A randomized parallel algorithm for single-source shortest-paths *

Philip N. Klein **Sairam Subramanian**

Department of Computer Science
Brown University, Providence, RI

Abstract

We give a randomized parallel algorithm for computing single-source shortest paths in weighted digraphs. We show that the exact shortest path problem can be efficiently reduced to solving a series of approximate shortest-path subproblems. Our algorithm for the approximate shortest-path problem is based on a technique used by Ullman and Yannakakis in a parallel algorithm for breadth-first search.

1 Introduction

One of the most fundamental and ubiquitous problems in combinatorial optimization is finding single-source shortest paths in a weighted graph. Aside from being important in its own right, the problem arises in algorithms for many other problems, especially those related to flow.

In view of the importance of the single-source shortest paths problem, it is unfortunate that all known *parallel* algorithms for this problem are very inefficient on sparse graphs. This inability to make efficient use of parallelism in computing shortest paths is of both theoretical and practical significance. A fast and efficient parallel algorithm for this problem remains a major goal in the design of parallel graph algorithms.

In this paper, we describe a parallel randomized algorithm for computing single-source shortest paths. Our algorithm achieves a significant speed-up even when only a linear number of processors are available. In the bounds below, we assume the concurrent-write PRAM as our model of parallel computation.

In stating our results, we specify the time required and the *work* required by the algorithms. Here *work* is defined to be the product of time and number of processors required; this corresponds to the time that would be required if the parallel processors were all simulated by a single processor. Work is a measure that reflects how efficiently the processors are being used by a parallel algorithm; one can compare the work to the time required by a sequential algorithm for the same problem.

Theorem 1.1 *There is a randomized parallel algorithm that, given a directed graph G with nonnegative integer lengths and given a source node s, computes shortest-path distances from s to all the other nodes in the graph. With probability $1 - n^{\Omega(1)}$, the algorithm takes time $O(\sqrt{n}\log^2 n \log L)$ and work $O(m\sqrt{n}\log^2 n \log L)$, where m, n, and L are, respectively, the number of edges in G, the number of nodes, and the maximum edge-length.*

By comparing the work required by our algorithm to the $O(m + n\log n)$ time required by a sequential algorithm for shortest paths, we see that our algorithm incurs an $O(\sqrt{n}\log^2 n \log L)$ factor overhead. Our algorithm is based on a parallel breadth-first search algorithm due to Ullman and Yannakakis [18].

Ullman and Yannakakis show that a breadth-first search tree can be constructed in $O(\sqrt{n}\,\mathrm{polylog}\, n)$ time using a linear number of processors. Their algorithm is limited, however, in that it cannot handle lengths on nodes or edges. Thus one of our main contributions is to extend their algorithm so that it can do so. We first show that using a simple rounding technique enables their algorithm to approximate shortest paths in the presence of lengths. We then show how to use this approximation algorithm along with a scaling technique

*Research supported by NSF grant CCR-9012357 and an NSF PYI award, together with PYI matching funds from Thinking Machines Corporation and Xerox Corporation. Additional support provided by ONR and DARPA contract N00014-83-K-0146 and ARPA Order No. 6320, Amendment 1.

to get exact shortest paths. The scaling technique is similar to the one used by Gabow [8] for sequential computation of shortest paths.

In other related work, a parallel algorithm for shortest paths was discovered by Spencer [17]. Actually, his algorithm involves a tradeoff between time and work. Spencer specifies his bounds in terms of a parameter ρ and an upper bound L on edge-lengths (assuming the lengths are positive integers). The time required is $O((n/\rho)\log n \log(\rho L))$ and the work is $O(n\rho^2 \log \rho \log(\rho L) + (m + n\log L)\mathrm{polylog}(n))$. For comparison, let us ignore logarithmic factors; in order to achieve about $\sqrt{n}$ time, Spencer's algorithm would need to do about n^2 work, whereas our algorithm would need to do about $m\sqrt{n}$ work. Thus our algorithm is considerably more efficient in the case where the graph is sparse (i.e. m not much bigger than n) and less efficient if the graph is dense (i.e. m bigger than $n^{1.5}$). Put another way, if the graph is sparse ($m = O(n)$) and one only has n processors available, our algorithm would use the n processors to achieve a speed-up of about $\sqrt{n}$ while Spencer's algorithm would achieve a speed-up of about $n^{1/4}$.

1.1 Subsequent work

In recent work Cohen [5] has given an algorithm to find $(1+\epsilon)$-approximate shortest paths that works in poly-logarithmic time and does close to linear work. Her algorithm makes use of the limited-search algorithm presented in this article. Thus for undirected graphs the work-complexity of Cohen's [5] algorithm is close to optimal. However, her algorithm does not generalize to directed graphs; and there seems to be no way to use it to compute exact shortest paths by repeated approximation even if the underlying graph is undirected.

More recently Shi and Spencer [16] have given a parallel shortest-path algorithm, for undirected graphs, with work-bounds that are similar to ours. Given $\log n \le t \le n$ their algorithm runs in $\tilde{O}(t)$ time with $\tilde{O}((n^3/t^2) + m)$ work, or in $\tilde{O}(t)$ time with $\tilde{O}((n^3/t^3) + mn/t)$ work.

To compare the relative efficiency of the two algorithms we set t to be $\sqrt{n}$. If we assume that the edge-lengths are polynomially bounded then for undirected graphs the two algorithms require roughly the same amount of work. However like Cohen's algorithm, their algorithm does not generalize to directed graphs either.

In other work [14] we have given a nearly work-optimal parallel algorithm for computing single-source shortest-paths if the underlying graph is planar. Combining the techniques of that paper with those of the present article, we can obtain a faster but somewhat less efficient randomized parallel algorithm for shortest paths in general graphs. For a given time bound t such that $t = \Omega(\log^c n \log L)$, the algorithm takes time t and work $O((mn/t)$ polylog $n \log L)$. In particular, one can compute shortest paths in time polylog in n and L using nm processors.

2 Background and overview

A word about terminology in order to forestall confusion. Unless otherwise stated, when we speak of the *length* of a path, we mean the sum of the lengths of its edges (rather than the number of its edges). Similarly, *distance* is measured according to edge-lengths. The *size* of a path, on the other hand, shall signify the number of nodes in the path, and the *minimum path-size* is the shortest distance measured in number of nodes traversed.

Unless otherwise stated, n and m denote, respectively the number of nodes and the number of edges in the input graph.

2.1 The model of parallel computation

We assume a model of parallel computation called the OR CRCW PRAM [2], in which multiple processors can simultaneously read and write to a shared memory. If multiple processors attempt to write multiple values to a single location, the value written is the bitwise OR of the values. This model enables us to do processor allocation using a randomized constant-time algorithm due to Hagerup and Raman [11] for approximate prefix summation. Their algorithm runs in constant time with probability $1 - 2^{-n^{\delta}}$ for some positive constant δ.

Assuming the slightly weaker ARBITRARY CRCW PRAM, in which an arbitrary processor succeeds in writing, would entail using an $O(\log^* n)$-time algorithm [10] for processor allocation. This change multiplies our time bounds by $O(\log^* n)$.

2.2 Parallel breadth-first search

The fastest known parallel algorithm for breadth-first search involves repeatedly squaring a matrix, where the element-wise operations are in the min–plus semiring. In fact, this algorithm can compute shortest paths. The technique is well-known; see [1, 7] for more details. The problem with this algorithm is that it requires too many processors; to achieve $O(\log n)$ time requires about n^3 processors. The processor bound has been improved somewhat [9] in the case of breadth-first search.

The most elementary parallel search technique is parallel breadth-first search, in which the nodes are visited level by level as the search progresses. Level 0 consists of the starting node s, level 1 consists of the neighbors of s, level 2 consists of the neighbors of the neighbors of s (or, rather, those not belonging to previous levels), and so forth.

Suppose there are p processors, To go from level i to level $i+1$, the processors first divide up the edges outgoing from the level i nodes. This step can be done with high probability in $O(\log^* p)$ time using the parallel load-balancing algorithm of [10]. Next, for each such edge, if the other endpoint has not yet been traversed, it is assigned to level $i+1$. This step can be done in $O(e_i/p)$ time, where e_i is the number of such edges. Hence the time required to traverse k levels is $O(e/p + k\log^* p)$ with high probability.

2.3 Limited search

The problem with this approach is that the time required grows linearly with the number of levels traversed. To keep the time small, we must resort to a limited search, in which we limit the number of levels traversed. A *k-limited shortest path* from s to t is a path from s to t that is no longer than any s-to-t path of size at most k. Note that a k-limited shortest path need not itself have size at most k. For any given error parameter $\epsilon > 0$ we define a k-limited *ϵ-approximate* shortest path to be one whose length is at most $1+\epsilon$ times the length of any path of size at most k. We show how to search such paths in parallel in $O(k\epsilon^{-1})$ iterations, loosely speaking. An estimate d of the distance from s to t is a k-limited ϵ-approximate shortest path distance if there is an s-to-t path of length at most d, and d is at most $1+\epsilon$ times the length of any path of size at most k.

In the case of all lengths being 1, it is easy to find k-limited shortest paths in k iterations; one just uses k iterations of parallel breadth-first search to find the actual shortest paths to all nodes at distance at most k from the source. We call this *k-limited breadth-first search*; it is essentially the method used in the algorithm of Ullman and Yannakakis.

2.4 Ullman and Yannakakis's algorithm

We now give a brief description of the basic version of Ullman and Yannakakis's parallel algorithm for breadth-first search. We describe our adaptation of their algorithm in greater detail in Section 4.1.

The algorithm is based on $\sqrt{n}\log n$-limited search. We are given a graph and a source s from which to find a breadth-first search tree. First, $O(\sqrt{n})$ randomly chosen nodes, along with the source, are designated as *distinguished*. (The constant hidden by the big-Oh determines the probability that the algorithm is correct, as we mention below.) From each of the distinguished nodes x in parallel, p processors conduct a $\sqrt{n}\log n$-limited search, identifying the minimum path-size from x to each node within path-size $\sqrt{n}$. By choosing $p = m/\sqrt{n}$, we can carry out all of these searches in $O(\sqrt{n}\log n)$ time using $p\sqrt{n} = m$ processors.

Second, an auxiliary graph is formed on the set of distinguished nodes, where the length of an edge from u to v is defined to be the minimum path-size from u to v found during the limited search. All-pairs shortest paths are computed for the auxiliary graph. This takes total work $O((\sqrt{n})^3\log n)$, and can easily be done in $O(\sqrt{n}\log n)$ time using m processors. In particular, we have the length of the shortest path in the auxiliary graph from the source s to each of the distinguished nodes.

Third, the minimum path-size from the source s to a node v is computed by taking the minimum, over all distinguished nodes x, of the auxiliary-graph shortest path length from s to x plus the minimum path-size from x to v found in x's limited search. One can show that, with high probability, this method yields the minimum path-size for all the nodes v.

The proof is as follows. Fix in advance one minimum-size path P_v from s to v, for every node v. Now consider the randomly selected distinguished nodes. With probability $1 - \frac{1}{n^c}$ (where c is any given constant, and determines the constant factor in the number of randomly chosen nodes), for each node v each subpath of P_v of size $\sqrt{n} \log n$ contains a distinguished node.

Thus the path P_v is broken up into subpaths starting and ending with distinguished nodes (except for the last subpath, which ends on v), such that each subpath has size at most $\sqrt{n} \log n$. These subpaths are traversed in the limited search. Hence the minimum path-size from s to the last distinguished node of P_v is correctly computed in the all-pairs computation, and the minimum path-size from s to v is correctly computed in the last step of the algorithm.

Ullman and Yannakakis's approach does not directly work with weighted graphs because there is no apparent way to find even the $\sqrt{n} \log n$-limited shortest paths within the available resource bounds. Our first contribution is a method that approximates these shortest paths.

We provide a procedure that, given a source node and a parameter k, finds estimates of the distances from the source to all nodes such that each estimate is at least the shortest path distance and no more than $1 + \epsilon$ times the k-limited shortest path distance. Given a graph with edge lengths having polynomial magnitude, the procedure runs in time

$$O\left(\frac{m \log n}{p} + k\epsilon^{-1} \log^* p\right)$$

using p processors.

By substituting $k = \sqrt{n} \log n$ and $p = O(m/\sqrt{n})$ and running $O(\sqrt{n})$ instances of the procedure in parallel, we can implement an approximate version of Ullman and Yannakakis' first step in $O(\sqrt{n}\epsilon^{-1} \log^* p)$ time using m processors. Note that this procedure is sufficient to make Ullman and Yannakakis's procedure work for finding approximate single-source shortest paths in weighted graphs; the second and third steps do not depend on the lengths being one. By applying essentially the same proof as sketched above for Ullman and Yannakakis' algorithm, it is easy to see that using these distances in forming the auxiliary graph results in the final estimated distances being at least actual distances and at most $(1+\epsilon)$ times actual distances. An earlier version of this paper [13] used this limited search idea to compute approximte single-source shortest paths.

Our second contribution is a reduction technique (similar to one due to Gabow [8]) that allows us to solve the exact problem by solving a series of approximate problems. In section 4 we show how to use this approximation algorithm to find exact single-source shortest paths.

2.5 Weighted parallel breadth-first search

There is an obvious approach to extending parallel breadth-first search to handle positive integral weights. To handle level i, the processors first divide up the outgoing edges as before. Then, for each such edge uv, the level of v is assigned to be i plus the length of uv (if this value is less than the level previously assigned to v). Thus level i consists of those nodes at distance i from the source.

2.6 Approximating k-limited shortest paths

Weighted parallel breadth-first search has the same disadvantage as ordinary parallel breadth-first search, only more so. The time required grows linearly with the distance traversed. This is especially a problem when the lengths are large. To alleviate this disadvantage, the obvious fix is to uniformly shrink all lengths. This in itself is not enough; the search crucially depends on the lengths being integral. Thus we must round the shrunken lengths up to nearest integers without substantially changing the length of the shortest path.

In Section 3, we outline a procedure, FullSearch(G, s, k, p, ϵ) for finding approximate k-limited shortest paths from source s in the directed graph G. Here ϵ is a positive number strictly less than one, and the estimated path lengths are at least the actual lengths and no more than $1 + \epsilon$ times the actual lengths. In this case, we say the estimates are accurate *to within a relative error of* ϵ, or, more briefly, to within ϵ.

Let p be the number of processors available and D be the sum of the edge lengths of G, procedure *FullSearch* finds approximate k-limited shortest paths in time

$$O(\frac{m \log D}{p} + k\epsilon^{-1}),$$

where D is the sum of edge-lengths.

When we use procedure FULLSEARCH for computing approximate k-limited shortest paths, we assign input parameters $k = \sqrt{n} \log n$ and $p = m \log D / \sqrt{n}\epsilon^{-1}$. With these parameters, procedure FULLSEARCH takes time $O(\sqrt{n}\epsilon^{-1} \log n)$. Since Ullman and Yannakakis's algorithm must carry out limited search from each of $O(\sqrt{n})$ nodes simultaneously, the number of processors required for all these calls is $O(m\epsilon \log D)$. Therefore the work done by the algorithm is $O(m\sqrt{n} \log n \log D)$.

In section 4 we address the problem of finding exact single-source shortest paths. We first show that the exact problem can be reduced to solving $O(\log n \log L)$ approximate problems by using a scaling technique similar to the one used by Gabow [8]. We then show how to use our approximation algorithm in conjunction with the scaling idea to solve the exact problem.

3 The procedure FULLSEARCH

In this section we give the procedure FULLSEARCH for finding approximate k-limited shortest paths from a given source node s. We first describe the sub-procedure SEARCH$(G, s, d, k, p, \epsilon)$. It determines the k-limited ϵ-approximate shortest path distances from s of those nodes whose distance is at least d and at most $2d$.

Procedure FULLSEARCH finds k-limited approximate shortest path distances of all nodes simply by calling SEARCH$(G, s, d, k, p, \epsilon)$ in parallel with $d = 1, 2, 4, \ldots$. Since D is the sum of the edge lengths we need to consider at most $\log D$ different values for d in order to determine approximate k-limited shortest paths to all the nodes. This can be accomplished with a $(1 + \log D)$- factor increase in the number of processors.

Using p processors, the procedure SEARCH$(G, s, d, k, p, \epsilon)$ takes time

$$O(\frac{m}{p} + k\epsilon^{-1}) \tag{1}$$

where m is the number of edges and n is the number of nodes in G.

The first step of the procedure is to define the *scale factor* α by

$$\alpha = \epsilon d/k \tag{2}$$

Next, we obtain approximate edge-lengths by rounding each length up to the nearest integer multiple of α. Zero lengths are rounded up to α.

$$\widehat{\ell}(e) = \begin{cases} \alpha\lceil \ell(e)/\alpha \rceil & \text{if } \ell(e) > 0 \\ \alpha & \text{if } \ell(e) = 0 \end{cases}$$

Since the resulting lengths are positive integer multiples of α, we can apply weighted breadth-first search where in each level we traverse a distance α. Search from the source for $\lceil 2(1+\epsilon)k/\epsilon \rceil$ levels, and label each node reached with the estimated distance, i.e. α times the number of levels the search required to reach the node. The estimated distance to a node not reached by the search is implicitly taken to be infinite.

The time required to traverse $\lceil 2(1+\epsilon)k/\epsilon \rceil$ levels is $O(m/p + (2k/\epsilon))$ with high probability using p processors. Next we show that the estimated distances computed are not too bad.

The following proposition follows immediately from the fact that the approximate edge-lengths are at least the actual edge-lengths.

Proposition 3.1 *For any path P, $\widehat{\ell}(P) \geq \ell(P)$.*

Proposition 3.1 ensures that we never underestimate distances. We might overestimate distances, but lengths of paths with at most k edges and distance at least d are not overestimated by much.

Lemma 3.1 *Suppose P is a source-to-v path of size k and length x, $d \leq x \leq 2d$. The estimated distance to v is at most $1 + \epsilon$ times the length of P.*

Proof: For each edge e of P, $\widehat{\ell}(e) \leq \ell(e) + \alpha$, so $\widehat{\ell}(P) \leq \ell(P) + k\alpha$. By choice of α, we have $k\alpha = \epsilon d$. Since $\ell(P)$ is at least d, we obtain $\widehat{\ell}(P) \leq (1+\epsilon)\ell(P)$. Furthermore, since $\ell(P) \leq 2d$, we have $\widehat{\ell}(P) \leq 2(1+\epsilon)d$, so $\lceil 2(1+\epsilon)k/\epsilon \rceil$ levels are sufficient to traverse P. Hence the estimated distance to v is at most $\widehat{\ell}(P)$. □

The procedure $\textsc{Search}(G, s, d, k, p, \epsilon)$ only accurately estimates the k-limited distances of nodes when those distances are between d and $2d$. To estimate the distances of all nodes, we call $\textsc{Search}$ in parallel with different values of d. Let D be the sum of edge-lengths. Procedure $\textsc{FullSearch}(G, s, k, p, \epsilon)$ simply makes $1 + \lceil \log D \rceil$ calls to $\textsc{Search}(G, s, d, k, p, \epsilon)$, each with d assigned a different power of 2 from 2^0 to $2^{\lceil \log D \rceil}$. Then, for each node, we take its estimated distance to be the minimum estimated distance obtained.

Lemma 3.2 *Procedure* $\textsc{FullSearch}$ *computes k-limited ϵ-approximate shortest path distances in time*

$$O(\frac{m \log D}{p} + k\epsilon^{-1}) \tag{3}$$

Proof: For any node v, the estimated distance to v is at least the actual distance, by Proposition 3.1. Suppose v is at distance $d(v)$ from the source. Let $d_0 = 2^{\lfloor \log d(v) \rfloor}$, so $d_0 \leq d(v) \leq 2d_0$. Then by Lemma 3.1, when $\textsc{Search}$ is called with $d = d_0$, the estimated distance to v is at most $(1+\epsilon)$ times the length of any path to v with k edges. The time bounds follow from the fact that the $1 + \lceil \log D \rceil$ calls to procedure $\textsc{Search}$ can all be made in parallel. □

4 Computing exact shortest paths by repeated approximation

In this section we show how to use procedure $\textsc{FullSearch}$ to compute exact shortest paths in a directed graph G.

Gabow has given [8] a scaling algorithm to compute single-source shortest paths in a graph with maximum edge-length L by solving a series of $\log L$ instances (involving the same graph) in which each shortest-path distance is guaranteed to be at most n, the number of nodes. For such an instance, any edge whose length exceeds n is superfluous and can be discarded. By using this reduction, we can assume henceforth that the sum of edge-lengths is at most n^3. This observation was made by D. Karger.

Next we show how to reduce an exact problem of this form to an approximate problem of the same form. The reduction again uses the central idea of Gabow's scaling algorithm, namely modifying edge-lengths according to node *prices* derived from a previous shortest-path calculation.

For intuition, think of an edge's length as the cost one must pay to traverse that edge. Suppose that for each node v, we have a *price* $p(v)$. Think of the price of v as the reward for entering the node, and the price for leaving it. When one takes into account the prices, the *net cost* $\ell_p(uv)$ of traversing an edge uv is

$$\ell_p(uv) = \ell(uv) + p(u) - p(v).$$

The net cost of traversing a path depends on the prices of the starting and ending nodes, not on the prices of intermediate nodes, because the path both enters and leaves each intermediate node. In particular, for a path P from x to v,

$$\ell_p(P) = p(x) + \ell(P) - p(v) \tag{4}$$

For any two nodes x and v, therefore, an x-to-v path is shortest with respect to the original edge-costs if and only if it is shortest with respect to the net costs. Thus introduction of prices yields an equivalent shortest-path problem.

Prices are useful because they can reduce the costs of the edges in shortest paths. Suppose, for example, that the price of each node v is the length of the shortest path from the source to v. Let P be this shortest path. By substituting 0 for p(source) and $\ell(P)$ for $p(v)$ in equation (4), we see that the net cost of the shortest source-to-v path is zero.

In solving a shortest-path problem, we cannot assign shortest-path distances as prices because we don't know those distances. However, since we have a subroutine for estimating shortest-path distances, we can use these estimates as prices. With respect to the resulting net costs, the shortest-path distances are smaller than before. This is the idea at the core of our reduction of exact to approximate shortest paths.

In assigning prices, however, we must ensure that the resulting net costs remain nonnegative. We therefore require that the distance estimates are underestimates and in particular that they are exact shortest-path distances with respect to smaller costs. These considerations lead us to the following definition.

We define the $(1-\epsilon)$-approximate shortest path problem to be computing distance estimates $d(v)$ from a given root such that:

- (A) There is an auxiliary graph such that the estimates are exact shortest-path distances in the auxiliary graph, and
- (B) If the distance in the original graph from one node to another is d, the distance in the auxiliary graph is between $(1-\epsilon)d$ and d.

Thus in the $(1-\epsilon)$-approximate problem we are interested in getting underestimates instead of overestimates. We now prove that the single-source shortest path problem is efficiently polylog-time-reducible to the $(1-\frac{1}{2})$-approximate shortest path problem.

Lemma 4.1 *Single-source directed shortest paths in a graph with nonnegative integral lengths and maximum shortest-path length D can be solved using $\log D$ calls to an algorithm for computing $\frac{1}{2}$-approximate shortest paths in the same graph (with different nonnegative integral lengths).*

Proof: We give a recursive algorithm and show it has recursion depth $\log D$. The algorithm first computes distance estimates $d(v)$ from the source. If these estimates are all zero, then by property (b) of the estimates, the exact distances are all zero. Otherwise the algorithm recursively computes exact shortest-path distances $\widehat{f}(v)$ in a graph with lengths $\ell(uv)$ defined by

$$\widehat{\ell}(uv) := \ell(uv) + \lceil d(u) \rceil - \lceil d(v) \rceil$$

Finally the algorithm computes exact shortest-path distances $f(v)$ in the original graph by setting $f(v) = \widehat{f}(v) + \lceil d(v) \rceil$.

The correctness of this procedure is proved as follows. For any node x and any path P from the source to x,

$$\begin{aligned} \widehat{\ell}(P) &= \sum_{uv \in P} \lceil d(u) \rceil + \ell(uv) - \lceil d(v) \rceil \\ &= \lceil d(\text{source}) \rceil + \ell(P) - \lceil d(x) \rceil \end{aligned} \tag{5}$$

Since $\lceil d(source) \rceil = 0$, it follows that the shortest path to x indeed has length $f(v)$.

Next we show that the new lengths $\widehat{\ell}$ are nonnegative. For any edge uv of the original graph, the u-to-v distance in the auxiliary graph is clearly at most $\ell(uv)$, so we have $d(v) \leq d(u) + \ell(uv)$. It follows that $\lceil d(v) \rceil \leq \lceil d(u) \rceil + \ell(uv)$, so $\widehat{\ell}(vw) \geq 0$.

To show that the recursion depth is $\log D$, it suffices to show that the maximum shortest-path distance with respect to the lengths $\widehat{\ell}$ is at most $D/2$. For any node x, let P be a shortest path to x according to lengths $\ell(uv)$. By property (b) of the distance estimates, $\lceil d(x) \rceil \geq (1-\frac{1}{2})\ell(P)$. Hence by (5), the distance to x according to lengths $\widehat{\ell}(uv)$ is at most $\ell(P)/2$. $\Box$

4.1 Using procedure FullSearch to solve the $(1-1/2)$-approximate shortest path problem

We now show how our approximation algorithm can be used to get underestimates that satisfy properties (A) and (B). We closely follow the outline of the algorithm of Ullman and Yannakakis. Let G be the input graph, and let n and m denote, respectively, the number of nodes and number of edges in G. The first two steps are as follows. Randomly select $(c+3)\sqrt{n}$ distinguished nodes, where c is a constant. Include the source node among the distinguished nodes. Use FullSearch to find $(1+\epsilon)$-approximate $\sqrt{n}\log n$-limited shortest paths from each distinguished node. The following lemma is the key idea in the algorithm of Ullman and Yannakakis.

Lemma 4.2 (Ullman and Yannakakis) *With probability at least $1 - n^{-c}$, for every pair u, v of nodes of G there is a shortest u-to-v path such that every subpath of size $\sqrt{n}\log n$ contains a distinguished node.*

Next, construct a graph H whose nodes are the distinguished nodes and where there is an edge uv of length $(1+\epsilon)^{-1}d$ if the estimated u-to-v distance is d. Use matrix powering to compute all-pairs shortest paths in H, and let H^* be the complete graph in which the length of the edge uv is the u-to-v distance in H.

We next show that distances in the union of H^* with the input graph G satisfies property (B).

Lemma 4.3 *For every pair u, v of nodes of G, if the u-to-v distance in G is d, the u-to-v distance in $H^* \cup G$ is between $(1-\epsilon)d$ and d.*

Proof: Since every path in G is a path in $H^* \cup G$, the u-to-v distance in $H^* \cup G$ is no more than the corresponding distance in G. Conversely, let P be a shortest u-to-v path in $H^* \cup G$. We must show that there is a corresponding path P'' in G whose length is not much more than that of P.

First, obtain P' from P by replacing each edge of H^* with the corresponding shortest path in H. Since the length of each edge of H^* equals the length of the corresponding shortest path in H, the length of P' equals the length of P.

Next, obtain P'' from P' by replacing each edge xy of H with the shortest x-to-y path P_{xy} of G. For each edge xy of H,

$$\begin{aligned}\text{length}(xy) &= (1+\epsilon)^{-1}\cdot \text{estimated } x\text{-to-}y \text{ distance in } G\\ &\geq (1+\epsilon)^{-1}\cdot \text{actual } x\text{-to-}y \text{ distance in } G\end{aligned}$$

Since we replace each such edge xy with a path of length at most $1+\epsilon$ times the length of xy, the resulting path P'' has length at most $1+\epsilon$ times the length of the shortest u-to-v path P in $H^* \cup G$. Let d be the u-to-v distance in G. We have shown that d is at most $1+\epsilon$ times the u-to-v distance in $H^* \cup G$; equivalently, the u-to-v distance in $H^* \cup G$ is at least $(1+\epsilon)^{-1}d$, which in turn is at least $(1-\epsilon)d$. □

Finally, we obtain single-source distances in $H^* \cup G$. To do this, we rely on the following lemma.

Lemma 4.4 *With probability at least $1 - n^{-c}$, for any node v, there is a shortest source-to-v path in $H^* \cup G$ that consists of at most $1 + \sqrt{n}\log n$ edges.*

Proof: Assume the high-probability event of Lemma 4.2 occurs, and let P be a shortest source-to-v path in $H^* \cup G$. Obtain P' from P as follows: for each maximal subpath P_i of P consisting solely of edges of G, replace P_i by the shortest path with the same endpoints whose existence is promised by Lemma 4.2. The resulting path P' has the following structure. It consists of subpaths in G of size at most $\sqrt{n}\log n$ starting and ending at distinguished nodes, interspersed with edges of H^*, and ending with a subpath in G of size at most $\sqrt{n}\log n$. Furthermore, since P' was obtained by replacing shortest paths in G with shortest paths in G, the length of P' equals the length of P.

Obtain P'' from P' as follows. For each subpath P_i in G of size at most $\sqrt{n}\log n$ starting and ending at distinguished nodes, replace P_i with the edge e_i of H with the same endpoints. Suppose P_i is a u-to-v path, and note that since the estimated u-to-v distance is at most $1+\epsilon$ times the length of P_i, and the length of e_i is defined to be $(1+\epsilon)^{-1}$ times the estimated distance, the length of e_i is at most the length of P_i. Hence the length of P'' is no more than the length of P'.

The structure of P'' is as follows. It consists of a subpath of edges of $H^* \cup H$, terminating in a subpath of G of size at most $\sqrt{n}\log n$. Obtain P''' from P'' by replacing the subpath of edges of $H^* \cup H$ by the single edge with the same endpoints. It follows that the length of P''' is no more than the length of P''' and P''' consists of at most $1 + \sqrt{n}\log n$ edges. □

Lemma 4.4 shows that to compute single-source distances in $H^* \cup G$, it is sufficient to consider paths of size at most $k = 1 + \sqrt{n}\log n$. Cohen [3] has observed that the following variant of Bellman-Ford correctly computes shortest paths in this case. Set $d(v) := \infty$ for each node v, except set $d(\text{source}) := 0$. Repeat the following step k times. For each node v other than the source, compute

$$d(v) := \min\{d(u) + \text{length}(uv) \ : \ uv \text{ an incoming edge of } v\}$$

Now we analyze the above algorithm. Carrying out FULLSEARCH for each distinguished node takes $O(\epsilon^{-1}\sqrt{n}\log n)$ time and $O(m\sqrt{n}\log n \log D)$ work. Since $D \leq n^3$, the work bound is $O(m\sqrt{n}\log^2 n)$. Constructing the graph H can be done within the same bounds. Since H has $O(\sqrt{n})$ nodes, computing all-pairs shortest paths in H can be done in $O(\sqrt{n}\log n)$ time and $O(n^{1.5}\log n)$ work. Finally, using the Bellman-Ford variant to compute single-source shortest paths requires $O(\sqrt{n}\log n)$ stages (as noted above). Each stage can be implemented in constant time with high probability by using Megiddo's [15] algorithm for computing minimum. Therefore the shortest paths in $H^* \cup G$ can be found in $O(\sqrt{n}\log n)$ time and $O(m\sqrt{n}\log n)$ work. We therefore get the bounds of Theorem 1.1.

5 Conclusion

In this paper, we have given an answer to one of the open problems posed by Ullman and Yannakakis: "Does any of this material generalize to the general shortest-path problem, where arcs initially have arbitrary weight?" In particular, we have addressed the case of nonnegative integral weights whose magnitudes are not too big. A more ambitious goal, articulated by Ullman and Yannakakis, is a fast algorithm that works for an arbitrary closed semiring (see [1]). This remains a tantalizing open problem.

Acknowledgments

Thanks to David Shmoys, Matt Pappas, Serge Plotkin, R. Ravi, and David Karger.

References

[1] A. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass. (1974).

[2] H. Bast, M. Dietzfelbinger, and T. Hagerup, "A perfect parallel dictionary," *17th Symposium on Mathematical Foundations of Computer Science* (1992).

[3] E. Cohen, "Efficient parallel shortest-paths in digraphs with a separator decomposition," *5th ACM Symposium on Parallel Algorithms and Architectures* (1993),pp. 57-67

[4] E. Cohen, "Fast algorithms for constructing t-spanners and paths with stretch t," *34th IEEE Symposium on Foundations of Computer Science* (1993).

[5] E. Cohen, "Polylog-time and near-linear work approximation scheme for undirected shortest paths," to appear in *Proceedings, 26th ACM Symposium on Theory of Computing* (1994).

[6] T.H. Cormen, C. E. Lieserson, R.E. Rivest, *Introduction to algorithms*, MIT Press (1990).

[7] D. Eppstein and Z. Galil, "Parallel algorithmic techniques for combinatorial computation,"preprint, Columbia University (1988).

[8] H. N. Gabow, "Scaling algorithms for network problems," *Journal of Computer and System Sciences*, 31:148–168, 1985.

[9] H. Gazit and G. L. Miller, "An improved parallel algorithm that computes the BFS numbering of a directed graph," *Information Processing Letters* 28 (1988), pp. 61-65.

[10] J. Gil, Y. Matias, and U. Vishkin, "Towards a theory of nearly constant time parallel algorithms," *Proceedings, 32nd Symposium on on Foundations of Computer Science* (1991), pp. 698-710.

[11] T. Hagerup and R. Raman "Waste makes haste: Tight bounds for loose parallel sorting,"*Proceedings, 33rd IEEE Symposium on Foundations of Computing* (1992) pp. 628-637.

[12] P. Klein, "A parallel randomized approximation scheme for shortest paths," Brown University Technical Report CS-91-56 (1991).

[13] P. Klein and S. Subramanian, "A parallel randomized approximation scheme for shortest paths," *Proceedings, 24th ACM Symposium on Theory of Computing* (1992) pp. 750-758.

[14] P. Klein and S. Subramanian, "A linear-processor polylog-time algorithm for shortest-paths in planar graphs," *Proceedings, 34th IEEE Symposium on Foundations of Computing* (1993) pp. 750-758.

[15] N. Megiddo, "Parallel algorithms for finding the maximum and the median almost surely in constant time," preprint 1982

[16] H. Shi and T. H. Spencer, "Time-work tradeoffs for the single-source shortest paths problem," Technical Report CS-TR-94-1, Department of Computer Science, University of Nebraska at Omaha (1994).

[17] T. H. Spencer, "More time-work tradeoffs for parallel graph algorithms," *Proceedings, 3rd ACM Symposium on Parallel Algorithms and Architectures* (1991) pp. 81-93.

[18] J. D. Ullman and M. Yannakakis, "High-probability parallel transitive-closure algorithms," *SIAM J. Comput. 20* (1991),pp. 100-125.