Finding Real-Valued Single-Source Shortest Paths in $o(n^3)$ Expected Time

Stavros G. Kolliopoulos and Clifford Stein *

Dartmouth College, Department of Computer Science, Hanover, NH 03755-3510.
E-mail: {stavros, cliff}@cs.dartmouth.edu

Abstract

Given an n-vertex, m-edge directed network G with real costs on the edges and a designated source vertex s, we give a new algorithm to compute shortest paths from s. Our algorithm is a simple deterministic one with $O(n^2 \log n)$ expected running time over a large class of input distributions. This is the first strongly polynomial algorithm in over 35 years to improve upon some aspect of the $O(nm)$ running time of the Bellman-Ford algorithm. The result extends to an $O(n^2 \log n)$ expected running time algorithm for finding the minimum mean cycle, an improvement over Karp's $O(nm)$ worst-case time bound when the underlying graph is dense. Both of our time bounds are shown to be achieved with high probability.

1 Introduction

Given a directed network $G = (V, E, c)$ where c is a real-valued function mapping edges to costs, the single-source shortest path problem (abbreviated SSSP) consists of finding, for each vertex $v \in V$, a simple path of minimum total cost from a designated source vertex s. This is an old and fundamental problem in network optimization with a plethora of applications in operations research (see, for example [1]). It also arises as a subproblem in other optimization problems such as network flows. A survey of over 200 shortest-path papers through the year 1984 appeared in [9].

The classic Bellman–Ford algorithm solves the SSSP problem in an n-vertex m-edge network in $O(nm)$ time [3, 11]. This simple algorithm has been widely used and studied for over 35 years, however, in all that time, no progress has been made in improving the worst case time bound for arbitrary real-valued shortest path problems.

*Research partly supported by NSF Award CCR-9308701, a Walter Burke Research Initiation Award and a Dartmouth College Research Initiation Award.

Our results. We cannot improve on the Bellman–Ford algorithm in the worst case for all networks. However, we provide a new deterministic algorithm for SSSP that, on any network with real edge costs chosen from a large class of input distributions, runs in $O(n^2 \log n)$ expected time. Moreover, the algorithm is shown to be reliable: it achieves the aforementioned time bound *with high probability,* i.e. with probability $1 - O(n^{-c})$ for some constant c. Thus we provide a probabilistic guarantee that our algorithm finishes its computation in $O(n^2 \log n)$ time. Our input model is that of a complete graph with random costs, therefore the running time of our algorithm outperforms the $O(n^3)$ bound of the Bellman–Ford algorithm. It is also faster than the recent scaling algorithm of Goldberg [17], which takes $O(n^{2.5})$ or more time under the similarity assumption [1] on dense graphs. To our knowledge, this is the first strongly polynomial algorithm that solves SSSP in $o(n^3)$ time for dense networks with arbitrary real-valued costs. In the same time bound, our algorithm will detect the existence of a negative-cost cycle. A conceptual contribution of our method is that one can solve real-weighted SSSP by solving $O(n)$ SSSP problems with nonnegative weights. When these problems can be solved in sublinear time, we get a faster algorithm.

Our model of random edge costs is Bloniarz's *endpoint-independent* model [5]. This model can be roughly defined as follows: the distribution according to which the cost of directed edge (u, v) is chosen does not depend on the head v. Bloniarz's model extended previous work and is arguably the most general model studied in the shortest path literature (see, for example, [5, 13, 27, 30, 29]). Recently Walley and Tan [31] defined a random cost model that is not comparable to Bloniarz's. The endpoint-independent model includes the common case of all edge costs being drawn independently from the same distribution. A more general example from the endpoint-independent model is a cost assignment in which the only restriction is that the costs of all edges emanating from a particular vertex v are chosen from the same distribution. Our method uses ideas from Bellman–Ford and from an algorithm of Moffat and Takaoka [27] originally intended for nonnegative-cost assignments, and turns out to be quite simple, drawing on the simplicity of these two algorithms.

In addition, we show how ideas from our shortest path algorithm can be used to obtain an $O(n^2 \log n)$ expected time implementation of Karp's algorithm [22] for the minimum mean cycle problem when the input satisfies the requirements of the endpoint-independent model. In this problem, given a directed network, a cycle is sought with the smallest *mean cost,* i.e., the ratio of its cost to the number of edges in the cycle. The worst-case time complexity of Karp's minimum mean cycle algorithm is $O(nm)$. Again we show that our algorithm executes within the $O(n^2 \log n)$ time bound with high probability.

Interestingly, our result yields the same bound for the average-case complexity of the SSSP and the minimum mean cycle problems as the one that is known for the assignment problem. Karp [23] gave an $O(n^2 \log n)$ expected time algorithm for the assignment problem on an input model very similar to the endpoint-independent one. The worst-case complexity of the assignment problem is also $O(n^3)$, the same as that of SSSP and mimimum mean cycle.

Finally we show that in a restricted model of computation, the Bellman–Ford algorithm is the best possible. We consider an *oblivious* model of computation, in which the decisions about which edges to relax are made in advance, before the input is seen. We show that in this model, any algorithm whose basic operation is edge relaxation has to perform $\Omega(nm)$ edge relaxations in the worst case to correctly compute shortest paths. The Bellman–Ford algorithm fits into this model.

Previous and Related Work. For arbitrary real costs the existence of negative-cost cycles, i.e. paths of negative cost in which every vertex has degree 2, makes the SSSP problem NP-hard [16]. In the absence of negative-cost cycles, the fastest strongly polynomial SSSP algorithm, as mentioned above, is attributed to Bellman and Ford [3, 11] and can be implemented to run in $O(nm)$ time, worst case. This is $O(n^3)$ for dense graphs. Until recently, all alternative implementations of Bellman–Ford first solved an assignment problem to find vertex potentials, which allows reweighting of edges so that all edge costs become nonnegative. Then Dijkstra's algorithm [10] is applied to the reweighted network. The bottleneck in this approach is the solution of the assignment problem. The first and fastest strongly polynomial-time algorithm for the assignment problem is Kuhn's Hungarian algorithm [25]. Implemented with Fibonacci heaps [12], this algorithm runs in $O(nm + n^2 \log n)$ time. Gabow and Tarjan [15] gave a scaling algorithm for the assignment problem that runs in $O(\sqrt{n}m \log(nN))$ time, where N is the largest absolute value of an edge cost. Recently Goldberg [17] proposed a scaling algorithm that finds shortest paths without solving an assignment problem first; this algorithm has a running time of $O(\sqrt{n}m \log N)$. All of these algorithms detect the existence of a negative-cost cycle. We also note that if the costs are nonnegative, faster algorithms are possible, as Dijkstra's algorithm [10] implemented with Fibonacci heaps [12] runs in $O(n \log n + m)$ time.

We are not aware of any work on the average-case complexity of the SSSP problem for real-valued edge costs. However, the all pairs shortest path problem with nonnegative edge costs is well studied and the relevant literature spans two decades. In the all pairs shortest path problem, (APSP), we are interested in computing the shortest paths between all $n(n-1)$ pairs of vertices. All previous work on the analysis of algorithms for networks with random nonnegative edge costs has been for the APSP problem. In most of these papers a SSSP routine is run n times, using each vertex as a source in turn. Because these algorithms require an $O(n^2 \log n)$ time preprocessing sorting phase, the running times are only reported for the APSP problem, but these algorithms do give ideas for the SSSP problem. A first APSP algorithm with an expected running time of $O(n^2 \log^2 n)$ on networks with independently and identically distributed edge costs was presented in a classical paper by Spira [29] (see [6] for minor corrections). This result was later refined [4] to take into account nonunique edge costs and improved in [30], where an $O(n^2 \log n \log \log n)$ expected time algorithm was given. Bloniarz [5] achieved an expected running time of $O(n^2 \log n \log^* n)$ and relaxed Spira's initial assumption that edge costs are drawn independently from any single but arbitrary distribution. He introduced the more general endpoint-independent randomness model. Hassin and Zemel [19] considered the case in which the edge costs are uniformly distributed independently and gave an $O(n^2 \log n)$ expected time algorithm. Their result extends under some assumptions to independently and identically distributed edge costs. Frieze and Grimmet [13] gave an $O(n^2 \log n)$ expected time algorithm for the case in which edge costs are identically and independently distributed with distribution function F, where $F(0) = 0$ and F is differentiable at 0. The fastest algorithm so far under the endpoint-independent model is due to Moffat and Takaoka [27] and runs in $O(n^2 \log n)$ expected time. Mehlhorn and Priebe [26] corrected an oversight in the analysis given by Moffat and Takaoka and provided a slightly modified version of the algorithm that runs in $O(n^2 \log n)$ time with high probability. They also showed that under weak assumptions $\Omega(n \log n)$ time is required with high probability for solving SSSP on networks with the endpoint-independent distribution. Recently some research has been done on randomized algorithms that use ideas from matrix multiplication [2, 28], but, for arbitrary cost assignments, only pseudopolynomial algorithms exist. The extent to which randomization can be used for faster algorithms, as was the case with e.g. minimum cut [21] and minimum spanning tree [20], is an open question.

The outline of the paper is as follows. In Section 2 we start with a high-level description of the new algorithm. Subsequently we present the randomness model, and give an implementation with fast average case. In Section 3 we present the lower bound for oblivious algorithms. In Section 4 we explain how the shortest path algorithmic ideas extend to the minimum mean cycle problem. In Section 5 we conclude with a discussion of our results and open questions. This paper is a revised and enhanced version of [24].

2 The Shortest-Path Algorithm

In this section we give an algorithm for SSSP with average-case running time $O(n^2 \log n)$ on a broad class of networks with random edge costs. We will give the algorithm in two parts. In Section 2.1 we give a modified version of the Bellman–Ford [3, 11] algorithm that reduces solving a shortest path problem to a sequence of n shortest path problems in a simpler network. Then in Section 2.2 we show how to solve this simpler shortest path problem in $O(n \log n)$ time, on average.

We use n to denote $|V|$ and m to denote $|E|$ for the network $G = (V, E, c)$ of interest. The *length* of path $p = (v_1, v_2, \ldots, v_k)$ is $k - 1$ while the *cost* is defined as $c(p) = \sum_{i=1}^{k-1} c(v_i, v_{i+1})$. The *shortest path* between a pair of vertices u, v is a minimum-cost simple path joining the two vertices. This minimum cost is called the *distance*, denoted by $d(u, v)$. Our algorithms will concentrate on finding the distance from the source s to each vertex, and we will use $d(v)$ to denote $d(s, v)$. Our method can easily be modified to find the actual paths without any asymptotic overhead.

```
procedure FAST_SSSP(Network G, source s)
    for all v ∈ V do
        sort the list of edges out of v;
        d^0(v):= ∞;
    d^0(s):= 0;
    for i=1 to n-1 do
        d^i(.):=PASS(G, d^{i-1});
    for all edges (u,v) ∈ E do
        if d^{n-1}(u) + c(u,v) < d^{n-1}(v) then
            report a negative-cost cycle and break;
```

Figure 1: Algorithm FAST_SSSP

2.1 Skeleton of the new algorithm

We begin by reviewing and modifying the Bellman–Ford algorithm. On a network in which negative costs are present, Bellman–Ford performs $n-1$ passes. Let $d^i(v)$ denote the distance estimate from s to v that has been found during the first i passes. During pass i, all edges (u, v) are relaxed in an arbitrary order. More precisely, pass i is implemented by first assigning $d^i(v) := d^{i-1}(v)$ for all v and then, for all edges (u, v), if

$$d^i(u) + c(u, v) < d^i(v), \tag{1}$$

then $d^i(v)$ is updated to $d^i(u)+c(u, v)$. This step is called *relaxing* edge (u, v). An extra n-th pass will reveal the existence of a negative-cost cycle, since any possible vertex cost decrease will be due to a non-simple path. Adding a simple preprocessing phase to Bellman–Ford, in which we sort the edge lists, we get the algorithm FAST_SSSP, which appears in Figure 2.1.

It is easy to show by induction that after the i-th pass, all shortest paths of length *at most* i have certainly been discovered, and possibly others. More precisely, let $\hat{d^i}(v)$ be the cost of the shortest path from s to v of length at most i. Then for all i and v, Bellman–Ford maintains the invariant that at the end of pass i, $d^i(v) \leq \hat{d^i}(v)$.

In our statement of FAST_SSSP in Figure 2.1, we treat as a black box procedure PASS, which updates the distance labels. The standard Bellman–Ford algorithm implements PASS by relaxing (applying Eq. (1) to) all m edges, yielding a total running time of $O(nm)$.

Implementing PASS. Our improvements will come from carefully implementing procedure PASS to run in $O(n \log n)$ expected time. We begin by observing that the correctness of Bellman–Ford is not harmed if we only compute the $\hat{d}^i(v)$'s during iteration i; after $n-1$ iterations, we will then have computed $\hat{d}^{n-1}(v)$, which is exactly what we want.

This allows us to make the following modifications:

Modification 1. *We can relax all edges in "parallel".* Typically, a sequential implementation of Bellman–Ford does not do this, as it only slows the algorithm down. However, we can accomplish this by replacing condition (1) with the following condition:

$$d^{i-1}(u) + c(u,v) < d^i(v). \tag{2}$$

If we apply this rule instead, it will clearly be the case that after pass i, $d^i(v) = \min\{\min_{u\in V}\{d^{i-1}(u) + c(u,v)\}, d^{i-1}(v)\}$. However, since we are only allowing the length of a path to grow by at most one edge in each pass, we can show inductively that we are actually computing $\hat{d}^i(v)$ in iteration i.

Modification 2. *During pass i only vertices already at length exactly $i-1$ from the source need to have their outgoing edges relaxed.* This is an idea that has been used before, see e.g. [18]. Consider a vertex v and let $j \le i-1$ be the minimum value such that $d^{i-1}(v) = d^{i-2}(v) = \ldots = d^{j-1}(v)$, i.e. a vertex that is at length exactly $j-1$ from the source. Then according to our rule, v had its outgoing edges relaxed for the last time during pass j, and hence for all neighbors w,

$$d^{i-1}(w) \le d^{j-1}(v) + c(v,w) = d^{i-1}(v) + c(v,w).$$

Thus, applying (2) to v's outgoing edges will not cause any vertex labels to be updated.

In light of the two modifications above, we observe that during pass i, relaxing all edges in the set I_v of incoming edges to vertex v may be too much. Of all the edges in I_v, only one is crucial, namely the one that will give the minimum value for $d^i(v)$. Of course, we do not know which edge this is *a priori*, but we will capture a set of edges that will contain it. To do this, we recast the implementation of pass i as a modified SSSP problem in an auxiliary network. Let a *single-source, two-level network* be a directed network $G_\varepsilon = (V_\varepsilon, E_\varepsilon, c_\varepsilon)$, where $V_\varepsilon = s \cup V_1 \cup V_2$, and edges connect only the source s to vertices in V_1 and the vertices in V_1 to vertices in V_2. The cost function c_ε ranges over the reals. We show how an auxiliary single-source, two-level network can be used in the implementation of Pass.

We define now a particular G_ε in terms of the input to Pass, namely $G = (V, E, c)$ and $d^{i-1}(.)$. Let V' be the set of vertices v in G whose current shortest path estimate derives from a path of length exactly $i-1$, i.e. $d^{i-1}(v) < d^{i-2}(v)$. Then V_1 contains a copy of every vertex in V'. V_2 contains a copy of every vertex $v \in V - s$. We introduce two types of edges in E_ε. First, for each pair $(x,y) \in V_1 \times V_2$, where x is a copy of u, y is a copy of v and $(u,v) \in E$, we add an edge (x,y) with $c_\varepsilon(x,y) = c(u,v)$. Second, for all $x \in V_1$, x a copy of v, we add an edge (s,x) with $c_\varepsilon(s,x) = d^{i-1}(v)$.

We use these definitions in the upcoming lemma.

Lemma 1 *Let $T = T(n, m)$ be the time needed to solve a* SSSP *problem in a single-source, two-level network with $\Theta(n)$ vertices and $O(m)$ edges. Then, routine* $\text{PASS}(G, d^{i-1})$ *can be implemented to compute $d^i(.)$ in $O(T + n)$ time. Hence,* FAST_SSSP *is correct and has time complexity $O(n^2 + nT + m \log n)$.*

Proof. The implementation of pass i reduces to solving SSSP in the auxiliary network G_ε, followed by a finishing step. Let SSSP_AUX(G_{aux}, S) be any routine that solves SSSP on a network G_{aux} in time T, where S is the set of vertices in G_{aux} with known distances. We implement PASS as follows:

procedure PASS(G, d^{i-1})

d_ε = SSSP_AUX(G_ε, $s \cup V_1$);

$d^i(v) = \min\{d_\varepsilon(v), d^{i-1}(v)\}$;

By the construction of G_ε, we know that the value of $d_\varepsilon(v)$ for $v \in V_1$ is equal to $d^{i-1}(v)$. Thus computing distances $d_\varepsilon(v)$ in G_ε for the vertices in V_2 is equivalent to computing

$$\min_u\{d^{i-1}(u) + c(u, v)\}, \ \forall v \tag{3}$$

in G. Note that this is the same as calculating the left hand side of (2) for all edges in G. Thus in the last step of PASS, when $d^i(v)$ is calculated as shown, it is equivalent to the process described above, where d^i is first initialized to d^{i-1} and then the test in (2) is applied to every edge. Inductively, if the given $d^{i-1}(.)$ is correct, $d^i(.)$ as output by PASS captures the shortest paths of length at most i in G, and thus routine PASS is correct.

For the running time, we notice that G_ε does not need to be explicitly constructed. All that is required is to identify the set of vertices whose current shortest path length is exactly $i - 1$, and hence the modified SSSP computation can easily be set up in $O(n)$ time. The last step requires an additional $O(n)$ time, so the total time required for PASS is $O(T + n)$, and the lemma follows. ▮

2.2 An implementation with fast average case

Our goal now is to implement the SSSP_AUX routine to run in $o(n^2)$ time on the particular network G_ε. We are unaware of a method achieving such a worst-case bound but we show how to do it in $O(n \log n)$ expected time on networks with random edge costs. In this section we define the class of probability measures for which our analysis holds and then present an algorithm by Moffat and Takaoka [27] and show that it can be used to efficiently find shortest paths in G_ε.

We define first the randomness model used for the analysis. The definition follows [5] except that we allow negative costs as well. Let $\mathcal{G}_n$ be the set of all n-vertex directed networks and suppose P is a probability measure on $\mathcal{G}_n$. We may identify $\mathcal{G}_n$ with the set of all $n \times n$ matrices with entries in $(-\infty, +\infty)$. P is uniquely characterized by its distribution function, $F_P : \mathcal{G}_n \to [0, 1]$, defined by

$$F_P(G) = P\{G' \in \mathcal{G}_n | c_{G'}(i, j) \leq c_G(i, j) \text{ for } 1 \leq i, j \leq n\}.$$

We say that P is *endpoint independent* if, for $1 \leq i, j, k \leq n$ and $G \in \mathcal{G}_n$, we have that

$$F_P(G) = F_P(G'),$$

where G' is obtained from G by interchanging the costs of edges (i, j) and (i, k). Intuitively exchanging endpoints of edges emanating from any fixed vertex does not affect the probability.

Several natural edge-cost assignments are endpoint independent, including the one used by Spira [29] in which edge costs are independently, identically distributed random variables. Another probability measure that meets the endpoint independence criterion is that in which each source vertex i has a probability measure P_i associated with it and the entries $c_G(i, j)$ of the matrix are drawn independently according to distribution P_i. The reader is referred to [5] for further examples.

We proceed with a high-level description of the Moffat-Takaoka method. Moffat and Takaoka give a SSSP algorithm with an expected running time $O(n \log n)$ under the above input model for a nonnegative-cost assignment, assuming that all of the edge lists are sorted. Their algorithm is similar to Dijkstra's [10] and exploits the fact that vertices extracted in increasing cost order from a priority queue cannot have their cost decreased later on; once a vertex v is removed from the priority queue, its distance estimate is the cost of the actual shortest path from the source to v. Of course this is not true, in general, when edges with negative costs are present.

We now review the algorithm of Moffat and Takaoka. For concreteness, we will refer to the algorithm as MT and assume that is takes, as input, a network G, and a set S of vertices of G whose shortest path distances have already been computed. Recall that it relies on all edges having nonnegative costs. The set S of *labeled* vertices is maintained throughout. These are the vertices for which the shortest paths are known. For every element v in S, a *candidate* edge (v, t) is maintained, which is known to be the least cost unexamined edge out of v. Every candidate edge gives rise to a candidate path; there are at most $|S|$ of them at any one time. The candidate paths are maintained in a priority queue. At every step we seek to expand S by picking the least cost candidate path p. If p with final edge (v, t) leads to an unlabeled vertex t, it is deemed to be *useful*, and t is added to S. Otherwise, the path is ignored, but in both cases the candidate path (v, t) is replaced by a different path ending in (v, t') with $c(v, t') \geq c(v, t)$. There are two extremes in the policy for picking this next candidate (v, t'): either simply select the next largest edge out of v or scan the sorted adjacency list until a useful edge (v, t') is found. Based on the policy selection, we obtain Spira's [29] and Dantzig's [8] methods respectively. The Moffat-Takaoka routine (MT) uses Dantzig's algorithm up to a critical point with respect to the size of S and then switches to Spira's. Using standard binary heaps, it can be shown that the original analysis of the algorithm by Moffat and Takaoka together with a correction by Mehlhorn and Priebe [26] yields the following lemma.

Lemma 2 ([27],[26]) *Let G be a network of n vertices with nonnegative edge costs drawn from an endpoint-independent distribution and let S be a set of vertices of G whose shortest path distances have* *been computed. Then* MT$(G,\ S)$ *solves* SSSP *on* G *in* $O(n \log n)$ *expected time, given that the edge lists are presorted by cost.*

Proof. See Theorem 1 in [27] and [26]. ∎

We note that, in general, the nonnegativity condition is necessary for correctness, as it is in Dijkstra's algorithm. However, we will now show that for a single-source, two-level network with real costs, such as G_ε, MT computes correctly shortest paths.

Lemma 3 *Let* $G_\varepsilon = (s \cup V_1 \cup V_2, E_\varepsilon, c_\varepsilon)$ *be the single-source, two-level network defined above. Then* MT$(G_\varepsilon,\ s \cup V_1)$ *solves* SSSP *on* G_ε *in* $O(n \log n)$ *expected time, given that the edge lists of vertices in* V_1 *are presorted by cost, and the edge costs are drawn from an endpoint-independent distribution.*

Proof. Given a set S of already labeled vertices, MT chooses to label next out of the vertices adjacent to S the one corresponding to the least cost candidate path. In the case of G_ε the only paths to vertices in V_2 are the ones passing through V_1, and MT will process each of them in sorted order until all the vertices in V_2 are labeled. Thus it will compute correctly shortest paths. For the running time, it suffices to notice that the analysis in Theorem 1 of [27] relies on the following fact. In an endpoint-independent distribution when $|S| = j$ each candidate leads to each of the $n - j$ unlabeled vertices with equal probability. This fact is not harmed by the negativity of a candidate edge. Thus by Lemma 2, MT on G_ε has $O(n \log n)$ expected time.

An alternative self-contained proof is the following. A sufficiently large constant C may be added to all edge costs in G_ε to make them nonnegative. This reweighting increases the costs of all paths to $v \in V_2$ by $2C$ and thus does not change the relative ordering among the paths; MT can be used to compute the $d_\varepsilon(v)$'s correctly in $O(n \log n)$ expected time. Therefore MT also takes $O(n \log n)$ expected time on the original, nonreweighted network. We note that in the context of our Fast_SSSP algorithm, the reweighting of all the auxiliary networks can be done in $O(n^2)$ worst case total time; however we do not include it in the algorithm to avoid unnecessary complication. ∎

By Lemmata 1 and 3 we obtain the following result.

Theorem 1 *Let* G *be a network of* n *vertices with real edge costs drawn from an endpoint-independent distribution. Algorithm* Fast_SSSP *solves* SSSP *on* G *in* $O(n^2 \log n)$ *expected time if no negative-cost cycle exists. Otherwise it reports the existence of a negative-cost cycle in the same time bound.*

Modifying the Moffat-Takaoka routine to use Fibonacci heaps [12], Mehlhorn and Priebe also obtained a high probability result. In our setting it can be phrased as follows.

Lemma 4 ([26],[27]) *Let* G *be a network of* n *vertices with nonnegative edge costs drawn from an endpoint-independent distribution, and let* S *be a set of vertices of* G *whose shortest path distances have* *been computed. If the edge lists are presorted by cost,* MT(G, S) *solves* SSSP *on* G*, and its running time is* $O(n \log n)$ *with probability* $1 - O(n^{-\beta})$ *for some constant* β*.*

Obtaining the high probability equivalent of Lemma 3 is straightforward. The constant β in Lemma 4 can be made greater than 1, in which case we obtain the following.

Theorem 2 *Let* G *be a network of* n *vertices with real edge costs drawn from an endpoint-independent distribution. Algorithm* FAST_SSSP *solves* SSSP *on* G *if no negative-cost cycle exists and its running time is* $O(n^2 \log n)$ *with probability* $1 - O(n^{-\gamma})$ *for some constant* γ*. Otherwise it reports the existence of a negative-cost cycle in the same time bound.*

The worst-case complexity of the Moffat-Takaoka subroutine is $O(n^2)$, therefore the worst-case running time of our algorithm is $O(n^3)$.

3 A Lower Bound

For over 30 years Bellman–Ford has been the fastest algorithm for the general SSSP problem with an $O(nm)$ worst-case running time. It is thus natural to wonder whether one can show a lower bound on the running time of SSSP algorithms in networks with real-valued edge costs. We do not know how to do this, but we can show a lower bound in a restricted model of computation that captures Bellman–Ford-like algorithms.

Observe that the Bellman–Ford algorithm disregards any information about the network except the number of edges, and performs a fixed sequence of edge relaxations until no vertex label can be decreased. Motivated by this, we define an *oblivious* algorithm for SSSP to be one that, given as input a network G with m edges numbered $1, \ldots, m$, decides on a sequence $S_m = e_{i_1} e_{i_2} \ldots e_{i_k}$ of k edge relaxations, $1 \leq e_{i_j} \leq m$, on the basis of m alone. An oblivious algorithm performs only the chosen relaxations. Obviously, Bellman–Ford falls into the oblivious class, as do generalizations that do not restrict the algorithm to operate in phases. In the following theorem we prove that the Bellman–Ford algorithm performs an asymptotically optimal number of relaxations within this model. We call an edge *optimal* if it belongs to a shortest path.

Theorem 3 *Let* $\mathcal{A}$ *be any oblivious* SSSP *algorithm. There exists a network* G *with* n *vertices and* m *edges on which* $\mathcal{A}$ *must perform* $\Omega(nm)$ *edge relaxations worst case to correctly output the shortest paths.*

Proof. We restrict our attention to networks G, with $m > 2n$, in which the maximum length of a shortest path is exactly $n - 1$, i.e., the shortest path tree is a single path. Let k be the length of the relaxation sequence $S_m = e_{i_1} e_{i_2} \ldots e_{i_k}$ that $\mathcal{A}$ performs on input G. A *subsequence* σ of S_m is any sequence of the form $\sigma = e_{i_{j_1}} e_{i_{j_2}} \ldots e_{i_{j_l}}$, with $1 \leq j_1 < j_2 < \ldots < j_l \leq k$. The correctness of any relaxation-based algorithm comes by guaranteeing that the edges in the shortest path tree are relaxed in an order that is a topological sort of the edges of the tree. In our case, in which the shortest path tree is a path, this means that the edges of the path must be relaxed in the linear order specified by the path. Before seeing the input, it is possible that any sequence s_n of $n-1$ out of the m edges can be the chain of optimal edges, and there are $\frac{m!}{(m-(n-1))!}$ such chains. $\mathcal{A}$ will correctly compute the shortest path distances only if s_n is a subsequence of S_m. The relaxation sequence of length k has exactly $\binom{k}{n-1}$ subsequences of length $n-1$. For the algorithm $\mathcal{A}$ to be correct on all networks, it must be that

$$\binom{k}{n-1} \geq \frac{m!}{(m-(n-1))!} \tag{4}$$

For simplicity of presentation we replace $n-1$ with r in the following. We want to compute k_0, the smallest possible value of k, for which (4) holds. Increasing the left hand side and decreasing the right hand side can only make this lower bound, k_0, smaller. Let $K(\rho(k))$ denote the minimum k such that a relation $\rho(k)$ holds. We assume, wlog, that $0 \leq n \leq k$. We can upper bound the left-hand side of (4) by using the identity [7]

$$\frac{k^k}{r^r(k-r)^{k-r}} \geq \binom{k}{r}. \tag{5}$$

We can lower bound the right-hand side of (4) by

$$\frac{m!}{(m-r)!} = m(m-1)\ldots(m-r+1) \geq (m-r)^r. \tag{6}$$

By (5) and (6), we get

$$k_0 = K\left(\binom{k}{r} \geq \frac{m!}{(m-r)!}\right) \geq K\left(\frac{k^k}{r^r(k-r)^{k-r}} \geq (m-r)^r\right) = k_1.$$

But $k^{k-r}/(k-r)^{k-r} \leq e^r$ thus we have that

$$\begin{aligned} k_1 &\geq K\left(\frac{k^r e^r}{r^r} \geq (m-r)^r\right) \\ &= K\left(k^r \geq \left(\frac{r(m-r)}{e}\right)^r\right) \\ &= K\left(k \geq \frac{r(m-r)}{e}\right). \end{aligned}$$

Substituting $n-1$ for r gives $k_0 = \Omega(nm)$. ▮

4 A Fast Algorithm for the Minimum Mean Cycle Problem

Given a directed network $G = (V, E, c)$, let the *length* $|W|$ of a cycle W be defined as the number of vertices in W. The minimum mean cycle problem consists of finding a cycle W with the minimum *mean cost,* i.e.

a cycle W that minimizes the ratio $\sum_{(i,j)\in W} c(i,j)/|W|$. The currently fastest algorithm is due to Karp [22] and finds a minimum mean cycle in $O(nm)$ worst-case time. We are not aware of any average-case results for this problem. In this section we show how to implement Karp's algorithm to run in $O(n^2 \log n)$ expected time on graphs with edge costs chosen from endpoint-independent distributions. The time bound also holds with high probability.

Theorem 4 *Let G be a network of n vertices with real edge costs drawn from an endpoint-independent distribution. The* minimum mean cycle *problem on G can be solved in $O(n^2 \log n)$ expected time.*

Proof. By inspection of Karp's result, one easily deduces that the time bottleneck of his algorithm is the computation of the quantities $D^i(v)$ by the recurrence

$$D^i(v) = \min_u \{D^{i-1}(u) + c(u,v)\} \ \forall v, \tag{7}$$

with i ranging from 1 to n. Here $D^i(.)$ denotes the minimum cost edge progression of length exactly i to v. Comparing recurrences (3) and (7) for a fixed i we see that they are of the same form except for the different meaning of the values computed by each. As a result we can cast the implementation of recurrence (7) as a single source problem on a two-level auxiliary network as the one described in Section 2.1 for recurrence (3). By Lemma 3 the auxiliary shortest path problem and by consequence the i-th phase of Karp's algorithm can be implemented to take $O(n \log n)$ expected time. ∎

By Lemma 4 we can easily derive a high probability result.

Theorem 5 *Let G be a network of n vertices with real edge costs drawn from an endpoint-independent distribution. There is an algorithm that solves the* minimum mean cycle *problem on G, whose running time is $O(n^2 \log n)$ with probability $1 - O(n^{-\gamma})$ for some constant γ.*

5 Discussion

In this paper we provide deterministic algorithms that improve on the average-case time complexity of two fundamental network problems: SSSP and minimum mean cycle. The algorithms are relatively simple and in both cases provide the first known strongly polynomial improvement over the worst-case cubic time bounds. In addition, our algorithms achieve the $O(n^2 \log n)$ time bound with high probability. Putting our work in perspective, we may contrast our results with the already known $O(n^2 \log n)$ result for the assignment problem [23]. Table 1 provides the best known running times (on dense graphs) for the assignment, SSSP and minimum mean cycle problems, both worst case and average case. These three problems are known to be conceptually related, and the pattern of $O(n^3)$ worst-case time seems to transfer to an $O(n^2 \log n)$ average-case time pattern for all three.

Problem	*Worst Case*	*Average Case*
assignment	$O(n^3)$ [25, 12]	$O(n^2 \log n)$ [23]
SSSP	$O(n^3)$ [3, 11]	$O(n^2 \log n)$ [this paper]
minimum mean cycle	$O(n^3)$ [22]	$O(n^2 \log n)$ [this paper]

Table 1: Best strongly polynomial time bounds for three related problems. The input model in [23] is defined as a special case of the endpoint-independent one. Worst-case time bounds are reported for dense graphs.

It is interesting to note that a linear time (ignoring log factors) reduction is known to exist from SSSP to the assignment problem [14]. We have been unable to use this reduction in conjunction with Karp's $O(n^2 \log n)$ assignment algorithm [23] to obtain a SSSP algorithm as fast as ours. The reduction is based on using an assignment algorithm to compute vertex potentials $\pi(.)$ that dominate the edge costs. In other words it must be

$$-c(u,v) \leq \pi(u) - \pi(v), \ \ \forall u, v.$$

Karp's algorithm in [23] is of the shortest augmenting path type. It starts by performing the standard transformation of adding a big positive constant to the real-valued costs. Thus the potentials it computes do not satisfy the relation above. On the other hand, if the first step of adding the constant is not taken the average-case analysis in [23] does not hold.

Open problems. We exhibited the first strongly polynomial algorithm in over 30 years that achieves $o(n^3)$ running time for SSSP with real-valued costs, although not for the worst case. The complexity gap between $O(nm)$ time for for dense graphs and $O(n^2 \log n)$ time is considerable. This raises the natural question of whether an $o(n^3)$ worst-case time algorithm for SSSP exists. If the answer is no, can one reason about a lower bound in a nonoblivious computational model? Another persistent open question is whether one can decouple negative cycle detection from shortest path computation and achieve an $o(n^3)$ worst-case time bound for the former problem.

Acknowledgements. The first author wishes to thank Donald Johnson for starting him working on shortest paths and for all the valuable discussions. The second author thanks Tom Sundquist for useful discussions.

References

[1] R. K. Ahuja, T. L. Magnanti and J. B. Orlin, "Network Flows," Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] N. Alon, Z. Galil and O. Margalit, On the exponent of the all pairs shortest path problem, *in* " Proc. 32nd IEEE Symp. Found. Comp. Sci.," 1991, 569-575.

[3] R. Bellman, On a routing problem, *Quarterly of Appl. Math.,* **16** (1958), 87-90.

[4] P. A. Bloniarz, A. Meyer and M. Fischer, Some observations on Spira's shortest path algorithm, Tech. Rept. 79-6, Comp. Sci. Dept., State University of New York at Albany, Albany, NY, 1979.

[5] P. A. Bloniarz, A shortest path algorithm with expected time $O(n^2 \log n \log^* n)$, *SIAM J. Comput.,* **12** (1983), 588-600.

[6] J. S. Carson and A. M. Law, A note on Spira's algorithm for the all-pairs shortest path problem,*SIAM J. Comput.,* **6** (1977), 696-699.

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

[8] G. B. Dantzig, On the shortest route through a network, *Management Sci.,* **6** (1960), 187-190.

[9] N. Deo and C. Pang, Shortest-path algorithms: taxonomy and annotation, *Networks,* **14** (1984), 275-323.

[10] E. W. Dijkstra, A note on two problems in connection with graphs, *Numerische Mathematics,* **1** (1959), 269-271.

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

[12] M. L. Fredman and R. E. Tarjan, Fibonacci heaps and their uses in improved network optimization problems, *J. ACM,* **34** (1987), 596-615.

[13] A. M. Frieze and G. R. Grimmet, The shortest-path problem for graphs with random arc-lengths, *Discrete Applied Mathematics,* **10** (1985), 57-77.

[14] H. N. Gabow, Scaling Algorithms for Network Problems, *J. Comput. Syst. Sci.,* **31** (1985), 148-168.

[15] H. N. Gabow and R. E. Tarjan, Faster scaling algorithms for network problems, *SIAM J. Comput.,* **18** (1989), 1013-1036.

[16] M. S. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-completeness", W. H. Freeman, San Francisco, CA, 1979.

[17] A. V. Goldberg, Scaling algorithms for the shortest path problem, *SIAM J. Comput.,* **24** (1995), 494-504.

[18] D. Goldfarb, J. Hao and S. R. Kai, "A computational comparison of label correcting and simplex algorithms for computing shortest paths", Technical Report, IEOR Department, Columbia University, 1986.

[19] R. Hassin and E. Zemel, On the shortest paths in graphs with random weights, *Math. Oper. Res.,* **10** (1985), 557-564.

[20] D. R. Karger, P. N. Klein and R. E. Tarjan, A randomized linear-time algorithm to find minimum spanning trees, *J. ACM,* **42** (1995), 321-328.

[21] D. R. Karger and C. Stein, A new approach to the minimum cut problem, *J. ACM,* **43** (1996), 601-640.

[22] R. M. Karp, A characterization of the minimum cycle mean in a digraph, *Discrete Math.* **23** (1978), 309-311.

[23] R. M. Karp, An Algorithm to Solve the $M \times N$ Assignment Problem in Expected Time $O(MN \log N)$, *Networks,* **10** (1980), 143-152.

[24] S. G. Kolliopoulos and C. Stein, Finding Real-Valued Single-Source Shortest Paths in $o(n^3)$ Expected Time, *in* "Proc. 5th MPS Integer Programming and Combinatorial Optimization Conference," *Lecture Notes in Computer Science,* W. H. Cunningham, S. T. McCormick, M. Queyranne eds., Vol. **1084**, Springer Verlag, 1996, 94-104.

[25] H. W. Kuhn, The Hungarian method for the assignment problem, *Naval Research Logistics Quarterly,* **2** (1955), 83-97.

[26] K. Mehlhorn and V. Priebe, On the all-pairs shortest path algorithm of Moffat and Takaoka, *Random Structures and Algorithms* **10** (1997), 205-220.

[27] A. Moffat and T. Takaoka, An all pairs shortest path algorithm with expected time $O(n^2 \log n)$, *SIAM J. Comput.,* **16** (1987), 1023-1031.

[28] R. Seidel, On the all-pairs-shortest-path problem, *in* "Proc. 24th ACM Symp. on Theory of Computing," 1992, 745-749.

[29] P. M. Spira, A new algorithm for finding all shortest paths in a graph of positive arcs in average time $O(n^2 \log^2 n)$, *SIAM J. Comput.,* **2** (1973), 28-32.

[30] T. Takaoka and A. M. Moffat, An $O(n^2 \log n \log \log n)$ expected time algorithm for the all shortest distance problem, *in* "Mathematical Foundations of Computer Science, Lecture Notes in Computer Science" (P. Dembinski, Ed.), pp. 643-655, Vol. **88**, Springer Verlag, Berlin, 1980.

[31] S. K. Walley and H. H. Tan, Shortest paths in random weighted graphs, *in* " Proc. 1st Int. Conf. on Computing and Combinatorics," 1995, 213-222.