Using selective path-doubling for parallel shortest-path computations

Edith Cohen
AT&T Bell Laboratories
Murray Hill, NJ 07974
edith@research.att.com

Revised version February 8, 1996

Abstract

We[1] consider parallel shortest-paths computations in weighted undirected graphs $G = (V, E)$, where $n = |V|$ and $m = |E|$. The standard $O(n^3)$ work path-doubling (Floyd-Warshall) algorithm consists of $O(\log n)$ phases, where in each phase, for every triplet of vertices $(u_1, u_2, u_3) \in V^3$, the distance between u_1 and u_3 is updated to be no more than the sum of the previous-phase distances between $\{u_1, u_2\}$ and $\{u_2, u_3\}$. We introduce a new $\mathcal{NC}$ algorithm that for $\delta = o(n)$, considers only $O(n\delta^2)$ triplets. Our algorithm performs $\tilde{O}(n\delta^2)$ work and augments E with $\tilde{O}(n\delta)$ new weighted edges such that between every pair of vertices, there exists a minimum weight path of size (number of edges) $\tilde{O}(n/\delta)$ (where $\tilde{O}(f) \equiv O(f \operatorname{polylog} n)$). To compute shortest-paths, we apply to the augmented graph algorithms that are efficient for small size shortest paths. We obtain a $\tilde{O}(t)$ time $\tilde{O}(|S|n^2 + n^3/t^2)$ work deterministic PRAM algorithm for computing shortest-paths from $|S|$ sources to all other vertices, where $t \leq n$ is a parameter. When the ratio of the largest edge weight and the smallest edge weight is $n^{O(\operatorname{polylog} n)}$, the algorithm computes shortest paths. When weights are arbitrary, it computes paths within a factor of $1 + n^{-\Omega(\operatorname{polylog} n)}$ of shortest.

[1] A preliminary version appeared in the Proc. of the 2nd Israeli Symposium on the Theory of Computing and Systems, 1993.

1. Introduction

One of the objectives in the design of parallel algorithms is to achieve efficient processor utilization. The amount of work performed by a parallel algorithm is measured by the product of time and number of processors used. The ultimate goal is to exhibit fast algorithm for a problem such that the work performed is as close as possible to the running time of the best known sequential algorithm for the problem. In this paper we consider parallel shortest-paths computations on undirected graphs. The algorithms and bounds discussed are for shared memory parallel machines (PRAM model). We use the notation $G = (V, E)$ for a graph where V is the set of vertices and E is the set of edges. Positive weights $w : E \rightarrow R_+$ are associated with the edges. We use the parameters $R = \max\{1, \lceil \log(\max_{e\in E} w(e)/\min_{e\in E} w(e)) \rceil\}$, $n = |V|$, and $m = |E|$. The weight of a path is the sum of weights of the path edges. The *distance* between a pair of vertices $\{u, v\} \subset V$ is the weight of the minimum-weight path connecting u and v and a *shortest-path* between u and v is a path whose weight equals the distance. We use the tilde notation to hide polylogarithmic factors in n, e.g., $\tilde{O}(f) \equiv O(f \operatorname{polylog} n)$. The single-source shortest-paths problem is computing shortest-paths between a designated source vertex $s \in V$ and all other vertices. We discuss single-source shortest-paths computations for each source $s \in S$ in a designated set of sources $S \subset V$. The classical sequential algorithm (Dijkstra [6]) for the problem runs in $\tilde{O}(|S|m)$ time. The best known $\mathcal{NC}$ algorithm, however, performs $\tilde{O}(n^3)$ work (Floyd-Warshall [6]) and essentially solves the more general all-pairs problem (that is, $|S| = n$). Hence, when $|S| = o(n)$ or $m = o(n^2)$, the $\tilde{O}(n^3)$ work performed by the known $\mathcal{NC}$ algorithm is far from the $\tilde{O}(|S|m)$ work achieved by the sequential algorithm. We mention that $\mathcal{NC}$ algorithms with better processor bounds are known for more restricted weights. Gazit and Miller [8] gave an $M(n)$ processor algorithm for the unweighted single-source problem (BFS tree computation), where $M(n)$ is the number of operations needed to multiply two $n \times n$ matrices (The current bound is $M(n) = o(n^{2.376})$ due to Coppersmith and Winograd [5]). Seidel [13] gave an $O(M(n))$ processor algorithm for the more general unweighted all-pairs problem. A recent $\mathcal{NC}$ algorithm by Galil and Margalit [7] solves the all-pairs problem for integral weights using $\tilde{O}(W^2 M(n))$ processors, where $W = \max_{e\in E} w(e)$. These algorithms actually compute shortest distances but can be modified to compute witnesses for shortest-paths using a technique of Alon, Galil, Margalit, and Naor [1].

Spencer [15] gave a parallel single-source shortest paths algorithms that exhibits tradeoffs between the amount of work performed and the time. His algorithm runs in $\tilde{O}(t)$ time using $O(Rn^3/t^3 + m/t)$ processors. Klein and Sairam [12] gave a randomized parallel approximation scheme for single-source shortest paths computation. Their algorithm reduces the amount of work performed at the cost of compromising the accuracy of the solution. Their algorithm runs in $\tilde{O}(\epsilon^{-2} n^{0.5})$ time, uses $\tilde{O}(m\epsilon^2)$ processors, and approximates shortest-paths distances to within a factor of $(1 + \epsilon)$. Within the same resource bounds the algorithm can be adapted to compute shortest-paths from up to $|S| = \min\{n^{0.5}, m/n\}$ sources. The Klein-Sairam algorithm is based on a randomized parallel algorithm for the unweighted single-source shortest-paths problem (BFS) by Ullman and Yannakakis [17]. The Ullman-Yannakakis [17] algorithm runs in $\tilde{O}(n^\alpha)$ time and uses $\tilde{O}(mn^{1-2\alpha})$ processors (if $m > n^{2-3\alpha}$). According to [12], similar time/work tradeoffs can be achieved for the [12] algorithm.

In this paper we present a $\tilde{O}(Rn/\delta)$ time, $O(|S|n\delta + \delta^3)$ processors, $\tilde{O}(|S|(m + n\delta))$ space, deterministic algorithm for the single-source shortest-paths problem from $|S|$ sources, where $1 \leq \delta \leq n$. Our algorithm exhibits better bounds than Spencer's algorithm when $\delta = \Omega(n^{0.5})$ and $|S| = \Omega(\text{polylog}\, n)$. Note that when the ratio of the largest edge weight to the smallest edge weight is less than $n^{O(\text{polylog}\, n)}$ (the parameter R is polylogarithmic), the running time is $\tilde{O}(t)$ and the algorithm uses $O(|S|n^2/t + n^3/t^3)$ processors, where $t = n/\delta$. For arbitrary weights (when the parameter R could be large), we present an algorithm that computes approximate distances to a factor of at most $(1+\epsilon)$. The approximation algorithm runs in time $\tilde{O}(n/\delta \log \epsilon^{-1})$ using $O(|S|n\delta + \delta^3)$ processors. By compromising the quality of approximations and allowing the time bound to have linear dependence on ϵ^{-1} (in contrast to $\log \epsilon^{-1}$), we improve on the above bounds: We utilize a subroutine introduced by Klein and Sairam [12] and obtain an algorithm that finds paths whose weights is within a factor of $(1+\epsilon)$ from the respective distances in $\tilde{O}(\epsilon^{-2}(n/\delta))$ time with work $O(|S|(m + n\delta) + \delta^3)$. Hence, when $m = \Omega(n\delta)$, and $|S| = \Omega(\delta^3/m)$, the algorithm performs almost optimal work with respect to the best sequential bound. We remark that for undirected graphs, the assumption of strictly positive weights does not limit the generality of the problem, since (i) the presence of negative edge-weights implies that distances are unbounded from below and (ii) zero-weight edges can be efficiently contracted using a connectivity algorithm. Our algorithm is intuitive and easy to implement. The only data structure needed is a standard adjacency lists or adjacency matrix representation of a weighted graph. In contrast, Spencer's algorithm [15] requires more complex data structures (computation and maintenance of a "nearby list" for each vertex).

We give an overview of the ideas and algorithms proposed here. A contribution of independent interest is our selective paths-doubling algorithm. The selective path-doubling algorithm is a variant of the Floyd-Warshall all-pairs shortest-paths algorithm. The Floyd-Warshall algorithm runs in $O(\log n)$ easily-parallelizable phases. In each phase, the algorithm considers all triples of vertices. For each triple $(u_1, u_2, u_3) \in V^3$, the algorithm updates the current distance between (u_1, u_3) to be no more than the sum of the previous-phase distances between (u_1, u_2) and (u_2, u_3). At each phase i and for all pairs of vertices, the current distances are the minimum weight of paths of size at most 2^i. A straightforward parallelization yields an $\mathcal{NC}$ algorithm that uses $\tilde{O}(n^3)$ work. This $\tilde{O}(n^3)$ bound is currently the best-known for $\mathcal{NC}$ computations of shortest-paths on weighted graphs. This bound results from the fact that all triples are considered. The selective path-doubling algorithm considers only a subset of the triples and hence, performs much less work: For a parameter $\delta = o(n)$, the selected triples are such that each vertex is a "middle" vertex in $\tilde{O}(\delta^2)$ triples. Therefore, the selective algorithm is in $\mathcal{NC}$ and performs $\tilde{O}(n\delta^2)$ work. The Floyd-Warshall algorithm can be viewed as computing $O(n^2)$ weighted edges between all pairs of vertices such that shortest distances are achieved on paths of size 1 in the augmented graph. The selective algorithm generates a set of weighted edges with the property that each distance between two vertices u and v is equal to the distance between u and v in the original graph and is obtained by a path with a small number of edges. More specifically, the selective algorithm computes a set E^* of $m + n\delta \log n/2$ weighted edges such that for any two vertices $\{u, v\} \subset V$, there is a path in (V, E^*) consisting of $\tilde{O}(Rn/\delta)$ edges whose weight is equal to the distance between u and v in G.

The shortest-paths algorithm proposed here consists of two stages. In the first stage we execute the selective path-doubling algorithm. In the second stage, we use a modified parallel version of the Bellman-Ford algorithm [6] on the augmented graph (V, E^*) to compute single-source shortest-paths from a vertex s. The Bellman-Ford algorithm runs in phases, where in each phase each vertex scans its neighbors and updates its current distance from s accordingly. At each phase i, the current distances are the minimum-weight of paths of size i. Hence, in general, $O(n)$ phases are needed. Thus, a parallel version of the algorithm runs in $\tilde{O}(n)$ time using $O(m)$ processors. Note, however, that if all shortest distances can be achieved on paths consisting of at most r edges, then r phases suffice and the parallel running time is improved accordingly. Since shortest-distances in (V, E^*) are achieved on paths with a small number of edges, the second stage can be performed by running the Bellman-Ford algorithm for $\tilde{O}(Rn/\delta)$ phases. In a straightforward implementation, an $\mathcal{NC}$ computation of each phase requires $2|E^*| = O(m + n\delta \log n)$ processors. We apply a modified algorithm that performs $\tilde{O}(Rn/\delta)$ time steps, but uses only $O(n\delta \log n)$ processors.

Our approximate single-source shortest-paths algorithm for arbitrary weights makes use of a scheme introduced by Klein and Sairam [12]. Their scheme reduces the problem to single-source shortest-paths computations on a collection of graphs, obtained from G by contracting and deleting edges. The sum of the sizes of the graphs in the collection is at most a factor of $O(\log(n\epsilon^{-1}))$ more than the size of G, and Each of the graphs has a ratio of $O(n\epsilon^{-1})$ between the largest and smallest edge-weights. The approximate algorithm, roughly, computes this collection and subsequently applies our exact algorithm to each of the graphs in the collection.

We discuss the use of our techniques to solve multiple sources shortest-paths problems, where the goal is to find all shortest paths from each vertex in a set of sources $S \subset V$ to all other vertices in V. The multiple sources problem can be solved by running parallel copies of the second stage for every source $s \in S$. We will see that each such copy uses $\tilde{O}(Rn^2)$ work (alternately, $\tilde{O}(\log(n\epsilon^{-1})n^2)$ work to obtain distances within a factor of $(1 + \epsilon)$). The latter bound has almost optimal work for dense graphs, but performs far from optimal work when $m = \tilde{o}(n^2)$. By replacing the second stage with a subroutine used by Klein and Sairam [12], we obtain a randomized procedure that finds paths whose weights is within a factor of $(1+\epsilon)$ from the respective distances in $\tilde{O}(\epsilon^{-2}(n/\delta))$ time with work $O(m + n\delta)$ per source. Hence, when $m = \Omega(n\delta)$, the algorithm performs almost optimal work with respect to the best sequential bound. We comment that for $\epsilon = \Omega(1/\operatorname{polylog}(n))$, these bounds are subsumed by a later result by the author [2].

The shortest paths algorithms stated in this paper actually compute the distances. It is straightforward, however, to add, within the same resource bounds, book-keeping operations such as to find the shortest paths explicitly (Compute a shortest-paths tree rooted at the source s or for the approximate problem, compute a tree such that the distance from s on the tree approximate the distances in the graph).

The paper is structured as follows: In Section 2 we state the selective path-doubling algorithm (first stage) and analyze its resource bounds when implemented on an EREW PRAM. In Section 3 we present the algorithm for the second stage and discuss the parallel implementation and resource bounds on an EREW PRAM of both the second stage algorithm and the combined algorithm. In Sections 4 and 5 we prove the correctness of the algorithms presented in the previous sections. In Section 6 we show how to obtain an approximation algorithm for general weights. In Section 7 we discuss the multiple sources shortest-paths problem. Section 8 contains some concluding remarks and open problems.

Notation. We denote the distance between u and v by $\text{dist}_G\{u,v\}$. We use the terms *shortest path* and *minimum-weight* path interchangeably for a path with a minimum weight. For a subset of edges $E' \subset E$, we denote $w(E') = \sum_{e \in E'} w(e)$. We interchangeably refer to a path p (or set of paths $\mathcal{P}$) as a set of either vertices or edges. By *size* of a path we always mean the number of edges in the path. By *end vertices* (resp. *end edges*) of a path p we refer to the first and last vertices (resp. edges) of p. We use the notation $\deg_{E'}(v) = |\{(u,v)|(u,v) \in E'\}|$ (for $E' \subset E$) and $N(v) = \{u \in V|(v,u) \in E\}$ (for $v \in V$). For a vertex $v \in V$, and a scalar $W \in R_+$, denote $N^W(v) = \{u \in V|(v,u) \in E \wedge w(v,u) \leq W\}$ and $\deg^W(v) = |N^W(v)|$. We treat the parameter δ as having an integral value.

2. The selective path-doubling algorithm

The input to the first-stage algorithm is an undirected graph $G = (V,E)$ with a weight function $w : E \to R_+$. The algorithm produces a set of edges E^* and a weight function $w : E^* \leftarrow R_+$ with the following properties:

i. $|E^*| \leq m + n\delta \log n/2$,

ii. for every two vertices $\{u,v\} \in V$, all paths between u and v in E^* have weight at least $\text{dist}_G\{u,v\}$, and

iii. for every two vertices $\{u,v\} \in V$ there is a path in E^* connecting them of size at most $L = 6nR\log^2 n/\delta + O(nR\log n/\delta) = \tilde{O}(nR/\delta)$ and weight $\text{dist}_G\{u,v\}$.

We give an overview of the algorithm and motivate some of the steps. The algorithm performs $\lfloor \log n \rfloor$ phases. In the kth phase ($k = 1,\ldots,\lfloor \log n \rfloor$) a set E_k of edges with weights $w : E_k \to R_+$ is generated. Each edge $(u,v) \in E_k$ has weight that equals the weight of a path of size 2^k in G between u and v. The algorithm obtains the set E_k by considering the weighted set of edges E_{k-1} that corresponds to paths of size 2^{k-1} and was computed during the $(k-1)$st phase. (We define $E_0 \equiv E$.) The algorithm generates an edge $(u_1,u_2) \in E_k$ of weight $w(u_1,u_2)$ if and only if there is a vertex $v \in V$ such that: $\{(u_1,v),(v,u_2)\} \subset E_{k-1}$, $w(u_1,u_2) = w(u_1,v) + w(v,u_2)$, $\deg^{w(u_1,v)}_{E_{k-1}}(v) \leq \delta$, $\deg^{w(u_1,v)}_{E_{k-1}}(u_1) \leq \delta$, $\deg^{w(v,u_2)}_{E_{k-1}}(v) \leq \delta$, and $\deg^{w(v,u_2)}_{E_{k-1}}(u_2) \leq \delta$. The motivation for the latter degree constraints is that they allow us to perform each phase in $\tilde{O}(n\delta^2)$ total work. To avoid adding clearly redundant parallel edges, we include in E_k only the copy of the edge (u,v) of smallest weight among all qualifying weights. We could also assume that if an edge e of weight $w(e)$ qualifies to be in E_k, but a copy of e is included in $\bigcup_{i \neq k} E_i$ and has weight smaller than $w(e)$, then e is removed from E_k. Note that the resource bounds obtained are valid even when this latter requirement is omitted and the algorithm produces parallel edges of different weights. When the data structure used to maintain the edges is a matrix, we would not want to maintain redundant parallel edges. If, however, adjacency lists are used, we may choose to avoid the unnecessary work involved in removing parallel edges.

The set E^* is chosen as a subset of $\cup_{i\geq 0} E_k$ that contains the set E. We are ready to establish that property ii holds for E^*:

Proposition 2.1 *For all pairs of vertices $\{u_1, u_2\} \in V$, the minimum-weight paths between u_1 and u_2 in the graphs $(V, \bigcup_i E_i)$ and (V, E^*) have weight* $\mathrm{dist}_G\{u_1, u_2\}$.

Proof: The edges $E \equiv E_0$ are included in $\bigcup_i E_i$ and E^* and have assigned weight according to w. Hence, $\mathrm{dist}_{(V,E^*)}\{u_1, u_2\} \leq \mathrm{dist}_G\{u_1, u_2\}$. Each edge $(u_1, u_2) \in E_\ell$ has weight that equals the weight of some path of size 2^ℓ (not necessarily of minimum weight) between u_1 and u_2. Hence $\mathrm{dist}_{(V,E^*)}\{u_1, u_2\} \geq \mathrm{dist}_G\{u_1, u_2\}$. ∎

Denote by $\hat{E} = \bigcup_{i\geq 0} E_i$ the set of all edges produced by Algorithm 2.4. In Section 4 we prove the following Theorem, that is, every pair of vertices has a minimum-weight path in $\hat{E}$ of size at most $L/2$.

Theorem 2.2 *For every pair of vertices $\{s, t\} \subset V$, there is a path in $(V, \hat{E})$, of weight* $\mathrm{dist}_G\{s, t\}$ *and size at most $L/2 = 3nR\log^2 n/\delta + O(nR\log n/\delta)$.*

The cardinality of the set $\hat{E}$ may, however, be $\Omega(n^2)$ even when m is much smaller. Hence, property iii holds for $\hat{E}$, but property i does not. The algorithm chooses the set E^* as the following subset of $\hat{E}$. An edge $e = (u, v) \in E_k$ $(k \geq 0)$ is in E^* if and only if $k = 0$ or $\deg_{E_k}^{w(e)}(u) \leq \delta$ and $\deg_{E_k}^{w(e)}(v) \leq \delta$. We show that property i holds for E^*: Clearly, for $k > 0$ we have for each $v \in V$ $\deg_{E_k \cap E^*}(v) \leq \delta$, and hence, $|E^* \cap E_k| \leq n\delta/2$. It follows that $|E^*| \leq |E| + n\delta \log n/2 = m + n\delta\log n/2$. It remains to show that property iii holds for E^*. We first prove the following:

Proposition 2.3 *If a path of size r and weight w between u and v exists in $(V, \hat{E})$, then a path of size at most $2r$ and the same weight w exists in (V, E^*) between u and v.*

Proof: Consider an edge $(u_1, u_2) \in E_k$. It follows directly from the construction that there exist edges $(u_1, v) \in E_{k-1} \cap E^*$ and $(v, u_2) \in E_{k-1} \cap E^*$ such that $w(u_1, u_2) = w(u_1, v) + w(v, u_2)$. ∎

Since for every pair of vertices there is a minimum-weight path in $(V, \hat{E})$ of size at most $L/2$, we can conclude that there is a minimum-weight path of size at most L in (V, E^*). Hence, property iii holds for the set E^*.

We give a formal statement of the algorithm:

Algorithm 2.4 [*Selective path-doubling*]

i. $E_0 \leftarrow E$

ii. For $i = 0, \ldots, \lfloor \log n \rfloor$:

(1) For $v \in V$: $h_i(v) \leftarrow \max\{W | \deg^W_{E_i}(v) \leq \delta\}$

(2) $E'_i \leftarrow \{e = (v_1, v_2) \in E_i | w(e) \leq h_i(v_1) \wedge w(e) \leq h_i(v_2)\}$

(3) $E_{i+1} \leftarrow \{(v_1, v_2) | \exists u \in V,\ (v_1, u) \in E'_i \wedge (u, v_2) \in E'_i\}$
For $(v_1, v_2) \in E_{i+1}$:
$w(v_1, v_2) \leftarrow \min\{w(v_1, u) + w(u, v_2) | \exists u \in V,\ (v_1, u) \in E'_i \wedge (u, v_2) \in E'_i\}$.

(4) If $E_{i+1} = \emptyset$, stop.

iii. For $v \in V$: $h_{i+1}(v) \leftarrow \max\{W | \deg^W_{E_{i+1}}(v) \leq \delta\}$
$E'_{i+1} \leftarrow \{e = (v_1, v_2) \in E_{i+1} | w(e) \leq h_{i+1}(v_1) \wedge w(e) \leq h_{i+1}(v_2)\}$

iv. $E^* \leftarrow E_0 \cup \bigcup_{i>0} E'_i$

Complexity. Algorithm 2.4 performs $O(\log n)$ phases. Step ii1 amounts to finding the δ lightest edges incident at each vertex. Hence, it can be performed within the same resource bounds as doing selection in a collection of sets with a total of $O(|E_i|)$ items, that is, on an EREW PRAM in $O(\log n \log^* n)$ time using $O(|E_i|/(\log n \log^* n))$ processors (see Cole [3]) or in $O(\log n)$ time using $O(|E_i|)$ processors (using sorting, see Cole [4]). It is easy to see that step ii2 can be performed in $O(\log n)$ time using $O(|E_i|/\log n)$ processors. To perform step ii3, we assign $\lceil \delta^2/\log n \rceil$ processors to each vertex. Each processor considers $\log n$ pairs of incident E'_i edges, in $\log n$ time steps, and generates at most $\log n$ new edges. Note that up to $O(n)$ processors can generate parallel copies of the same edge. In each set of parallel edges, only the parallel edge of smallest weight is retained. The computation of these minimum weight edges can be done within the same bounds as computing the minimum element in each set in a collection of sets of total cardinality $n\delta^2$. Hence, it can be done in $O(\log n)$ time, using $O(n\delta^2/\log n)$ processors (see, e.g., Karp and Ramachandran [10]). It follows that the first phase can be executed using $O(m)$ total work in $\Omega(\log n \log^* n)$ time, and all consecutive phases in $O(\log n)$ time using $O(n\delta^2)$ processors or in $O(n\delta^2)$ work using $\Omega(\log n \log^* n)$ time. Hence, Algorithm 2.4 runs on an EREW PRAM in

$$O(\log^2 n) \text{ time, using } O(m/\log^2 n + n\delta^2) \text{ processors.}$$

When Algorithm 2.4 is used in conjunction with the second-stage to perform a single-source shortest paths computation, we use a slower version and utilize less processors. Note that if we allow for $t = \Omega(\log^2 n \log^* n)$ time units we need $O(m/t + n\delta^2 \log n/t)$ processors (see Brent's principle [6]).

3. Computing single-source shortest-paths

In this section we present the second stage algorithm that computes shortest-paths from a designated source vertex s to all other vertices. The second stage algorithm is provided as input with the weighted graph $G^* = (V, E^*)$ produced by applying the first-stage algorithm to a graph G. Consider computing all minimum-weight paths from s in G^* using the Bellman-Ford single-source shortest-paths algorithm. Since all distances can be obtained by paths of size at most $L = 6nR\log^2 n/\delta + O(n \log nR/\delta)$, the algorithm needs only examine paths consisting of at most L edges. Hence, it suffices to run the Bellman-Ford algorithm for $\lceil L \rceil$ phases. We first analyze the resource bounds of an EREW implementation of the Bellman-Ford algorithm. Later on we state a modified algorithm that achieves a better processor bound. The correctness proof of the modified algorithm is given in Section 5.

Algorithm 3.1 [*L-phases Bellman-Ford algorithm*]

i. For $v \in V$, if $(s, v) \in E^*$ then $d(v) \leftarrow w(s, v)$ else $d(v) \leftarrow \infty$.

ii. For $i = 1, \ldots, \lceil L \rceil$:

For $v \in V$ do (in parallel):
$$d(v) \leftarrow \min\{d(v), \min\{d(u) + w(u, v) | (u, v) \in E^*\}\}$$

When the algorithm terminates we have $d(v) = \text{dist}_{G^*}\{s, v\} = \text{dist}_G\{s, v\}$, for all $v \in V$. We refer to each iteration at step ii as a phase. We show that Algorithm 3.1 can run on an EREW PRAM in $O(L \log n) = O(nR\log^3 n/\delta)$ time, using $O(|E^*|/\log n) = O(m/\log n + n\delta)$ processors. Each phase can be performed in $O(\log n)$ time as follows. We allocate to each vertex $v \in V$ $\lceil \deg_{E^*}(v)/\log n \rceil$ processors. Each such processor is assigned to a set of at most $\lceil \log n \rceil$ incident edges (any arbitrary partition of incident edges will do). At each phase, for each vertex v, each of the processors allocated to v computes (in $O(\log n)$ steps) for each assigned edge (u, v) the quantity $d(u) + w(u, v)$. Thereafter, the $\lceil \deg_{E^*}(v)/\log n \rceil$ processors associated with v compute the minimum of these $\deg_{E^*}(v)$ quantities. This is done in parallel for all vertices. Finding the minimum of $k \le n$ quantities using $k/\log n$ processors takes $O(\log n)$ time. Hence, each phase takes $O(\log n)$ time. The total number of processors used is $\sum_{v \in V} \lceil \deg_{E^*}(v)/\log n \rceil \le 2|E^*|/\log n + n$.

We propose a modification of Algorithm 3.1 that when implemented on an EREW PRAM, runs in the same $O(nR\log^3 n/\delta)$ time but uses only $O(n\delta)$ processors. We give some definitions and then state the modified algorithm.

Definition 3.2 i. We refer to an operation of the form $d(v) \leftarrow \min\{d(v), x, d(u) + w(u, v)\}$, where x is some additional quantity, as *scanning* the edge $(u, v) \in E^*$ *at* the vertex v.

ii. We denote by $\mathrm{scan}_v\{e_1, \ldots, e_k\}$, where e_i $(1 \le i \le k)$ are incident at v, the following parallel operation of simultaneously scanning the edges $\{e_1, \ldots, e_k\}$ at a vertex v:

$$d(v) \leftarrow \min \left\{ d(v), \min_{i=1}^{k} \{d(u) + w(u,v) | e_i = (u,v)\} \right\} .$$

When $k \le m$, the scan operation can be performed on an EREW PRAM in $O(\log n)$ time using $k/\log n$ processors.

We give an overview of the algorithm. Recall that in Algorithm 3.1 every edge is scanned at every phase. In the more efficient modified algorithm, at each vertex $v \in V$, the incident edges are partitioned into $\delta(\lfloor \log n \rfloor + 2\lceil \log(\deg_E(v)/\delta) \rceil)$ sets, where the edges constituting each set are maintained in a circular queue. The edges in each queue are scanned in a round robin fashion: In each phase, for each queue, the current edge is scanned and the next edge becomes current. Each of the (at most $\delta(\lfloor \log n \rfloor)$) incident edges at v that are in $E^* \setminus E$ is assigned singly to a queue. The E-edges incident at v are partitioned into $2\delta\lceil \log(\deg_E(v)/\delta) \rceil$ queues. Note that $\deg_E(v)$ could be larger than the number of queues and hence, some of the E edges are not scanned in every phase. We divide the E-edges incident at v into $\lceil \log(\deg_E(v)/\delta) \rceil$ sets A_v^i $(i = 1, \ldots, \lceil \log(\deg_E(v)/\delta) \rceil)$ as follows: A_v^i contains all the incident edges e such that $(2^{i-1} - 1)\delta < \deg_E^{w(e)}(v) \le (2^i - 1)\delta$ $(i = 1, \ldots, \lceil \log(\deg_E(v)/\delta) \rceil - 1)$. For $i = \lceil \log(\deg_E(v)/\delta) \rceil$, A_v^i contains all the incident edges e such that $(2^{i-1}-1)\delta < \deg_E^{w(e)}(v) \le (2^i)\delta$. It is easy to see that $|A_v^i| \le 2^i\delta$. The edges in each set A_v^i are allotted 2δ queues such that each queue contains either $\lfloor |A_v^i|/(2\delta) \rfloor$ or $\lceil |A_v^i|/(2\delta) \rceil$ edges. Note that edges in A_v^i are assigned to queues of size at most 2^{i-1}. We allocate $(\delta(\lfloor \log n \rfloor + 2\lceil \log(\deg_E(v)/\delta) \rceil))/\log n$ processors to each vertex, where each processor is assigned at most $\lceil \log n \rceil$ queues. In a single phase, a processor scans the current edge in each of the queues assigned to it and then makes the next edges current in the respective queues.

Proposition 3.3 *At each vertex v, incident edges in $E^* \setminus E$ are scanned in every phase and incident edges in E are scanned at least once every $\lceil \deg_E^{w(v)}(v)/\delta \rceil$ consecutive phases.*

Proof: The edges in $E^* \setminus E$ are assigned singly to a queue and hence are scanned in every phase. An edge $e \in A_v^i$ is assigned to a queue of size at most

$$|A_v^i|/(2\delta) \le 2^{i-1} \le \lceil \deg_E^{w(e)}(v)/\delta \rceil .$$

Hence, e is scanned at least once every $\lceil \deg_E^{w(v)}(v)/\delta \rceil$ consecutive phases. ∎

We give a formal statement of the algorithm.

Algorithm 3.4 [Modified Bellman-Ford algorithm]

- *Initialization:*

i. For $v \in V$: if $(s,v) \in E^*$ then $d(v) \leftarrow w(s,v)$ else $d(v) \leftarrow \infty$.
   $P_v \leftarrow \{(u,v) \in E^* \setminus E | u \in V\}$
   For $i = 1, \ldots, \lceil \log(\deg_E(v)/\delta) \rceil$:
      $A_v^i \leftarrow \{e = (v,u) \in E | (2^{i-1} - 1)\delta < \deg_E^{w(e)}(v) \leq (2^i - 1)\delta\}$
      If $i = \lceil \log(\deg_E(v)/\delta) \rceil$, $A_v^i \leftarrow \{e = (v,u) \in E | (2^{i-1} - 1)\delta < \deg_E^{w(e)}(v)\}$
      Partition the edges in A_v^i into 2δ circular queues:
      Q_v^{ij} $(j = 1, \ldots, 2\delta)$ of sizes $\{\lfloor |A_v^i|/(2\delta) \rfloor, \lceil |A_v^i|/(2\delta) \rceil\}$

- *Main part (iterating phases):*

ii. For $k = 1, \ldots, \lceil L + 6nR/\delta \rceil$:
   For $v \in V$ do (in parallel):
      $\text{scan}_v \{P_v \cup \{\text{Next}(Q_v^{ij}) | 1 \leq i \leq \lceil \log(\deg_E(v)/\delta) \rceil, 1 \leq j \leq 2\delta\}\}$

In Subsection 5 we prove that Algorithm 3.4 is correct, that is, at termination we have $d(v) = \text{dist}_G\{s,v\}$, for all $v \in V$.

Complexity of the second stage. We state resource bounds of an implementation of Algorithm 3.4 on an EREW PRAM. At each phase, n parallel scan operations are performed, one at each vertex. Observe that each such scan involves at most $\delta(\lfloor \log n \rfloor + 2 \log n)$ edges and is done by $\delta(\lfloor \log n \rfloor + 2 \log n)/\log n$ processors. Hence, each phase requires $O(\log \delta + \log\log n) = O(\log n)$ time using $O(n\delta)$ processors. The algorithm performs $\lceil L + 6nR/\delta \rceil = O(nR \log^2 n/\delta)$ phases, and hence, the computation is performed in

$$O(nR \log^3 n/\delta) \text{ time using } O(n\delta) \text{ processors.}$$

Complexity of single-source shortest paths. When used in conjunction with the second-stage, we use a slower version of the first-stage algorithm (Algorithm 2.4). When allowing for $t = O(nR \log^3 n/\delta)$ time units, Algorithm 2.4 requires $O(m\delta/(nR \log^3 n) + \delta^3/(R \log^2 n))$ processors.
It follows that the single-source shortest-paths problem can be solved in

$$O(nR \log^3 n/\delta) \text{ time, using } O(n\delta + \delta^3/(R \log^2 n)) \text{ processors.}$$

4. Correctness of the first-stage algorithm

In this section we prove Theorem 2.2. We first provide intuition. The theorem claims that even when a minimum-weight path p in G has many edges, Algorithm 2.4 performs enough "edge-doubling" steps to guarantee that the augmented graph $(V, \hat{E})$ has a corresponding minimum-weight path of small size. Consider, for simplicity, an unweighted graph. In the first phase the algorithm doubles all subpaths of size 2 that are not incident at vertices of degree larger than δ. The edges that are not doubled at the first phase will not be considered later on. Hence, we should make sure that there are not too many such edges on any shortest path. Consider a shortest path p. It is easy to verify that each vertex $v \in V$ can be adjacent to at most 3 vertices from p. It follows that $\sum_{u \in p} \deg(u) \leq 3n$, and hence, p contains at most $3n/\delta$ vertices of degree δ or larger. The proof of the theorem relies on a generalization of the above statement.

We give some definitions and then present the proof of the theorem. Consider a minimum weight path $p = (u_0, \ldots, u_k)$ between a pair of vertices $\{u_0, u_k\}$ in the graph. We interchangeably use the notation $p = (e_1, \ldots, e_k)$ for the edges on the path p (where $e_i = (u_{i-1}, u_i)$ for $1 \leq i \leq k$). For a vertex $u_i \in p$, we denote by $h_p(u_i) = \max\{w(e_i), w(e_{i+1})\}$, the weight of the heaviest p-edge incident at u_i. We denote by

$$E_p = \bigcup_{0 \leq i \leq k} \{(u_i, v) \in E | w(u_i, v) \leq h_p(u_i)\} ,$$

the set of edges incident at vertices of p such that the weight of an edge incident at $u \in p$ does not exceed $h_p(u)$.

Proposition 4.1 *A vertex $v \in V$ can be adjacent, using E_p edges, to at most $6R$ vertices from p.*

Proof: Consider a scalar $W \in R_+$ and a vertex $v \in V$. A simple case analysis (aided by Figure 1(A)-(D)) establishes that since p is a shortest path,

$$\left| N_{E_p}(v) \cap \{u_i | W \leq h_p(u_i) \leq 2W\} \right| \leq 6 .$$

Denote $V_W = N_{E_p}(v) \cap \{u_i | W \leq h_p(u_i) \leq 2W\}$ and by E_W the E_p edges between v and V_W. Associate every edge $e \in E_W$ with an adjacent p edge of weight at least $w(e)$. The E_W and associated edges are marked by thick solid lines. Note that $|V_W| = |E_W|$, since each vertex in V_W corresponds to exactly one E_W edge. Consider the left-most and right-most V_W vertices in p (see Figure (A)). By definition $a \leq 2W$ and $b \leq 2W$. Since p is a shortest paths, the segment of p between these vertices must have weight at most $a + b \leq 4W$. Associated p edges have weight at least W, hence, the p-segment is either a continuous segment of 4 associated edges of weight W (as in Figure (B)), or contains at most 3 associated edges (see Figure (C)). Case (B) is infeasible since there are at most 5 associated vertices and there must be 6 associated edges (the edges associated with a, b must have weight $2W$.) Assume to the contrary that it is case (C) and $|E_W| > 6$. This implies that at least three of the p-segments $\{x, y, z, w\}$ must be nonempty. It follows, that a subgraph of the form (D), where $b > 0$, is a subset of the E_W and associated edges. Since p is a shortest path, we have $a + b + c \leq d + g$. By definition of associated edges, $d \leq a$ and $g \leq c$. Hence, a contradiction. This concludes the claim that $|E_W| \leq 6$. Therefore,

$$\deg_{E_p}(v) = \left| N_{E_p}(v) \right| \leq 6 \max\{1, \lceil \log(\max_{e \in p} w(e) / \min_{e \in p} w(e)) \rceil\} \leq 6R .$$

∎

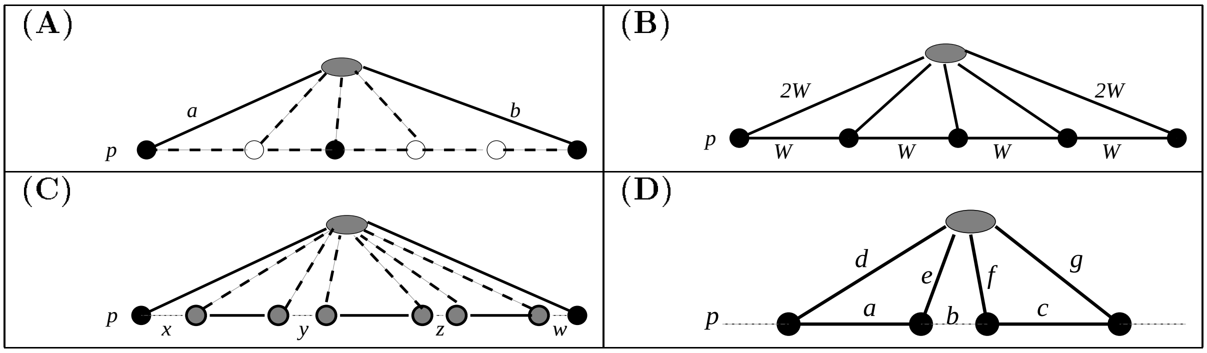

Figure 1: Cases for proving $|Z_W| \leq 6$

Corollary 4.2 *It follows that*

$$\sum_{u\in p} \deg_{E_p}(u) \leq \sum_{v\in V} \deg_{E_p}(v) \leq 6nR \ .$$

Hence, p contains at most $6nR/\delta$ vertices with δ or more incident E_p edges

$$\Big|\{u \in p | \deg_{E_p}(u)| \geq \delta\}\Big| \leq 6nR/\delta \ .$$

We will show that a similar statement holds for the following generalization of a shortest path:

Definition 4.3 Consider a collection of paths $p_i = (u^i_0, \ldots, u^i_{n_i})$ (the respective edges are $p_i = (e^i_1, \ldots, e^i_{n_i})$) $(i = 1, \ldots, r)$ in an undirected graph $G = (V, E)$ with an associated weight function $w : E \leftarrow R_+$. We say that the ordered set $\mathcal{P} = (p_1, \ldots, p_r)$ is a *fractured shortest path* if

i. for $1 \leq i \leq r$, p_i is a shortest path between its end vertices $\{u^i_0, u^i_{n_i}\}$, and

ii. for every two vertices on different paths $u^{i_1}_{j_1} \in p_{i_1}$, $u^{i_2}_{j_2} \in p_{i_2}$ (suppose that $i_1 < i_2$) we have
$$\operatorname*{dist}_G\{u^{i_1}_{j_1}, u^{i_2}_{j_2}\} > \sum_{j_1<k\leq n_{i_1}} w(e^{i_1}_k) + \sum_{1\leq k\leq j_2} w(e^{i_2}_k) + \sum_{i_1<k<i_2} w(p_k) \ .$$

We refer to the paths p_i $(1 \leq i \leq r)$ as the *segments* of the fractured shortest path $\mathcal{P}$.

For a given fractured shortest path $\mathcal{P}$ we define $h_{\mathcal{P}}(u)$, for $u \in \mathcal{P}$, to be the weight of the heaviest $\mathcal{P}$ edge incident at u. We define

$$E_{\mathcal{P}} = \bigcup_{u\in\mathcal{P}} \{(u, v) \in E | w(u, v) \leq h_{\mathcal{P}}(u)\} \ .$$

Proposition 4.4 *A fractured shortest path $\mathcal{P}$ contains at most $6nR/\delta$ vertices with δ or more incident $E_{\mathcal{P}}$-edges*

$$|\{u \in \mathcal{P} | \deg_{E_{\mathcal{P}}}(u) \geq \delta\}| \leq 6nR/\delta \ .$$

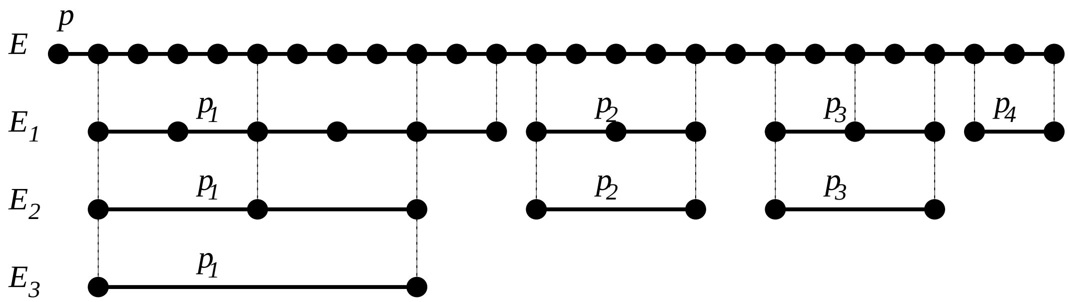

Figure 2: A path p and respective fractured shortest paths in E_1, E_2, E_3.

Proof: The arguments used in the proofs of Proposition 4.1 and Corollary 4.2 carry over to fractured shortest paths. The key to the proof is to verify that likewise for shortest paths, the conditions in Definition 4.3 guarantee that the fractured shortest path $\mathcal{P}$ is such that for every $v \in V$, $\deg_{E_{\mathcal{P}}}(v) \leq 6R$. ∎

The following proposition establishes that at any stage ℓ of the algorithm and for any shortest path p in G, the (V, E_ℓ) subpaths obtained when E_ℓ edges shortcut nonoverlapping portions of p, form a fractured shortest path in (V, E_ℓ). Figure 2 illustrates the claim.

Proposition 4.5 *Suppose that $p = (v_0, \ldots, v_k)$ (with respective edges $(e_1, \ldots, e_k)$) is a shortest path in $G = (V, E)$ and E_ℓ is the set of edges generated at the ℓth phase of Algorithm 2.4 when applied to G. Consider a set of paths $\mathcal{P} = (p_1, \ldots, p_r)$ in (V, E_ℓ) of the form*

$$p_i = (v_{a_i}, v_{a_i+2^\ell}, v_{a_i+2\cdot 2^\ell}, \ldots, v_{a_i+b_i\cdot 2^\ell}) \quad (i = 1, \ldots, r) ,$$

where $a_r + b_r 2^\ell \leq k$ and $a_i + b_i 2^\ell < a_{i+1}$ $(i = 1, \ldots, r-1)$, such that

$$\text{for } i = 1, \ldots, r \text{ and } j = 0, \ldots, b_i - 1, \quad w(v_{a_i+j2^\ell}, v_{a_i+(j+1)2^\ell}) = \sum_{h=a_i+j2^\ell+1}^{a_i+(j+1)2^\ell} w(e_h) .$$

It follows that the set $\mathcal{P}$ constitutes a fractured shortest path in (V, E_ℓ).

Proof: We show that the first condition of Definition 4.3 holds for $\mathcal{P}$. The construction implies that for every pair of vertices $\{u, v\}$,

$$\operatorname*{dist}_{(V,E_\ell)}\{u, v\} \geq \operatorname*{dist}_G\{u, v\} .$$

It is easy to see that for all i,

$$\operatorname*{dist}_G\{v_{a_i}, v_{a_i+b_i\cdot 2^\ell}\} = \operatorname*{dist}_{(V,E_\ell)}\{v_{a_i}, v_{a_i+b_i\cdot 2^\ell}\} = w(p_i).$$

Hence, $p_1, \ldots, p_r$ are shortest paths in (V, E_ℓ). It remains to prove that the second condition of Definition 4.3 holds. Consider two vertices on different paths $v_{a_i+k_1\cdot 2^\ell} \in p_i$ and $v_{a_j+k_2\cdot 2^\ell} \in p_j$, where $i < j$. Since p is a shortest path,

$$\operatorname*{dist}_G\{v_{a_i+k_1\cdot 2^\ell}, v_{a_j+k_2\cdot 2^\ell}\} = \sum_{h=a_i+k_1\cdot 2^\ell+1}^{a_j+k_2\cdot 2^\ell} w(e_h) .$$

Hence,

$$\operatorname*{dist}_{(V,E_\ell)}\{v_{a_i+k_1\cdot 2^\ell}, v_{a_j+k_2\cdot 2^\ell}\} \geq \operatorname*{dist}_{G}\{v_{a_i+k_1\cdot 2^\ell}, v_{a_j+k_2\cdot 2^\ell}\}$$

$$> \sum_{h=a_i+2^\ell k_1+1}^{a_i+2^\ell b_i} w(e_h) + \sum_{h=a_j+1}^{a_j+2^\ell k_2} w(e_h) + \sum_{i<h<j} w(p_h) \ .$$

The last inequality is strict since weights of edges are positive. ∎

To prove Theorem 2.2 we consider two vertices $\{s,t\} \subset V$ and construct a path p^* in $(V, \hat{E})$ of weight $\mathrm{dist}_G\{s,t\}$ and size at most $L/2 = 3nR\log^2 n/\delta + O(nR\log n/\delta)$. We base our construction of p^* on a shortest path p_{st} between s and t in $G = (V, E)$. The edges we select for p^* correspond to a partition of p_{st} to edge-disjoint segments, where the end-vertices of each segment constitute an edge in $\hat{E}$ whose weight equals the weight of the segment in (V, E) (we say that the selected edge *covers* the segment). The sizes of segments are powers of two, where segments of size 2^i are covered by edges in E_i ($1 \leq i \leq \lceil \log n \rceil$). In order to construct p^*, we choose, for $i = 0, \ldots, \lfloor \log n \rfloor$, sets of edges $F_i \subset E_i$. At the ith step we consider an ordered set of paths $\mathcal{P}_i$ in the graph (V, E_i). The segments of $\mathcal{P}_i$ relate to the path p_{st} in G as in the statement of Proposition 4.5, and hence, constitute a fractured shortest path in (V, E_i). The end vertices of segments of $\mathcal{P}_i$ correspond to the end vertices of portions of p_{st} not covered by the already-chosen edges $\bigcup_{j<i} F_j$. We choose the set $F_i \subset E_i$ to be of cardinality at most $\tilde{O}(nR/\delta)$ and such that the segments in $\mathcal{P}_i \setminus F_i$ can be covered by paths consisting of E_{i+1} edges.

We elaborate on this process. Consider the fractured shortest path $\mathcal{P}_i$ in (V, E_i), and assume it has k_i segment. We point on a set F_i of at most $18nR/\delta + k_i$ edges that when removed from $\mathcal{P}_i$, the refined segments obtained (the segments of $\mathcal{P}_i \setminus F_i$) are (i) of even or zero size and (ii) every edge $e = (v_1, v_2)$ in such a segment is such that $\deg_{E_i}^{w(e)}(v_1) \leq \delta$ and $\deg_{E_i}^{w(e)}(v_2) \leq \delta$ (equivalently, $e \in E'_i$).
We choose the set F_i to include all $\mathcal{P}_i$ edges that are in $E_i \setminus E'_i$. We also include in F_i one end-edge from each of the odd-size segments obtained when $E_i \setminus E'_i$ edges are removed from $\mathcal{P}_i$. It follows from the construction that $\mathcal{P}_i \setminus F_i$ satisfies properties (i) and (ii). We bound the cardinality of the set F_i: Proposition 4.4 asserts that there are at most $6nR/\delta$ vertices such that $v \in \mathcal{P}_i$ and $\deg_{E_{\mathcal{P}_i}}(v) \geq \delta$. Hence, $|\mathcal{P}_i \cap \{E_i \setminus E'_i\}| \leq 12nR/\delta$ and the number of segments in $\mathcal{P}_i \setminus \{E_i \setminus E'_i\}$ is at most $k_i + 6nR/\delta$. Therefore, the total number of edges in F_i is at most $18nR/\delta + k_i$.
Denote by $p_1, \ldots, p_r$ the set of non-zero size segments in $\mathcal{P}_i \setminus F_i$. Since all the edges in the paths p_j ($1 \leq j \leq r$) are in E'_i, each subpath of size 2 is covered by an edge in E_{i+1}. Hence, for each $1 \leq j \leq r$, the list of vertices at even positions in the path p_j constitutes a valid path p'_j in E_{i+1} between the two end vertices of p_j. Note that $w(p_j) = w(p'_j)$. It follows from Proposition 4.5 that $\mathcal{P}_{i+1} = (p'_1, \ldots, p'_r)$ is a fractured shortest path in (V, E_{i+1}). It is easy to see that the total number of edges in $\mathcal{P}_{i+1}$ is at most half of that of $\mathcal{P}_i$.

To construct the path p^*, we apply the above process where the initial fractured shortest path is $\mathcal{P}_0 \equiv p_{st}$. (Hence, $\mathcal{P}_0$ has only one segment.) We iterate the above process and construct $F_0, \ldots, F_i, \ldots$ and $\mathcal{P}_0, \ldots, \mathcal{P}_i, \ldots$. We stop when $\mathcal{P}_i$ is empty. This happens within at most $\lfloor \log n \rfloor$ iterations (since $|\mathcal{P}_0| \leq n$ and $|\mathcal{P}_{i+1}| \leq |\mathcal{P}_i|/2$ for $i \geq 0$).

It can be easily verified that $\bigcup_i F_i$ constitutes an s-t path in $\hat{E}$ whose weight equals $\mathrm{dist}_G\{s,t\}$. We bound the cardinality of $\bigcup_i F_i$. Denote by k_i the number of segments in $\mathcal{P}_i$ ($i \geq 0$). We showed that $k_0 = 1$ and for all i, $k_{i+1} \leq k_i + 6nR/\delta$. Hence, $k_i \leq 1 + 6nRi/\delta$. It follows that $|F_i| \leq (18+6i)nR/\delta + 1$. Therefore,

$$\sum_i |F_i| \leq 3nR\log^2 n/\delta + 17nR\log n/\delta = L/2 \ .$$

This concludes the proof of the theorem.

5. Correctness of the second-stage algorithm

Consider an application of Algorithm 3.4 to a weighted graph $G^* = (V, E^*)$ that was produced by applying Algorithm 2.4 to an input graph G. We show that when Algorithm 3.4 terminates, we have $d(u) = \mathrm{dist}_G\{s,u\}$ for all $u \in V$.

Definition 5.1 A path $p = (s \equiv u_0, \ldots, u_r)$ is *scanned* by Algorithm 3.4 within $k \geq r$ phases, if there exists a sequence of phases $1 \leq j_1 < j_2 < \cdots < j_r \leq k$ such that the edge (u_i, u_{i+1}) is scanned at vertex u_{i+1} in the j_ith phase.

The following proposition is immediate:

Proposition 5.2 *Consider a run of Algorithm 3.4. If some path p from s to u was scanned prior to the current phase, then the current value of $d(u)$ is at most $w(p) = \sum_{e \in p} w(e)$.*

It follows that to prove the correctness of Algorithm 3.4 it suffices to show that for every vertex $u \in V$, there exists a minimum-weight path $\hat{p}$ in (V, E^*) between s and u that is scanned within $L + 6nR/\delta$ phases.

Consider a vertex u and let p^* be the minimum-weight path between s and u in $(V, \bigcup_i E_i)$ that is constructed by the procedure described in Section 4 (the correctness proof of Algorithm 2.4). Let $p_{su} = (s = v_0, \ldots, v_k = u)$ be the shortest path from s to u in G that the construction was based on. Recall that the construction is such that the E-edges of p^* are a subseries of the edges of p_{su}. Consider a minimum-weight path $\hat{p} = (s = u_0, \ldots, u_r = u)$ in (V, E^*) obtained from p^* by replacing each p^* edge in $\bigcup_i E_i \setminus E^*$ by the appropriate path of size two and the same weight in E^*, as described in Proposition 2.3. The replacements can be done such that edges in $E_1 \setminus E^*$ are replaced by E-edges contained in p_{su}. Note that (see Section 2): (i) $r \leq L$ and (ii) the E-edges in $\hat{p}$ constitute a subseries of p_{su}.

To complete the correctness proof, we show that the path $\hat{p}$ is scanned within $\lceil L + 6nR/\delta \rceil$ phases. For an edge $e = (u_i, u_{i+1}) \in \hat{p}$ define

$$\mathrm{delay}(e) = \begin{cases} 1 & e \in E^* \setminus E \\ \lceil \deg_E^{w(u_{i+1})}(u_{i+1})/\delta \rceil & e \in E \end{cases}$$

Proposition 3.3 asserts that an edge (u_i, u_{i+1}) is scanned at u_{i+1} at least once every delay(e) phases. Hence, the path $\hat{p}$ is scanned within

$$\sum_{e\in\hat{p}} \text{delay}(e) \leq r + \sum_{\{0\leq i\leq r-1 | e_{i+1}\equiv(u_i,u_{i+1})\in E\}} \deg_E^{w(e_{i+1})}(u_{i+1})/\delta \leq L + 6nR/\delta$$

phases. The second inequality follows from the fact that $E \wedge \hat{p}$ is a subseries of the shortest path p_{su} and hence, using Corollary 4.2, we have

$$\sum_{\{0\leq i\leq r-1 | e_{i+1}\equiv(u_i,u_{i+1})\in E\}} \deg_E^{w(e_{i+1})}(u_{i+1}) \leq \sum_{i=0}^{k-1} \deg_E^{w(v_i,v_{i+1})}(v_{i+1}) \leq 6nR \ .$$

This concludes the correctness proof of Algorithm 3.4.

6. Approximate distances for general weights

In this section we consider graphs with arbitrary edge weights. We present an algorithm that computes paths from a single source whose weights is within a factor of $(1+\epsilon)$ from the respective distances. The approximate algorithm is based on combining a modified version of our single-source shortest path algorithm with a technique used by Klein and Sairam [12]. For a weighted graph $G = (V, E)$ with weights $w : E \to R_+$, a designated source $s \in V$, and $\epsilon > 0$. The algorithm computes quantities $\tilde{\text{dist}}\{s, u\}$ (for $u \in V$), such that

$$\forall u \in V, \text{dist}\{s, u\} \leq \tilde{\text{dist}}\{s, u\} \leq (1+\epsilon)\,\text{dist}\{s, u\} \ .$$

A subtree of G rooted at s such that the distances on the tree from s are at most $\tilde{\text{dist}}\{s, u\}$ (for $u \in V$) can be generated using the same resource bounds. The resource bounds of our approximate single-source shortest-paths algorithm are

$$O(n \log(n\epsilon^{-1}) \log^3 n/\delta) \text{ time, using } O(n \log(n\epsilon^{-1})\delta + \delta^3/\log^3 n) \text{ processors.}$$

Hence, the approximate algorithm is efficient when $\epsilon^{-1} = n^{O(\text{polylog}\, n)}$.

Klein and Sairam [12] used the following idea. Consider a weighted graph and suppose that we are interested in pairs of vertices with distances which are in $[W, 2W]$, and that it suffices to know distances in that range approximately, within a factor of $(1+\epsilon)$. We generate a graph from G such that edges of weight smaller than $\epsilon W/(2n)$ are contracted and edges of weight larger than $2W$ are deleted. Denote this graph by $G_W = (V_W, E_W)$. We refer to the vertices V_W as supervertices and view them as subsets of vertices of G such that a subset contain all the G vertices contracted to form the corresponding supervertex. For a vertex $v \in V$ denote by $S(v) \in V_W$ ($S(v) \subset V$) the supervertex in G_W that v is a member of. Note that the different sets in $S(v)$ ($v \in V$) form a partition of V. It is easy to verify that (i) if two vertices $\{u, v\} \subset V$ have distance $d \leq 2W$ in G, then $\{S(u), S(v)\} \subset V_W$ have distance in $[d - \epsilon W/2), d]$ in G_W, and (ii) if $\{S(u), S(v)\} \subset V_W$ have distance $d \leq 2W$ in G_W then $\{u, v\} \subset V$ have distance $[d, d + \epsilon W/2)]$ in G. Hence, by performing shortest-paths computations on G_W we can approximate to within a factor of $(1+\epsilon)$ distances between pairs of vertices whose distances in G are in the range $[W, 2W]$. Note that graphs of the form G_W have a maximum ratio of at most $O(n\epsilon^{-1})$ between the largest and smallest edge-weights. Hence, the single-source shortest paths algorithm of previous sections is efficient for such graphs.

Consider two supervertices $\{S(u), S(v)\} \in V_W$ such that $d = \mathrm{dist}_{G_W}\{S(u), S(v)\} \in [W, 2W]$ and a corresponding minimum weight path p_W. We remark that a path p between u and v in G of weight $[d, d+\epsilon W/2)]$ can be obtained as follows. The construction of G_W is such that an edge $e' \in E_W$ corresponds to an edge $(u, v) \in E$ such that $w(e) = w(e')$ and $e' = ((S(u), S(v)))$. We refer to the vertices $\{u, v\} \subset V$ as the *V-end points* of e'. The E-edges that correspond to the edges of p_W form a set of paths $p_1, \ldots, p_k$ in G (where $k \geq 1$), such that the first vertex of p_1 is u, the last vertex of p_k is v (p_1 and p_k are possibly of zero size), and for $1 \leq i < k$, the last vertex of p_i and the first vertex of p_j are in the same supervertex $S_i \in V_W$. Let q_i $(1 \leq i < k)$ be any simple path in G between the last vertex of p_i and the first vertex of p_{i+1} that contains only edges of weight less than $\epsilon W/(2n)$ (the existence of these paths is immediate from the construction.) The sum of the sizes of the paths q_i is at most n and hence, $\sum_{1\leq i<k} w(q_i) < \epsilon W/2$. It follows that $p = \bigcup_{1\leq i\leq k} p_i \cup \bigcup_{1\leq i<k} q_i$ is a path between u and v in G such that $w(p_W) \leq w(p) \leq w(p_W) + \epsilon W/2$.

Klein and Sairam [12] gave a subroutine that produces efficiently the collection of weighted graphs $G_1, \ldots, G_k$ that includes (in the same order) all the different graphs among $G_{2^\ell w_{\min}}$ $(\ell \geq 0)$ with a nonempty set of edges, where $w_{\min} = \min_{e\in E} w(e)$. (It is easy to see that $k \leq m$.) Their subroutine is based on combining a parallel prefix computation with a connected components algorithm (they use the bounds of e.g., Shiloach and Vishkin [14] connected components algorithm) and generates these graphs in $O(\log^2 n)$ time on a CRCW PRAM or in $O(\log^3 n)$ time on an EREW PRAM, using $O(m)$ processors. It is immediate that the supervertices in G_i $(1 \leq i \leq k)$ are unions of disjoint set of supervertices in G_j $(j < i)$. For a pair of vertices u_1, u_2 denote by $w^*(u_1, u_2)$ the weight of the minimum-weight edge e such that the removal of all edges in G of weight larger than $w(e)$ disconnects u_1 and u_2. Obviously $w^*(u_1, u_2) \leq \mathrm{dist}_G\{u_1, u_2\} \leq n w^*(u_1, u_2)$. The Klein-Sairam subroutine that generates $G_1, \ldots, G_k$ can also produce, within the same bounds, a spanning tree T of G, where for any pair of vertices u_1, u_2, the weights of all edges in the T-path between u_1 and u_2 do not exceed $w^*(u_1, u_2)$. Obviously, $\mathrm{dist}_T\{u_1, u_2\} \leq n\, \mathrm{dist}_G\{u_1, u_2\}$. Subsequently we assume that the subroutine computes $\mathrm{dist}_T\{u_1, u_2\}$ for all pairs of vertices (this can be done in $O(n^2 \log n)$ work).

For G_i denote by $[a_i, b_i]$ the smallest interval such that $[W, 2W] \subset [a_i, b_i]$ for all W where $G_i = G_W$ $(1 \leq i \leq k)$. Denote by $S_i(v)$ $(v \in V, 1 \leq i \leq k)$ the supervertex in G_i containing v (if one exists). Denote by G'_i the connected component that contains s in G_i $(1 \leq i \leq k)$. We discuss some properties of the graphs G_i and G'_i $(1 \leq i \leq k)$. It is easy to see that a pair of vertices $\{u_1, u_2\} \subset V$ are members of different supervertices of the same connected component in G_W only if $\epsilon W/(2n) \leq w^*(u_1, u_2) \leq 2W$. Hence, for at most $\log(n\epsilon^{-1})$ values of $1 \leq i \leq k$ the vertices $\{u_1, u_2\} \subset V$ are such that $S_i(u_1) \neq S_i(u_2)$ but $S_i(u_1)$ and $S_i(u_2)$ are in the same connected component of G_i. It follows that for at most $\log(n\epsilon^{-1})$ of the graphs G'_i, a vertex $v \in V$ is a member of a supervertex other than the supervertex containing s. Denote by n_i and m_i the number of vertices and edges, respectively, in G'_i $(1 \leq i \leq k)$. We have $\sum_i m_i \leq m \log(n/\epsilon)$ and $\sum_i n_i \leq n \log(n\epsilon^{-1})$.

We are ready to present the approximate single-source shortest paths algorithm.

Algorithm 6.1 [Approximate single-source shortest paths]

i. Apply the Klein-Sairam subroutine discussed above to compute the graphs $G'_1, \ldots, G'_k$ and the tree T.

ii. {First stage} For $1 \leq i \leq k$, in parallel, apply to the graph G'_i the following modification of the first stage algorithm (Algorithm 2.4:
The basic operation of Algorithm 2.4 is the generation of new edges by considering paths of size two among the edges of the current phase. The modified algorithm differs from Algorithm 2.4 only in the performance of this operation. Recall that the edges of G'_i correspond to edges in G between their V-end points. When generating new edges, the modified Algorithm 2.4 keeps track of the V-vertices the endpoints of the new edge correspond to and associates weights with the new edges as follows. Suppose that the modified algorithm considers a path of size two consisting of the edges $e'_1 = (S_i(v_1), S_i(v_2))$ and $e'_2 = (S_i(u_1), S_i(u_2))$ such that the corresponding V-end points are (v_1, v_2) and (u_1, u_2), respectively, and $S_i(v_2) = S_i(u_1)$. The modified algorithm generates the new edge $e' = (S_i(v_1), S_i(u_2))$, and keeps record of (v_1, u_2) as being the V-end points of e'. The modified algorithm determines the weight of the edge e' to be $w(e') = w(e'_1) + w(e'_2) + \text{dist}_T\{v_2, u_1\}$.
For $1 \leq i \leq k$, denote by $G^*_i = (V_i, E^*_i)$ the weighted graphs obtained by applying the modified Algorithm 2.4 to G'_i. Maintain the information about the V-end points of the edges in E^*_i.

iii. {Second stage} In parallel for $1 \leq i \leq k$ do as follows:
Apply the second stage algorithm (Algorithm 3.4) to G^*_i with respect to the source vertex $S_i(s)$. Compute a shortest-paths tree T_i of G^*_i rooted at $S_i(s)$, that is, for each supervertex $S_i(u) \in V_i$ compute the ingoing tree edge $e(S_i(u)) \in E^*_i$ and a predecessor supervertex $\text{pred}(S_i(u)) \in V_i$ such that

$$\underset{G^*_i}{\text{dist}}\{S_i(s), S_i(u)\} = w(e(S_i(u))) + \underset{G^*_i}{\text{dist}}\{S_i(s), \text{pred}(S_i(u))\} .$$

The computation of T_i can be performed within the resource bounds of Algorithm 3.4.
For every supervertex $S_i(u)$, define the local source of $S_i(u)$ to be the vertex $s'(S_i(u)) \in S_i(u)$ that corresponds to the $S_i(u)$ V-end point of the edge $e(S_i(u))$. Define the local source of the root vertex to be $s'(S_i(s)) \equiv s$.
Define $t'(S_i(u)) \in \text{pred}(S_i(u))$ to be the V-end point of $e(S_i(u))$ that is not in $S_i(u)$.
For every supervertex $S_i(u) \in V_i$ update the weight of the T_i edge connecting it to its father as follows:

$$w(e(S_i(u))) \leftarrow w(e(S_i(u))) + \underset{T}{\text{dist}}\{s'(\text{pred}(S_i(u))), t'(S_i(u))\} .$$

Compute $\text{dist}_{T_i}\{S_i(s), S_i(u)\}$ for every supervertex $S_i(u) \in V_i$.
Let $V'_i \subset V_i$ be the set of supervertices such that $\text{dist}_{T_i}\{S_i(s), S_i(u)\} \in [a_i(1-\epsilon), b_i]$.

For all vertices $v \in \bigcup_{A \in V_i'} A \subset V$, define

$$\tilde{\text{dist}}_G\{s,v\} = \text{dist}_{T_i}\{S_i(s), S_i(v)\} + \text{dist}_{T}\{s'(S_i(v)), v\} \ .$$

It is easy to verify that for every $v \in V$,

$$\text{dist}_G\{s,v\} \leq \tilde{\text{dist}}_G\{s,v\} \leq (1+\epsilon)\,\text{dist}_G\{s,v\} \ ,$$

and that $\tilde{\text{dist}}_G\{s,v\}$ is the weight of some path between s and v in G. Book keeping operations can be added such that within the same resource bounds we obtain an approximate shortest-paths tree that explicitly represents these paths.

Complexity. The complexity of step i is dominated by the other steps of the algorithm. In step ii, the weighting of edges done by the modified algorithm is such that the weight of a new edge differs only by a factor of $(1+\epsilon)$ than what the weight would have been in a normal execution of Algorithm 2.4. Thus, it is easy to verify that the complexity analysis done in Sections 2 and 4 still holds and hence, the modified algorithm runs within the same asymptotic resource bounds as Algorithm 2.4. It follows that when G_i' gets an allotment of $O(m_i/t + n_i\delta^2/t)$ processors, the modified first stage can be performed in $O(t)$ time (if $t = \Omega(\log^2 n \log^* n)$). The total number of processors used is $O(\log(n\epsilon^{-1})(m+n\delta^2)/t)$. In step iii, we allocate $n_i\delta$ processors to G_i^* for the run of algorithm 3.4 ($1 \leq i \leq k$). The resulting running time is $O(n\log(n\epsilon^{-1})\log^3 n/\delta)$. Since $\sum_i n_i \leq n\log(n\epsilon^{-1})$, the number of processors used is $O(n\delta\log(n\epsilon^{-1}))$. The final step of computing the approximate distances can be done for G_i' in $O(\log n)$ time using $O(|\bigcup_{A \in V_i'} A|)$ work. We had shown above that

$$\forall v \in V, |\{1 \leq i \leq k | v \in \bigcup_{A \in V_i'} A\}| \leq \log(n\epsilon^{-1}) \ .$$

Hence, the total work performed in the final step is

$$\sum_{1 \leq i \leq k} |\bigcup_{A \in V_i'} A| \leq n\log(n\epsilon^{-1}) \ .$$

It follows that step iii can be performed in $O(n\log(n\epsilon^{-1})\log^3 n/\delta)$ time using $O(n\delta\log(n\epsilon^{-1}))$ processors.

7. Shortest paths from multiple sources

This section is concerned with computing shortest paths from each vertex in a given set $S \subset V$ of sources to all other vertices. The randomized BFS algorithm of Ullman and Yannakakis [17] runs in $\tilde{O}(t)$ time (we assume $t \leq n^{0.5}$), and uses $\tilde{O}(mn/t + n^3/t^4)$ work. By carefully examining their algorithm we can see that within the same resource bounds, the algorithm can be modified to compute shortest paths information from up to $O(n/t)$

different sources at the cost of $\tilde{O}(n/t)$ work per pair of vertices. Thus, within the resource bounds their algorithm can handle $\tilde{O}(n^2/t, m + n^2/t^3)$ pairs and therefore find all shortest paths from a set of $\tilde{O}(\min\{n/t, m/n + n/t^3\})$ sources. The cost of adding a "source" is $\tilde{O}(m + n^2/t^3)$ work. The cost of computing a shortest path from a source to some vertex is $\tilde{O}(n/t)$ regardless of the current number of sources. It seems from [12], that the randomized approximation algorithm of Klein and Sairam [12] for the weighted case handles additional sources and source-sink pairs in essentially the same resource bounds.

We apply the techniques and algorithms presented in previous sections to solve the multiple sources problem. We assume that the weights are such that the ratio of the largest weight to the smallest weight is $n^{O(\mathrm{polylog}\, n)}$ or otherwise, the bounds are valid for an approximate distances to within a factor of $1 + O(n^{-\mathrm{polylog}\, n})$. We first suggest a straightforward extension of our single-source shortest-paths algorithm to the multiple sources problem. Recall that only the second stage (Algorithm 3.4) does source-specific computation and it runs in $\tilde{O}(n/\delta)$ time, using $O(n\delta)$ processors. Hence, $\tilde{O}(n^2)$ total work is performed in the second stage for all values of the parameter δ. The first stage (Algorithm 2.4) performs $\tilde{O}(n\delta^2)$ total work. Hence, the problem of finding all paths from each vertex in a set of $O(\delta^2/n)$ sources can be solved within the same resource bounds as solving the problem for a single source. Note that when $n\delta^2 = \tilde{O}(m\delta^2/n)$, that is, $m = \tilde{\Omega}(n^2)$ (graph is dense), the total work performed has an almost optimal speedup (up to polylogarithmic factors) with respect to the best known sequential bounds.

We showed how $\Omega(\delta^2/n)$ sources can be handled at the cost of $\tilde{O}(n^2)$ work per source. When $m = \tilde{o}(n^2)$, the $\tilde{O}(n^2)$ bound is far from the $O(m)$ work per source performed by the sequential algorithm. If we relax our requirements and seek paths that are at most a factor of $(1+\epsilon)$ away from shortest where $\epsilon^{-1} = O(\mathrm{polylog}\, n)$, we can solve multiple sources problem using $\tilde{O}(m + n\delta)$ work per source. To do so, we utilize the following CRCW PRAM subroutine proposed as a part of an algorithm by Klein and Sairam [12, 11]. For an integer $1 \le k \le n$ and a scalar $\epsilon > 0$, it computes paths p_{sv} from a source $s \in V$ to all other vertices $(v \in V)$ such that for each path p_{sv}, the weight of p_{sv} is at most $(1+\epsilon)$ times the weight of the shortest path of size at most k from s to v. Their subroutine runs in time $O(m \log n/p + k\epsilon^{-2} \log n \log^* n)$ using p processors. If we choose $p = m\epsilon^2/(k \log^* n)$, their subroutine runs in $O(k\epsilon^{-2} \log n \log^* n)$ time and performs $O(m \log^2 n)$ work. Hence, the second stage of our algorithm can be replaced by an approximate computation done by the above subroutine with $k = \tilde{O}(n/\delta)$. This can be done in $\tilde{O}(\epsilon^{-2} n/\delta)$ time using $p = (m + n\delta)\epsilon^2\delta/n$ processors. It follows that the additional work for computing all shortest paths approximately from a new source (in $\tilde{O}(n/\delta)$ time) is $\tilde{O}(m + n\delta)$. Thus, within the resource bounds of the first stage we can approximately find all shortest paths from a set of $O(n\delta^2/(m + n\delta))$ sources.

For unweighted graphs, if we are willing to compromise the precision even further and be content with paths that are shortest up to a small multiplicative factor and an additive term of $\tilde{O}(n/\delta)$, we can compute shortest paths from new sources in $\tilde{O}(n\delta)$ work per source, even when $m = \tilde{\Omega}(n\delta)$. This can be achieved as follows. First, we contract all vertices V_0 in G where, $V_0 = \{v \in V | \deg_E(v) > \delta\}$. By contracting $v \in V$, we refer to the process of merging all the neighbors of v into a supervertex. The new supervertices can be computed by applying a connected components algorithm to the graph induced by edges incident at

V_0 vertices. We denote the graph with contracted vertices by $G' = (V', E')$. For $v \in V$, we denote by $v' \in V'$ the supervertex containing v. We apply the shortest paths algorithm mentioned in the previous paragraph to the graph G'. Since $|E'| \leq n\delta$, this can be done in the desired $\tilde{O}(n\delta)$ work per source. To obtain an approximate distance from u to v in G, we compute the distance in G' from the supervertex containing u to the supervertex containing v. To prove the correctness of the above procedure, it suffices to show that for any pair of vertices $\{u, v\} \subset V$,

$$\operatorname*{dist}_G\{u, v\} - 6n/\delta \leq \operatorname*{dist}_{G'}\{u', v'\} \leq \operatorname*{dist}_G\{u, v\} .$$

The rightmost inequality is immediate. We justify the leftmost inequality. In Section 4 we had seen that every shortest path in G between some pair of vertices $\{v, u\} \subset V$, contains at most $O(n/\delta)$ vertices of degree larger than δ. We generalize this and show that on any path p between $\{v, u\} \subset V$, there are at least $\mathrm{dist}_G\{u, v\} - 3n/\delta$ vertices of degree smaller than δ. It is easy to see that the latter claim concludes the correctness proof of the algorithm. To prove the claim, consider a path $p = (u \equiv u_0, u_1, \ldots, u_r \equiv v)$ from u to v. Let $i_k = \min\{0 \leq j \leq r | \operatorname{dist}\{u, u_j\} = k\}$ ($k = 0, \ldots, \mathrm{dist}_G\{u, v\}$). It is easy to see that u_{i_k} are well defined (for $k = 0, \ldots, \mathrm{dist}_G\{u, v\}$). By definition, a vertex $v \in V$ can be adjacent to at most 3 vertices in $U' = \{u_{i_k} | 0 \leq k \leq \operatorname{dist}\{u, v\}\}$. Hence, $\sum_{w \in U'} |N(w)| \leq 3n$. It follows that at most $3n/\delta$ vertices in U' have degree δ or larger. Hence, at least $\mathrm{dist}_G\{u, v\} - 3n/\delta$ vertices in p have degree smaller than δ. Since the latter holds for every such path p, we conclude that $\mathrm{dist}_{G'}\{u', v'\} \geq \mathrm{dist}_G\{u, v\} - 6n/\delta$.

The PRAM resource bounds discussed in this paper for the undirected multiple-sources shortest-paths problem are summarized in Table 1. In the statement of the bounds presented in this paper we assume that either $R = O(\operatorname{polylog} n)$ or only approximate distances are obtained to within a factor of $1+O(n^{-\operatorname{polylog} n})$. The bounds stated for [12] assume $m > n^{2-3\alpha}$ and $\alpha \leq 0.5$ and are based on extrapolating from [17]. The bound stated for [15] assumes $R = O(\operatorname{polylog} n)$.

8. Discussion

To conclude, we discuss some issues and open problems that arise from this work.

Exact shortest-paths. Our shortest-paths algorithm runs efficiently on graphs where the edge weights are such that R (that is, the ratio of the largest edge weight to the smallest edge weight) is small (e.g., bounded by a quasipolynomial in n). For general weights, our modified algorithm computes near-shortest paths. An interesting open question is whether the dependence on the weights can be eliminated. It is easy to verify that the complexity analysis of the algorithm holds for more general weights, where R is substituted in the resource bounds by the following parameter r. Define r to be the minimum number such that there are r scalars $a_1, \ldots, a_r$ with the property that

$$\forall e \in E, w(e) \in \bigcup_{1 \leq i \leq r} [a_i, 2a_i] .$$

alg.	*number of sources*	*time*	*processors*	*work*
Deterministic algorithms				
e.g.,[6]	1	$\tilde{O}(m)$	1	$\tilde{O}(m)$
	s	$\tilde{O}(sm)$	1	$\tilde{O}(sm)$
[7]	n	$\tilde{O}(1)$	$W^2M(n)$	$\tilde{O}(W^2M(n))$
e.g.,[6]	n	$\tilde{O}(1)$	$O(n^3)$	$\tilde{O}(n^3)$
[15]	1	$\tilde{O}(t)$	$O(n^3/t^3+m/t)$	$\tilde{O}(n^3/t^2+m)$
This	$\tilde{O}(n/t^2)$	$\tilde{O}(t)$	$O(n^2/t+n^3/t^3)$	$\tilde{O}(n^2+n^3/t^2)$
paper	s	$\tilde{O}(t)$	$O(n^3/t^3+sn^2/t)$	$\tilde{O}(sn^2+n^3/t^2)$
Approximate shortest-paths to a factor of $(1+\epsilon)$ *with high probability*				
[12]	$\min\{n/t, m/n\}$	$\tilde{O}(\epsilon^{-2}t)$	$O(\epsilon^2 mn/t^2)$	$\tilde{O}(mn/t)$
Approximate shortest-paths to a factor of $(1+\epsilon)$				
This	$\tilde{O}((n^2+n^3/t^2)/(m+n^2/t))$	$\tilde{O}(\epsilon^{-2}t)$	$O(\epsilon^2(n^2/t+n^3/t^3))$	$\tilde{O}(n^2+n^3/t^2)$
paper	$\tilde{\Omega}((n^2+n^3/t^2)/(m+n^2/t))$	$\tilde{O}(\epsilon^{-2}t)$	$O(s\epsilon^2(m/t+n^2/t^2))$	$\tilde{O}(sm+sn^2/t)$

Table 1: Parallel bounds for shortest paths from s sources

Minimum-hop diameter. A weighted graph has *minimum-hop diameter (hop-diameter)* λ, if for every pair of vertices, the distance between them is achieved by paths of size at most λ. A motivation for this definition is that parallel single-source shortest-paths computations can be performed efficiently on graphs with a small hop-diameter. Namely, in $\tilde{O}(\lambda)$ time using $O(m)$ processors (by a parallel Bellman-Ford algorithm), and, for near-shortest paths, in $\tilde{O}(\lambda)$ time using $O(m/\lambda)$ processors (by the technique of Klein and Sairam [12], see Section 7). The first stage (Algorithm 2.4) with parameter $1 \leq \lambda \leq n$ (we choose $\delta = \tilde{O}(Rn/\lambda)$) produces $\tilde{O}(Rn^2/\lambda)$ additional edges such that the distances in the augmented graph are equal to the original distances, and the hop-diameter of the augmented graph is λ. The computation is performed in polylogarithmic time using $\tilde{O}(n^3R^2/\lambda^2)$ total work. In the unweighted case, the algorithm of Ullman and Yannakakis [17] takes $\tilde{O}(\lambda)$ time, and $\tilde{O}(mn/\lambda)$ work (when $m > n^2/\lambda^3$), and produces a graph of hop-diameter λ by adding $O(n^2/\lambda^2)$ edges. We remark that for every weighted graph G a corresponding weighted graph with distances that equal distances in G, and hop-diameter $O(\lambda)$ can be obtained by adding $O(n^2/\lambda^2)$ edges as follows. Randomly select a subset of n/λ vertices and add a complete graph on these vertices, where the weights on the additional edges equal the distances in the original graph. It follows from standard arguments (see [9]) that with high probability, the augmented graph has the desired hop-diameter. The number of edges added in the latter process is only $O(n^2/\lambda^2)$. It is not clear, however, how to generate these weights efficiently in parallel. An interesting open question is to design algorithms that produce, efficiently in parallel, small augmentations of an input graph such that distances are preserved and the augmented graph has a small hop-diameter. This question was partially answered (for approximately-shortest paths) in a later work by the author [2].

Directed graphs. The single-source shortest-paths algorithms of Spencer [16] and (for unweighted graphs) Ullman and Yannakakis [17] are applicable to directed graphs. Our algorithm, however, is suitable only for undirected graphs. The difficulty in extending our algorithm to directed graphs stems from the first stage (Algorithm 2.4). The analysis of the first stage algorithm is based on a generalization of the fact that for any shortest path p, the sum of degrees of vertices on p is bounded by $O(n)$. It is easy to construct directed graphs, however, where the sums of indegrees and outdegrees of vertices on a shortest path are both $\Omega(n^2)$.

Acknowledgement. The author would like to thank David Johnson, John Overdeck, Vijaya Ramachandran, Peter Shor, Mario Szegedy, Alex Wang, Mihalis Yannakakis, and the anonymous referee for remarks on earlier versions of this manuscript, bibliographic pointers, and/or relevant discussions.

References.

[1] N. Alon, Z. Galil, O. Margalit, and M. Naor. Witnesses for boolean matrix multiplication and for shortest paths. In *Proc. 33rd IEEE Annual Symposium on Foundations of Computer Science*, pages 417–426. IEEE, 1992.

[2] E. Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest-paths. In *Proc. 26th Annual ACM Symposium on Theory of Computing*, pages 16–26. ACM, 1994.

[3] R. J. Cole. An optimally efficient selection algorithm. *Information Processing Let.*, 26:255–259, 1988.

[4] R. J. Cole. Parallel merge sort. *SIAM J. Comput.*, 17:770–785, 1988.

[5] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9:251–280, 1990.

[6] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to algorithms*. McGraw-Hill Book Co., New York, 1990.

[7] Z. Galil and O. Margalit. A faster algorithm for the all pairs shortest path problem for undirected graphs. Manuscript.

[8] H. Gazit and G. L. Miller. An improved parallel algorithm that computes the BFS numbering of a directed graph. *Information Processing Let.*, 28(2):61–65, 1988.

[9] H. Greene and D. E. Knuth. *Mathematics for the analysis of algorithms*. Birkhauser, Boston, 1982.

[10] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In J. van Leeuwenm, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 17, pages 869–941. MIT Press/Elsevier, Cambridge, MA, 1991.

[11] P. N. Klein. A parallel randomized approximation scheme for shortest paths. Draft of journal version, 1992.

[12] P. N. Klein and S. Sairam. A parallel randomized approximation scheme for shortest paths. In *Proc. 24th Annual ACM Symposium on Theory of Computing*, pages 750–758. ACM, 1992.

[13] R. Seidel. On the all-pairs shortest-path problem. In *Proc. 24th Annual ACM Symposium on Theory of Computing*, pages 745–749. ACM, 1992.

[14] Y. Shiloach and U. Vishkin. An $O(\log n)$ parallel connectivity algorithm. *J. Alg.*, 3:57–67, 1982.

[15] T. H. Spencer. More time-work tradeoffs for parallel graph algorithms. In *Proc. 3rd annual ACM Symposium on Parallel Algorithms and Architectures*, pages 81–93. ACM, 1991.

[16] T. H. Spencer. Time-work tradeoffs for parallel graph algorithms. In *Proc. 2nd ACM-SIAM Symposium on Discrete Algorithms*, pages 425–432. ACM-SIAM, 1991.

[17] J. D. Ullman and M. Yannakakis. High-probability parallel transitive closure algorithms. *SIAM J. Comput.*, 20:100–125, 1991.