May 1992

Report No. STAN-CS-92-1429

PB96-150370

Scaling Algorithms for the Shortest Paths Problem

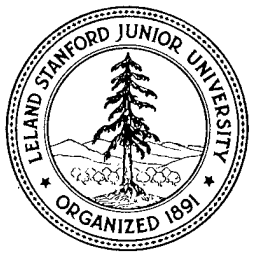
by

Andrew Goldberg

Department of Computer Science

Stanford University

Stanford, California 94305



M97-05-2962

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 1992	

4. TITLE AND SUBTITLE

Scaling Algorithms for the shortest Paths Problem

5. FUNDING NUMBERS

6. AUTHOR(S)

Andrew V. Goldberg

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Computer Science Department
Stanford University
Stanford, CA 94305

8. PERFORMING ORGANIZATION REPORT NUMBER

STAN-CS-92-1429

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

ONR
Arlington, VA 22217

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Abstract

We give an $O(\sqrt{n}m \log N)$ algorithm for the single-source shortest paths problem with integral arc lengths. (Here n and m is the number of nodes and arcs in the input network and N is essentially the absolute value of the most negative arc length.) This improves previous bounds for the problem.

14. SUBJECT TERMS

15. NUMBER OF PAGES
11

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

Scaling Algorithms for the Shortest Paths Problem

*Andrew V. Goldberg**
Computer Science Department
Stanford University
Stanford, CA, 94305

May 1992

*Part of this work was done while the author was visiting IBM Almaden Research Center and supported by ONR Contract N00014-91-C-0026.
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, a grant from Powell Foundation, and by a grant from Mitsubishi Electric Laboratories.

Abstract

We give an $O(\sqrt{n}m \log N)$ algorithm for the single-source shortest paths problem with integral arc lengths. (Here n and m is the number of nodes and arcs in the input network and N is essentially the absolute value of the most negative arc length.) This improves previous bounds for the problem.

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 $O(\sqrt{n}m \log N)$ 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 for the minimum-cost flow problem [2, 11, 13, 14, 15].

Previously known algorithms for the problem are as follows. The classical Bellman-Ford algorithm [1, 7] runs in $O(nm)$ time. Our bound is better than this bound for $N = o(2^{\sqrt{n}})$. The scaling algorithm of Gabow and Tarjan [10] is dominated by an assignment subroutine, which takes $O(\sqrt{n}m \log(nC))$ time, where $[-C, \ldots, C]$ is the range of the arc costs. Our bound dominates this bound (note that $N \leq C - 1$).

Our bound is competitive even with bounds for special-purpose algorithms on planar graphs. The fastest shortest paths algorithm currently known for planar graphs [8] runs in $O(n^{1.5} \log n)$ time. Our algorithm runs in $O(n^{1.5} \log N)$ time on planar graphs, which is better for $N = o(n)$.

The previous best time bounds for scaling algorithms for the shortest paths problem match those for the assignment problem. Our improved bound for the former problem implies that either this problem is computationally simpler or the bound for the latter problem can be improved.

Our framework is very flexible. In Section 8, for example, we use the framework to design a different algorithm for the problem that runs in $O(\sqrt{n}m \log(nN))$ time. Although our algorithms do not use sophisticated data structures such as those for manipulating sets and paths, the flexibility of the framework would make it easy to use such data structures if needed. This flexibility may lead to better running time bounds.

2 Preliminaries

The input to the single-source shortest paths problem is (G, s, l), where $G = (V, E)$ is a directed graph, $\ell : E \rightarrow \mathbf{R}$ is a length function, and $s \in V$ is the source node. (See *e.g.* [3, 17].) The

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

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* $l_p : E \Rightarrow \mathbf{R}$ 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

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

We call an arc (v, w) *improvable* if $l_p(v, w) \leq -\epsilon$, and we call a node w *improvable* if there is an improvable arc entering w.

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 [9]. We call such a problem *Dijkstra's* shortest paths problem [5]. 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 Framework

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. The high-level description of the method is given in Figure 1.

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

```
procedure COMPUTE-PRICES(G, s, l);
    [initialization]
    ε ← 2^{1+⌊log₂ N⌋};
    ∀v ∈ E,  p(v) ← 0;
    [loop]
    while ε > 1 do
        ε ← ε/2;
        p ← REFINE(ε, p);
    end;
    return(p);
end.
```

Figure 1: High-level description of the shortest paths method.

Proof. The lemma follows from the fact that $\ell_p(a)$ is integral and $\ell_p(a) > -1$. ■

Corollary 3.2 *The method terminates in $O(\log N)$ iterations.*

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 [16]; if a component contains a negative reduced cost arc, G has a negative length cycle; otherwise, contract each component.

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 (2ϵ)-feasible price function into an ϵ-feasible one. The CUT-RELABEL operation takes a closed set S and decreases prices of all nodes in S by ϵ.

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 ϵ. Before the relabeling, S is closed and therefore $l_p(a) > 0$. After the relabeling, $l_p(a) > -\epsilon$. ■

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.

Lemma 5.2 *Let p be a (2ϵ)-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 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 ϵ, and, by (2ϵ)-feasibility of p, $l_{p'}(v, x) > -\epsilon$. ■

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).
4. If v is improvable return "the problem is infeasible" and halt.

It is easy to see that given a (2ϵ)-feasible price function, this algorithm computes an ϵ-feasible one in $O(nm)$ time.

As we shall see, it is possible to find either a set X, such that relabeling X eliminates many improvable nodes, or a chain containing many improvable arcs. In the next section we describe a technique that can be applied to a chain of improvable arcs.

6 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 $1 \leq i < j \leq t$, 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 relabels S_i. If w_i is improvable after the relabeling, the algorithm concludes that the problem is infeasible.

Lemma 6.1 *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 ϵ. 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. ■

Lemmas 5.1 and 6.1 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, assigns the price of the new node to be zero, and eliminates self-loops involving the new node. Reduced costs of the other 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 $p(v)$ 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 6.1 $S_i \subseteq S_j$ 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 [4] of Dijkstra's algorithm. Our implementation uses a priority queue that holds items with integer key values in the range $[0, \ldots, 3n]$; 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.

```
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.
```

Figure 2: The scan operation.

Note that if p is (2ϵ)-feasible and $l_p(a) > 2n\epsilon$, then a can be deleted from the graph. 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.

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

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 Figure 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 ϵ and $shift(Q, -\epsilon)$ is executed. This concludes iteration i.

Next we prove correctness of the implementation.

Lemma 6.2 *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 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 value at the beginning of the iteration, then v is reachable from w_i by Lemma 6.1. 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 6.3 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 $O(1 + n' + m')$, where n' is the number of nodes being contracted and m' is the number of arcs between these nodes. The number of CONTRACT operations is $t < n$; the sum of m' values over all CONTRACT operations is at most m and the sum of n' values is below $2n$. Thus the total cost of contract operations is $O(m)$.

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

7 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. The algorithm reduces k by at least $\sqrt{k}$ at each iteration. An iteration takes linear time and is based on the results of sections 5 and 6 and the following lemma, which is related to Dilworth's Theorem (see *e.g.* [6]).

Lemma 7.1 *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 $\sqrt{n}$. Furthermore, such an S can be found in $O(m)$ time.*

Proof. Define a length function l' on E_p by

$$l'(a) = \begin{cases} -1 & \text{if } a \text{ is improvable} \\ 0 & \text{otherwise.} \end{cases}$$

The absolute value of the path length with respect to l' is equal to the number of improvable arcs on the path.

Add a source node r to G_p and arcs of zero length from r to all nodes in V. Call the resulting graph G''; note that G' is acyclic. Let $d' : V \rightarrow \mathbf{R}$ 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.

```
procedure REFINE(ε, p);
    k ← the number of improvable nodes;
    repeat
        DECYCLE(G_p);
        S ← a chain or a set as in Lemma 7.1;
        if S is a chain
          ELIMINATE-CHAIN(S);
        else
          CUT-RELABEL(S);
        k ← the number of improvable nodes;
    until k = 0;
    return(p);
end.
```

Figure 3: An efficient implementation of REFINE.

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 $O(\sqrt{n}m)$ implementation of REFINE is described in Figure 3. 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 7.1, and eliminating at least $\sqrt{k}$ improvable nodes in S using techniques of sections 4, 5, and 6.

Lemma 7.2 *The implementation of* REFINE *described in this section runs in* $O(\sqrt{n}m)$ *time.*

Proof. Each iteration of REFINE take $O(m)$ time by the results of the previous sections. 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

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

■

Corollary 3.2 and Lemma 7.2 imply the following result.

Theorem 7.3 *The shortest paths algorithm with* REFINE *implemented as described in this section runs in* $O(\sqrt{n}m \log N)$ *time.*

8 Tighten Operation

In this section we describe an alternative to the CUT-RELABEL operation, which we call TIGHTEN. This operation is motivated by the operation described in [13] in the context of minimum cost flows. Let p be a (2ϵ)-feasible price function. Assume that we eliminated cycles in G_p, and let l', d', and D be as in the proof of Lemma 7.1. Define a new price function p' by

$$p'(v) = p(v) + \epsilon \frac{d'(v)}{D}.$$

TIGHTEN computes d' and replaces p by p'. This takes $O(m)$ time.

For any v, $0 \leq p(v) - p'(v) \leq \epsilon$. Thus if $l_p(v,w) > 0$ then $l_{p'}(v,w) > -\epsilon$. If $-\epsilon < l_p(v,w) \leq 0$, then $l'(v,w) = 0$ and thus $d'(w) \leq d'(v)$; therefore $l_{p'}(v,w) \geq l_p(v,w) > -\epsilon$. Finally if $l_p(v,w) \leq -\epsilon$, then $l'(v,w) = -1$ and thus $d'(w) \leq d'(v) - 1$; therefore $l_{p'}(v,w) \geq l_p(v,w) - \epsilon/D$.

This implies that TIGHTEN creates no improvable arcs. Since $D \leq n-1$, the reduced cost of every existing improvable arc increases by at least $\frac{\epsilon}{2n}$. Therefore the implementation of REFINE based on TIGHTEN takes $O(nm)$ time.

Note that TIGHTEN does not maintain integrality of p, so the method cannot terminate when ϵ reaches 1. However, if $\epsilon = O(1/n)$ and l is integral, an ϵ-feasible price function can be converted into a feasible price function using rounding and a Dijkstra's shortest paths computation. Therefore the overall running time of the algorithm is $O(nm \log(nN))$.

This bound can be improved by using TIGHTEN in combination with ELIMINATE-CHAIN. Implemented this way, REFINE works as follows. It starts by removing cycles from G_p and computing d', D, and k. If $D \geq \sqrt{k}$, then ELIMINATE-CHAIN is applied to the appropriate chain, eliminating at least $\sqrt{k}$ improvable nodes. If $D < \sqrt{k}$, then TIGHTEN is applied, increasing the reduced cost of every improvable arc by at least $\frac{\epsilon}{2\sqrt{k}}$.

The first case cannot occur more than $O(\sqrt{n})$ times since all improvable nodes will be eliminated. The second case cannot occur more than $O(\sqrt{n})$ times since reduced cost of every improvable arc will increase by at least ϵ and these arcs will no longer be improvable. The resulting algorithm runs in $O(\sqrt{n}m \log(nN))$ time.

9 Concluding Remarks

We described a framework for designing scaling algorithms and two operations, CUT-RELABEL and TIGHTEN, that 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. Our algorithms are easy to implement and

may have practical implications; this work was in fact motivated by an experimental study of minimum-cost flow algorithms [12].

If one is interested only in the algorithms based on CUT-RELABEL, bit scaling can be used instead of ϵ-feasibility. This version of the algorithm rounds lengths up to a certain precision, initially the smallest power of two that is greater or equal to N. Each iteration of the algorithm starts with a price function that is feasible with respect to the current (rounded) lengths. 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 lengths. The resulting price function is 1-feasible with respect to the current length function; the feasibility is restored using the CUT-RELABEL operations as described above. The bounds obtained for the algorithms that use the relabel operation remain valid, but some aspects of the implementation and analysis become slightly simpler. In particular, the only possible negative reduced cost value is -1, and the length function l' defined in the proof of Lemma 7.1 is the same as l_p. Also, no rounding is required for computing the priority queue keys. However, the TIGHTEN operation cannot be defined in a natural way if bit scaling is used.

Our definition of ϵ-feasibility corresponds to that of ϵ-optimality for minimum cost flows [11, 14]. If one follows [11, 14] 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 outer loop of the method.

In conclusion we would like to mention a natural variation of the TIGHTEN operation related to continuous optimization techniques. Suppose we use l_p instead of l' and redefine p' by

$$p'(v) = p + \delta d'(v).$$

We can interpret d' as the direction we want to move in, and δ as a parameter that determines the step size. Then we can define a penalty function Φ whose value is determined by the reduced costs, and pick δ to achieve a large decrease in Φ. For example, we can define Φ to be absolute value of the most negative reduced cost and set $\delta = 1 + \frac{\Phi}{D+\Phi}$. Then an application of TIGHTEN reduces Φ to at most

$$\Phi\frac{D}{D+\Phi} < \Phi\frac{n}{n+1}.$$

(The last inequality follows from the fact that $D < n\Phi$.) The resulting algorithm runs in $O(nm\log(nN))$ time. Using a different penalty function may give a different result.

Acknowledgments

I am grateful to Tomasz Radzik for suggesting an important idea for the proof of Lemma 7.2, 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 ot this paper.

References

[1] R. E. Bellman. On a Routing Problem. *Quart. Appl. Math.*, 16:87–90, 1958.

[2] R. G. Bland and D. L. Jensen. On the Computational Behavior of a Polynomial-Time Network Flow Algorithm. *Math. Prog.*, 54:1–41, 1992.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[4] R. B. Dial. Algorithm 360: Shortest Path Forest with Topological Ordering. *Comm. ACM*, 12:632–633, 1969.

[5] E. W. Dijkstra. A Note on Two Problems in Connection with Graphs. *Numer. Math.*, 1:269–271, 1959.

[6] R. P. Dilworth. A Decomposition Theorem for Partially Ordered Sets. *Annals for Math.*, 51:161–166, 1950.

[7] L. R. Ford, Jr. and D. R. Fulkerson. *Flows in Networks.* Princeton Univ. Press, Princeton, NJ, 1962.

[8] G. Frederickson. Fast Algorithms for Shortest Paths in Planar Graphs, with Applications. *SIAM J. Comput.*, 16:1004–1022, 1987.

[9] 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.

[10] H. N. Gabow and R. E. Tarjan. Faster Scaling Algorithms for Network Problems. *SIAM J. Comput.*, pages 1013–1036, 1989.

[11] A. V. Goldberg. *Efficient Graph Algorithms for Sequential and Parallel Computers.* PhD thesis, M.I.T., January 1987. (Also available as Technical Report TR-374, Lab. for Computer Science, M.I.T., 1987).

[12] A. V. Goldberg and M. Kharitonov. On Implementing Scaling Push-Relabel Algorithms for the Minimum-Cost Flow Problem. Technical Report STAN-CS-92-1418, Department of Computer Science, Stanford University, 1992.

[13] A. V. Goldberg and R. E. Tarjan. Finding Minimum-Cost Circulations by Canceling Negative Cycles. *J. Assoc. Comput. Mach.*, 36, 1989. A preliminary version appeared in *Proc. 20th ACM Symp. on Theory of Comp.*, 388-397, 1988.

[14] A. V. Goldberg and R. E. Tarjan. Finding Minimum-Cost Circulations by Successive Approximation. *Math. of Oper. Res.*, 15:430–466, 1990. A preliminary version appeared in *Proc. 19th ACM Symp. on Theory of Comp.*, 7-18, 1987.

[15] H. Röck. Scaling Techniques for Minimal Cost Network Flows. In U. Pape, editor, *Discrete Structures and Algorithms*, pages 181–191. Carl Hansen, Münich, 1980.

[16] R. E. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.*, 1:146–160, 1972.

[17] R. E. Tarjan. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.