A linear-processor polylog-time algorithm for shortest paths in planar graphs

Philip N. Klein* Brown University **Sairam Subramanian†** Brown University

Abstract

We introduce a new technique for efficiently searching a sparse graph in parallel. Using this technique, we give an algorithm requiring polylog time and a linear number of processors to solve single-source shortest paths in directed planar graphs, bounded-genus graphs, and 2-dimensional overlap graphs. More generally, the algorithm works for any graph provided with a decomposition tree constructed using size-$O(\sqrt{n}$ polylog $n)$ separators.

1 Introduction

Computing single-source shortest paths with nonnegative lengths, in addition to being a fundamental problem in its own right, arises frequently in solving other problems. The sequential algorithms for this problem are quite efficient; they require only slightly more than linear time [1,7,8,13]. Unfortunately, we don't know how to solve this problem as efficiently in parallel. Polylog-time algorithms using a polynomial number of processors are known, but the processor bounds are large. These algorithms use repeated squaring of $n \times n$ matrices, and hence require about n^3 processors. Thus the *work* done by these algorithms (work = time $\times$ number of processors, essentially the total number of operations) far exceeds the work done by the sequential algorithms. An *efficient* parallel algorithm for a problem is one that does work exceeding the sequential time by no more than a polylog factor. This polylog factor represents the overhead in going from a sequential to a parallel solution.

It is a longstanding open problem to find an efficient polylog-time algorithm for shortest paths. The gap between what we can and cannot achieve is particularly evident when we consider sparse graphs, graphs where the number of edges is a small factor times the number of nodes. For such graphs, the work required by known polylog-time algorithms is essentially the cube of the sequential time required. Put another way, using p processors yields a speed-up of roughly $p^{1/3}$ instead of p.

We give more efficient randomized polylog-time algorithms for shortest paths in sparse graphs with moderate edge-lengths.

Theorem 1 For m-edge directed graphs with integral nonnegative edge-lengths summing to L, single-source shortest paths can be computed in time $O(\text{polylog } (m+L))$ with m^2 processors.

Our main result, concerns planar graphs.

Theorem 2 For n-node planar graphs with integral nonnegative edge-lengths summing to L, single-source shortest paths can be solved in time $O(\text{polylog } (m+L))$ using n processors.

Our proof of Theorem 2 does not depend on planarity. and can be generalized to apply also to any graphs for which $\tilde{O}(\sqrt{n})$-node separators[1] can be found efficiently in polylog time, including constant-genus graphs and two-dimensional geometric overlap graphs.[2]

The best previously known polylog-time algorithm for single-source shortest paths in undirected graphs with such separators is that of Pan and Reif [24,25], which does $\tilde{O}(n^{1.5})$ work. Cohen [5] has given an algorithm with similar bounds for the directed case. The previous bounds are no better for breadth-first search, the special case where every edge-length is one.

Our shortest-path algorithm applied to planar graphs in particular has several further applications.

*Research supported by NSF PYI award CCR-9157620, together with PYI matching funds from Honeywell, Thinking Machines Corporation and Xerox Corporation. Additional support provided by ARPA contract N00014-91-J-4052 ARPA Order No. 8225.

†Research supported in part by a National Science Foundation Presidential Young Investigator Award CCR-9047466 with matching funds from IBM, by NSF research grant CCR-9007851, by Army Research Office grant DAAL03-91-G-0035, and by the Office of Naval Research and the Defense Advanced Research Projects Agency under contract N00014-91-J-4052 and ARPA order 8225.

[1] We use the $\tilde{O}(\cdot)$ notation to elide polylog factors.

[2] Use of separators in (sequentially) computing shortest paths was pioneered by Frederickson [6].

It can be used to obtain an efficient polylog-time algorithm for minimum s-t cut in planar graphs, using Johnson's parallel version of an algorithm of Reif [14, 15,26]. In the case when s and t belong to the same face, we can also efficiently obtain a maximum s-t flow, using the algorithm of Hassin [11]. Best previously known parallel algorithms for these problems relied on the algorithm of Pan and Reif, and hence required $\tilde{O}(n^{1.5})$ work.

Breadth-first search also has several applications. For example, using our algorithm in conjunction with the method of Miller [17], we obtain the first polylog-time, linear-processor algorithm for finding $O(\sqrt{n})$ separators in planar graphs. The best NC algorithm previously known was that of Gazit and Miller [9], which takes $O(\log^2 n)$ time and requires $O(n^{1+\epsilon})$ processors (for any given constant $\epsilon > 0$).[3] Another application of parallel breadth-first search is in implementing in parallel a method due to Baker [3] for approximation schemes for a variety of problems in planar graphs. Her method involves doing a breadth-first search on a graph derived from the planar input graph. Since the graph still has small separators, our method applies. The second stage of her method involves solving the problem exactly on k-outerplanar graphs. This stage can be done efficiently in parallel using the algorithm of Lagergren [19]

Our method also provides an algorithm for multiple-source reachability in directed graphs with separators. This generalizes the result of Kao and Klein [16] on planar graphs (see also Guattery and Miller [10]).

We can extend the algorithm presented here to handle negative lengths, as long as no negative-length edge appears in a directed cycle. Using this algorithm, we can solve longest-path in a dag and, by using the technique of Hassin and Johson [12], we can find the maximum flow in an undirected network where s and t are not necessarily on the same face, both in polylog time using a linear number of processors.

The top-level description of our shortest-path algorithm is as follows. We show that the exact shortest-paths problem can be quickly and efficiently reduced to a certain kind of approximate shortest-path problem. The reduction resembles a scaling algorithm due to Gabow [23] for shortest paths. To solve the approximate problem, we show how to construct a graph in which shortest-path distances approximate shortest-path distances in the original graph, and in which for every pair of nodes v, w a shortest path from v to w exists that has only a polylog number of edges. As Cohen observed [5], a simple parallel variant of the Bellman-Ford algorithm then suffices to quickly find shortest paths in such a graph from any given source. To construct this graph, we use similar but denser graphs on the separators. This idea was used by Lingas [21] in the context of breadth-first search and later by Cohen [5] for shortest-paths. However, their constructions required roughly $n^{1.5}$ work, whereas ours requires roughly linear work.

[3] Their algorithm can also be used to find separators of size $O(\sqrt{n}$ polylog $(n))$ using only $O(n)$ processors. Indeed, we use this in our algorithm.

2 The ingredients

For a graph G, we use $V(G)$ to denote the nodes of G, and $|G|$ to denote the number of edges in G. An *r-edge path* is one with at most r edges.

Separators

For a graph G and a node-subset S, an *S-balanced node-separator* is a set of nodes whose removal breaks G into two pieces such that each piece contains at most an α fraction of the nodes of S (for some suitable constant $1/2 \leq \alpha < 1$). Several families of graphs have such separators of size $O\sqrt{n})$ where n is the number of nodes in the graph. Examples include planar graphs [22], bounded-genus graphs, two-dimensional overlap graphs [18], and graphs excluding a fixed graph as a minor [2]. Of these families, for all but the last there are NC algorithms for finding such separators. If one is willing to settle for separators of size $O(\sqrt{n}$ polylog $(n))$, the algorithm of Gazit and Miller [9] for planar graphs requires only a linear number of processors. Separators of size $O(\sqrt{n})$ can be found for overlap graphs by an algorithm using only a linear number of processors [18].

2.1 Decomposition tree

Let G be a constant-degree graph all of whose subgraphs belong to such a separator family. By repeatedly finding separators, one obtains a decomposition of G. Following tradition, we find it useful to represent the structure of the decomposition by means of a rooted tree, the decomposition tree. We refer to *vertices* of the decomposition tree instead of *nodes* in order to avoid confusion with the nodes of G. Each vertex v of the decomposition tree is labeled with a node-induced subgraph $G(v)$ of G, called the *pertinent graph* of v, and a subset $S(v)$ of the nodes of $G(v)$, called the *splitting set* of v. Essentially $S(v)$ consists of the nodes used to separate $G(v)$, together with some nodes used in separators belonging to ancestors of v.

We define the decomposition tree recursively as follows. Suppose S is a subset of G's nodes such that

$|S| = \tilde{O}(\sqrt{n})$. If G contains only one node, then the decomposition tree for G and S consists of a single vertex v labeled with pertinent graph G and splitting set S. Otherwise, let X be a separator for G that breaks G into up to three pieces such that each piece contains at most an α fraction of the nodes of G and S.[4] Let G_1, G_2, G_3 be the pieces with the separator nodes X added in. For $i = 1, 2, 3$, let $S_i = (X \cup S) \cap V(G_i)$. Let T_i be the decomposition tree for G_i and S_i. Then the decomposition tree for G and S is obtained $T_1 \cup T_2 \cup T_3$ by adding a new vertex v labeled with G and with splitting set $S \cup X$, and having as its three children the roots of T_1, T_2, and T_3.

The point of the splitting sets is as follows. A path cannot "escape" a pertinent graph $G(v)$ without first passing through a node of the splitting set of v's parent $p(v)$. More formally, any path in G(root of decomposition) from a node in $G(v)$ to a node outside $G(v)$ has an initial subpath consisting of nodes of $G(v)$ and ending in a node of $S(p(v))$. We call this the *splitting property* of the decomposition tree with respect to the graphs $G(\cdot)$.

A vertex v in the tree is said to be in level i if the distance of v from the root of the tree is i. Let $T(G_0)$ be the decomposition tree for G_0 and the empty set. Let $N = |V(G_0)|$. It is not hard to show that $T(G_0)$ has the following properties:
- The number of levels in the decomposition tree is $O(\log N)$, and
- For each level i, the sum of the sizes of the pertinent graphs at that level is $O(N)$, and the sum of the squares of the sizes of the splitting sets is $\tilde{O}(N)$.

2.2 Reducing exact to approximate shortest paths

For this paper, the $(1-\epsilon)$-approximate shortest path problem is to compute distance estimates $d(v)$ from a given root such that (a) there is an auxiliary graph containing the nodes and edges of the input graph and such that the estimates are exact shortest-path distances in the auxiliary graph, and (b) If the distance in the original graph between two nodes is d, the distance in the auxiliary graph is between $(1-\epsilon)d$ and d. We now prove that the single-source shortest path problem is efficiently polylog-time-reducible to the $\frac{1}{2}$-approximate shortest path problem.

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

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

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

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

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

$$\begin{aligned} \widehat{\ell}(P) &= \sum_{uv \in P} \lceil d(u) \rceil + \ell(uv) - \lceil d(v) \rceil \\ &= \lceil d(\text{source}) \rceil + \ell(P) - \lceil d(x) \rceil \qquad (1) \end{aligned}$$

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

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

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

2.3 Limited Bellman-Ford search from a source-path

Given a graph G with a source, and given a positive integer r, we consider the problem of computing, for each node v, the shortest source-to-v path consisting of at most r edges. As was observed by Cohen [4,5], this problem can be solved in $O(r \log |G|)$ time and $O(r|G|)$ work by a variant of the Bellman-Ford algorithm. We initialize all distance estimates $d(v)$ to ∞, except that $d(\text{source}) = 0$. In parallel, we *relax* all the edges in G, i.e. we compute new distance estimates $d'(v)$ by setting $d'(v) := \min_{\text{out-edges } uv} d(u) + \ell(uv)$ for each node v. A simple induction shows that after this step

[4] It is easy to see that such a separator can be obtained by first finding a G-balanced separator and then finding an S-balanced separator in the piece that contains more than half the nodes of S.

is repeated r times, the resulting distance estimates are as desired.

We now show how to adapt this idea to give a shortest-path procedure that finds approximate r-edge distances in a graph G to a path $P = \eta_1 \dots \eta_s$.

Lemma 2 There is a procedure that takes as inputs a graph G, a path P in G, a positive integer r, a distance bound d^*, and an error parameter ϵ. The procedure computes the following for each ordinary node v in G and each distance $d \le d^*$ that is a multiple of ϵd^*:
(A) either a node η in P such that

(1) there is a v-to-η path of length at most d, and
(2) for every node η' occuring before η in P, every v-to-η' path consisting of at most r edges has length more than $d - \epsilon d^*$,

(B) or ∞, if there is no v-to-P path of length at most d and size at most r.
The procedure takes time $O(r \log m)$ and work $O(m\epsilon^{-1}r^2)$, where m is the number of edges in G.

First the procedure discards edges of length more than d^*, as such edges obviously do not participate in paths of length at most d^*. Next it uses rounding to replace the edge-lengths with small integers. Let $\ell(vw)$ denote the length of edge vw, and define $\widehat{\ell}(vw) := \lceil \ell(vw)/\alpha \rceil$, where $\alpha = \epsilon d^*/r$. Let $i^* = r\epsilon^{-1}$, and note that $\widehat{\ell}(vw) \le i^*$.

Let the source-path be $P = \eta_1\eta_2 \dots \eta_s$. We give a procedure using lengths $\widehat{\ell}(\cdot)$ that computes a table T_v of size $i^* + 1$ for every node v, satisfying the following *correctness condition*: for each nonnegative $i \le i^*$, $T_v[i]$ is the minimum index j such that there is a v-to-η_j path of length at most i and size at most r. If there is no such index, $T_v[i] = \infty$.

Before giving this procedure, we show how it can be used to prove Lemma 2. For any distance $d \le d^*$ that is a multiple of ϵd^*, let $i = \lceil d/\alpha \rceil$. Then for a node v, if $T_v[i]$ is ∞ then there is no no v-to-P path of length at most d and size at most r. If $T_v[i] = j$ then there is a v-to-η_j path that has $\widehat{\ell}$-length at most d/α. Hence η_j satisfies (1) of Lemma 2. Moreover, for any η' occuring before η_j in P, every v-to-η' path Q of size at most r satisfies

$$i < \sum_{vw \in Q} \lceil \ell(vw)/\alpha \rceil \le (\sum_{vw \in Q} \ell(vw)/\alpha) + r$$

Substituting for i and α yields $\ell(Q) > d - \epsilon d^*$. Hence η_j satisfies condition (2) of Lemma 2.

Now we give the procedure. To initialize the tables, for every i we set $T_v[i] := \infty$ for all nodes v not in the path P, and set $T_{\eta_j}[i] := j$ for nodes η_j in the path.

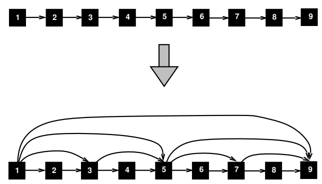

Figure 1: Augmenting a segment with jump edges.

The basic step is relaxing all edges by simultaneously updating each table T_v as follows.

$$T'_v[i] := \min\{T_v[i], \min\{T_w[i - \ell(vw)] \ : \ \ell(vw) \le i\}\}. \quad (2)$$

It is a simple induction on r to show that, after r iterations of applying (2) to all the nodes of G simultaneously, the correctness condition holds. Each of these r iterations requires time $O(\log m)$ and work $O(i^* m)$, where m is the number of edges of G. Substituting for i^* yields the bounds of Lemma 2. A similar procedure can be used to compute r-limited shortest-paths from P to the nodes of G.

2.4 Summaries

A *summary* H of G is a graph consisting of nodes of G (*ordinary nodes*) and *segment nodes*. Each segment node is a *copy* of some node of G. We define all such copies to be distinct.[5] Each edge uv of H corresponds to a path in G between the nodes corresponding to u and v. Note that distinct edges need not map to disjoint paths. The mapping from edges of H to paths of G is represented by a *derivation dag*, described later.

The segment nodes of H are partitioned into disjoint paths, called *segments*. The only edges in H between segment nodes are the edges belonging to the segments. These edges are called *segment edges* to distinguish them from *ordinary edges*, which connect ordinary nodes to segment nodes or to other ordinary nodes. The *segment graph* of H is the graph obtained by coalescing each segment into a single node. A *supernode* of H is either an ordinary node or a segment. Thus the nodeset of the segment graph of H consists of the supernodes of H. Note that the segment graph may have multiple edges between the same endpoint. Note also that it has no edges between segments. We define the *edge-multiplicity* of the segment graph to be equal to the maximum number of edges between any two nodes of the segment graph.

A *hop* in H is a path from one ordinary node to another all of whose internal nodes are segment nodes belonging to a single segment. Since there are no edges

[5] Hence when two summaries are unioned, segment nodes are never coalesced, as happens to ordinary nodes appearing in both summaries.

between segments, any path in H between ordinary nodes can be decomposed into hops. A k-hop path is one consisting of at most k hops.

2.5 Applying limited Bellman-Ford to a summary

In computing limited shortest paths in a summary, we cannot afford to spend lots of time traversing the many edges in a segment. For purposes of computing shortest paths, therefore, we shall assume the segments have been *augmented* with *jump edges* as shown in Figure 1. Each jump edge is assigned a length equal to the sum of the lengths jumped, so distances are not affected. These jump edges have the property that any subpath of the segment can be replaced by a path of at most $2\log s$ jump edges, where s is the size of the segment. Moreover, given any subset X of the ndoes of the segment, there is a node-induced subgraph consisting of $|X|\log s$ nodes and $2|X|\log s$ edges that has the same property with respect to X: any path in the original segment starting and ending at nodes of X can be replaced by a path in the subgraph consisting of at most $2\log s$ edges. This is called the subgraph *relevant* to X. We assume in the following lemmas that the segments of the input summary H have been thus augmented. We also assume that sufficient preprocessing has taken place that, given a subset X of the nodes of a segment, it is easy to obtain the relevant subgraph. We omit the details.

Lemma 3 Given an augmented summary H and a source node v of H, k-hop-limited shortest paths to all ordinary nodes of H can be found in time $O(k\log^2 n_0)$ and work $O(km\log^3 n_0)$, where n_0 is the number of nodes in H and m is the number of ordinary edges.

Proof: In order to achieve a work bound that depends only on the number m of ordinary edges, rather than on the total number of edges in the summary, we must restrict our attention to only some of the segment nodes and edges. Call a segment node *active* if it has incident ordinary edges. For each segment, we consider only the subgraph relevant to the sement's active nodes. hence the total number of nodes and edgs under consideration is $O(m\log n_0)$. By applying limited Bellman-Ford search from the source node v with $r = 2k\log n_0$, we obtain the desired shortest paths. □

Next we make use of the shortest-path approximation algorithm of Lemma 2 for searching from a path. Our goal is to construct a representation of nearly shortest k-hop paths that go through the path.

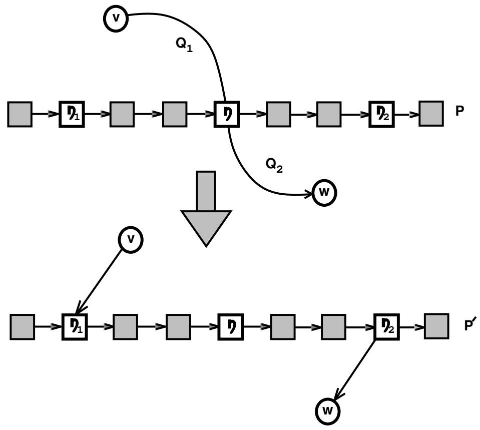

Figure 2: The P-substitute satisfies correctness properties (III) and (IV).

Lemma 4 There is a procedure that, given an augmented summary H, a distance bound d^*, an accuracy parameter ϵ, an ordinary path P in H of length at most $\epsilon d^*/4$, and a positive integer k, constructs a *P-substitute* for H, consisting of a *copy* P' of P and a set of *pseudoedges* between ordinary nodes and nodes of P', with the following properties:

I. Each ordinary node has $O(\epsilon^{-1})$ pseudoedges to and from P'.
II. Each edge in P' has cost $1-\epsilon$ times the corresponding edge in P.
III. Each pseudoedge corresponds to a path in H and has length $1-\epsilon$ times the length of the path.
IV. For ordinary nodes v and w, if there is a v-to-w path of length $d \in [d^*/2, d^*]$ consisting of at most k hops and containing a node of P, then there is a v-to-w path of length at most d consisting of a pseudoedge, followed by a subpath of P', followed by a pseudoedge.

The procedure constructs the P-substitute in time $O(k\log^2 n_0)$ and work $O(m\epsilon^{-1}k^2\log^3 n_0)$, where n_0 is the number of nodes in H.

Proof: As before, we consider only relevant subgraphs of active nodes within each segment, so the number of edges under consideration is $O(m\log n_0)$. The procedure to construct the P-substitute is as follows. Apply the procedure of Lemma 2 to H' with $r = (2 + 2\log n_0)k$, distance bound d^*, and error parameter $\bar\epsilon = \epsilon/8$. For each ordinary node v we construct $\bar\epsilon^{-1}$ outgoing edges, one for each distance $d \le d^*$ that is a multiple of $\bar\epsilon d^*$. The edge corresponding to distance d is $v\eta$, where η is the copy in P' of the node of P satisfying (1) and (2) of Lemma 2, namely:

(1) there is a v-to-η path of length at most d, and
(2) for every node η' occuring before η in P, every v-to-η' path consisting of at most r edges has length more than $d - \bar{\epsilon}d^*$.

The length assigned to the pseudoedge is $1-\epsilon$ times the length of the path it represents, namely the path of (1). For those d for which (Lemma 2,B) holds, there is no corresponding pseudoedge.

We have described the construction of outgoing pseudoedges. Incoming pseudoedges are constructed analogously. The time and work for this procedure are dominated by the time and work required by the two calls to the procedure of Lemma 2. The bounds of Lemma 4 therefore follow.

Now we show the construction satisfies the correctness properties. Properties (I) and (II) are immediate. Let Q be a v-to-w path of length between $d^*/2$ and d^* consisting of at most k hops and containing a node η of P. Write $Q = Q_1\eta Q_2$, and let $d_1 = \lceil \ell(Q_1)/\bar{\epsilon}d^* \rceil \bar{\epsilon}d^*$, and $d_2 = \lceil \ell(Q_2)/\bar{\epsilon}d^* \rceil \bar{\epsilon}d^*$. Let $v\eta_1$ be the outgoing pseudoedge in the substitute that corresponds to distance d_1, and let $\eta_2 w$ be the incoming pseudoedge that corresponds to distance d_2. By construction $\ell(Q_1) - \bar{\epsilon}d \leq d_1 \leq \ell(Q_1) + \bar{\epsilon}d$ and $\ell(Q_2) - \bar{\epsilon}d \leq d_2 \leq \ell(Q_2) + \bar{\epsilon}d$. Therefore by condition (2) of Lemma 2 (as shown in Figure 2) η occurs no earlier than η_1 and no later than η_2 on P. Thus η_1 occurs no later than η_2 on P. Let P'' be the subpath of P' from η_1 to η_2. The path consisting of the pseudoedge $v\eta_1$ followed by P'' followed by $\eta_2 w$ has length at most $(1-\epsilon)[d_1 + d_2 + \ell(P)]$, which is at most $\ell(Q)$. □

2.6 The derivation dag

In manipulating a summary H of a graph G, it is useful to maintain the mapping from edges of H to paths in G. A *derivation dag* for a graph G and summary H is a directed acyclic graph (dag) whose vertices represent paths in G. The vertices with no predecessors represent edges of G. Every other vertex v has an ordered pair of predecessors; the path represented by v is the path obtained by composition from the paths represented by v's predecessors. We require that for each edge of H the corresponding path of G is represented by some vertex in the derivation dag. For brevity, we abuse the terminology and say that the edge of H is represented by the vertex. We also require that the the derivation dag be shallow and not be much bigger than the graph G. In particular, the derivation dag should have depth polylogarithmic in $|G|$ and size $\tilde{O}(|G|)$. Here *depth* is the maximum number of edges in a directed path in the dag.

In connection with a derivation dag, we will use the term *constructing* a path P to mean adding to the dag a vertex representing P (including adding arcs and additional vertices as needed.) We shall have need on occasion to construct a subpath P' of a path P already represented. This is facilitated by the fact that one can always find a logarithmic number of paths represented in the dag whose composition is P'.

We list some of the operations that can be performed with the help of the derivation dag. Each of these operations can be performed by doing a constant number of upward and downward sweeps of the dag. Hence for each operation, the work is proportional to the size of the dag, and the parallel time is proportional the its depth. By our requirements about the size and depth of the derivation dag, the time is therefore polylogarithmic in $|G|$, and the work is $\tilde{O}(|G|)$. In this extended abstract, we omit the details concerning carrying out the operations.

Derivation dag operations:

- Given a set of represented paths (i.e. given the corresponding vertices of the derivation dag), find the set of underlying edges in G belonging to these paths.
- Given a set S of nodes of G, find the set of represented paths containing nodes in S.
- Given a length parameter r and a set X of represented paths, partition the paths in X into node-disjoint segments of length at most r. The partitioning has the following minimality property: for two adjacent segments of the same path of X, if the two segments were combined into one, the resulting segment would be longer than r.
- Given a set of represented paths X and a set of nodes S of G, for each path π in X determine one of the nodes in S it contains (if any), and find and construct the two subpaths of π, (1) up to and (2) after the node.

2.7 Near-covers

The edges of a summary H of G need not map to disjoint paths in G. We say two edges of H *intersect* if the corresponding paths in G share a node. We say G is *sparse* if every subgraph of G has at most c times as many edges as nodes, where c is a constant. For example, constant-degree graphs and planar graphs are sparse. If G is sparse but H is dense, it intuitively follows that many of the edges of H intersect. We use this idea in the lemmas below, which are the basis for our shortest-path approximation algorithm.

The following lemma can be proved using an argument due to Leighton [20] for lower-bounding the crossing number of a graph.

Lemma 5 Let H be a summary of a sparse graph G. Suppose the segment graph of H has m edges and n

nodes and maximum edge-multiplicity 1, and $m = \Omega(n)$. Then $\Omega(m^3/n^2)$ pairs of ordinary edges of H intersect.

A *near-cover* of H is a set of ordinary edges A such that at least half the ordinary edges of H intersect some edge in A.

Lemma 6 Let H be a summary of a sparse graph G. Suppose the segment graph of H has m edges and n nodes and maximum edge-multiplicity r, and $m = \Omega(rn)$ Then a near-cover of H of size $O(r^2n^2/m)$ can be found in expected time $O($ polylog $|G|)$ and work $O(r(|G| + |H|)$ polylog $|G|)$.

Proof Sketch: We can write $H = H_1 \cup H_2 \cup \ldots \cup H_r$, where for each i the segment graph of H_i has at least m/r ordinary edges and at most one edge between each pair of nodes. It suffices to find a near-cover of each H_i of size $O(rn^2/m)$. Let $m_i \geq m/r$ be the number of ordinary edges in H_i. Let E be a subset of the ordinary edges of H_i such that $|E| = \Omega(n)$. It follows from Lemma 5 that there are $\Omega(|E|^3/n^2)$ pairs of intersecting edges in E. Using this fact, it can be proved that a random subset of the edges of H_i has probability exceeding a constant of being a near-cover of H_i, where each edge is chosen with probability $O(n^2/m_i^2)$. This gives us the following procedure to find a near-cover in H_i: randomly select edges with the appropriate probability, and use the derivation dag to determine if it is a near-cover. If so, and if the number of selected edges is $O(n^2/m_i)$, the procedure is done. Otherwise, repeat. The expected number of repetitions is constant. □

3 The shortest-path approximation algorithm

In this section we describe our randomized algorithm for single-source shortest path approximation. Throughout this section we use G_0 to denote the (sparse) input graph, and N to denote $|V(G_0)|$. We use L to denote the sum of lengths. We use G to denote a subgraph of G_0. Our goal is an algorithm that requires $O($polylog $(N + L))$ time and uses N processors. It is sufficient to give an algorithm requiring $O($polylog $(N + L))$ time and $O(N$ polylog $(N + L))$ work, because then the processor bound can be reduced to N at the expense of increasing the running time by an $O($polylog $(N + L))$ factor.

We want the correctess and running time of our algorithm to hold with high probability, i.e. with probability $1 - N^{-c}$, where c is any given constant. For this reason the time bounds for our procedures depend on $\log N$. For example, the procedure of Lemma 6 for finding a near-cover in a graph G takes expected time $O($polylog $|G|)$. By having this procedure start over whenever its running time significantly exceeds the expected time, we get a procedure that only starts over $O(\log N)$ times with high probability, for a total running time of $O($polylog $N)$.

For an accuracy parameter ϵ, the algorithm computes $(1 - \epsilon)^{O(\log^2 N)}$-approximate shortest-path distances. We therefore assign $\epsilon := \Theta(1/\log^2 N)$ so as to get $\frac{1}{2}$-approximate distances. Our procedures also refer to a parameter k, a limit on number of hops as in Lemma 3 and 4. In order to achieve correctness with high probability, we assign $k := \hat{c} \log N$ where $\hat{c}$ is a constant to be determined.

We use H to denote a summary of G, and we use n and m to denote the number of nodes and edges in the segment. For every summary H under consideration, each segment has $O(\epsilon^{-1}n)$ nodes and at most $O(\epsilon^{-1})$ edges to and from each ordinary node (cf. Lemma 4). Hence the maximum edge-multiplicity of the segment graph of H is $O(\epsilon^{-1})$. We use $V_o(H)$ and $V_s(H)$ to denote, respectively, the set of ordinary nodes of H and the set of supernodes of H (nodes of the segment graph of H).

For a set S of nodes, *S-distances* are the shortest-path distances (in a given graph) between pairs of nodes in S. An *$(1 - \epsilon)$-accurate underestimate* of a distance d is an estimate between $(1 - \epsilon)d$ and d; the definition of an $(1 + \epsilon)$-accurate overestimate is analogous.

Lemma 7 Consider two graphs, A and B, where $V(A) \cap V(B) = V'$. Suppose that V'-distances in B are $(1-\epsilon_2)$-accurate underestimates of V'-distances in A, and, for some $S \subset V(A)$, S-distances in A are $(1 - \epsilon_1)$-accurate underestimates of some other set of distances δ. Then

1. $V(A)$-distances in $A \cup B$ are $(1-\epsilon_1)(1-\epsilon_2)$-accurate underestimates of δ, and
2. V-distances in B equal V-distances in $A \cup B$.

3.1 Reducing the size and diameter of a summary

This subsection contains the core of our algorithm for shortest-path approximation. In particular, we describe two procedures, Compact and SmallPath. The first takes as parameters a summary H and a node-subset S, and produces a summary whose size is proportional to S that approximates distances in H. The second takes a summary H and produces another summary of size proportional to that of H that approximates distances in H and that obeys the following crucial property: for any two ordinary nodes

u and v, if there is a u-to-v path in the output summary, there is a shortest such path that consists of only $O(\log^2 N)$ hops.

Both these procedures are based on a random-sampling technique due to Ullman and Yannakakis [27], combined with our procedure ABBREVIATE. Intuitively, the goal of ABBREVIATE is to search k-hop paths between n nodes in a summary H with m edges. Naively this would take work proportional to nm, where m is typically around n^2. Our method performs the search in around n^2 work, using the fact that the summary has a near-cover of size roughly n^2/m. Namely, it finds a near-cover and replaces its edges with paths; it then does searches both to and from all these paths using the procedure of Lemma 4, effectively searching all k-hop paths that intersect the edges of the near-cover. The work for the search is roughly m per path, for a total of roughly n^2. In order to search the paths that do not intersect the edges of the near-cover, it deletes the edges intersecting the near-cover and recurses.

Lemma 8 *There is a procedure that given a distance bound d^* and a summary H such that no ordinary edge in H has length more than d^* produces a summary $\widehat{H}$ with the following properties:*

- A1: $V(H) \cap V(\widehat{H})$ is the set of ordinary nodes of H.
- A2: $\widehat{H}$ has at most $\max\{0, n/4 \log L - cn^2/\epsilon^3 m\}$ segments, where c is a constant.
- A3: For $u, v \in V_0(H)$, if the u-to-v distance in H is d such that $d^*/2 \le d \le d^*$, the u-to-v distance in $\widehat{H}$ is at least $(1-\epsilon)d$.
- A4: For $u, v \in V_o(H)$, if there is a k-hop u-to-v path of length d in H such that $d^*/2 \le d \le d^*$, then the one-hop u-to-v distance in $\widehat{H}$ is at most d.

The procedure runs in time $O(k \text{ polylog } N)$ and requires work $O((|G| + k^2\epsilon^{-4}n^2) \text{ polylog } N \log L)$ with high probability.

Proof: The procedure is recursive. The base case is when the number of ordinary edges is $8cn\epsilon^{-3}\log L$. In this case, the procedure uses the procedure of Lemma 3 to find k-hop-limited shortest paths from every ordinary node to every other ordinary node. The graph $\widehat{H}$ consists of the ordinary nodes of H, together with an edge vw for every pair of ordinary nodes such that a limited shortest path was found from v to w; the length of the edge is taken to be the length of the path. In this case, properties (A1) through (A4) clearly hold. From lemma 3 it can be seen that the time required for this computation is $O(k \log^2 n_0)$ and the work required is $O(k^2 n^2 \epsilon^{-3} \log^3 n_0 \log L)$.

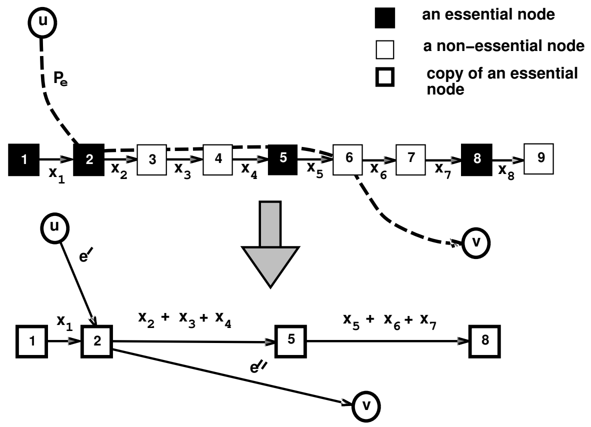

Figure 3: Constructing H_A from H.

The recursive case is as follows. First it uses the procedure of Lemma 6 to find a near-cover A of size $cn^2/4\epsilon^2 m$ for H (this is where c is determined), i.e. a set of ordinary edges of H that intersect at least half the ordinary edges. Let H' be the summary obtained from H by deleting all the intersected edges. Let m' be the number of ordinary edges of H'. Since A is a near-cover, $m' \le m/2$. The procedure calls itself recursively on H', returning a summary $\widehat{H}'$. By the inductive hypothesis,

- A1′: $V(H) \cap V(\widehat{H}') = V_0(H)$.
- A2′: $\widehat{H}'$ has at most $\max\{0, n/4 \log L - cn^2/\epsilon^3 m'\}$ segments.

We will show how the procedure obtains $\widehat{H}$ from $\widehat{H}'$ by adding $cn^2/\epsilon^3 m$ segments and putting in edges between these segments and the ordinary nodes. Thus property (A1) will hold. The number of segments in $\widehat{H}$ will be $\max\{0, n/4 \log N - cn^2/\epsilon^3 m'\} + cn^2/\epsilon^3 m.$, which is at most $\max\{cn^2/\epsilon^3 m, n/4 \log N - cn^2/\epsilon^3 m\}$ since $m' \le m/2$. Moreover, since $m > 8cn\epsilon^{-3}\log L$ (else the base case applies), the second term of the $\max\{\}$ dominates the first. Thus (A2) holds.

As for (A3) and (A4), the u-to-v paths in H not represented in H' (and hence not represented in $\widehat{H}'$) are those containing edges intersecting A. The added segments will help represent these paths. Recall that an ordinary edge e is said to intersect an edge $a \in A$ if the path in G that e represents shares a node with the path that a represents. Using the derivation dag, the procedure determines one such node v_e for each intersected edge e. For each such node, the procedure determines one edge $a \in A$ whose path contains the node. We say that node is *essential* to a.

Next the procedure uses the derivation dag to process the paths represented by the edges in A. It decomposes these paths into node-disjoint subpaths of length at most $\epsilon d^*/4$. Since each edge has length at most d^*, each of the $cn^2/4\epsilon^2 m$ paths gives rise to at most $4/\epsilon$ subpaths, for a total of $cn^2/\epsilon^3 m$ subpaths. The procedure applies a splicing operation to each such subpath, splicing out all inessential nodes. When edges are merged due to splicing, their lengths are summed, so the distances between essential nodes are maintained. The spliced subpaths will be segments in $\widehat{H}$.

In order to determine which ordinary edges to include in $\widehat{H}$, the procedure first constructs an auxiliary graph H_A from H, and then applies the procedure of Lemma 4 to it. The auxiliary graph is obtained from H by adding the spliced subpaths and some edges between them and the ordinary nodes. These are determined as follows. The procedure processes the paths P_e represented by the intersected edges e. For each such path P_e, the procedure constructs the two subpaths P'_e and P''_e obtained by splitting P_e at the node v_e. Since v_e is essential to one of the paths represented by the edges in A, a copy of it appears in one of the spliced subpaths. The procedure adds to H_A the edges e' and e'' corresponding to P'_e and P''_e, attaching these edges to the copy of v_e, as shown in Figure 3.

Next the procedure calls the procedure of Lemma 4 to find P-substitutes for each of these subpaths P. These P-substitutes are added to $\widehat{H}'$ as segments to obtain $\widehat{H}$. As before, we splice out segment nodes that have no ordinary edges attached to them. Thus the number of segments added is $cn^2/\epsilon^3 m$ as required. The amount of work required to construct one P-substitute is $O(m\epsilon^{-1}k^2 \log^3 n_0)$, so the amount of work required to construct $cn^2/\epsilon^3 m$ substitutes is $O(ck^2n^2\epsilon^{-4} \log^3 n_0)$. Since the depth of recursion is logarithmic, the lemma follows. □

Corollary 1 There is a procedure ABBREVIATE that from a summary H produces a summary $\widehat{H}$ with the following properties:

- B1: $V(H) \cap V(\widehat{H}) = V'$.
- B2: $\widehat{H}$ has at most $n/4$ segments.
- B3: For $u, v \in V'$, if the u-to-v distance in H is d, the u-to-v distance in $\widehat{H}$ is at least $(1-\epsilon)d$.
- B4: For $u, v \in V'$, if the k-hop u-to-v distance in H is d, then the one-hop u-to-v distance in $\widehat{H}$ is at most d.

The procedure runs in time $O(k \text{ polylog } N)$ and requires work $O((|G| + k^2\epsilon^{-4}n^2) \text{ polylog } N \log^2 L)$ with high probability.

Proof: The procedure ABBREVIATE is as follows. Let H_i be the subgraph of H consisting of edges of length at most $L/2^i$. There are at most $\log L$ nonempty subgraphs. The procedure simply applies the procedure of Lemma 8 to each H_i with the distance bound $d^* = L/2^i$, and unions the results $\widehat{H}_i$ to form the summary $\widehat{H}$. The only nodes the $\widehat{H}_i$'s have in common are the ordinary nodes of H. Properties (B1) through (B3) are immediate from properties (A1) through (A3) of Lemma 8.

As for property (B4), let P be a k-hop u-to-v path in H, such that $L/2^{i+1} \le l(P) \le L/2^i$. Surely, P exists in H_i. Furthermore, $d^*/2 \le l(P) \le d^*$ (where $d^* = L/2^i$). Hence by property (A4) the one-hop distance in $\widehat{H}_i$ (and hence in $\widehat{H}$) is at most the length of P. This proves property (B4). The work bound follows from the fact that there are at most $\log L$ calls to the prcedure of lemma 8. □

Now we introduce the random-selection technique of Ullman and Yannakakis [27]. Let H be a summary of G. For each pair of ordinary nodes u, v, let us designate a shortest u-to-v path P_{uv}. For a summary H of G, suppose we randomly select a subset consisting of half the ordinary nodes of H. Define a *gap* to be a path with no internal nodes being selected. Say a gap is *large* if it consists of at least k hops, where k is the global we defined to be $\hat{c} \log N$ for a constant $\hat{c}$. The following proposition determines the value of $\hat{c}$.

Proposition 1 With high probability, no designated path P_{uv} contains a large gap.

Proof: For each P_{uv} and each ordinary node w in P_{uv}, consider the $\hat{c} \log N$-hop subpath of P_{uv} starting at w. Say this subpath *fails* if it is a gap. One can show that the probability of a subpath failing is $\exp(-\Omega(\hat{c} \log N))$. Since there are at most N^3 of these subpaths, the probability that any fail is at most $N^3 \exp(-\Omega(\hat{c} \log N))$. By choice of the constant $\hat{c}$, therefore, one can ensure that with high probability no subpath fails. □

Lemma 9 There is a procedure COMPACT that, given a summary H and a subset S of the ordinary nodes of H, produces a summary H_S that with high probability has the following properties:

- C1: S-distances in H_S are $(1-\epsilon)^{\log_{8/7} n}$-accurate underestimates of S-distances in H, where n is the number of supernodes in H.
- C2: H_S has at most $8|S|$ supernodes.

The procedure runs in time $O(\text{polylog } N)$ and work $O(|G| + \epsilon^{-4}n^2) \text{ polylog } N \log L^2)$ with high probability.

Proof: If $|S| \geq n/8$, then H itself obeys (C1) and (C2), so the procedure returns H. Otherwise, the procedure selects a random subset consisting of half the ordinary nodes of H. Also consider the nodes of S as selected. The procedure applies ABBREVIATE to H, obtaining $\widehat{H}$. The procedure constructs H' from $\widehat{H}$ by deleting all the ordinary nodes that were not selected. Finally the procedure obtains H_S by recursing on H'.

The proof of correctness is by induction on n. By (B2) of Corollary 1, H' has $n/4$ segments. H' has at most $|S| + n/2$ ordinary nodes, and $|S| < n/8$, so H' has a total of at most $\frac{7}{8}n$ supernodes. By the inductive hypothesis, H_S has at most $8|S|$ supernodes so property (C2) holds.

By the inductive hypothesis, S-distances in H_S are $(1-\epsilon)^{(\log_{8/7} n)-1}$-accurate underestimates of S-distances in H'. We claim that with high probability S-distances in H' are $(1-\epsilon)$-accurate underestimates of S-distances in H. It follows that property (C1) holds.

It remains to prove the claim. It follows from (B3) that S-distances in $\widehat{H}$ and hence in H' are no less than $1-\epsilon$ times the corresponding distances in H. We must prove that they are no greater than the corresponding distances. By Proposition 1, for every $u, v \in S$ there is a shortest u-to-v path P_{uv} that decomposes into subpaths, where each subpath consists of at most k hops and starts and ends at selected nodes. By property (B4), for each such subpath there is a one-hop path in $\widehat{H}$ with the same endpoints and having no greater length. Since H' contains all segments in $\widehat{H}$ and all selected nodes, each such one-hop path is contained in H'. This proves the claim. $\square$

Lemma 10 There is a procedure SMALLPATH that from a summary H produces a summary H^* that with high probability has the following properties (n is the number of supernodes in H):

- D1: $V_o(H)$-distances in H^* are $(1-\epsilon)^{\log_{4/3} n}$-accurate underestimates of $V_o(H)$-distances in H.
- D2: For any ordinary nodes u, v, a shortest u-to-v path exists in H^* that consists of at most $O(\log N \log n)$ hops
- D3: H^* has at most $2n$ supernodes.

The procedure runs in time $O(\text{polylog } N)$ and work $O(|G| + \epsilon^{-4}n^2) \text{ polylog } N \log L^2)$ with high probability.

Proof: As in Lemma 9, the procedure selects a random subset of half the ordinary nodes of H. Then the procedure applies ABBREVIATE to H with $k = c \log N$, obtaining $\widehat{H}$. The procedure obtains a subgraph H' of $\widehat{H}$ by deleting all unselected ordinary nodes. The procedure calls itself recursively on H', obtaining H'^*, and returns the union $H^* = H \cup H'^*$.

By (B2), H' has $n/4$ segments. It has at most $n/2$ ordinary nodes, so it has at most $(3/4)n$ supernodes in total. By (D3) of the inductive hypothesis applied to H', $|V_s(H'^*)| \leq 2(3/4)n$. The only supernodes in H^* that are not in H'^* are the unselected ordinary nodes, and there are at most $n/2$ such nodes. Thus the total number of supernodes in H^* is at most $2(3/4)n + n/2 = 2n$. Thus property (D3) holds.

Since H^* contains H as a subgraph, clearly $V_o(H)$-distances in H^* are no greater than the corresponding distances in H. Conversely, (B3) implies that $V_o(H)$-distances in $\widehat{H}$ are at least $1-\epsilon$ times the corresponding distances in H, and so certainly distances in H' between ordinary nodes are at least $1-\epsilon$ times the corresponding distances in H, since H' is a subgraph of $\widehat{H}$. By (D1) of the inductive hypothesis, distances in H'^* between ordinary nodes are at least $(1-\epsilon)^{(\log_{4/3} n)-1}$ times the corresponding distances in H', and hence at least $(1-\epsilon)^{\log_{4/3} n}$ times the corresponding distances in H, proving (D1).

It remains to prove (D2). We use an argument like that used in the proof of Lemma 9.

Claim 1 For any path Q in $H \cup H'^*$ between ordinary nodes, there exists an equally short path Q' with the same endpoints such that every subpath of Q' consisting of at least $c \log N$ hops contains a node in H'^*.

Proof: Write $Q = Q_1 v_1 Q_2 v_2 \cdots v_{f-1} Q_f$, where the v_i's are the selected nodes. Then each Q_i not contained in H'^* belongs wholly to H. Replace each such Q_i with the designated shortest path P_{uv} with the same endpoints. The claim follows by Proposition 1.

Property (D2) follows by induction from the following claim.

Claim 2 For any path Q in $H \cup H'^*$ between ordinary nodes, there exists an equally short path Q'' that either consists of at most $c \log N$ hops or else consists of at most $c \log N$ hops followed by a path wholly in H'^* followed by at most $c \log N$ hops.

Proof: Let Q' be the path of Claim 1. If Q' consists of at most $c \log N$ hops, we are done. Otherwise, it contains a first selected node u and a last selected node v; moreover, the prefix of Q' ending at u has at most $c \log N$ hops and similarly for the suffix of Q' starting at v. By Lemma 7, the u-to-v subpath of Q' can be replaced by a path lying wholly in H'^*. $\square$

3.2 Constructing a graph with small diameter

Our goal in this subsection is to solve the $\frac{1}{2}$-approximate shortest-path problem: given a graph with a source, compute distance underestimates from the source to each of the nodes. These estimates must be actual shortest-path distances in some graph. It follows from Lemma 1 that our algorithm to this problem yields an algorithm of comparable performance for the exact shortest-path problem.

Given a graph G_0, our goal is to construct a graph $\overline{G}_0$ that is not too much bigger than G_0, such that $V(G_0)$-distances in $\overline{G}_0$ are $(1-\epsilon)^{O(\log^2 N)}$-underestimates of $V(G_0)$-distances in G_0 itself, and such that for any nodes $v, w \in V(G_0)$, if a shortest v-to-w path exists in $\overline{G}_0$, one exists that has only $O(\text{polylog}\, N)$ hops, and hence, using the augmentation described in Subsection 2.5, only $O(\text{polylog}\, N)$ edges. Single-source shortest paths can then be solved in $\overline{G}_0$ using limited Bellman-Ford. The time required is $O(\text{polylog}\, N)$ because only $O(\text{polylog}\, N)$ stages must take place. The work required is $O(|\overline{G}_0| \text{ polylog}\, N)$.

If G_0 is a sparse graph, we can construct such a $\overline{G}_0$ of size $O(|V(G_0)|^2)$, as shown in Lemma 10. If G_0 is not a sparse graph, one can apply a standard transformation to reduce its degree to three; the resulting graph, however, has as many nodes as the original graph had edges. We thus prove Theorem 1.

If in addition G_0 and its subgraphs have size-$\tilde{O}(\sqrt{n})$ separators, and we have been provided with a decomposition tree for G_0 (or can compute one quickly in parallel), then we show how to construct such a $\overline{G}_0$ of size $\tilde{O}(|G_0|)$. Essentially we apply Lemma 10 to each splitting set of each vertex in the decomposition tree. It is easy to show that because shortest paths between nodes in the same splitting set are small, shortest paths between arbitrary nodes are only slightly larger. Of course, the construction for nodes of the separator must reflect paths between nodes not in the separator. We therefore use bottom-up processing of the decomposition tree to carry out these constructions.

Namely we work up the decomposition tree T of G, constructing a summary for each vertex v of T. The summary for one vertex is obtained by applying procedures COMPACT and SMALLPATH to the union of the summaries for the children of that vertex. We combine all these summaries, constructing a "substitute graph" $G'(v)$ in which distances are underestimates of distances in the pertinent graph $G(v)$ of vertex v and in which shortest paths exist that have only polylog size. For a vertex v of T, let $h(v)$ denote the height of v. The following lemma can be proved by induction on $h(v)$ using this idea.

Lemma 11 For a vertex v of the decomposition tree, we can compute a graph $\overline{G}(v)$ and a summary $H^*(v)$ with the following properties.

- E1: $V(G(v))$-distances in $\overline{G}(v)$ are $(1-\epsilon)^{2h(v)\log N}$-accurate underestimates of $V(G(v))$-distances in $G(v)$.
- E2: $S(v)$-distances in $H^*(v)$ equal $S(v)$-distances in $\overline{G}(v)$.
- E3: For any $a, b \in S(v)$, a shortest a-b path in $H^*(v)$ exists that has $O(\log^2 N)$ hops.
- E4: The number of supernodes in $|H^*(v)|$ is $O(|S(v)|)$.
- E5: For any two children v_i and v_j of v in the decomposition tree, any path in $\overline{G}(v)$ between $\overline{G}(v_1)$ and $\overline{G}(v_2)$ must contain a node of $S(v)$.
- E6: For any $a \in S(v)$ and $b \in V(G(v))$, if there is an a-b path in $\overline{G}(v)$, there is a shortest path of size $O(h(v)\log^3 N)$.

The time required is $O(\text{polylog}\, N)$, and the work is $O(\epsilon^{-4}|G(v)| \text{ polylog}\, N \log^2 L)$.

Recall the splitting property we described when we introduced decomposition trees. Using a simple top-down induction using property E5 of Lemma 11, one can show that the decomposition tree has the splitting property with respect to the graphs $\overline{G}(\cdot)$.

Let G_0 be the original graph. It follows from property E1 of Lemma 11 that $V(G_0)$-distances in the graph $\overline{G}_0 = \overline{G}(\text{root})$ are $(1-\epsilon)^{O(\log^2 N)}$-accurate underestimates of $V(G_0)$-distances in G_0. We show that there are small shortest paths in $\overline{G}_0$ between nodes of $V(G_0)$.

Lemma 12 For any nodes a, b of graph G_0, a shortest path in $\overline{G}_0$ exists that has $O(\log^4 N)$ edges.

Proof: Let v_a be the lowest vertex in the decomposition tree such that $G(v_a)$ contains a, and similarly for v_b. Let v be the lowest common ancestor of v_a and v_b in the decomposition tree. Let P be a shortest a-b path in $\overline{G}_0$. The splitting property for the graphs $\overline{G}(\cdot)$ implies that P goes through a node c of $S(v)$. By applying property E6 to the a-c subpath and the c-b subpath, we obtain the lemma. □

Acknowledgments

We benefited greatly from John Reif's encouragement and insights. We also are grateful to Andrea Pietracaprina for bringing a result of Leighton [20] to our attention. Thanks also to R. Ravi and Cliff Stein for comments on drafts of this paper, and to Gary Miller for his comments on an earlier version.

References

[1] R. Ahuja, K. Mehlhorn, J. Orlin, and R. E. Tarjan, "Faster algorithms for the shortest path problem," *Journal of the Association for Computing Machinery* 37 (1990), 213–223.

[2] N. Alon, P. Seymour, and R. Thomas, "A separator theorem for graphs with an excluded minor and its applications," *Proc. 22nd Annual ACM Symposium on Theory of Computing* (1990), 293–299.

[3] B. S. Baker, "Approximation algorithms for NP-complete problems on planar graphs," *Proc. 24th Annual IEEE Symposium on Foundations of Computer Science* (1983), 265–273.

[4] E. Cohen, "Parallel algorithms with improved work for shortest-paths from multiple sources," *Proc. 2nd Israel Symposium on Theory of Computing and Systems* (1993), 57–67.

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

[6] G.N. Frederickson, "Fast algorithms for shortest paths in planar graphs, with applications," *SIAM Journal on Computing* 16 (1987), 1004–1022.

[7] M. L. Fredman and D. E. Willard, "Trans-dichotomous algorithms for minimum spanning trees and shortest paths," *Proc. 31st Annual IEEE Symposium on Foundations of Computer Science* (1990), 719–725.

[8] M.L. Fredman and R.E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the Association for Computing Machinery* 34 (1987), 596–615.

[9] H. Gazit and G. L. Miller, "A parallel algorithm for finding a separator in planar graphs," *Proc. 28th Annual IEEE Symposium on Foundations of Computer Science* (1987), 238–248.

[10] S. Guattery and G. L. Miller, "A contraction procedure for planar directed graphs," *Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures* (1992), 431–441.

[11] R. Hassin, "Maximum flow in (s,t) planar networks," *Information Processing Letters* 13 (1981), 107.

[12] R. Hassin and D. B. Johnson, "An $O(n \log^2 n)$ algorithm for maximum flow in undirected planar networks," *SIAM Journal on Computing* 14 (1985), 612–624.

[13] D.B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *Journal of the Association for Computing Machinery* 24 (1977), 1–13.

[14] D.B. Johnson, "Parallel algorithms for minimum cuts and maximum flows in planar networks," *Journal of the Association for Computing Machinery* 34 (1987), 950–967.

[15] D.B. Johnson and S. M. Venkatesan, "Parallel algorithms for minimum cuts and maximum flows in planar networks," *Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science* (1982), 244–254.

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

[17] G. L. Miller, "Finding small simple cycle separators for 2-connected planar graphs," *Journal of Computer and System Sciences* 32 (1986), 265–279.

[18] G. L. Miller, S. Teng, and S. Vavasis, "A unified geometric approach to graph separators," *Proc. 31st Annual IEEE Symposium on Foundations of Computer Science* (1991), 538–547.

[19] J. Lagergren, "Efficient parallel algorithms for tree-decomposition and related problems," *Proc. 30th Annual IEEE Symp. on Foundations of Computer Science* (1990), 173–182.

[20] F.T. Leighton, *Complexity issues in VLSI: optimal layouts for the shuffle-exchange graph and other networks*, MIT Press, 1983.

[21] A. Lingas, "Fast parallel algorithms for planar directed graphs," *Proc. SIGAL International Symposium on Algorithms* (1990), 447–457.

[22] R.J. Lipton and R.E. Tarjan, "A separator theorem for planar graphs," *SIAM Journal of Applied Mathematics* 36 (1979), 177–189.

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

[24] V. Pan and J. H. Reif, "Fast and efficient solution of path algebra problems," *Journal of Computer and System Sciences* 38 (1989), 494–510.

[25] V. Pan and J. H. Reif, "The parallel computation of minimum cost paths in graphs by stream contraction," *Information Processing Letters* 40 (1991), 79–83.

[26] J. H. Reif, "Minimum s-t cut of a planar undirected network in $O(n \log^2 n)$ time," *SIAM Journal on Computing* 12 (1983), 71–81.

[27] J. D. Ullman and M. Yannakakis, "High-probability parallel transitive-closure algorithms," *SIAM Journal on Computing* 20 (1991), 100–125.