Efficient parallel shortest-paths in digraphs with a separator decomposition

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

Revised, August, 1995

Abstract

We consider[1] shortest paths and reachability problems on directed graphs with real-valued edge weights. For sparser graphs, the known $\mathcal{NC}$ algorithms for these problems perform much more work than their sequential counterparts. In this paper we present efficient parallel algorithms for families of graphs, where a separator decomposition is either provided with the input or is easily obtainable. (A separator is a subset of the vertices that its removal splits the graph into connected components, such that the number of vertices in each component is at most a fixed fraction of the number of vertices in the graph. A separator decomposition is a recursive decomposition of the graph using separators.) Let $G = (V, E)$, where $n = |V|$, be a weighted directed graph with a k^{μ}-separator decomposition (where subgraphs with k vertices have separators of size $O(k^{\mu})$). We present an $\mathcal{NC}$ algorithm that computes shortest paths from s sources to all other vertices using $\tilde{O}(n^{3\mu} + s(n + n^{2\mu}))$ work. A sequential version of our algorithm improves over previously-known time bounds as well. Reachability from s sources can be computed using $\tilde{O}(M(n^{\mu}) + s(n + n^{2\mu}))$ work, where $M(r) = o(r^{2.37})$ is the best known work bound for $r \times r$ matrix multiplication. The algorithm is based on augmenting G with a set of $\tilde{O}(n^{2\mu})$ edges such that in the augmented graph, all distances can be obtained by paths of size $O(\log n)$. The above bounds, with $\mu = 0.5$, are applicable to planar graphs, since a $k^{0.5}$-separator decomposition can be computed within these bounds. We obtain further improvements for graphs with planar embeddings where all vertices lie on a small number of faces.

[1]An extended abstract appeared in Proc. 5th annual ACM Symposium on Parallel Algorithms and Architectures, 1993

1. Introduction

The reachability problem on a directed graph $G = (V, E)$ is to determine, for a specified source vertex $v \in V$, all vertices reachable from v in G. When real-valued edge weights $w : E \to \mathcal{R}$ are associated with the edges, the single-source shortest-paths problem is to determine directed paths of minimum weight from v to all other vertices, where the weight of a path is defined to be the sum of the weights of the edges on the path. Let $m = |E|$ and $n = |V|$. We use the notation $\tilde{O}(f) \equiv O(f \operatorname{polylog} f)$. Although the reachability and shortest path problems from s sources can be solved sequentially in $\tilde{O}(ms)$ time ($\tilde{O}(mn)$ time if negative weights are allowed), the best $\mathcal{NC}$ algorithms for shortest paths use $\tilde{O}(n^3)$ work and for reachability $\tilde{O}(M(n))$ work, where $M(r) = o(r^{2.37})$ (see Coppersmith and Winograd [2]) is the best known work bound for $r \times r$ matrix multiplication. Hence, when $m \ll n^2$ or $s \ll n$ the parallel algorithms perform considerably more work. This reality is referred to in the literature as the "transitive closure bottleneck." In this paper we present efficient $\mathcal{NC}$ algorithms that overcome this bottleneck for families of graphs that have a recursive decomposition using small separators.

Our results. A separator of a graph $G = (V, E)$ is a subset of vertices $S \subset V$ such that the subgraph induced by $V \setminus S$ is not connected. It is typically required that the number of vertices in each connected component is at most a fixed fraction of the number of vertices in the graph. For a function $f : \mathcal{N} \to \mathcal{R}$, an $f(k)$-separator decomposition of G is, roughly, a recursive decomposition of G using separators, where subgraphs of size k have separators of size $O(f(k))$. Consider a directed graph $G = (V, E)$ with weights $w : E \to \mathcal{R}$ and a k^μ-separator decomposition of G. We present an algorithm that finds minimum weight paths from s sources by first performing a preprocessing step in $O(\log^2 n)$ time and $O((n + n^{3\mu}) \log n)$ work, if $\mu \neq 1/3$, or $O(n \log^2 n)$ work, if $\mu = 1/3$. After the preprocessing, the algorithm takes $O(\log^2 n)$ time and uses $O(s(n + n^{2\mu}))$ work, if $\mu \neq 1/2$, or $O(sn \log n)$ work, if $\mu = 1/2$. An alternative algorithm performs the preprocessing stage in $O(\log^3 n)$ time using $O(n + n^{3\mu})$ work if $\mu \neq 1/3$ and $O(n \log n)$ work otherwise. See Table 1 for a summary. For graph reachability, the preprocessing can be done using $O((n + M(n^\mu)) \log^2 n)$ work if $\omega\mu \neq 1$ and $O(n \log^3 n)$ otherwise, where ω is the matrix multiplication exponent $M(r) = O(r^\omega)$. The parallel bounds discussed in this paper are on the EREW PRAM model unless explicitly stated otherwise. An example of a family of graphs with a readily available separator decomposition is d'-dimensional grid graphs, where $d \leq d'$ dimensions have more than a constant width. For these graphs there is a trivial $k^{(d-1)/d}$-separator decomposition. Other examples are bounded tree-width graphs with a tree decomposition (see e.g., Robertson and Seymour [19]) and r-overlap graphs, proposed by Miller, Teng, and Vavasis [15]. Miller *et al.* had shown that r-overlap graphs embedded in d dimensions have a separator bound of $O(r^{1/d} n^{(d-1)/d})$ and these separators can be computed by a randomized algorithm in polylogarithmic time using linear work. r-overlap graphs constitute an important family of graphs that in two dimensions includes the class of planar graphs, and in d dimensions includes all subgraphs of the d-dimensional grid. We comment that sequential versions of our algorithms are an improvement over previous sequential algorithms. For general directed graphs with real edge-weights, the best known

time	*preprocessing work*		*work per source*	
$\log^2 n$	$\mu \neq 1/3$	$(n + n^{3\mu}) \log n$	$\mu = 1/2$	$n \log n$
	$\mu = 1/3$	$n \log^2 n$	$\mu \neq 1/2$	$n + n^{2\mu}$
$\log^3 n$	$\mu \neq 1/3$	$n + n^{3\mu}$		
	$\mu = 1/3$	$n \log n$		

Table 1: Shortest paths in graphs with k^μ-separators

sequential time bound for computing shortest-paths from s sources is $O(mn + n^2 \log n)$, using a Fibonacci heap implementation of Johnson's algorithm (see [3]). For planar graphs Frederickson[5] gave a $O(n^2)$ time algorithm. As far as we know, these are also the previously best known bounds for graphs with a separator decomposition (in particular, for the examples above).

An interesting application of our algorithm outside the shortest paths realm is obtaining faster sequential algorithms for solving linear systems of inequalities where each inequality involves at most two variables, when the underlying graph has a separator decomposition. The underlying graph of such a system has a vertex corresponding to each variable and an edge to each inequality between the vertices corresponding to the variables participating in the inequality. The fastest known algorithm for solving such systems, due to Cohen and Megiddo [1], runs in $\tilde{O}(n^3 + mn)$ expected time, where n is the number of variables and m is the number of inequalities. The $\tilde{O}(n^3)$ factor in the running time arises from the work bound of a Floyd-Warshall parallel shortest-paths computation. The algorithm can use instead, however, the work bound of any polylog-time directed all-pairs shortest paths algorithm that is applicable to the underlying graph and to path algebra problems over semirings. Hence, our results imply that when the underlying graph of the system of inequalities has a k^μ-separator decomposition, the system can be solved in $\tilde{O}(n^{1+2\mu} + mn)$ time.

For planar graphs, a separator decomposition such that $\mu = 0.5$ can be found in $O(n^{1+\epsilon})$ work and $O(\log^3 n)$ time on a CRCW PRAM for any $\epsilon > 0$ (see Gazit and Miller [7]). Hence, shortest paths from s sources can be found in $O(\log^3 n)$ time and $O(n^{1.5} + sn \log n)$ work on a CRCW PRAM. We further improve over the above bounds for families of graphs that have planar embeddings such that all vertices lie on $q \ll n$ faces. The improvement is achieved by combining our algorithm with an algorithm by Pantziou, Spirakis, and Zaroliagis [18] that is based on a sequential algorithm by Frederickson [6]. Their algorithm reduces the problem to computing shortest-paths in a graph of size $O(q)$. It suffices to compute a $k^{0.5}$-separator decomposition of a planar graph with $O(q)$ vertices. We use the algorithm of Gazit and Miller [7] to compute this decomposition in $O(\log^4 q)$ time and $O(q^{1+\epsilon})$ work on a CREW PRAM. We show that a representation of all-pairs shortest paths (by a compact routing table) can be produced in $O(\log^4 q + \log^2 n)$ time using $O(nq \log^2 n)$ work on a CREW PRAM, shortest-path trees from s sources can be produced in $O(\log^4 q + \log^2 n)$ time using $O(n \log^2 n + q^{1.5} + s(n + q \log q))$ work, and distances between k pairs of vertices can be produced in $O(\log^4 q + \log^2 n)$ time using $O(n \log^2 n + q^2 \log q + k \log n)$ work.

We give some comments. (i) Note that if negative weight cycles are present, some of the distances are not defined. It is simple, however, to adopt the algorithm to detect negative weight cycles within the same resource bounds. (ii) The algorithm as stated computed only distances, but it can be easily adapted to explicitly find minimum weight paths. (iii) Our algorithm is applicable to general path algebra problems over semirings (see Pan and Reif [16] and [3] for applications). (iv) The separator decomposition for a graph G depends only on the undirected unweighted skeleton of G, and hence needs to be computed only once for a group of instances which differ in the weights and direction of edges. (v) In addition to providing significant asymptotic improvements over previous algorithms, our algorithm are intuitive and simple, and hence, may be of value in practice. (Particularly in situations where a separator-decomposition is readily available, for example, multi-dimensional grid-like graphs or planar graphs.)

Previous work. Pan and Reif [16, 17] extended their parallel nested dissection algorithm and presented a class of parallel algorithms for path algebra problems on *undirected* graphs. Their algorithms rely on the availability of a separator decomposition of the undirected graph G and exhibit comparable bounds to ours. Lingas [13] considered *planar* directed graphs where the edge weights are restricted to be *nonnegative and have polynomial size.* His algorithm assumes that a separator decomposition of the graph is available, where the separators have to be simple cycle separators. The work bounds given are worse by a factor of $O(\log^2 n)$ than our algorithm with $\mu = 0.5$. For planar digraphs for which we are provided with embeddings such that all vertices are on $q \ll n$ faces, Pantziou, Spirakis, and Zaroliagis [18] presented a CREW parallel algorithm for all-pairs shortest paths information with real-valued edge weights. Their algorithm is based on parallelizing the hammock decomposition technique introduced by Frederickson [6]. Their algorithm runs in $O(\log^2 n)$ time and employs $O(nq + q^3)$ processors. The transitive closure can be found using $O(nq + M(q))$ processors. Another outcome of the parallel hammock decomposition technique is an algorithm by Djidjev, Pantziou, and Zaroliagis [4] that computes distances between k specified pairs of vertices in a planar graph in $O(\log^2 n)$ time using $O((n + q^3)\log^2 n)$ work. Kao and Klein [10] gave a $\tilde{O}(n)$ work $\mathcal{NC}$ algorithm for single source reachability in planar digraphs. Their algorithm is based on a $\tilde{O}(n)$ work strongly connected components algorithm by Kao [11]. Klein and Sairam [12] gave a parallel approximation algorithm for single source shortest paths on undirected planar graph with nonnegative weights. Their algorithm finds approximate paths to within a factor of $(1 + \epsilon)$ (for a fixed ϵ) and runs in $\tilde{O}(n^{1/10})$ time performing $\tilde{O}(n)$ work.

Overview. We give an overview of our algorithm. The algorithm is given as input a directed graph $G = (V, E)$ with real weights $w : E \to \mathcal{R}$, along with a separator decomposition tree T_G of the undirected skeleton of G. The tree T_G is a rooted binary tree with subgraphs of G associated with its nodes. The root of T_G contains the original graph. With each node $t \in T_G$ we associate a separator for the subgraph at the node t. The subgraphs at the children of t correspond to the partition induced by the separator at t. The algorithm uses T_G to recursively construct a set of edges $E^+ \subset V \times V$ with weights $w : E^+ \to \mathcal{R}$. The set E^+ is such that distances in $G = (V, E)$ are the same as in $G^+ = (V, E \cup E^+)$. In addition, every distance in G^+ can be achieved by a path with at most $O(d_G)$ edges, where d_G is the height of the tree T_G (the size of the longest path from the root to a leaf). Given the graph G^+, single source shortest path information can be obtained in $O(d_G \log n)$ time with $O(|E \cup E^+|d_G)$ work per source, using a straightforward parallel version of the Bellman-Ford algorithm (see [3]). We show how to reduce the work to $O(|E \cup E^+|)$. In Section 2 we give some definitions. In Section 3 we define the set E^+ with respect to G and the tree T_G and prove that E^+ has the desired properties. We also discuss an efficient way of computing shortest-paths using the set E^+. In Section 4 we present algorithms for computing the set E^+. In section 5 we analyze the resource bounds of these algorithms when the decomposition tree is such that for some fixed $\mu < 1$, at each node, if the subgraph has k vertices, the separator is of size $O(k^\mu)$. In Section 6 we discuss algorithms for planar graphs.

2. Preliminaries

In this section we give some definitions and notation.

2.1. Notation

For a graph $G = (V, E)$ and $V' \subset V$, we denote by $\mathcal{E}(V') \subset E$ the set of edges with two end points in V', we denote by $N(V') \subset V$, the neighborhood of V', that is, all vertices in or adjacent to V'. For a directed path $p = (v_1, \ldots, v_r)$ and $1 \le i < j \le r$ we denote by p_{ij} the subpath $(v_i, \ldots, v_j)$. A subset of vertices $V' \subset V$ separates $V_1 \subset V \setminus V'$ and $V_2 \subset V \setminus V'$, if every (undirected) path between a vertex in V_1 and a vertex in V_2 contains a vertex in V'. Let $w : E \to \mathcal{R}$ be weights on the edges of G. For $E' \subset E$, we denote $w(E') = \sum_{e \in E'} w(e)$. We refer to paths as sequences of vertices or edges. The *size* of a path is the number of edges in the path. The weight of a path $p \subset E$ is $w(p)$. The distance from $v_1 \in V$ to $v_2 \in V$, denoted $\mathrm{dist}_G(v_1, v_2)$, is the weight of the minimum-weight path (shortest path) from v_1 to v_2. We use the notation $\mathcal{R}$ for the set of real numbers and $\mathcal{Z}_+$ for the set of nonnegative integers.

2.2. The minimum-weight diameter of a graph

The *minimum-weight diameter* $\mathrm{diam}(G) \in \mathcal{Z}_+$ of a graph $G = (V, E)$ with weights $w : E \to \mathcal{R}$ is the maximum, over all pairs $(v_1, v_2) \in V \times V$ of the minimum size of a path from v_1 to v_2 of weight $\mathrm{dist}(v_1, v_2)$.

$$\mathrm{diam}(G) = \max_{(v_1,v_2)\in V\times V} \min\{|p| \mid p \text{ is a path from } v_1 \text{ to } v_2 \text{ s.t. } w(p) = \mathrm{dist}_G(v_1, v_2)\}$$

Single source shortest paths computation can be performed more efficiently in parallel on graphs with a small minimum-weight diameter, namely, in $O(\mathrm{diam}(G) \log n)$ time using $O(|E| \mathrm{diam}(G))$ work. These bounds are achieved by running a parallel version of the Bellman-Ford shortest-paths algorithm (see [3, 9]). Note that we need to run the algorithm for only $\mathrm{diam}(G)$ phases and each phase can be performed in $O(\log n)$ time and $O(|E|)$ work.

2.3. Separator decomposition tree of a graph

Consider an undirected graph $G = (V, E)$. A *separator decomposition tree* of G (see Figure 1 for an example) is a rooted binary tree T_G such that each node $t \in T_G$ is labeled by two subsets of vertices $\mathcal{V}(t) \subset V$ and $\mathcal{S}(t) \subset \mathcal{V}(t)$. For $t \in T_G$, denote by $G(t) = (\mathcal{V}(t), \mathcal{E}(\mathcal{V}(t)))$ the subgraph of G induced by $\mathcal{V}(t)$. The subset $\mathcal{S}(t)$ is a separator of $G(t)$. The root node $r \in T_G$ is labeled with $\mathcal{V}(r) = V$ and $\mathcal{S}(r)$ is a separator for the graph G. The labels for the children of a node $t \in T_G$ are determined as follows. Let $V_1 \subset \mathcal{V}(t)$ and $V_2 \subset \mathcal{V}(t)$ be such that $\mathcal{S}(t)$ separates V_1 from V_2 in $G(t)$. (Note that V_1 and V_2 are not necessarily connected.) Denote by t_1, t_2 the children of t in T_G. The labels of t_i (i=1,2) are $\mathcal{V}(t_i) = V_i \cup (\mathcal{S}(t) \cap N(V_i))$ and $\mathcal{S}(t_i)$ is a separator of $G(t_i)$. The leaves of T_G have only $\mathcal{V}()$ labels. The avoid ambiguity, throughout the paper we refer to the vertices of G as "vertices" and to the vertices of the separator tree T_G as "nodes." We denote by d_G the height of the tree T_G (size of a maximum size path from the root to a leaf).

We associate sets of *boundary* vertices $\mathcal{B}(t)$ with each node $t \in T_G$. The boundaries are defined as follows: The boundary of the root $r \in T_G$ is $\mathcal{B}(r) = \emptyset$. The boundary of a node $t \in T_G$ with parent t' is $\mathcal{B}(t) = (\mathcal{S}(t') \cup \mathcal{B}(t')) \cap \mathcal{V}(t)$.

It is easy to see that the boundary sets have the following properties:

Proposition 2.1 *i. For all $t \in T_G$, $\mathcal{B}(t) = \mathcal{V}(t) \cap \bigcup_{t' \in W} \mathcal{S}(t')$, where $W \subset T_G$ is the set of ancestors of t.*

ii. For all $t \in T_G$, $\mathcal{B}(t)$ separates $\mathcal{V}(t) \setminus \mathcal{B}(t)$ from $V \setminus \mathcal{V}(t)$ in G.

3. Augmenting G using the decomposition tree

Consider a directed graph $G = (V, E)$ with weights $w : E \to \mathcal{R}$ and a separator decomposition tree T_G of the undirected skeleton of G. We define an augmentation of G, with respect to T_G, by a set $E^+ \subset V \times V$ of edges with weights $w : E^+ \to R$. The augmented graph $G^+ = (V, E \cup E^+)$ is such that distances in G^+ are equal to the respective distances in G and G^+ has a small minimum-weight diameter. In Subsection 3.1 we define the set E^+ and prove it has some desired properties. In Subsection 3.2 we discuss the resource bounds of computing shortest-paths from a set of sources when the set E^+ is given.

3.1. The set E^+

For $t \in T_G$, denote by E_t a set of weighted edges as follows: $E_t = \mathcal{B}(t) \times \mathcal{B}(t) \cup \mathcal{S}(t) \times \mathcal{S}(t)$, where $e = (v_1, v_2) \in E_t$ has weight $w(e) = \mathrm{dist}_{G(t)}(v_1, v_2)$. Define $E^+ = \bigcup_{t \in T_G} E_t$. We may assume that E^+ does not contain parallel edges by including in E^+ only the parallel edge of minimum weight. In Section 4 we present an algorithm to compute the set E^+. The following theorem establishes that the set E^+ has the desired properties.

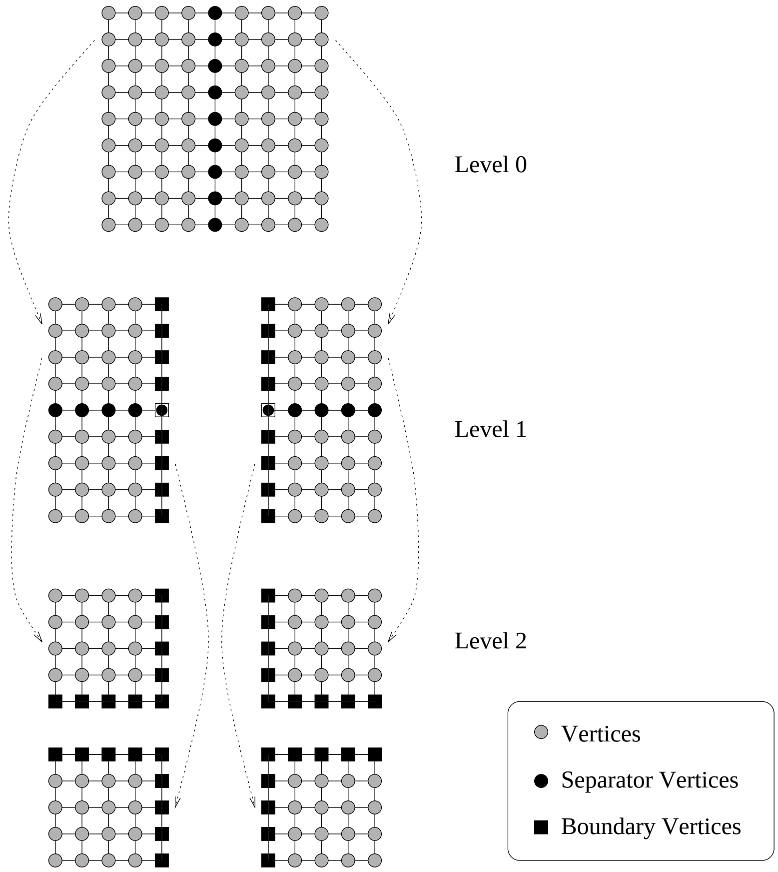

Figure 1: A separator decomposition tree of a 9×9 grid graph

Theorem 3.1 *The graph $G^+ = (V, E \cup E^+)$ has the following properties.*

i. For every $(v_1, v_2) \subset V \times V$, $\mathrm{dist}_G(v_1, v_2) = \mathrm{dist}_{G^+}(v_1, v_2)$, and

ii. $\mathrm{diam}(G^+)$, the minimum-weight diameter of G^+, is such that $\mathrm{diam}(G^+) \leq 4d_G + 2\ell + 1$, where ℓ is the maximum, over all leaves $t \in T_G$ of the minimum-weight diameter of $G(t)$.

Part i of Theorem 3.1 immediately follows from the fact that all the edges in E^+ have weight which is not smaller than the respective distances in G. In the remaining part of the subsection we prove part ii of Theorem 3.1

We define two functions level : $V \cup T_G \to \{0, \ldots, d_G\}$ and node : $V \to T_G$. For a node $t \in T_G$, denote by level(t) the size of the path in T_G from t to the root. For $v \in V$, define

$$\mathrm{level}(v) = \min\{i | \exists t \in T_G \text{ s.t. } v \in \mathcal{S}(t) \wedge \mathrm{level}(t) = i\} ,$$

the minimum level of all nodes $t \in T_G$ such that $v \in \mathcal{S}(t)$. If $v \notin \bigcup_{t \in T_G} S(t)$ (v does not belong to any separator), then level(v) is undefined. It is easy to see that for all $t \in T_G$, if $v \in \mathcal{B}(t)$ then level$(v) <$ level(t) and if $v \in \mathcal{S}(t)$ then level$(v) \leq$ level(t). For $v \in V$, we define node$(v) \in T_G$ as follows: node$(v) = t$ if and only if $v \in \mathcal{S}(t) \wedge \text{level}(v) = \text{level}(t)$. If $v \notin \bigcup_{t \in T_G} S(t)$, we define node$(v)$ to be the leaf $t \in T_G$ such that $v \in \mathcal{V}(t)$. It follows from the construction of T_G and properties of the level function that node(v) is uniquely defined.

Consider a directed path $p = (v_1, v_2, \ldots, v_r)$ in G. We say that the path p has a *shortcut* if $e = (v_1, v_r) \in E^+ \cup E$ and $w(e) \leq w(p)$. The following proposition states some sufficient conditions for shortcuts.

Proposition 3.2 *A path $p = (v_1, v_2, \ldots, v_r)$ in G has a shortcut if any one of the following conditions holds:*

i. $\text{level}(v_1) = \text{level}(v_r)$ *and for all* $1 \leq k \leq r$, $\text{level}(v_k) \geq \text{level}(v_1)$

ii. $\text{level}(v_1) > \text{level}(v_r)$ *and for all* $1 < k < r$, $\text{level}(v_k) > \text{level}(v_1)$

iii. $\text{level}(v_1) < \text{level}(v_r)$ *and for all* $1 < k < r$, $\text{level}(v_k) > \text{level}(v_r)$

Proof: In order to prove that a path p has a shortcut, it suffices to show that there exists some $t \in T_G$, such that $p \subset \mathcal{V}(t)$ and either $\{v_1, v_r\} \subset \mathcal{S}(t)$ or $\{v_1, v_r\} \subset \mathcal{B}(t)$. We first prove part i. Note that $p \cap \mathcal{B}(\text{node}(v_1)) = \emptyset$ (since the level of boundary vertices of a node is lower than the level of the node). Hence, $p \subset \mathcal{V}(\text{node}(v_1))$ and $\text{node}(v_1) = \text{node}(v_r)$. The definition of the level function implies that $\{v_1, v_r\} \subset \mathcal{S}(\text{node}(v_1))$. It follows that p has a shortcut. We prove part ii of the claim. The correctness of part iii can be shown by a symmetric argument. The properties of the level function assert that $\{v_1, \ldots, v_{r-1}\} \cap \mathcal{B}(\text{node}(v_1)) = \emptyset$, $\{v_2, \ldots, v_{r-1}\} \cap \mathcal{S}(\text{node}(v_1)) = \emptyset$, $v_r \in \mathcal{B}(\text{node}(v_1))$, and $v_1 \in \mathcal{S}(\text{node}(v_1))$. Hence, one of the children, t, of node(v_1) is such that $p \subset \mathcal{V}(t)$ and $\{v_1, v_r\} \subset \mathcal{B}(t)$. Thus, p has a shortcut. ∎

We are ready to conclude the proof of Theorem 3.1 and show that $\text{diam}(G^+) \leq 4d_G + 2\ell + 1$. We first establish the following claim. Consider a path $\hat{p} = (v = v_1, \ldots, v_r = u)$ such that the levels of at least one vertex in $\hat{p}$ and all of $\{v_2, \ldots, v_{r-1}\}$ are undefined. It follows that for some leaf $t \in T_G$, $p \subset \mathcal{V}(t)$, and hence, there is a path $p' = (v = v'_1, \ldots, v'_k = u)$ such that $w(p') \leq w(\hat{p})$ and $k \leq \ell$. Consider a path $p = (v_1, \ldots, v_r)$ in G. We say that p can be *replaced* by a path p' in G^+, if p' is from v_1 to v_r and $w(p') \leq w(p)$. It suffices to show that p can be replaced by a path of size at most $4d_G + 2\ell + 1$. If level(v_i) is undefined for all $1 \leq i \leq r$, then the claim implies that p can be replaced by a path of size at most ℓ. Otherwise, let $i_1 = \min\{1 \leq i | \text{level}(v_i) \text{ is defined}\}$ and let $i_2 = \max\{i | \text{level}(v_i) \text{ is defined}\}$. The claim asserts that the subpaths p_{1i_1} and p_{i_2r} can each be replaced by paths of size at most ℓ. If $i_1 = i_2$ then p can be replaces by a path of size at most 2ℓ and we are done. Otherwise, to establish that p can be replaced by a path of size at most $4d_G + 2\ell + 1$, it suffices to show that the subpath $p_{i_1i_2}$ can be replaced by a path of size $4d_G + 1$. We explicitly point on $h \leq 4d_G + 2$ indices $i_1 = j_1 < j_2 < \cdots < j_h = i_2$ such that for all $1 \leq i \leq h-1$. $p_{j_ij_{i+1}}$ has a shortcut. For each $i_1 \leq j < i_2$ such that level(v_j) is defined, we assign to v_j a *right shortcut* as follows. The right shortcut of v_j is a vertex v_k such that $j < k \leq i_2$, level(v_k) is defined, and the subpath p_{jk} has a shortcut. Below we define the right shortcut of a vertex v_j. In the following, when level(v) is undefined, we treat it as level$(v) = \infty$.

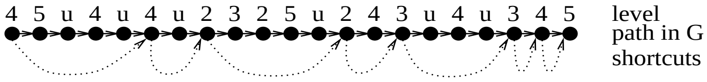

Figure 2: A path with level labels and corresponding right shortcuts

i. Let

$$k = \max\{i | i > j \wedge \text{level}(v_i) = \text{level}(v_j) \wedge \forall i' \text{ s.t. } i \le i' \le j, \text{level}(v_{i'}) \ge \text{level}(v_j)\} \ .$$

If k is defined, the right shortcut from v_j is v_k. It follows from Part i of Proposition 3.2 that p_{jk} has a shortcut.

ii. Otherwise, if k is not defined, then if there exists $i > j$ such that $\text{level}(v_i) < \text{level}(v_j)$, let

$$k = \min\{i > j | \, \text{level}(v_i) < \text{level}(v_j)\} \ ,$$

and define v_k to be the right shortcut of v_j. It follows from Part ii of Proposition 3.2 that p_{jk} has a shortcut.

iii. Otherwise, for all $i > j$, $\text{level}(v_i) > \text{level}(v_j)$. Let

$$k = \max\{i > j | \forall i' \text{ s.t. } j \le i' \le i, \text{level}(v_{i'}) > \text{level}(v_i)\} \ ,$$

and define v_k to be the right shortcut of v_j. Since $\text{level}(v_{i_2})$ is defined, $\text{level}(v_k)$ must be defined. It follows from Part iii of Proposition 3.2 that p_{jk} has a shortcut.

Consider the path p' in E^+ formed by starting from v_{i_1} and following right shortcuts (note that $\text{level}(v_{i_1})$ is defined). The path p' must end in v_{i_2}, and the level function of all vertices in p' is defined. Consider the sequence of level labels of vertices on the path. It is easy to see that any subsequence of consecutive vertices with the same level is of size at most 2. The sequence is bitonic: first nonincreasing up to a point and then nondecreasing. We conclude that the size of p' is at most $4d_G + 1$. See Figure 2 for an example of right shortcuts.

3.2. Computing shortest-paths using E^+

We discuss how we employ the set E^+ of weighted edges to compute shortest-paths from s sources. We consider a single source vertex $u \in V$. In Subsection 2.2 we argued that distances from a single source in a graph G can be compute in $O(\text{diam}(G) \log n)$ time using $O(m \, \text{diam}(G))$ work. These bounds are achieved by running for each source, a parallel version of the Bellman-Ford shortest-paths algorithm (see [3, 9]). We sketch the algorithm. The algorithm runs in $O(\text{diam}(G))$ phases. In each phase, for each node $v \in V$, we consider all edges $e \in E$ entering v. For each such edge, $e = (v', v)$ we update the distance of v from u to be the minimum of the current distance and the sum of $w(e)$ and the current distance of v' from u. We refer to the above operation as *scanning* the edge e. Each phase takes $O(m)$ time sequentially and it can be parallelized such that it takes $O(\log n)$ time and $O(m)$ work.

The Bellman-Ford algorithm sketched above scans all edges in every phase. We show that for the augmented graph $G^+ = (V, E \cup E^+)$, however, it suffices to scan the edges only in a small number of phases. In the proof of Theorem 3.1 we pointed on minimum-weight paths of a specific structure. We defined the function level : $V \rightarrow \{0, \ldots, d_G\}$, where d_G is the height of the separator tree T_G. We showed that for every pair of vertices in G^+, there exists a minimum-weight path between them with the following structure. The path may start and end with a sequence of at most ℓ edges from E (where ℓ is as in Theorem 3.1). All other edges are E^+ edges. The levels of the vertices on the path form a bitonic sequence: first nonincreasing then nondecreasing, where vertices of the same level may appear at most 2 consecutive times. It is easy to see that it suffices to run a version of the Bellman-Ford algorithm that scans edges as follows. We run the algorithm for $2\ell + 4d_G + 1$ phases. In each of the first ℓ phases and the last ℓ phases we scan the edges E. For $1 \leq i \leq 2d_G + 1$, in the $(i + \ell)$th phase we scan all edges $e = (v_1, v_2)$ as follows: If i is odd, we scan e if $\mathrm{level}(v_1) = \mathrm{level}(v_2) = d_G - (i-1)/2$ If i is even, e is scanned if $\mathrm{level}(v_1) = d_G - i/2 - 1$ and $\mathrm{level}(v_2) < \mathrm{level}(v_1)$. For $1 \leq i \leq 2d_G$, in the $(i + \ell + 2d_G)$th phase we scan all edges $e = (v_1, v_2)$ as follows: If i is odd, e is scanned if $\mathrm{level}(v_1) = (i-1)/2$ and $\mathrm{level}(v_2) > \mathrm{level}(v_1)$. If i is even, e is scanned if $\mathrm{level}(v_1) = \mathrm{level}(v_2) = i/2$

The work performed by the modified Bellman-Ford algorithm is $O(\ell|E|+|E^+|)$ per source, and the time is $O(\log n)$ per phase and hence is $O((\ell + d_G)\log n)$.

4. Computing the set E^+

In this section we present two algorithms that input a directed graph $G = (V, E)$ with edge-weights $w : E \rightarrow \mathcal{R}$ and a separator tree T_G for G, and compute the weighted set of edges $E^+ = \bigcup_{t \in T_G} E_t$ as defined in Section 3. The first algorithm computes the set E^+ recursively, going from the leaves to the root. The running time is $O(d_G \log^2 n)$. The second algorithm runs in time $O(d_G \log n)$ but performs a factor of $O(\log n)$ more work.

4.1. Computing E^+ from the leaves up

The computation of E^+ is done by first computing the sets E_t for the leaves of T_G (this can be done independently and in parallel for all leaves), and then d_G parallel phases are performed, where at phase i $(1 \leq i \leq d_G)$, E_t is computed for all nodes $t \in T_G$ at level $d_G - i$, using the procedure below and the previously computed E_t sets for all nodes at level $d_G - i + 1$.

Below, we state a procedure that inputs a node $t \in T_G$ with children $\{t_1, t_2\} \subset T_G$ and the sets E_{t_1} and E_{t_2}, and constructs the set E_t.

We discuss the procedure for processing a single node t. Let $t_1 \in T_G$ and $t_2 \in T_G$ be the children of T. We assume that for $i = 1, 2$, we are provided with the set E_{t_i}, and in particular, with the distances in $G(t_i)$ between the pairs $\mathcal{B}(t_i) \times \mathcal{B}(t_i)$. The goal is to obtain the distances in $G(t)$ between the pairs $\mathcal{B}(t) \times \mathcal{B}(t) \cup \mathcal{S}(t) \times \mathcal{S}(t)$. The following procedure performs this computation.

Algorithm 4.1 [Process the node $t \in T_G$]

i. Construct the graph $H_S = (\mathcal{S}(t), \mathcal{S}(t) \times \mathcal{S}(t))$ with weights $w_{H_S} : \mathcal{S}(t) \times \mathcal{S}(t) \rightarrow \mathcal{R}$ as follows. For every $e = (v_1, v_2) \in \mathcal{S}(t) \times \mathcal{S}(t))$ define:
$w_{H_S}(e) \leftarrow \min\{\mathrm{dist}_{G(t_1)}(v_1, v_2), \mathrm{dist}_{G(t_2)}(v_1, v_2)\}$.
(These distances are available since for $i = 1, 2$, $\{v_1, v_2\} \subset \mathcal{B}(t_i)$.)

ii. Perform an all-pairs shortest-paths computation on H_S. For every $e = (v_1, v_2) \in \mathcal{S}(t) \times \mathcal{S}(t)$, output $w(e) \leftarrow \mathrm{dist}_{H_S}(v_1, v_2)$.

iii. Construct the graph $H = (V_H, E_H)$ where $V_H = \mathcal{B}(t) \cup \mathcal{S}(t)$ and $E_H = \mathcal{B}(t) \times \mathcal{S}(t) \cup \mathcal{S}(t) \times \mathcal{B}(t) \cup \mathcal{S}(t) \times \mathcal{S}(t)$ with weights $w_H : E_H \rightarrow \mathcal{R}$ as follows.
For $e = (v_1, v_2) \in E_H$: if $e \in \mathcal{S}(t) \times \mathcal{S}(t))$, then $w_H(e) \leftarrow \mathrm{dist}_{H_S}(v_1, v_2)$. Otherwise, let $i \in \{1, 2\}$ be such that $\{v_1, v_2\} \subset \mathcal{V}(t_i)$ (this must be the case for exactly one of $i = 1$ or $i = 2$). Let $w_H(e) \leftarrow \mathrm{dist}_{G(t_i)}(v_1, v_2)$. (These distances are available since $\{v_1, v_2\} \subset \mathcal{B}(t_i)$.)

iv. For every vertex $v \in \mathcal{B}(t)$ perform a single-source 3-limited shortest-paths computations in H of distances from v and distances to v.

v. For every $e = (v_1, v_2) \in \mathcal{B}(t) \times \mathcal{B}(t))$, output

$$w(e) = \begin{cases} \min\{\mathrm{dist}_{G(t_1)}(v_1, v_2), \mathrm{dist}^3_H(v_1, v_2)\} & \text{if } \{v_1, v_2\} \subset \mathcal{B}(t_1) \\ \min\{\mathrm{dist}_{G(t_2)}(v_1, v_2), \mathrm{dist}^3_H(v_1, v_2)\} & \text{if } \{v_1, v_2\} \subset \mathcal{B}(t_2) \\ \mathrm{dist}^3_H(v_1, v_2) & \text{otherwise} \end{cases}$$

Complexity. Using the Floyd-Warshall algorithm (see [3, 9]), step ii can be implemented such that it uses $O(|\mathcal{S}(t)|^3 \log(|\mathcal{S}(t)|))$ work and runs in $O(\log^2 n)$ time. The work can be further reduced to $O(|\mathcal{S}(t)|^3)$, if a recent algorithm of Han, Pan, and Reif [8] is used. Step iv can be implemented (e.g., by using, for each source, 3 iterations of a parallel Bellman-Ford algorithm, see [3, 9]) such that it uses $|\mathcal{B}(t)|^2 |\mathcal{S}(t)|)$ work and runs in $O(\log(|\mathcal{B}(t)| + |\mathcal{S}(t)|))$ time. Hence, Algorithm 4.1 can be implemented such that it uses $O(|\mathcal{S}(t)|^3 + |\mathcal{B}(t)|^2 |\mathcal{S}(t)|)$ work and runs in $O(\log^2 n)$ time. If the algorithm is used for reachability or transitive closure computations, we can perform step ii in $O(\log^2 |\mathcal{S}(t)|)$ time using $M(|\mathcal{S}(t)|) \log |\mathcal{S}(t)|$ work, and step iv in $O(\log |\mathcal{S}(t)| + |\mathcal{B}(t))$ time using $M(|\mathcal{S}(t)| + |\mathcal{B}(t)|)$ work. Hence, in $O(\log^2 n)$ time using $M(|\mathcal{S}(t)|) \log n + M(|\mathcal{B}(t)|)$ work.

Correctness. The following proposition establishes the correctness of the algorithm.

Proposition 4.2 *For every $e = (v_1, v_2) \in E_t$, $w(e)$ computed by the algorithm is such that $w(e) = \mathrm{dist}_{G(t)}(v_1, v_2)$.*

Proof: First we show that for $\{v_1, v_2\} \subset \mathcal{S}(t)$, $\mathrm{dist}_{H_s}(v_1, v_2) = \mathrm{dist}_{G(t)}(v_1, v_2)$. Consider a path $p = (u_1, \ldots, u_r) \subset \mathcal{V}(t)$ such that $p \cap \mathcal{S}(t) = \{u_1, u_r\}$. It follows from definitions that $\{u_1, u_r\} \subset \mathcal{B}(t_1) \cap \mathcal{B}(t_2)$ and either $p \subset \mathcal{V}(t_1)$ or $p \subset \mathcal{V}(t_2)$. Hence,

$$w(p) \geq \min\{\mathrm{dist}_{G(t_1)}(u_1, u_r), \mathrm{dist}_{G(t_2)}(u_1, u_r)\} = w_{H_S}(u_1, u_r) \geq \mathrm{dist}_{H_S}(u_1, u_r) \ .$$

Consider a path $p = (v_1 = u_1, \ldots, u_r - v_2) \subset \mathcal{V}(t)$ such that $p \cap \mathcal{S}(t) = \{u_{i_1}, \ldots, u_{i_k}\}$, where $1 = i_1 < \cdots < i_k = r$. It suffices to show that $w(p) \geq \mathrm{dist}_{H_S}(u_1, u_r)$. The argument above implies that for $1 \leq j \leq k-1$, $w(p_{i_j i_{j+1}}) \geq \mathrm{dist}_{H_S}(u_{i_j}, u_{i_{j+1}})$. Therefore,

$$w(p) = \sum_{j=1}^{k-1} w(p_{i_j i_{j+1}}) \geq \sum_{j=1}^{k-1} \mathrm{dist}_{H_S}(u_{i_j}, u_{i_{j+1}}) \geq \mathrm{dist}_{H_S}(u_1, u_r) \ .$$

It remains to show that for $e = (v_1, v_2) \subset \mathcal{B}(t) \times \mathcal{B}(t)$, $w(e) = \mathrm{dist}_{G(t)}(v_1, v_2)$. Let $p = (v_1 = u_1, \ldots, u_r = v_2)$ be a path from v_1 to v_2 in $G(t)$ we show that $w(e) \leq w(p)$. If $p \cap \mathcal{S}(t) = \emptyset$ then for either $i = 1$ or $i = 2$, $p \subset \mathcal{V}(t_i)$ and $\{v_1, v_2\} \subset \mathcal{B}(t_i)$, hence, $w(p) \geq \mathrm{dist}_{G(t_i)}(v_1, v_2) \geq w(e)$. Otherwise, let $i_1 = \min\{1 \leq i \leq r | u_i \in \mathcal{S}(t)\}$ and let $i_2 = \max\{1 \leq i \leq r | u_i \in \mathcal{S}(t)\}$. By definition, $\{e_1 = (u_1, u_{i_1}), e_2 = (u_{i_1}, u_{i_2}), e_3 = (u_{i_2}, u_r)\} \subset E_H$. (If $i_1 = i_2$, do not consider the edge e_2.) It follows that p_{1i_1} and $p_{i_2 r}$ must be each contained in either $\mathcal{V}(t_1)$ or in $\mathcal{V}(t_2)$. Hence $w(p_{1i_1}) \geq w_H(e_1)$, $w(p_{i_2 r}) \geq w_H(e_3)$, and (using the first part of the proof) $w_H(e_2) = \mathrm{dist}_{H_S}(u_{i_1}, u_{i_2}) \leq w(p_{i_1 i_2})$. It follows that $w(p) \geq w_H(e_1) + w_H(e_2) + w_H(e_3)$. Hence, there is a path of size at most 3 in H with weight at most the weight of p. ∎

4.2. A faster algorithm for computing E^+

The algorithm in Subsection 4.1 computes the set E^+ by performing d_G parallel phases, where in phase i we compute E_t for all the nodes $t \in T_G$ of level $d_G - i$. Therefore, in each phase, the algorithm performs complete all-pair shortest-path computations that take $O(\log^2 n)$ time. The following algorithm computes E^+ by performing simultaneously, for all nodes $t \in T_G$, an all-pairs shortest paths computation on the boundary and separator vertices of $G(t)$. Roughly, computing E_t for each node $t \in T_G$ before it is done with computing E_{t_1}, E_{t_2} for the children t_1, t_2 of t. This enables us to save a factor of $O(d_G)$ in the running time. We assume that the all-pairs shortest-paths computation utilized consists of $O(\log n)$ path doubling steps. After each doubling step, we update the weights in each $G(t)$ using the weights given to the edge by the children of t.

Algorithm 4.3 [Compute the weighted set E^+]

i. For all $t \in T_G$ pardo:
   Construct the graph $H(t) = (V_H(t), E_H(t))$,
   where $V_H(t) \leftarrow \mathcal{S}(t) \cup \mathcal{B}(t)$ and $E_H(t) \leftarrow V_H(t) \times V_H(t)$.
   Assign weights to every $e = (v_1, v_2) \in E_H(t)$ as follows:

$$w_t(e) \leftarrow \begin{cases} \mathrm{dist}_{G(t)}(v_1, v_2) & \text{if } t \text{ is a leaf} \\ \min\{w(e), \infty\} & \text{otherwise} \end{cases}$$

ii. Repeat $2\lceil\log n\rceil + 2d_G$ times:

(1) For all $t \in T_G$ apply a path doubling step to $H(t)$:
Pardo for all $e = (u_1, u_2) \in E_H(t)$:
$w(e) \leftarrow \min\{w_t(e), \min_{u \in V_H(t)} w_t(u_1, u) + w_t(u, u_2)\}$.

(2) For all $t \in T_G$ pardo:
For every $e \in E_H(t)$ pardo :
$w_t(e) \leftarrow \min\{w_{t_1}(e), w_{t_2}(e), w_t(e)\}$.
(where $t_1, t_2 \in T_G$ are the children of t and for $i \in \{1, 2\}$, $w_{t_i}(e) \equiv \infty$ if $e \notin E_H(t_i)$.)

iii. For all $e \in E^+$, $w(e) \leftarrow \min_{\{t \in T_G | e \in E_H(t)\}} w_t(e)$

Complexity. Let $L_G \subset T_G$ be the set of leaves of T_G, and let ℓ be the maximum minimum-weight diameter of $\{G(t) | t \in L_G\}$. Step i can be implemented in $O(\max_{t \in L_G} \log |\mathcal{V}(t)|)$ time and $O(\ell \sum_{t \in L_G} (|\mathcal{S}(t)| + |\mathcal{B}(t)|)|\mathcal{V}(t)|)$ work, using ℓ iterations of the Bellman-Ford algorithm (see [3, 9]) or in $\sum_{t \in L_G} (|\mathcal{V}(t)|^3) \max_{t \in L_G} \log |\mathcal{V}(t)|)$ work and $O(\log \ell \max_{t \in L_G} \log |\mathcal{V}(t)|)$ time using the Floyd-Warshall algorithm. Step ii1 can be implemented such that it uses $\sum_{t \in T_G} (|\mathcal{S}(t)| + |\mathcal{B}(t)|)^3$ work and runs in

$$\max_{t \in T_G} O(\log(|\mathcal{S}(t)| + |\mathcal{B}(t)|)) = O(\log n)$$

time, by using a single iteration of the Floyd-Warshall algorithm (see [3, 9]). The complexity bounds of other steps are dominated by step ii1. Note that step ii1 is performed $O(d_G + \log n)$ times. It follows that Algorithm 4.3 can be implemented such that it uses

$$O(\sum_{t \in T_G} (|\mathcal{S}(t)| + |\mathcal{B}(t)|)^3 \log n + \sum_{t \in L_G} (|\mathcal{V}(t)|^3) \log n)$$

work and runs in $O(\log^2 n + d_G \log n)$ time. If the algorithm is used for reachability or transitive closure computations, we can perform step ii1 in $O(\log(|\mathcal{S}(t)| + |\mathcal{B}(t)|)$ time using $O(M(|\mathcal{S}(t)| + |\mathcal{B}(t)|))$ work, and step i in $O(\sum_{t \in L_G} (M(|\mathcal{V}(t)|) \max_{t \in L_G} \log |\mathcal{V}(t)|))$ work. Hence, the algorithm runs in $O(\log^2 n + d_G \log n)$ time using

$$O(\sum_{t \in T_G} M(|\mathcal{S}(t)| + |\mathcal{B}(t)|) \log n + \sum_{t \in L_G} M(|\mathcal{V}(t)|) \log n) \text{ work.}$$

Remark 4.4 In some cases, the time can be improved using the following consideration. Observe that Algorithm 4.3 as stated performs some redundant work. We say that two edges (u_1, u_2) and (u_2, u_3) are *paired* at some iteration, if during this iteration the algorithm updates the weight assigned to (u_1, u_3) to be at most the sum of the weights currently assigned to (u_1, u_2) and (u_2, u_3). For a triple of vertices u_1, u_2, u_3 define

$$C_{u_1 u_2 u_3} = \{t \in T_G | \{u_1, u_2, u_3\} \subset V_H(t)\} \ .$$

The algorithm pairs the edges (u_1, u_2) and (u_2, u_3) $|C_{u_1u_2u_3}|$ times in each computation of step ii1. These pairings are done with respect to the weights w_t ($t \in C_{u_1u_2u_3}$). It suffices to do the pairing once with respect to the weights $\min_{t \in C_{u_1u_2u_3}} w_t$. We modify the algorithm such that it maintains a single copy of each edge in $\bigcup_{t \in T_G} E_H(t)$ and a compactly represented table indicating which edges are to be paired, where two edges (u_1, u_2) and (u_2, u_3) are paired if and only if $|C_{u_1u_2u_3}| > 0$. (Note that this table depends only on the sets $V_H(t)$ for $t \in T_G$ and needs to be computed once). It follows from standard PRAM techniques that the computation of the table and each iteration of step ii1 (that is, pairing all qualified pairs) can be performed in $O(\log n)$ time and

$$O(|\{\{u_1, u_2, u_3\} \subset V | \exists t \in T_G, \{u_1, u_2, u_3\} \subset V_H(t)\}|) \text{ work.}$$

Correctness. The following proposition establishes the correctness of Algorithm 4.3.

Proposition 4.5 *At step iii of the algorithm, for every $t \in T_G$ and $e = (v_1, v_2) \in E_H(t)$, $w_t(e) = \text{dist}_{G(t)}(v_1, v_2)$.*

In the remaining part of this section we prove Proposition 4.5. For a node $t \in T_G$, denote by $\text{height}(t)$ the maximum size of a path from t to a leaf in the subtree of T_G rooted at t. The following claim implies the proposition. Consider a node $t \in T_G$ and an edge $e = (v_1, v_2) \in E_H(t)$. Let $p = (v_1 = u_1, \ldots, u_r = v_2)$ be a path in $G(t)$ from v_1 to v_2 such that $w(p) = \text{dist}_{G(t)}(v_1, v_2)$. We claim that after at most $2\lceil \log(r-1) \rceil + 2\,\text{height}(t)$ iterations of step ii, $w_t(e) = w(p)$. We prove the claim by induction on $\text{height}(t)$. The base case is when t is a leaf ($\text{height}(t) = 0$), and the claim is immediate from the initialization (step i). If $\text{height}(t) > 0$, we assume (by the induction hypothesis) that the claim holds for the children t_1, t_2 of t. If $p \cap \mathcal{S}(t) \subset \{u_1, u_r\}$, it follows from definitions that $\{u_1, u_2\} \subset \mathcal{B}(t_1) \cap \mathcal{B}(t_2)$ and either $p \subset \mathcal{V}(t_1)$ or $p \subset \mathcal{V}(t_2)$. Hence, the claim follows from the induction hypothesis. Otherwise, let $1 = i_1 < \cdots < i_k = r$ be such that $(p \setminus \{u_1, u_r\}) \cap \mathcal{S}(t) = \{u_{i_2}, \ldots, u_{i_{k-1}}\}$. It follows from the definitions that for $1 \le j \le k-1$, the subpath $p_{i_j i_{j+1}}$ is contained in $\mathcal{V}(t_i)$ (for either $i = 1$ or $i = 2$) and that $e_j \equiv (u_{i_j}, u_{i_{j+1}}) \in E_H(t_i)$ and $e_j \in E_H(t)$. The induction hypothesis asserts that for $1 \le j \le k-1$, after $2\lceil \log(i_{j+1} - i_j) \rceil + 2\max\{\text{height}(t_1), \text{height}(t_2)\}$ iterations, $\min\{w_{t_1}(e_j), w_{t_2}(e_j)\} = w(p_{i_j i_{j+1}})$ and hence, $w_t(e_j) = w(p_{i_j i_{j+1}})$.

View the path in $H(t)$ consisting of the edges $e_1, \ldots, e_{k-1}$ as a list. We define the *size* of the edge e_j to be $|e_j| = i_{j+1} - i_j$ (the size of the subpath $p_{i_j i_{j+1}}$). We say that an edge e_j becomes *active* when it is assigned the weight $w(p_{i_j i_{j+1}})$. The edge e_j becomes active after at most $2\lceil \log(i_{j+1} - i_j) \rceil + 2\max\{\text{height}(t_1), \text{height}(t_2)\}$ iterations. Consider performing the following "pairing phase" on the list of edges in each iteration.

i. Choose a collection of disjoint pairs of active edges in the list such that the two edges in each pair are consecutive.

ii. Replace each such pair of edges $e'_1 = (v_1, v_2)$ and $e'_2 = (v_2, v_3)$ with a new edge $e' = (v_1, v_3) \in E_H$, where the weight of e' is defined to be the sum of the weights currently assigned to e'_1 and e'_2, and the size of e' is the sum of the sizes of e'_1 and e'_2. Newly formed edges are active.

Consider performing pairing phases with a sequence of iterations. The sum of the sizes of the edges on the list is always $r-1$. It is easy to see that during the execution of the sequence, the weight currently assigned to each edge e in the list is not smaller than the current value of $w_t(e)$. Consider a sequence that reduces the list to a single edge $e' = (u_1, u_r)$. The weight of the edge e' equals $w(p)$. Hence, it suffices to show that there exists a sequence of at most $2\lceil \log(r-1) \rceil + 2\,\mathrm{height}(t)$ pairing phases that reduces the list $e_1, \ldots, e_{k-1}$ to a single edge.

We use the following proposition:

Proposition 4.6 *There exists a sequence of pairing phases such that for $i \geq 1$, after*

$$2i + 2\max\{\mathrm{height}(t_1), \mathrm{height}(t_2)\}$$

phases, all edges of size at most 2^i are active and the list does not contain two consecutive edges of size 2^{i-1} or less.

Proof: The part of the claim regarding the active edges is immediate from the definition. We prove the proposition by induction on i. The base case where $i = 1$ is immediate, since all edges of size 1 are initially active and an obvious pairing achieves the goal (e.g., any maximal pairing of active edges). Assume that the proposition holds for $i = k$. Consider all edges of size at most 2^k after phase $2k + 2\max\{\mathrm{height}(t_1), \mathrm{height}(t_2)\}$. We need to show that after 2 appropriately chosen pairing phases there are no two consecutive edges of size 2^k or less. Consider a maximal sublist of edges of size 2^k or less. (It suffice to consider these sublists independently). It follows from the induction hypothesis that each edge of size at most 2^{k-1} is either the rightmost edge of the sublist or has an edge of size larger than 2^{k-1} to its right. In the first pairing phase, pair all edges of weight no larger than 2^{k-1} with the edge to their right (if the edge to the right is active) and if the two rightmost edges are nboth ot paired, pair them together. As a result, all the edges in the new sublist have size larger than 2^{k-1}, except, perhaps, the rightmost edge. In the second phase, pair up edges at even positions with the edge to their right (except for the rightmost edge). The resulting sublist is such that all edges are of size larger than 2^k with the possible exception of the rightmost edge. ∎

We conclude the proof of Proposition 4.5. We substitute $i = \lceil (r-1) \rceil$ in Proposition 4.6. We obtain that after

$$2\lceil (r-1) \rceil + 2\max\{\mathrm{height}(t_1), \mathrm{height}(t_2)\}$$

phases the list consists of at most 3 edges, all active. (Since the sum of sizes of the edges is $r-1$). Hence, after two additional pairing phases the list would consists of a single edge. Proposition 4.5 follows using the fact that $\mathrm{height}(t) = \max\{\mathrm{height}(t_1), \mathrm{height}(t_2)\} + 1$.

5. Graphs with n^μ separators

We analyze the performance of the algorithms of Sections 4 (computing the set E^+) and Section 3 (computing shortest-paths using E^+) when for two constants $0 < \mu < 1$ and $1/2 < \alpha < 1$, the separator decomposition tree T_G has the following properties:

i. For all $t \in T_G$, $|\mathcal{S}(t)| = O(|\mathcal{V}(t)|^\mu)$.

ii. For all $\{t, t'\} \subset T_G$ such that t' is a child of t, $|\mathcal{V}(t')| \leq \alpha |\mathcal{V}(t)|$.

iii. If $t \in T_G$ is a leaf then $|\mathcal{V}(t)| = O(1)$.

We prove the following:

Theorem 5.1 *Under the assumptions above, the set E^+ can be computed within the following bounds:*

i. Algorithm 4.3 runs in $O(\log^2 n)$ time and performs $O((n + n^{3\mu}) \log n)$ work if $\mu \neq 1/3$ and $O(n \log^2 n)$ work if $\mu = 1/3$. For the reachability problem, the algorithm performs $O((M(n^\mu) + n) \log^2 n)$ work if $\omega\mu \neq 1$ and $O(n \log^3 n)$ work if $\omega\mu = 1$.

ii. The algorithm of Subsection 4.1 runs in $O(\log^3 n)$ time and performs $O(n^{3\mu})$ work if $3\mu > 1$, $O(n \log n)$ work if $3\mu = 1$, and $O(n)$ work if $3\mu < 1$.

iii. The sets E and E^+ are such that $|E| = O(n + n^{2\mu})$ and $|E^+| = O(n + n^{2\mu})$. We have $d_G = O(\log n)$.

Corollary 5.2 *Distances from a set of s sources to all other vertices in G^+ can be computed in $O(\log^2 n)$ time using $O(s(n + n^{2\mu}))$ work if $\mu \neq 1/2$ and $O(sn \log n)$ work if $\mu = 1/2$.*

Proof: The corollary follows from the analysis done in Subsection 3.2, and substituting $d_G = O(\log n)$, $\ell = O(1)$, and the above bounds for $|E|$ and $|E^+|$. ∎

The remaining part of the section contains a proof of Theorem 5.1. The above properties guarantee that the height of the separator tree is $d_G \leq \log_{\alpha^{-1}} n$. It follows that Algorithm 4.3 runs in time $O(\log^2 n)$ and the algorithm of Subsection 4.1 runs in $O(\log^3 n)$ time. For $0 \leq i \leq d_G$,

For $0 \leq i \leq \lfloor \log_{(1-\alpha)^{-1}} n \rfloor$, denote by $T_i \subset T_G$, the set of all nodes $t \in T_G$ such that $(1-\alpha)^{i-1} n < |\mathcal{V}(t)| \leq (1-\alpha)^i n$. Observe that (i) T_i is a forest consisting of subtrees of height $\lceil \log_{(1-\alpha)} \alpha \rceil$, and (ii) if $t \in T_k$ for some k, then the children of t are in $T_k \cup T_{k+1}$. Denote by $P_k \subset T_k$, all nodes $t \in T_k$ such that the parent of t in T_G is not in T_k. Denote $n_k = \sum_{t \in P_k} |\mathcal{V}(t)|$. Observe that $\sum_{t \in T_k} |\mathcal{V}(t)| = O(n_k)$.

$$s_k = \sum_{t \in T_k} |\mathcal{S}(t)| = O\left(\frac{n_k}{(1-\alpha)^k n} (1-\alpha)^{k\mu} n^\mu \right) .$$

It is immediate from the definitions that $n_0 = n$ and $n_{k+1} \leq n_k + s_k$. We show that for all k, $n_k = O(n)$. Note that

$$n_{k+1} \leq n_k + s_k \leq n_k (1 + O(((1-\alpha)^k n)^{\mu - 1})) .$$

Hence,

$$n_{k+1} \le n \prod_{i=0}^{k} (1 + O(n^{\mu-1}((1-\alpha)^{-1})^{i(1-\mu)})) \ .$$

Using the fact that $k \le \log_{(1-\alpha)^{-1}} n$ we derive

$$n_k \le n \prod_{i=0}^{\lfloor \log_{(1-\alpha)^{-1}} n \rfloor} (1 + O(n^{\mu-1}((1-\alpha)^{-1})^{(\lfloor \log_{(1-\alpha)^{-1}} n \rfloor - i)(1-\mu)}))$$

$$\le n \prod_{i \ge 0} (1 + O((1-\alpha)^{i(1-\mu)})) \le n 2^{O(\sum_{i \ge 0} (1-\alpha)^{i(1-\mu)})} = O(n) \ .$$

Therefore, $s_k = O(n^{\mu}(1-\alpha)^{k(\mu-1)})$.

We bound the work of performing Algorithm 4.3. The work bound is obtained using the analysis done in Subsection 4.2. First note that $\sum_{t \in L_G}(|\mathcal{V}(t)|^3) = O(n)$, where $L_G \subset T_G$ are the leaves of T_G. We use the implementation discussed in Remark 4.4. It follows that the work is bounded by $O(|P| \log n)$, where

$$|P| = |\{\{u_1, u_2, u_3\} \subset V | \exists t \in T_G, \{u_1, u_2, u_3\} \subset \mathcal{S}(t) \cup \mathcal{B}(t)\}| \ .$$

We bound the number $|P|$ of such triplets. It follows from definitions that if $\{u_1, u_2, u_3\} \subset \mathcal{S}(t) \cup \mathcal{B}(t)$ for some $t \in T_G$, then $\{u_1, u_2, u_3\}$ are separator vertices at ancestors of the node t. Recall that if $v \in \mathcal{S}(t)$ then v occurs as a boundary vertex in both children of t and if $v \in \mathcal{B}(t) \setminus \mathcal{S}(t)$ then v occurs only in one child. Hence,

$$|P| = O(\sum_{k=0}^{\lfloor \log_{(1-\alpha)^{-1}} n \rfloor} s_k 2 \sum_{j=k}^{\lfloor \log_{(1-\alpha)^{-1}} n \rfloor} ((1-\alpha)^j n)^{\mu} 2 \sum_{i=j}^{\lfloor \log_{(1-\alpha)^{-1}} n \rfloor} ((1-\alpha)^i n)^{\mu}) =$$

$$O(n^{3\mu} \sum_{k=0}^{\lfloor \log_{(1-\alpha)^{-1}} n \rfloor} (1-\alpha)^{k(3\mu-1)}) = \begin{cases} O(n^{3\mu}) & \text{if } 3\mu > 1 \\ O(n \log n) & \text{if } 3\mu = 1 \\ O(n) & \text{if } 3\mu < 1. \end{cases}$$

Remark 4.4 assumes a specific path doubling algorithm which considers all edges that are to be paired. In some cases, a faster doubling algorithm is available (e.g., for reachability problems we can use fast matrix multiplication algorithms). We analyze the work bound of Algorithm 4.3 when the weights are such that a doubling step on a graph with k nodes can be performed in $O(\log k)$ time and $O(k^{\omega})$ work ($2 \le \omega < 3$). It follows from the analysis in Subsection 4.2 that the work is bounded by

$$\sum_{k=0}^{\lfloor \log_{(1-\alpha)^{-1}} n \rfloor} \sum_{t \in T_k} O(|\mathcal{S}(t)|^{\omega} \log n + |\mathcal{B}(t)|^{\omega} \log n) \ .$$

Consider $0 \le k \le \lfloor \log_{(1-\alpha)^{-1}} n \rfloor$. We have

$$\sum_{t \in T_k} O(|\mathcal{S}(t)|^{\omega}) = O(s_k/((1-\alpha)^k n)^{\mu}((1-\alpha)^k n)^{\omega\mu}) = O((1-\alpha)^{k(\omega\mu-1)} n^{\omega\mu}) \ .$$

(Since for all $t \in T_k$, $|\mathcal{S}(t)| = O(((1-\alpha)^k n)^\mu)$ and the sum is maximized when the elements are as large as possible.) Hence,

$$\sum_{t \in T_G} |\mathcal{S}(t)|^\omega = \sum_{k=0}^{\lfloor \log_{(1-\alpha)^{-1}} n \rfloor} \sum_{t \in T_k} |\mathcal{S}(t)|^\omega = \begin{cases} O(n^{\omega\mu}) & \text{if } \omega\mu > 1 \\ O(n \log n) & \text{if } \omega\mu = 1 \\ O(n) & \text{if } \omega\mu < 1. \end{cases}$$

We bound $\sum_{t \in T_k} O(|\mathcal{B}(t)|^\omega)$. Recall that for a node $t \in T_G$, (i) $\mathcal{B}(t)$ consists of separator vertices of ancestors of t, and (ii) each occurrence of v in $\mathcal{S}(t)$ contributes to the boundaries of at most two of t's descendants at each level. Hence, $\sum_{t \in T_k} |\mathcal{B}(t)| = O(\sum_{i \le k} s_i)$. The sum $\sum_{t \in T_k} O(|\mathcal{B}(t)|^\omega)$ is maximized when the boundaries are as large as possible. It is easy to see that the choice (in accordance with the restrictions above) of boundary sizes at T_k that maximizes the sum is such that for $i \le k$, there are $O(s_i/((1-\alpha)^i n)^\mu) = O((1-\alpha)^{-i})$ nodes $t \in T_k$, for which $|\mathcal{B}(t)| = O(((1-\alpha)^i n)^\mu)$. It follows that

$$\sum_{t \in T_k} O(|\mathcal{B}(t)|^\omega) = O(\sum_{i \le k} (1-\alpha)^{-i}((1-\alpha)^i n)^{\omega\mu}) =$$

$$O(n^{\omega\mu} \sum_{i \le k} (1-\alpha)^{i(\omega\mu - 1)}) = \begin{cases} O(n^{\omega\mu}(1-\alpha)^{k(\omega\mu-1)}) & \text{if } \omega\mu < 1 \\ O(kn) & \text{if } \omega\mu = 1 \\ O(n^{\omega\mu}) & \text{if } \omega\mu > 1. \end{cases}$$

Therefore,

$$\sum_{t \in T_G} |\mathcal{B}(t)|^\omega = O(\sum_{k=0}^{\lfloor \log_{(1-\alpha)^{-1}} n \rfloor} \sum_{t \in T_k} |\mathcal{B}(t)|^\omega) = \begin{cases} O(n \log n) & \text{if } \omega\mu < 1 \\ O(n \log^2 n) & \text{if } \omega\mu = 1 \\ O(n^{\omega\mu} \log n) & \text{if } \omega\mu > 1. \end{cases}$$

It follows that the work performed by Algorithm 4.3 is bounded by $O(n \log^2 n)$, if $\omega\mu < 1$, by $O(n \log^3 n)$, if $\omega\mu = 1$, and by $O(n^{\omega\mu} \log^2 n)$ otherwise. Recall that algorithm 4.3 runs in $O(\log^2 n)$ time. Hence, the computation of the set E^+ for the reachability problem can be performed in $O(\log^2 n)$ time and $O(n \log^3 n)$ work if $\omega\mu = 1$, and $O(M(n^\mu)\log^2 n + n \log^2 n)$ work otherwise.

We bound the work performed by the algorithm of Subsection 4.1 (e.g., applying, sequentially for each layer, Algorithm 4.1 to the nodes of that layer). The algorithm performs

$$\sum_{t \in T_G} |\mathcal{B}(t)|^2 |\mathcal{S}(t)| + |\mathcal{S}(t)|^3$$

work. We have that

$$\sum_{t \in T_G} |\mathcal{B}(t)|^2 |\mathcal{S}(t)| = \sum_{k=0}^{\lfloor \log_{(1-\alpha)^{-1}} n \rfloor} O(((1-\alpha)^k n)^\mu \sum_{t \in T_k} |\mathcal{B}(t)|^2)$$

Using the derivation above we obtain that for $0 \le k \le \lfloor \log_{(1-\alpha)^{-1}} n \rfloor$,

$$O(((1-\alpha)^k n)^\mu \sum_{t \in T_k} |\mathcal{B}(t)|^2) = \begin{cases} O(n^{3\mu}(1-\alpha)^k(3\mu - 1)) & \text{if } 2\mu < 1 \\ O(n^{1.5} k (1-\alpha)^{0.5k}) & \text{if } 2\mu = 1 \\ O(n^{3\mu}(1-\alpha)^{\mu k}) & \text{otherwise.} \end{cases}$$

Hence, the work performed by the algorithm is

$$\begin{cases} O(n^{3\mu}) & \text{if } 3\mu > 1 \\ O(n\log n) & \text{if } 3\mu = 1 \\ O(n) & \text{if } 3\mu < 1. \end{cases}$$

We bound the size of the set $E^+ = \bigcup_{t\in T_G} E_t$. For each node $t \in T_G$, $E_t = \mathcal{S}(t) \times \mathcal{S}(t) \cup \mathcal{B}(t) \times \mathcal{B}(t)$. Therefore,

$$E^+ = \bigcup_{k=0}^{\lfloor \log_{(1-\alpha)^{-1}} n\rfloor} \bigcup_{t\in T_k} \left(\mathcal{S}(t) \times \mathcal{S}(t) \cup \mathcal{B}(t) \times \mathcal{B}(t)\right) .$$

Consider a node t. If a vertex v is in $\mathcal{S}(t)$, then v is a boundary vertex of both children of t. If $v \in \mathcal{B}(t) \setminus \mathcal{S}(t)$ then v is a boundary vertex in exactly one of the children of t. If two vertices v_1, v_2 are such that $\{v_1, v_2\} \subset \mathcal{S}(t) \cup \mathcal{B}(t)$ for some node $t \in T_G$, then for some t' (either t or an ancestor of t) we have $\{v_1, v_2\} \subset \mathcal{S}(t') \cup \mathcal{B}(t')$ and $\{v_1, v_2\} \cap \mathcal{S}(t') \neq \emptyset$ (assume $v_1 \in \mathcal{S}(t')$). In addition, for some t'' (either t' or an ancestor of t') we have that $v_2 \in \mathcal{S}(t'')$, Therefore, to bound the size of E^+, it suffices to bound for each node t, and each occurrence of a vertex v in $\mathcal{S}(t)$, the number of vertices that are in $\mathcal{S}(t')$ in every descendant t' of t such that $v \in \mathcal{B}(t')$. Hence,

$$|E^+| \leq \sum_{k=0}^{\lfloor \log_{(1-\alpha)^{-1}} n\rfloor} s_k(\sum_{j=k}^{\lfloor \log_{(1-\alpha)^{-1}} n\rfloor} O(((1-\alpha)^j n)^\mu)) =$$

$$= n^{2\mu} \sum_{k=0}^{\lfloor \log_{(1-\alpha)^{-1}} n\rfloor} O((1-\alpha)^{k(2\mu-1)}) = \begin{cases} O(n^{2\mu}) & \text{if } 2\mu > 1 \\ O(n\log n) & \text{if } 2\mu = 1 \\ O(n) & \text{if } 2\mu < 1. \end{cases}$$

We bound the number of edges $|E|$. Consider a node $t \in T_G$ with children t_1, t_2. It is immediate that if an edge e is such that $e \in \mathcal{E}(\mathcal{V}(t)) \setminus (\mathcal{E}(\mathcal{V}(t_1)) \cup \mathcal{E}(\mathcal{V}(t_2)))$, then $e \in \mathcal{S}(t) \times \mathcal{S}(t)$. Denote by $L_G \subset T_G$ the leaves of T_G. It follows that the number of edges is bounded by the sum of $\sum_{t\in L_G} |\mathcal{V}(t)|^2 = O(n)$ and $\sum_{k=0}^{\lfloor \log_{(1-\alpha)^{-1}} n\rfloor} \sum_{t\in T_k} |\mathcal{S}(t)|^2$. Using the considerations above we obtain that $\sum_{t\in T_G} |\mathcal{S}(t)|^2$ is $O(n^{2\mu})$ if $2\mu > 1$, is $O(n\log n)$ if $\mu = 1/2$ and is $O(n)$ if $2\mu < 1$. Hence, $|E| = O(n + n^{2\mu})$ if $2\mu \neq 1$ and $O(n\log n)$ if $2\mu = 1$.

6. Planar graphs

Planar graphs are known to have separators of size $O(n^{0.5})$ (see Lipton and Tarjan [14]) and hence, they fall into the class of graphs for which our analysis is applicable. Gazit and Miller [7] gave a deterministic $O(\log^3 n)$ time $O(n^{1+\epsilon})$ work CRCW PRAM algorithm that computes an $O(n^{0.5})$ separator tree for a planar graph. It follows that shortest paths from s sources in a planar digraph can be found using $O(\log^3 n)$ time $O(n^{1.5})$ work preprocessing, and then $O(\log^2 n)$ time using $O(sn\log n)$ work. Hence, in $O(\log^3 n)$ time and $O(n^{1.5} + sn\log n)$ work. The preprocessing work bound for reachability is $O(M(n^{0.5})\log^2 n)$.

Further improvements can be obtained for graphs where we are given a planar embedding such that all vertices lie on q faces. An efficient sequential algorithm for such graph was given by Frederickson [6]. A parallel version by Pantziou, Spirakis, and Zaroliagis [18] followed. We achieve further improvements for the parallel version by plugging our shortest-paths algorithm in one of the steps of the Pantziou *et al.* algorithm. We sketch the general ideas behind these algorithms.

Frederickson [6] presented a sequential algorithm that computes a compact representation of shortest-paths and is based on a novel hammock decomposition technique. The graph is decomposed into $O(q)$ subgraphs ("hammocks") such that each hammock forms an outerplanar graph in the given embedding. The total size of the hammocks is $O(n)$ and every hammock is attached to the rest of the graph through at most 4 vertices. Frederickson showed how to compute the hammock decomposition and a representation (via compact routing tables) of all-pairs shortest-paths within a hammock in linear time. The hammock decomposition and the paths information between the attachment vertices of each hammock is used to construct a graph G' with $O(q)$ edges and vertices. The graph G' is produced by replacing every hammock in the hammock decomposition of G by a complete directed graph on 4 vertices (the attachment vertices of the hammock) with weights corresponding to distances within the hammock. (In fact, Frederickson considered a planar version of G' also with $O(q)$ vertices). All pairs shortest-paths information in G' can be computed in $O(q^2)$ time and be used to construct a representation of shortest-paths in the original graph, via compact routing tables, in $O(qn)$ time. Frederickson also showed that shortest-paths information within hammocks and in the graph G' constitutes an alternate encoding of all-pairs shortest paths in G. This encoding can be produced in only $O(n + q^2)$ time.

Pantziou, Spirakis, and Zaroliagis [18] showed how to compute the hammock decomposition on a CREW PRAM in $O(\log n \log^* n)$ time using $O(n)$ processors. They also showed that a representation (compact routing table) of all-pairs shortest-paths in an outerplanar graph can be performed on a CREW PRAM in $O(\log^2 n)$ time using $O(n \log^2 n)$ work. Their algorithm performs the basic steps of Frederickson's algorithm using their parallel procedures. The algorithm computes the hammock decomposition and all-pairs distances within each hammock. Following that, it produces the graph G'. The computation of all-pairs shortest-paths distances in G' can be performed in $O(\log^2 q)$ time using $O(q^3)$ work (using the algorithm of [8]). The all-pairs shortest path information on G' and individually in each hammock is used by Pantziou *et al.* to produce a compact routing table of all-pairs shortest-paths in G in $O(\log^2 n)$ time using $O(nq \log^2 n)$ work. The resulting bounds (on a CREW PRAM) are $O(\log^2 n)$ time and $O(nq \log^2 n + q^3)$ work for all-pairs shortest paths computation and $O(nq \log^2 n + M(q) \log n)$ work for transitive closure information. Djidjev, Pantziou, and Zaroliagis [4] gave an $O(\log^2 n)$ time $O((n + q^3) \log^2 n + k \log n)$ work algorithm to compute distances between k specified pairs of vertices. Their algorithm consists of $O((n + q^3) \log^2 n)$ preprocessing and then can answer a query about a specific pair in $O(\log n)$ work. The preprocessing work bound for reachability is $O((n + M(q)) \log^2 n)$. Their algorithm is based on the parallel hammock decomposition and on a separator decomposition of outerplanar graphs that allows to compute distances efficiently after $O(n \log^2 n)$ preprocessing work.

We can improve over the above bounds as follows. Observe that the graph G' is "almost" planar. It is easy to produce a planar graph G'' such that a $k^{0.5}$-separator decomposition of G'' corresponds to a $k^{0.5}$-separator decomposition of G'. The graph G'' can be obtained from G' by replacing every hammock by a circle of size 4 (which corresponds to the 4 attachment vertices in the order they appear in the corresponding outerplanar graph) and a middle vertex connected to the attachment vertices. A separator of G'' can be transformed to a separator of G by replacing each middle vertex by the 4 attachment vertices of the corresponding hammock. It is easy to verify that a $k^{0.5}$-separator decomposition of G'' corresponds to a $k^{0.5}$-separator decomposition of G'. We use the planar separator algorithm of Gazit and Miller [7] to compute a separator decomposition tree of G''. It follows that our algorithm can be used to compute shortest-paths in G'. We first compute a $k^{0.5}$-separator decomposition tree of G' (which can be obtained in $O(\log^3 q)$ time and $O(q^{1+\epsilon})$ work on a CRCW PRAM [7] and hence in $O(\log^4 q)$ time on a CREW PRAM). Combining this with our algorithm we obtain that shortest-paths from s sources to all other vertices in G' can be computed in $O(\log^4 q)$ time using $O(q^{1.5} + sq \log q)$ work. Reachability from s sources can be computed using $O(M(q) \log^2 q + sq \log q)$ work. Hence, all-pairs shortest-paths in G' can be obtained using $O(q^2 \log q)$ work.

It follows that the alternate encoding of shortest paths suggested by Frederickson [6] (using all-pairs shortest-paths information in each hammock and in G') can be computed in $O(q^2 \log q + n \log^2 n)$ work and $O(\log^4 q + \log^2 n)$ time on a CREW PRAM. Given the above information and an appropriate representation of distances in each hammock (see Djidjev *et al.* [4]), we can produce the distance between two given vertices as follows: In constant work if the vertices belong to different hammocks and in $O(\log n)$ work and time if the vertices are in the same hammock. Hence, distances between k specified pairs of vertices can be computed in $O(\log^4 q + \log^2 n)$ time using $O(q^2 \log q + n \log^2 n)$ preprocessing work and $O(k \log n)$ additional work. The resulting bounds for a representation (via a compact routing table) of all-pairs shortest paths in G are $O(\log^4 q + \log^2 n)$ time and $O(nq \log^2 n)$ work on a CREW PRAM. Shortest-paths trees from s sources can be computed on a CREW PRAM in $O(\log^4 q + \log^2 n)$ time using $O(n \log^2 n + s(n + q \log q) + q^{1.5})$ work. We sketch how the latter bounds are achieved. We compute, for each source u, single-source shortest-paths in G' from the 4 attachment vertices of the hammock containing u. We also compute, for each hammock, distances and shortest-paths information in the subgraph induced by each hammock, from the attachment vertices to all other vertices in the hammock (this information can be produced in $O(\log^2 n)$ time and $O(n \log^2 n)$ work). Given the above, a shortest-path tree from u can be computed using $O(n)$ work.

As for finding an embedding of a planar graph with a small number $q \ll n$ of faces, Frederickson [6] showed that the problem of finding the minimum q is NP-complete. He provided, however, a sequential algorithm that finds an embedding where the number of faces is a constant factor away from the minimum possible. Pantziou *et al.* [18] provided a parallel version of the approximation algorithm that uses $O(n)$ processors and runs in $O(\log^6 n)$ time on a CREW PRAM. The latter can be combined with the above algorithm when a planar embedding is not provided with the input.

Acknowledgements. The author would like to thank Noga Alon, Andrew Goldberg, Seffi Naor, and Mario Szegedy for helpful discussions, Christos Zaroliagis for many comments and references, and John Reif for comments, references, and suggesting to reduce the time bound by a factor of $O(\log n)$.

References.

[1] E. Cohen and N. Megiddo. Improved algorithms for linear inequalities with two variables per inequality. In *Proc. 23rd Annual ACM Symposium on Theory of Computing*, pages 145–155. ACM, 1991. Full version to appear in SIAM J. Comp.

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

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

[4] H. N. Djidjev, G. E. Pantziou, and C. D. Zaroliagis. Computing shortest-paths and distances in planar graphs. In *Proc. 18th ICALP*, pages 327–339. Springer Verlag, LNCS 510, 1991.

[5] G. N. Frederickson. Fast algorithms for shortest paths in planar graphs with applications. *SIAM J. Comput.*, 16:1004–1022, 1987.

[6] G. N. Frederickson. Planar graph decomposition and all pairs shortest paths. *J. Assoc. Comput. Mach.*, 38:162–204, 1991.

[7] H. Gazit and G. L. Miller. A deterministic parallel algorithm for finding a separator in planar graphs. Technical Report CMU-CS-91-103, Carnegie-Mellon University, 1991.

[8] Y. Han, V. Pan, and J. Reif. Efficient parallel algorithms for computing all pair shortest paths in directed graphs. In *Proc. 4th annual ACM Symposium on Parallel Algorithms and Architectures*, pages 353–362. ACM, 1992.

[9] J. JáJá. *Parallel algorithms.* Addison-Wesley Publishing Co., Reading, MA, 1992.

[10] M-Y Kao and P. N. Klein. Towards overcoming the transitive-closure bottleneck: efficient parallel algorithms for planar digraphs. In *Proc. 22nd Annual ACM Symposium on Theory of Computing*, pages 181–192. ACM, 1990.

[11] M-Y Kao and G. E. Shannon. Local reorientation, global order, and planar topology. In *Proc. 21st Annual ACM Symposium on Theory of Computing*, pages 286–296. ACM, 1989.

[12] P. N. Klein and S. Sairam. A linear-processor polylog-time algorithm for shortest paths in planar graphs. In *Proc. 34th IEEE Annual Symposium on Foundations of Computer Science*, pages 259–270. IEEE, 1993.

[13] A. Lingas. Efficient parallel algorithms for path problems in planar directed graphs. In *SIGAL, Lecture Note in Computer Science 450*, pages 447–457, 1990.

[14] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Applied Math.*, 36:177–189, 1979.

[15] G. L. Miller, S-H Teng, and S. A. Vavasis. A unified geometric approach to graph separators. In *Proc. 32nd IEEE Annual Symposium on Foundations of Computer Science*, pages 538–547. IEEE, 1991.

[16] V. Pan and J. Reif. Fast and efficient solution of path algebra problems. *J. Comput. System Sci.*, 38:494–510, 1989.

[17] V. Pan and J. Reif. The parallel computation of minimum cost paths in graphs by stream contraction. *Information Processing Let.*, 40:79–83, 1991.

[18] G. E. Pantziou, P. G. Spirakis, and C. D. Zaroliagis. Efficient parallel algorithms for shortest paths in planar digraphs. *BIT*, 32:215–236, 1992.

[19] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Alg.*, 7:309–322, 1986.