Our algorithm uses an $O(m)$-time subroutine DECYCLE(G_p) that works as follows. Find strongly connected components of G_p (see, e.g., [22]); if a component contains a negative reduced cost arc, G has a negative length cycle; otherwise contract each component. (Note that the prices of nodes in each contracted component change by the same amount, so the reduced costs of arcs with both ends in the same component do not change.)

Suppose G_p is acyclic. Then G_p defines a partial order on V and on the subset of improvable nodes. This motivates the following definitions. A set of nodes S is *closed* if every node reachable in G_p from a node in S belongs to S. A set of nodes (arcs) S is a *chain* if there is a path in G_p containing every element of S.

5. Cut-relabel operation. In this section we study the CUT-RELABEL operation which is used by our method to transform a 1-feasible price function into a feasible one. The CUT-RELABEL operation takes a closed set S and decreases prices of all nodes in S by 1.[3] Note that the operation preserves integrality of the prices (and therefore integrality of the reduced costs).

LEMMA 5.1. *The* CUT-RELABEL *operation does not create any improvable arcs.*

Proof. The only arcs whose reduced cost is decreased by CUT-RELABEL are the arcs leaving S. Let a be such an arc. The relabeling decreases $l_p(a)$ by 1. Before the relabeling, S is closed and therefore $l_p(a) > 0$. By integrality, $l_p(a) \geq 1$. After the relabeling, $l_p(a) \geq 0$. □

The above lemma implies that CUT-RELABEL does not create improvable nodes. The next lemma shows how to use this operation to reduce the number of improvable nodes.

[3] Alternatively, the operation can decrease prices of all nodes of S by the maximum amount ϵ' such that Lemma 5.1 holds.

LEMMA 5.2. *Let p be a 1-feasible price function. Let S be a closed set of nodes, and let $X \subseteq S$ be a set of improvable nodes such that every improvable arc entering a node of X crosses the cut defined by S. After the set S is relabeled, nodes in X are no longer improvable.*

Proof. Let p' be the price function after the relabeling. Let $w \in X$ and let (v, w) be an improvable arc with respect to p. By the statement of the lemma, $v \notin S$. Thus the relabeling increases l_p by 1, and, by 1-feasibility of p, $l_{p'}(v, x) \geq 0$. □

A simple algorithm based on CUT-RELABEL applies the following procedure to every improvable node v.

1. DECYCLE(G_p).
2. $S \leftarrow$ set of nodes reachable from $\{v\}$ in G_p.
3. CUT-RELABEL(S).

It is easy to see that given a 1-feasible price function, this algorithm computes a feasible one in $O(nm)$ time.

6. Faster algorithm. In this section we introduce an $O(\sqrt{n}m \log N)$ algorithm for finding a feasible price function. Let k denote the number of improvable nodes. At each iteration, the algorithm either finds a closed set S such that applying CUT-RELABEL to S reduces the number of improvable nodes by at least $\sqrt{k}$, or a chain S such that applying ELIMINATE-CHAIN to S reduces the number of improvable nodes by at least $\sqrt{k}$. (The ELIMINATE-CHAIN operation is described in the next section.) An iteration takes linear time and is based on the results of §§5 and 7 and the following lemma, which is related to Dilworth's theorem (see, e.g., [7]).

LEMMA 6.1. *Suppose G_p is acyclic. Then there exists a chain $S \subseteq E$ such that S contains at least $\sqrt{k}$ improvable arcs or a closed set $S \subseteq V$ such that relabeling S reduces the number of improvable nodes by at least $\sqrt{k}$. Furthermore, such an S can be found in $O(m)$ time.*

Proof. Construct a graph G' by adding a source node r to G_p and arcs from r to all nodes in V. Note that G' is acyclic. Define $l'(a) = l_p(a)$ for all $a \in E_p$ and $l'(a) = 0$ for the newly added arcs a. The absolute value of the path length with respect to l' is equal to the number of improvable arcs on the path. Let $d' : V \to \mathbf{R}$ give the shortest paths distances from r with respect to l' in G'. Since G' is acyclic, d' can be computed in linear time. Define $D = \max_v |d'|$.

If $D \geq \sqrt{k}$, then a shortest path from r to a node v with $d'(v) = -D$ contains a chain with at least $\sqrt{k}$ improvable arcs.

If $D < \sqrt{k}$, then the partitioning of the set of improvable nodes according to the value of d' on these nodes contains at most $\sqrt{k}$ nonempty subsets. Let X be a subset containing the maximum number of improvable nodes and let i be the value of d' on X. Observe that X contains at least $\sqrt{k}$ improvable nodes. Define $S = \{v \in V | d'(v) \leq i\}$.

Clearly $X \subseteq S$. Also, S is closed. This is because if $v \in S$ and there is a path from v to w in G_p, then the length of this path with respect to l' is nonpositive, so $d'(w) \leq d'(v) \leq i$ and therefore $w \in S$.

We show that after CUT-RELABEL is applied to S, nodes in X are no longer improvable. Let $x \in X$ and let (v, x) be an improvable arc. Then $l'(v, x) = -1$ and therefore $d'(v) > d'(x) = i$. Thus $v \notin S$ and (v, w) is not improvable after relabeling of S. □

The efficient implementation of REFINE is described in Fig. 1. The implementation reduces the number of improvable nodes k by at least $\sqrt{k}$ at each iteration by eliminating cycles in G_p, finding S as in Lemma 6.1, and eliminating at least $\sqrt{k}$ improvable nodes in S using techniques of §§4, 5, and 7. In §7 below we describe a linear time implementation of ELIMINATE-CHAIN. This implies that an iteration REFINE runs in linear time.

LEMMA 6.2. *The implementation of* REFINE *described in this section runs in $O(\sqrt{n}m)$ time.*

```
procedure REFINE(p);
    k ← the number of improvable nodes;
    repeat
        DECYCLE(G_p);
        S ← a chain or a set as in Lemma 6.1;
        if S is a chain then
          ELIMINATE-CHAIN(S);
        else
          CUT-RELABEL(S);
        k ← the number of improvable nodes;
    until k = 0;
    return(p);
end.
```

FIG. 1. *An efficient implementation of* REFINE.

Proof. We need to bound the number of iterations of REFINE. Each iteration reduces k by at least $\sqrt{k}$, and $O(\sqrt{k})$ iterations reduce k by at least a factor of two. The total number of iterations is bounded by

$$\sum_{i=0}^{\infty} \sqrt{\frac{n}{2^i}} = O(\sqrt{n}). \qquad \square$$

Lemmas 3.2 and 6.2 imply the following result.

THEOREM 6.3. *The shortest paths algorithm with* REFINE *implemented as described in this section runs in* $O(\sqrt{n}m \log N)$ *time.*

7. Eliminate-chain subroutine. Suppose that G_p is acyclic and let Γ be a path in G_p. Let $(v_1, w_1), \ldots, (v_t, w_t)$ be the collection of all improvable arcs on Γ such that for $1 \le i < j \le t$, the path visits v_j before v_i (i.e., v_1 is visited last). By definition, nodes $w_1, \ldots, w_t$ are improvable. In this section we describe a subroutine ELIMINATE-CHAIN that modifies p so that the nodes $w_1, \ldots, w_t$ are no longer improvable and no new improvable nodes are created, or finds a negative length cycle in G. The subroutine runs in $O(m)$ time.

At iteration i, ELIMINATE-CHAIN finds the set S_i of all nodes reachable from w_i in the admissible graph and applies CUT-RELABEL to S_i. If w_i is improvable after the relabeling, the algorithm concludes that the problem is infeasible.

LEMMA 7.1. *The path* Γ *is always admissible. If* w_i *is improvable after iteration* i*, then the problem is infeasible.*

Proof. The price function is modified only by CUT-RELABEL. At iteration i, S_i contains w_i, all its successors on Γ, and no other nodes of Γ (by induction on i). Therefore $l_p(v_i, w_i)$ changes exactly once during iteration i, when it increases by 1. The arc (v_i, w_i) is improvable before the change, and admissible after the change. Reduced costs of other arcs on Γ do not change during the execution of ELIMINATE-CHAIN.

Suppose w_i is improvable immediately after iteration i. Then there must be a node v such that (v, w_i) is improvable and $v \in S_i$. By construction of S_i, there must be an admissible path from w_i to v. This path together with the arc (v, w_i) forms a negative length cycle. $\square$

Lemmas 5.1 and 7.1 imply that the implementation of ELIMINATE-CHAIN is correct. Next we show how to refine this implementation to achieve $O(m)$ running time. The key fact that allows such an implementation is that the sets S_i are nested.

First, we contract the set of nodes S_i at every iteration. The reason for contracting is to allow us to change the prices of nodes in S_i efficiently (these prices change by the same

amount). The CONTRACT(S_i) operation collapses all nodes of S_i into one node s_i and assigns the price of the new node to be zero. (The price of s_i is actually an increment to the prices of the nodes in S_i.) Reduced costs of the arcs adjacent to the new node remain the same as immediately before CONTRACT. Note that we have at most one contracted node at any point during ELIMINATE-CHAIN, but contracted nodes can be nested.

The UNCONTRACT(s_i) operation, applied to a contracted node s_i, restores the graph as it was just before the corresponding CONTRACT operation and adds $p(s_i)$ to prices of all nodes in S_i. At the end of the chain elimination process, we apply UNCONTRACT until the original graph is restored.

Contraction is used for efficiency only and does not change the price function computed by ELIMINATE-CHAIN, because by Lemma 7.1 $S_i \subseteq S_j$ for $1 \leq i < j \leq t$.

Second, we implement the search for the nodes reachable from w_i's in the admissible graph in a way similar to Dial's implementation [5] of Dijkstra's algorithm.[4] Our implementation uses a priority queue that holds items with integer key values in the range $[0, \ldots, 2n]$; the amortized cost of the priority queue operations is constant. We assume the following queue operations.

- enqueue(v, Q): add a node v to a priority queue Q.
- min(Q): return the minimum key value of elements on Q.
- extract-min(Q): remove a node with the minimum key value from Q.
- decrease-key(v, x): decrease the value of key(v) to x.
- shift(Q, δ): add δ to the key values of all elements of Q.

All of these operations except shift are standard; a constant time implementation of shift is trivial.

Note that if p is 1-feasible and $l_p(a) \geq 2n$, then a can be deleted from the graph. This is because in the current iteration, the reduced cost of an arc can decrease by at most n; at the next iteration, by at most $n/2$ (measured in the current units), and so on. Thus the reduced cost of a will remain nonnegative from now on. We assume that such arcs are deleted as soon as their reduced costs become large enough.

We define the key assignment function h that maps reduced costs into integers as follows.

$$h(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{otherwise.} \end{cases}$$

During the chain elimination computation, each node is *unlabeled*, *labeled*, or *scanned*. Unlabeled nodes have infinite keys; other nodes have finite keys. The priority queue Q contains labeled nodes. Initially all nodes are unlabeled. At the beginning of iteration i, key(w_i) is set to zero and w_i is added to Q. While Q is not empty and the minimum key value of the queue nodes is zero, a node with the minimum key value is extracted from the queue and scanned as in Dijkstra's algorithm except that $h(l_p(a))$ is used instead of $l_p(a)$ (see Fig. 2). When this process stops, the scanned nodes are contracted, the new node is marked as scanned, and its key is set to zero. Then the price of the new node is decreased by 1 and shift(Q, -1) is executed. This concludes iteration i.

Next we prove correctness of the implementation.

LEMMA 7.2. *The sets S_i are computed correctly for every $i = 1, \ldots, t$.*

Proof. For convenience we define $S_0 = \emptyset$. Consider an iteration i. It is enough to show that S_i is correct if $1 \leq i \leq t$ and S_{i-1} is correct.

Let v be a node on Q with the zero key value. We claim that v is reachable from w_i in the current admissible graph. To see this, consider two cases. If v was a node on Q with zero key

[4] In §9 we show that Dial's implementation can be used directly. The implementation described in this section, however, gives a better insight into the method.

```
procedure SCAN(v);
    for all (v, w) do
        if key(w) = ∞ then
            mark w as labeled;
            key(w) ← l_p(v, w);
            insert(w, Q);
        else if w is labeled and key(w) < h(l_p(v, w)) then
            decrease-key(w, l_p(v, w));
    mark v as scanned;
end.
```

FIG. 2. *The scan operation.*

value at the beginning of the iteration, then v is reachable from w_i by Lemma 7.1. Otherwise, key of v became zero when an arc (u, v) was scanned. We can make an inductive assumption that u is reachable from w_i. By definition of h, $h(u, v) = 0$ implies that $l_p(u, v) \leq 0$, and therefore v is reachable from w_i.

Let Γ be an admissible path originating at w_i. It is easy to see by induction on the number of arcs on Γ that all nodes on Γ are scanned and added to S_i.

It follows that at the end of iteration i, S_i contains all nodes reachable from w_i in the admissible graph. □

LEMMA 7.3. ELIMINATE-CHAIN *runs in* $O(m)$ *time.*

Proof. Each node is scanned at most once because a scanned node is marked as such and never added to Q. A contracted node is never scanned. The time to scan a (noncontracted) node is proportional to degree of the node, so the total scan time is $O(m)$.

The time of a CONTRACT operation is $O(1 + n')$, where n' is the number of nodes being contracted. The number of CONTRACT operations is at most n and the sum of n' values over all CONTRACT operations is at most $2n$. Thus the total cost of contract operations is $O(n)$.

The cost of an UNCONTRACT operation is $O(1 + n')$, where n' is the same as in the corresponding CONTRACT operation. Thus the total time for these operations is $O(2n)$. □

8. Alternative chain elimination. In this section we describe an algorithm based on an alternative implementation of REFINE. We call this implementation REFINE-P. The algorithm runs in $O(\sqrt{nm} \log N)$ time.

REFINE-P works in iterations, which we call *passes*. At the beginning of every pass we check for negative cycles and eliminate zero length admissible cycles using DECYCLE. Then we compute distances d' defined in the proof of Lemma 6.1. Given a nonnegative integer M, we define the *key function*

$$\delta(v) = \min(-d'(v), M) \quad \forall v \in V.$$

(We discuss the choice of initial value of M later.) Sometimes we refer to $\delta(v)$ as the *key* of v. Let V_M denote the set of nodes with key value M. At each iteration of a pass, CUT-RELABEL is applied to V_M. Then keys of nodes in V_M and all nodes reachable from V_M in the admissible graph are changed to $M - 1$ and M is decreased by one. This process is repeated until M reaches zero; at this point the pass terminates. A pass can be implemented to run in linear time; the implementation is similar to that of ELIMINATE-CHAIN. We leave the details to the reader.

The next lemma implies that CUT-RELABEL in used correctly in a pass.

LEMMA 8.1. *Immediately before a* CUT-RELABEL *operation is applied by a pass,* V_M *is closed with respect to the current admissible graph.*

Proof. Before the first CUT-RELABEL operation, V_M is closed by of the definition of δ. The admissible graph is changed only by the CUT-RELABEL operations, and after every such operation a search is done to enforce the closeness of V_M. □

Note that the function d' is well defined if the admissible graph does not have negative cycles.

LEMMA 8.2. *If at the beginning of an iteration of a pass the admissible graph is acyclic, then*

$$\delta(v) = \min(-d'(v), M) \quad \forall v \in V.$$

Proof. The proof is by induction on the number of iterations. Keys are initialized so that the statement of the lemma holds before the first iteration. Suppose that the statement is true immediately before iteration i, and show that it holds immediately after the iteration.

The d' value of nodes in V_M increases by one, and the keys of these nodes are decreased by one at the end of the iteration. The d' values of a node outside V_M changes only if this node becomes reachable from V_M in the admissible graph, in which case the new d' value of this node is $-(M-1)$ or less. The keys of the nodes that become reachable are correctly set to $M-1$. □

Recall that $D = \max_V |d'|$.

LEMMA 8.3. *Suppose that the value of M at the beginning of a pass is equal to t such that $0 < t \leq D$, and the admissible graph does not contain negative cycles throughout the pass. Then the pass decreases the number of improvable nodes by at least t.*

Proof. Given $v, w \in V$, we say that $v \succ w$ if there is a negative reduced cost path from v to w in the admissible graph. If the admissible graph does not contain negative cycles, then "$\succ$" defines a partial order on V.

Consider the beginning of an iteration of a pass. Let v be a maximum element (with respect to "$\succ$") of the set of nodes with key value M. By the previous lemma, v is an improvable node. By the choice of v, if (u, v) is an improvable arc then $u \notin V_M$. Therefore v is no longer improvable at the end of the iteration.

Each iteration of the pass reduces the number of improvable nodes, and the number of iterations is t. □

Next we discuss the choice of initial value of M. Define d_i to be the number of improvable nodes with d' value of $-i$ (in the beginning of a pass). If the initial value of M is i, $0 < i \leq D$, and there are no negative cycles, the number of improvable nodes is reduced by at least d_i by the first application of CUT-RELABEL. Combining this observation with the above lemma, we conclude that the pass reduces the number of improvable nodes by $\max(i, d_i)$. A more careful analysis shows that the improvement is at least $i + d_i - 1$, since all improvable nodes with an initial d' value of i and at least one improvable node for each value of j, $0 < j < i$, are no longer improvable after a pass. Define $k_i = i + d_i - 1$, and set M to the index that maximizes k_i. By an argument of Lemma 6.1, $k_M = \Omega(\sqrt{n})$. This implies the following theorem.

THEOREM 8.4. *With the above choice of the initial value of M, the alternative implementation of* REFINE *runs in $O(\sqrt{n}m)$ time.*

We would like to note that in practice, a pass is likely to reduce the number of improvable nodes by more then k_i, and it may be more advantageous to chose higer initial values for M. The algorithm performance is likely to be better than the above worst-case bound suggests.

9. Chain elimination using Dijkstra's algorithm. In this section we show yet another implementation of ELIMINATE-CHAIN. This implementation uses Dial's implementation of Dijkstra's algorithm [5], and does not use the CUT-RELABEL operation explicitly.

Let Γ be a path in G_p. An auxiliary network A is defined as follows.

- Let d' be the distance function on Γ with respect to l_p from the beginning of Γ to all nodes on Γ.

- Define $l'(a) = \max(0, l_p(a))$.
- Define $d'(v) = 0$ for v not on Γ.
- Add a source node t, connect t to all $v \in V$ and define $l'(t, v) = n + d'(v)$.

ELIMINATE-CHAIN works as follows.

1. Construct the auxiliary network A.
2. Compute shortest paths distances d in A with respect to l'.
3. $\forall v \in V$, $p'(v) \leftarrow p(v) + d(v) - n$.
4. Replace p by p'.

LEMMA 9.1. *The above version of* ELIMINATE-CHAIN *can be implemented to run in linear time.*

Proof. The fact that all steps of ELIMINATE-CHAIN except for the shortest paths computation take linear time is obvious. The shortest paths computation takes linear time if Dial's implementation [5] of Dijkstra's algorithm is used. This is because l' is nonnegative and the source is connected to the other nodes by arcs of length at most n. □

LEMMA 9.2.

1. *p' is integral.*
2. *$\forall a \in E$, $l_{p'} \geq -1$.*
3. ELIMINATE-CHAIN *does not create improvable arcs.*

Proof. The first claim follows from the fact that l' is integral. The last two claims follow from the observation that l'_d is nonnegative and, for $a \in E$, $l'_d(a) - l_{p'}(a) = 1$ if a is improvable and 0 otherwise. □

LEMMA 9.3. *If the problem is feasible, then $\forall v$ on Γ $p'(v) = p(v) + d'(v)$.*

Proof. Clearly $p'(v) \leq p(v) + d'(v)$. Assume for contradiction that for some node v on Γ, $p'(v) < p(v) + d'(v)$. For the shortest path P in A from t to v, we have $l'(P) < n + d'(v)$ and therefore $l_p(P) < n + d'(v)$. Let (t, w) be the first arc of P, and let Q be P with (t, w) deleted. We have

$$l_p(Q) = l_p(P) - n - d'(w) < d'(v) - d'(w).$$

Note that since l' is nonnegative, w must be a successor of v on Γ. Let R be the part of Γ between v and w. By the definition of d',

$$l_p(R) = d'(w) - d'(v).$$

Thus $l_p(Q) + l_p(R) < 0$. This is a contradiction because the paths Q and R form a cycle. □

LEMMA 9.4. *If the problem is feasible and v is an improvable node on Γ with respect to p, then v is not improvable with respect to p'.*

Proof. Assume for contradiction $\exists (u, v) \in E : l_{p'}(u, v) < 0$. Let P be the shortest path in A from t to u, let (t, w) be the first arc on P, and let Q be P with (t, w) deleted. Note that $d(u) \leq d(v)$, because otherwise $l_{p'}(u, v)$ cannot be negative. Therefore w must be a successor of v on Γ. Let R be the portion of Γ between v and w.

Since Q is a shortest path, we have $l'_d(Q) = 0$. This implies $l_{p'}(Q) \leq 0$. By the previous lemma $l_{p'}(R) = 0$. Therefore the cycle formed by R, Q, and (u, v) has a negative reduced cost with respect to p'. This is a contradiction. □

Remark. Implications of Lemma 9.4 are stronger than those of Lemma 7.1: if the problem is feasible, the former lemma guarantees that all improvable nodes on Γ are "fixed," and the latter guarantees only that the nodes that are heads of the improvable arcs on Γ are "fixed."

10. Extensions to the minimum-cost circulation and assignment problems. Our shortest path method extends to the minimum-cost circulation problem. The intuitive difference is that when a shortest path algorithm finds a negative cycle, it terminates; when the corresponding minimum-cost circulation algorithm finds a negative cycle, it increases the flow around the

cycle so that an arc on the cycle becomes saturated, and continues. In our discussion below, we assume that the reader is familiar with [17], [18]. We denote the reduced costs by c_p and the residual graph by G_f.

We define admissible arcs to be residual arcs with negative reduced costs, as in [17], [18]. Without loss of generality, we assume that a feasible initial circulation is available. A simple algorithm based on the CUT-RELABEL operation does the following at each iteration. First, it cancels admissible cycles; this can be done in $O(m \log n)$ time (see, e.g., [17]). Next, the algorithm picks an improvable node v, finds the set S of nodes reachable from v in the admissible graph, and executes CUT-RELABEL(S). The resulting algorithm runs in $O(nm \log n \log(nC))$ time (note that the initial flow may have residual arcs with reduced cost of $-C$ with respect to the zero price function). We can also use the TIGHTEN operation to obtain a minimum-cost flow algorithm with the same running time. These algorithms are variations of the tighten-and-cancel algorithms of [17].

In the above minimum-cost flow algorithms, the admissible graph changes due to flow augmentations in addition to price changes. Because of this fact, our analysis of the improved algorithms for the shortest paths problem does not seem to extend to the minimum-cost flow problem. In the special case of the assignment problem, the analysis of the improved shortest path algorithm can be extended to obtain an $O(\sqrt{n}m \log(nC))$ time algorithm. This bound matches the fastest known scaling bound [13], but the algorithm is different. The idea is to define the admissible graph and improvable arcs so that an improvable node has exactly one improvable arc going into it and the residual capacity of this arc is one. This is possible because of the special structure of the assignment problem. When an admissible cycle is canceled, all improvable arcs on this cycle are saturated and there are no improvable nodes on the cycle after the cancellation.

11. Concluding remarks. We described a framework for designing scaling algorithms. The CUT-RELABEL operation can be used to design algorithms within this framework. The framework is very flexible and can be used to design numerous algorithms for the problem. Using these results, we improved the time bound for the problem. We believe that further investigation of this framework is a promising research direction.

One can apply the version of ELIMINATE-CHAIN described in §9 without using scaling. It can be shown that in this case if the problem is feasible, all negative reduced costs of arcs on Γ are changed to nonnegative ones, and reduced costs of other arcs do not become more negative. This suggests a possibility of solving the general shortest paths problem in $O(\sqrt{m})$ Dijkstra shortest paths computations. The problem, however, is that our way of dealing with the first case of Lemma 6.1 does not work without scaling.

Our definition of ϵ-feasibility corresponds to that of ϵ-optimality for minimum cost flows [14], [18]. If one follows [14], [18] faithfully, however, one would define ϵ-feasibility using $l_p(a) \geq -\epsilon$ instead of (2) and not consider arcs with zero reduced costs admissible. Under these definitions, the admissible graph cannot have zero length cycles, so there is no need for DECYCLE. However, these definitions seem to lead to an $O(\log(nN))$ bound on the number of iterations of the scaling loop of the method. The *tighten* operation described in [17] also leads to an implementation of the method that runs in $O(\log(nN))$ iterations of the scaling loop.

The techniques introduced in this paper have a practical impact. In particular, the techniques of §8 proved to be crucial in our implementation of price update computation in a minimum-cost flow algorithm [15], which resulted in a significant improvement of performance.

Preliminary experiments with the algorithm of this paper, conducted as a part of the experimental study described in [3], suggest that the algorithm is not the best one to use in

practice. Although on some problem families the algorithm significantly outperformed the classical methods, it was dominated by the algorithm of [16] on all problem classes studied.

The algorithms we discussed scale ϵ by a factor of two. Any factor greater than one can be used instead without affecting the asymptotic time bounds. The method can be modified to maintain a tentative shortest path tree. When the algorithm terminates, this tree is the shortest path tree. This eliminates the need for the Dijkstra computation at the end of the algorithm.

Acknowledgments. I am grateful to Tomasz Radzik for suggesting an important idea for the proof of Lemma 6.2 and the stronger statement of Lemma 9.4, and to Bob Tarjan for suggesting a clean implementation of DECYCLE. I would also like to thank Serge Plotkin, Éva Tardos, and David Shmoys for useful discussions and comments on a draft of this paper.

REFERENCES

[1] R. E. BELLMAN, *On a routing problem*, Quart. Appl. Math., 16 (1958), pp. 87–90.
[2] R. G. BLAND AND D. L. JENSEN, *On the computational behavior of a polynomial-time network flow algorithm*, Math. Programming, 54 (1992), pp. 1–41.
[3] B. V. CHERKASSKY, A. V. GOLDBERG, AND T. RADZIK, *Shortest Paths Algorithms: Theory and Experimental Evaluation*, Technical report STAN-CS-93-1480, Department of Computer Science, Stanford University, Stanford, CA, 1993.
[4] T. H. CORMEN, C. E. LEISERSON, AND R. L. RIVEST, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.
[5] R. B. DIAL, *Algorithm 360: Shortest path forest with topological ordering*, Comm. ACM, 12 (1969), pp. 632–633.
[6] E. W. DIJKSTRA, *A note on two problems in connection with graphs*, Numer. Math., 1 (1959), pp. 269–271.
[7] R. P. DILWORTH, *A decomposition theorem for partially ordered sets*, Ann. Math., 51 (1950), pp. 161–166.
[8] L. R. FORD, JR. AND D. R. FULKERSON, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1962.
[9] G. FREDERICKSON, *Fast algorithms for shortest paths in planar graphs, with applications*, SIAM J. Comput., 16 (1987), pp. 1004–1022.
[10] M. L. FREDMAN AND R. E. TARJAN, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. Assoc. Comput. Mach., 34 (1987), pp. 596–615.
[11] 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, 1990, pp. 719–725.
[12] H. N. GABOW, *Scaling algorithms for network problems*, J. Comput. Systems Sci., 31 (1985), pp. 148–168.
[13] H. N. GABOW AND R. E. TARJAN, *Faster scaling algorithms for network problems*, SIAM J. Comput., 18 (1989), pp. 1013–1036.
[14] A. V. GOLDBERG, *Efficient Graph Algorithms for Sequential and Parallel Computers*, Ph.D. Thesis, M.I.T., Cambridge, MA, January 1987 (Also available as Technical report TR-374, Lab. for Computer Science, M.I.T., 1987).
[15] ———, *An efficient implementation of a scaling minimum-cost flow algorithm*, in Proc. 3rd Integer Prog. and Combinatorial Opt. Conf., 1993, pp. 251–266.
[16] A. V. GOLDBERG AND T. RADZIK, *A heuristic improvement of the Bellman-Ford algorithm*, Appl. Math. Lett., 6 (1993), pp. 3–6.
[17] A. V. GOLDBERG AND R. E. TARJAN, *Finding minimum-cost circulations by canceling negative cycles*, J. Assoc. Comput. Mach., 36 (1989), pp. 873–886.
[18] ———, *Finding minimum-cost circulations by successive approximation*, Math. Oper. Res., 15 (1990), pp. 430–466.
[19] R. J. LIPTON AND R. E. TARJAN, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 177–189.
[20] S. T. MCCORMICK, *Approximate binary search algorithms for mean cuts and cycles*, Oper. Res. Lett., 14 (1993), pp. 129–132.
[21] H. RÖCK, *Scaling techniques for minimal cost network flows*, in U. Pape, ed., Discrete Structures and Algorithms, Carl Hansen, Münich, 1980, pp. 181–191.
[22] R. E. TARJAN, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.
[23] ———, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.