SIAM J. COMPUT.
Vol. 28, No. 1, pp. 263–277


# NEAR-LINEAR TIME CONSTRUCTION OF SPARSE NEIGHBORHOOD COVERS*

BARUCH AWERBUCH†, BONNIE BERGER‡, LENORE COWEN§, AND DAVID PELEG¶

**Abstract.** This paper introduces a near-linear time sequential algorithm for constructing a sparse neighborhood cover. This implies analogous improvements (from quadratic to near-linear time) for any problem whose solution relies on network decompositions, including small edge cuts in planar graphs, approximate shortest paths, and weight- and distance-preserving graph spanners. In particular, an $O(\log n)$ approximation to the $k$-shortest paths problem on an $n$-vertex, $E$-edge graph is obtained that runs in $\tilde{O}(n + E + k)$ time.

**Key words.** neighborhood covers, network decompositions, approximate shortest paths, spanners

**AMS subject classifications.** 05C85, 68R10, 05C65, 05C70

**PII.** S0097539794271898

## 1. Introduction.

**1.1. Background.** An *r-neighborhood* of a vertex in an undirected weighted graph is the set of vertices within distance $r$ away from it in the graph. An *r-neighborhood cover* is a set of overlapping sets of vertices in the graph (called *clusters*) with the property that every vertex has its entire $r$-neighborhood contained in one of these clusters. An $r$-neighborhood cover efficiently represents the local neighborhoods in a graph when all clusters in the cover have low diameter and the overlap among clusters (measured by the maximum number of clusters that intersect at a vertex) is minimized.

There is an inherent tradeoff between achieving low diameter and low overlap. If the clusters are chosen to be all $r$-neighborhoods (namely, the balls of radius $r$ around every vertex), then all clusters have diameter $r$, but each vertex can be in as many as $n$ clusters, where $n$ is the number of vertices in the network. On the other hand, the entire network can be considered a single cluster, in which case each vertex is in just one cluster, but the diameter of this cluster can be as large as $n$.

Sparse neighborhood covers were first introduced in [9] in the context of distributed computing. A general $r$-neighborhood cover construction with $O(r \log n)$ cluster diameter and $O(\log n)$ cluster overlap is presented in [9]. It is also shown

*Received by the editors July 15, 1994; accepted for publication (in revised form) March 2, 1997; published electronically June 15, 1998.

http://www.siam.org/journals/sicomp/28-1/27189.html

†Laboratory for Computer Science, MIT, Cambridge, MA 02139. Current address: The Johns Hopkins University, Baltimore, MD 21218 (baruch@blaze.cs.jhu.edu). This work was supported by Air Force contract AFOSR F49620-92-J-0125, NSF contract 9114440-CCR, DARPA contracts N00014-91-J-1698 and N00014-J-92-1799, and a special grant from IBM.

‡Department of Mathematics and Laboratory for Computer Science, MIT, Cambridge, MA 02139 (bab@theory.lcs.mit.edu). This research was supported in part by an NSF Postdoctoral Research Fellowship and an ONR grant provided to the Radcliffe Bunting Institute.

§Department of Mathematical Sciences and Department of Computer Science, The Johns Hopkins University, Baltimore, MD 21218. This research was supported in part by an NSF postdoctoral fellowship.

¶Department of Applied Mathematics and Computer Science, The Weizmann Institute, Rehovot 76100, Israel (peleg@wisdom.weizmann.ac.il). This research was supported in part by an Allon Fellowship, a Bantrell Fellowship, a Minerva Fellowship, and a Walter and Elise Haas Career Development Award.

therein how to achieve an essentially optimal tradeoff between the diameter and overlap parameters. However, the construction of [9] takes $O(nE)$ time and space, where $E$ is the number of edges of the graph.

**1.2. Summary of results.** In this paper, we present a new efficient construction method that achieves the optimal tradeoff in near-linear time. More specifically, the main result of this paper is a deterministic sequential algorithm for the construction of a sparse $r$-neighborhood cover with $O(r \log n)$ cluster diameter and $O(\log n)$ cluster overlap with running time and space $O(E \log n + n \log^2 n)$. The algorithm can also be applied for constructing an $r$-neighborhood cover with $O(\beta r)$ diameter and $O(\beta n^{1/\beta})$ overlap, in time and space $O((E\beta + n\beta^2)n^{2/\beta})$, where $\beta$ is given an input to the algorithm, representing the desired tradeoff between diameter and overlap.

To achieve this, we refine the "set coarsening" method of [9] and show how to construct the cover by only producing breadth-first search (BFS) trees from carefully selected vertices. The novel aspect of our approach is that a BFS tree is carved out in such a way that one can bound the overlap with other BFS trees. It turns out that for $r > 1$, bounding the overlap between BFS trees poses significant difficulties. In order to derive good complexity bounds, we need to charge the work performed in overlapping regions against the work performed in disjoint regions. Also, we need to lower-bound the fraction of $r$-neighborhoods that have "escaped carving" in a given iteration.

While most previous applications of neighborhood covers were in the distributed domain, the new algorithm makes the $r$-neighborhood cover a viable data structure for speeding up certain *sequential* algorithms. In particular, the results in this paper automatically imply analogous improvements (i.e., from quadratic to near-linear time) to many of the results in the literature that rely on neighborhood covers as a "black box." These include finding small edge cuts in planar graphs in $\tilde{O}(E)$ time and space[1] (as a consequence of using the algorithm of this paper in conjunction with the algorithm in [24], thus improving the $O(n^2)$ time and space complexity obtained by using the construction method of [9] in [24]), and weight- and distance-preserving graph spanners with $\tilde{O}(E)$ running time and space (see [5, 12], down from $O(nE)$ time using [11] or a combination of [5] and [9]).

Another application that is derived explicitly in the sequel is a sequential algorithm for approximating $k$-pairs shortest paths. We describe this application next.

**1.3. Approximating $k$-pairs shortest paths.** The *$k$-pairs shortest paths* problem, as discussed in the sequel, is defined as follows. Given an undirected graph $G(V, \mathcal{E})$ (where $|V| = n$ and $|\mathcal{E}| = E$), nonnegative edge weights, and $k$ pairs of vertices, find the length of the shortest path between each of these pairs.

Another version, henceforth referred to as the *explicit shortest paths (ESP)* problem, requires the algorithm to actually produce the paths. Clearly, the time complexity of the ESP version is lower-bounded by the length of the output, which is proportional to the total length of the requested paths.

The best known bounds for exact solution of the $k$-pairs shortest paths problem are based on one of two approaches: implementations of Dijkstra's algorithm and solutions based on matrix multiplication.

The first approach involves running Dijkstra's algorithm with a binary heap implementation of the priority queue, yielding time $O(kE \log n)$ [13], or with a Fibonacci heap implementation, yielding time $O(kn \log n + kE)$ [14]. For the all-pairs shortest

[1] We use the $\tilde{O}(R)$ notation to denote $O(R \log^{O(1)} R)$, for cleaner statement of bounds.

paths problem, the time bounds become $O(n^2 \log n + nE)$ and $O(nE \log n)$, respectively [13, 14].

There have been other improvements on the running time for the all-pairs shortest paths problem for some special case graphs [2, 10, 15, 16, 17, 18, 19, 22, 26], but they all have worst-case time $O(nE)$. Karger, Koller, and Phillips [19] show that $\Omega(nE)$ is a lower bound for directed graphs on the running time of any algorithm which is "path-comparison based," which is evidence that it is hard to improve exact algorithms with this approach.

The second approach is based on computing (exact) all-pairs shortest paths relying on matrix multiplication. Alon, Galil, and Margalit [3] describe an $O((nW)^{2.688})$ time algorithm for the case of integer edge weights whose absolute value is less than $W$. This result has been improved to give an algorithm for the unweighted case which runs in time $O(M(n) \log n)$, where $M(n)$ is the time for matrix multiplication (currently known to be $o(n^{2.376})$) [25]. Subsequently, this result has been further improved in [4], solving exact all-pairs shortest paths in the case of integer edge weights between 0 and $W$ in $O(W^2 M(n) \log n)$ time. Notice that the algorithm presented in this paper runs faster than these recent algorithms in both the unweighted and weighted cases.

Hence the bottlenecks in the running time of the exact algorithms stem either from repeated application of Dijkstra's single-source shortest paths algorithm or from matrix multiplication. We bypass these bottlenecks at the cost of producing an approximate solution.

There has also been recent work on approximation algorithms for shortest paths in the domain of parallel computation. Klein [20] extended the work of Ullman and Yannakakis [27] to the weighted case, obtaining a randomized PRAM algorithm that can $(1+\epsilon)$-approximate shortest paths between $k$ pairs of vertices in $O(\sqrt{n}\epsilon^{-2} \log n \cdot \log^* n)$ time using $(kE \log n)\epsilon^2$ processors (where $\epsilon$ is constant). When this algorithm is run sequentially, the time is not as good as known sequential algorithms for the exact problem. Hence, Klein's approximation algorithm is strictly of interest for parallel computation.

As a corollary of our near-linear cost construction of sparse neighborhood covers, we obtain an $O(\log n)$-approximation algorithm for this problem that runs in $\tilde{O}(E + k)$ time. Our improvement in running time is achieved through our ability to construct a data structure of $\delta = \lceil \log \text{Diam}(G) \rceil$ sparse neighborhood covers (where $\text{Diam}(G)$ is the diameter of the graph) in time $O((E \log n + n \log^2 n)\delta)$, thereby allowing us to then quickly retrieve short paths. (The previous construction of [9] has been insufficient to produce any improvement over existing algorithms for the shortest paths application.) Once the data structure is in place, we show how to query it for the approximate distance between two vertices in $O(\log n \log \delta)$ time. (For ESP, the explicit version of the problem, the cost increases (inherently) by the length of the path.) Thus we approximate $k$-pairs shortest paths in time $O((E \log n + n \log^2 n)\delta + k(\log n \log \delta))$ (plus the total length of the $k$ paths, for the ESP version).

More generally, a tradeoff can be achieved between the quality of the approximation and the running time; for example, a $32\beta$ approximation takes $O((E\beta + n\beta^2)n^{2/\beta}\delta + k\beta n^{1/\beta} \log \delta)$ time, for any $\beta \geq 1$. Note that we can achieve a better running time than known exact algorithms even for paths that are only a constant factor times the length of the shortest path. Further work in this direction is reported by Cohen [12].

We remark that there are no algorithms known for the single-pair shortest path problem that run asymptotically faster than the best single-source algorithms in the worst case [13]. Consequently, classical algorithms for $k$-pairs shortest paths have

done just as well using single-source shortest paths algorithms repeatedly.

**1.4. Structure of this paper.** The remainder of the paper is organized as follows. In section 2, we provide the graph-theoretic definition of neighborhood covers [9]. Section 3 presents an overview of the sparse neighborhood covers construction algorithm, followed by a formal presentation of the algorithm. The analysis of the algorithm is given in section 4. Finally, section 5 presents the application of the new algorithm to near-shortest path computation.

**2. Network covers.** Let $\mathrm{dist}(u,v)$ denote the minimum distance between $u$ and $v$ in the graph $G$. For a cluster of vertices $S_i \subseteq V$, let $\mathrm{dist}_{S_i}(u,v)$ denote the minimum distance from $u$ to $v$ in the subgraph induced by $S_i$. The *diameter* of the cluster is defined as $\mathrm{Diam}(S_i) = \max_{u,v \in S_i}(\mathrm{dist}_{S_i}(u,v))$.

The *$r$-neighborhood* of a vertex $v$ is defined as $N_r(v) = \{u \mid \mathrm{dist}\ (u,v) \leq r\}$.

A *sparse neighborhood cover* is defined as follows. A *$(\beta, r)$-neighborhood cover* is a collection of sets (also called *clusters*) of vertices $S_1, \ldots, S_l$, with the following properties.

1. For every vertex $v$ there exists some $1 \leq i \leq l$ s.t. $N_r(v) \subseteq S_i$.
2. For every $i$, $\mathrm{Diam}(S_i) \leq O(\beta r)$.

The *overlap* of the cover is the maximum number of clusters a vertex belongs to. A $(\beta, r)$-neighborhood cover is said to be *sparse* if its overlap is at most $\beta n^{1/\beta}$.

For the applications, $\beta$ can be used to control the tradeoffs between complexity and the quality of the approximation, and we typically set $\beta = \log n$. Then a sparse $(\log n, r)$-neighborhood cover is a collection of sets that contain all $r$-neighborhoods such that the diameter of the sets is bounded by $O(r \log n)$, and each vertex is contained in at most $O(\log n)$ sets.

We remark that the diameter/sparsity tradeoff in this definition is tight to within a constant factor. In particular, for $\beta = \log n$, there exist graphs for which any $(\log n, r)$-neighborhood cover places some vertex in at least $\Omega(\log n)$ sets [21].

A *sparse neighborhood covers data structure* is a family of sparse neighborhood covers for a fixed $\beta$ and different values of $r$. Applications typically require the construction of sparse $r$-neighborhood covers for $O(\log \mathrm{Diam}(G))$ successively doubled values of $r$ (namely, $r = 1, 2, 4, 8, \ldots$).

## 3. The cover construction algorithm.

**3.1. Overview.** Let us start with an overview of the algorithm. The algorithm builds the cover $\mathcal{X}$ in phases. Each phase produces a new subcollection $\mathcal{Y}$ of sets to be added to the cover. All sets added to the cover have low diameter, and each vertex will appear in at most one set per phase. In addition, an $\Omega(n^{-1/\beta})$ fraction of the vertices whose $r$-neighborhoods have so far remained uncovered by any set already in the cover, will have their $r$-neighborhoods placed in some set constructed in the current phase. Hence, in $\beta n^{1/\beta}$ phases the algorithm finishes covering the $r$-neighborhood of all vertices in the graph. Since each vertex appears in at most one set per phase, each vertex will appear in at most $\beta n^{1/\beta}$ sets in the output cover.

Fix a phase of the algorithm, and let $U$ be the collection of vertices whose $r$-neighborhood remains uncovered. Let $\mathcal{Y}$ denote the subcollection of sets we will place in the cover in the current phase.

We now sketch how to construct the sets $\mathcal{Y}$, in near-linear time. Low diameter sets $Y$ are constructed one at a time, in a breadth-first manner, and added to $\mathcal{Y}$. The key element is a new "guarded" BFS technique, which carefully controls how new sets are grown around existing sets in $\mathcal{Y}$. In particular, a two-layer "shield" is grown

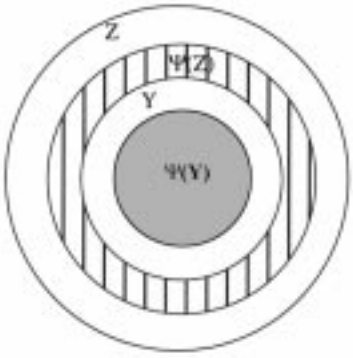


FIG. 1. *The layers of a grown cluster. The sets $Y$, $\Psi(Z)$, and $Z$ all grow in successive bands of width $r$ around $\Psi(Y)$.*

around each set $Y$. The shield serves a dual purpose: it keeps different sets in $\mathcal{Y}$ from overlapping, and it also insures that several sets do not duplicate work by performing BFS forays again and again into the same area of the graph.

Section 3.2 describes how the algorithm constructs a single set $Y$ in the cover, and section 3.3 shows how a new nonoverlapping set $Y$ grows around existing sets $Y$ in the collection $\mathcal{Y}$. The details of the algorithm, including the code for the procedures, are presented in section 3.4. A formal correctness proof and complexity analysis of the algorithm appear in section 4.

**3.2. Growth of a single set.** We show how to grow a single set $Y$ for the cover. Procedure `Cluster` grows $Y$ iteratively, in layers, and outer layers form a shield around $Y$. We keep track of four layers around an "internal kernel" called $\Psi(Y)$. The set $\Psi(Y)$ consists of those vertices whose $r$-neighborhood is fully subsumed in the set $Y$; $Z$ is the $2r$-neighborhood of $Y$; and $\Psi(Z)$ consists of those vertices whose $r$-neighborhood is subsumed by $Z$ (see Figure 1). Note that

$$\Psi(Y) \subset Y \subset Z \quad \text{and} \quad \Psi(Y) \subset \Psi(Z) \subset Z.$$

We will sometimes refer to the set $Z$ as the set $Y$'s associated *cluster*.

The set $Y$ starts growing as follows. Initially $\Psi(Y)$ is just the single vertex $v$, and $Y$ is its $r$-neighborhood. The sets $Y$, $\Psi(Z)$, and $Z$ all grow in successive bands of width $r$ around $\Psi(Y)$. Specifically, if the stopping condition of Procedure `Cluster` is not met, then at the next growth iteration, $Y$ grows to include the whole of the old $Z$, $\Psi(Y)$ grows to the old $\Psi(Z)$, the algorithm grows the BFS to construct a new $Z$ which contains the entire $2r$-neighborhood of the new $Y$, and the new $\Psi(Z)$ consists of those vertices whose $r$-neighborhood lies in the new $Z$. Note that $Y$, $\Psi(Z)$, and $Z$ in this picture are entire balls, not just rings; i.e., they contain all vertices within their borders. However, also note that it is not necessarily the case that $Y \subset \Psi(Z)$ (see section 3.3).

The stopping condition of Procedure `Cluster` is actually the conjunction of three separate conditions. The first stopping condition says that the number of still uncovered vertices in $Y$ must be an appropriately large (specifically, $\Omega(n^{-1/\beta})$) fraction of those in $Z$. The second condition says that the number of still uncovered vertices in $\Psi(Y)$, i.e., the interior of the set $Y$, must be a sufficiently large fraction of those in

$\Psi(Z)$. These two conditions help guarantee that the cover is sparse, as will be argued below. A third stopping condition makes sure that the BFS exploration of the outer $Z$ layer does not involve too much computation. Define $\lambda(v)$ to be the degree of vertex $v$ in the graph $G$. For a set of vertices $W$, let $\lambda(W) = \sum_{v \in W} \lambda(v)$. The final stopping condition says that $\lambda(Y)$ is a sufficiently large fraction times $\lambda(Z)$. We argue below that all three stopping conditions are guaranteed to be met simultaneously within $O(\beta)$ iterations of Procedure `Cluster`. Thus, the diameter of a $Y$ will be at most $O(r\beta)$.

**3.3. A subcollection of sets.** In this section, we show how a set $Y$ and its cluster grow around previously built nonoverlapping sets $Y$ from the same subcollection $\mathcal{Y}$ in the cover $\mathcal{X}$. A new cluster is always grown starting from a vertex $v$ which lies outside the shielding ball $\Psi(Z)$ of any previous $Y$. Notice that this insures that the entire $r$-neighborhood of $v$ lies outside any previous $Y$.

When a new cluster then grows into territory already occupied by a previous cluster, its BFS growth is curtailed by the previous cluster's layers. Each of the successive shields around an existing cluster permit a different degree of penetration.

- No cluster is allowed to grow into a previous cluster's $Y$. This ensures that the sets put in $\mathcal{Y}$ are disjoint. (See Figure 2.)
- A cluster's layer $\Psi(Z)$ does not grow into previous sets $\Psi(Z)$.
- The $Z$ level is entirely permeable. (We control the cost of repeated BFS forays into previous $Z$ instead by a stopping condition on the growth of a single cluster.)

We point out, to aid comprehension, that the relation $Y \subseteq \Psi(Z)$ may not hold for clusters grown after the first cluster (see Figure 3), since $\Psi(Z)$ is restricted to include new vertices if they have not already been placed in some previous cluster's associated set $\Psi(Z)$.

Figures 2–4 give snapshots of how new clusters grow. Figure 2 depicts how a new cluster begins growing, with $\Psi(Y)$ selected to be any single vertex outside the previous sets $\Psi(Z)$. Notice that this ensures that $Y$, the $r$-neighborhood of $\Psi(Y)$, will not overlap with previous $Y$'s. Hence previous $Y$'s form an impenetrable barrier (bold line) that nothing else can enter. In addition, the new cluster's $\Psi(Z) \setminus Y$ (striped region) does not enter previous sets $\Psi(Z)$.

Figure 3 illustrates the formation of the second layer of the new cluster (when the stopping condition was not met in the first iteration). Notice that the new $\Psi(Y)$ does not contain all of $Y$ from the previous iteration, since $\Psi(Y)$ is the old $\Psi(Z)$, which does not extend through previous clusters' $\Psi(Z)$ sets. Even though $Y$ and $Z$ do not extend into the territory of previous clusters' $Y$ sets, $Y$ contains the $r$-neighborhood of $\Psi(Y)$, and $Z$ contains the $r$-neighborhood of $\Psi(Z)$.

Finally, Figure 4 illustrates how a third cluster begins to grow. As before, the kernels $\Psi(Y)$ are shaded, the sets $Y$ are marked with a bold line, the $\Psi(Z)$ is striped, and $Z$ is within the outside ring. Excepting the participation of vertices in the outer $Z$ ring, every vertex participates in the BFSs at most twice: a vertex can lie in at most one set $Y$, and prior to being placed in $Y$, it could lie in at most one $\Psi(Z)$. The BFS performed in the $Z$ layer can be charged to the $Y$ cluster for which it was grown. When subcollection $\mathcal{Y}$ is complete, every vertex appears in a $\Psi(Z)$, for some cluster $Z$, and the stopping conditions ensure that a sufficiently large fraction of the vertices appear in $\Psi(Y)$ for some $Y$. Thus every vertex appears in at most one $Y$, and the same fraction of the vertices have their $r$-neighborhoods contained in $Y$ as well.

We stop growing clusters in the subcollection $\mathcal{Y}$ when every vertex is in $\Psi(Z)$ for

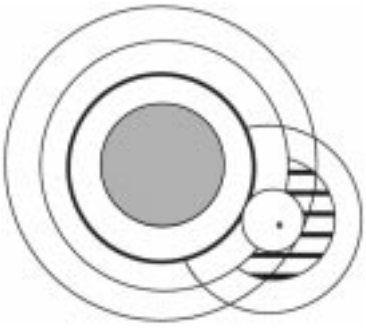

FIG. 2. *A new cluster begins growing around a single-vertex set* $\Psi(Y)$ *outside the previous sets* $\Psi(Z)$.

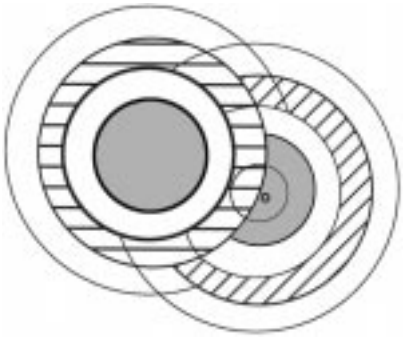

FIG. 3. *Formation of the second layer of the new cluster.*

some cluster $Z$.

**3.4. Details of the algorithm.** This subsection introduces Algorithm `All_Cover` that, given $\beta$ and $r$, constructs a sparse $(\beta, r)$-neighborhood cover. The algorithm is built from two intermediate subprocedures: Procedure `Cluster` (see Figure 5), which grows a single set to be placed in the cover, and Procedure `Cover`, which calls `Cluster` to produce a subcollection $\mathcal{Y}$ of the sets in the cover $\mathcal{X}$ (see Figure 6). Procedure `Cover` produces what was earlier called a single phase, in the overview section.

The input to the algorithm is a graph $G = (V, \mathcal{E})$ (where $|V| = n$ and $|\mathcal{E}| = E$) and integers $r, \beta \geq 1$. The output collection of cover clusters, $\mathcal{X}$, is initially empty. The algorithm maintains the set of "remaining" vertices $R$. These are the vertices whose neighborhoods are not yet subsumed by the constructed cover. Initially, $R = V$, and the algorithm terminates once $R = \emptyset$. The code for Algorithm `All_Cover` appears in Figure 7.

The algorithm operates in (at most $\beta n^{1/\beta}$) phases. Each phase consists of the activation of Procedure `Cover`$(R)$, which adds a subcollection of output clusters $\mathcal{Y}$ to $\mathcal{X}$ and removes the set of vertices $\Psi(R)$ whose $r$-neighborhood appears in some $Y \in \mathcal{Y}$ from $R$.

The collection of clusters $\mathcal{Y}$ returned by Procedure `Cover` are placed in the sparse neighborhood cover built by Algorithm `All_Cover`. The collection of clusters $\mathcal{Z}$ returned by Procedure `Cover` themselves form an *average degree cover* (see section 6).

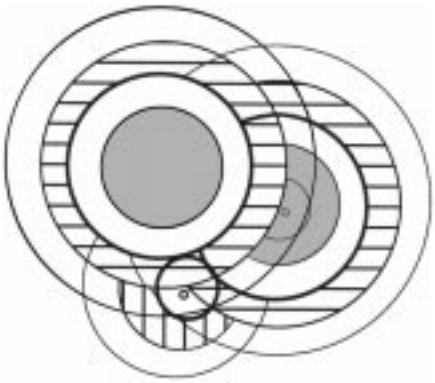

FIG. 4. *The growth of a third cluster. As before, the kernels $\Psi(Y)$ are shaded, the sets $Y$ are marked with a bold line, $\Psi(Z)$ is striped, and $Z$ is the outside ring.*

$\Psi(Z) \leftarrow \{v\}$
$Z \leftarrow \{N_r(v)\}$
**repeat**
  $\Psi(Y) \leftarrow \Psi(Z)$
  $Y \leftarrow Z$
  Perform a multiorigin BFS w.r.t. $Y$
    to depth $2r$ in $G(V \setminus \bigcup \mathcal{Y})$
  Add all vertices encountered to $Z$
  $\Psi(Z) \leftarrow \{v \mid v \in U \cap Z, \ \ \mathrm{dist}(v, Y) \leq r\}$
**until** $|Z| \leq n^{1/\beta} \cdot |Y|$
  **and** $|\Psi(Z)| \leq |R|^{1/\beta} \cdot |\Psi(Y)|$
  **and** $\lambda(Z) \leq n^{1/\beta} \cdot \lambda(Y)$
**return** $(\Psi(Y), Y, \Psi(Z), Z)$

FIG. 5. *Procedure* `Cluster`$(R, U, v, \mathcal{Y})$.

**4. Analysis.** In this section, we first analyze the properties of the procedures in section 3.4 and prove that Algorithm `All_Cover` constructs a sparse neighborhood cover. The complexity analysis then follows in section 4.2.

**4.1. Correctness of the algorithm.**

LEMMA 4.1. *A set $Y$ produced by Procedure* `Cluster` *has diameter at most $O(\beta r)$.*

*Proof.* Recall the three stopping conditions on the growth of a cluster. If the first stopping condition fails to be met, this means that $|Z| \geq n^{1/\beta} \cdot |Y|$. Since each time the stopping condition fails we increase $Y$ by a factor of $n^{1/\beta}$, this stopping condition can fail at most $\beta$ times.

If the second stopping condition fails to be met, this means that $|\Psi(Z)| \geq |R|^{1/\beta} \cdot |\Psi(Y)|$. Note that whenever Procedure `Cluster` is invoked within Procedure `Cover`, we have $U \subseteq R$. Since each time the second stopping condition fails we increase $\Psi(Y)$ by a factor of $|R|^{1/\beta}$, and $\Psi(Y)$ is restricted to $U$, this stopping condition can again fail at most $\beta$ times.

The third stopping condition says that $\lambda(Z)$ is less than or equal to $n^{1/\beta}$ times

$U \leftarrow R$
$\Psi(R) \leftarrow \emptyset$.
$\mathcal{Y}, \mathcal{Z} \leftarrow \emptyset$.
**while** $U \neq \emptyset$ **do**
  Select an arbitrary vertex $v \in U$
  $(\Psi(Y), Y, \Psi(Z), Z) \leftarrow$ `Cluster`$(R, U, v, \mathcal{Y})$
  $\Psi(R) \leftarrow \Psi(R) \cup \Psi(Y)$
  $U \leftarrow U \setminus \Psi(Z)$
  $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{Y\}$
  $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{Z\}$
**end-while**
**return** $(\Psi(R), \mathcal{Y}, \mathcal{Z})$

FIG. 6. *Procedure* `Cover`$(R)$.

$R \leftarrow V$
$\mathcal{X} \leftarrow \emptyset$
**repeat**
  $(\Psi(R), \mathcal{Y}, \mathcal{Z}) \leftarrow$ `Cover`$(R)$
  $\mathcal{X} \leftarrow \mathcal{X} \cup \mathcal{Y}$
  $R \leftarrow R \setminus \Psi(R)$
**until** $R = \emptyset$
**return** $\mathcal{X}$

FIG. 7. *Algorithm* `All_Cover`.

$\lambda(Y)$, and since the sum of the degrees of the entire graph is at most $E \leq n^2$, the third stopping condition can successively fail at most $2\beta$ times.

Therefore, the number of iterations of the growth process for which *any* of the stopping conditions fail is at worst $4\beta$, and so, for one of the first $4\beta + 1$ iterations, all three stopping conditions will hold, and the cluster will stop growing. Since each time we grow $Y$, we add an additional distance $r$ to the radius, the diameter of the sets is bounded by $O(\beta r)$. □

LEMMA 4.2. *The $r$-neighborhood of any vertex in a cluster's set $Y$ is contained in $\Psi(Z)$ for that cluster or some previous cluster's set $\Psi(Z)$.*

*Proof.* Let $v$ be a vertex in the $r$-neighborhood of $Y$. Then $v$ is certainly in the $2r$-neighborhood of $Y$, so Procedure `Cluster` places $v \in Z$ for that cluster. Procedure `Cluster` then makes $\Psi(Z) = \{v \mid v \in U \cap Z, \ \ \mathrm{dist}(v, Y) \leq r\}$, so if $v$ is not placed in $\Psi(Z)$, it means $v$ is not in $U$. Now we examine the set $U$ as it is modified over each invocation of Procedure `Cover`. The set $U$ initially contains the entire graph $G$, and Procedure `Cover` only deletes those vertices from $U$ which lie in some previous cluster's set $\Psi(Z)$. □

LEMMA 4.3. *For every $w \in G$, each invocation of Procedure* `Cover` *places $w$ into at most one set $Y$.*

*Proof.* The proof is by contradiction. Suppose $w$ was placed in $Y_1$ and $Y_2$, and without loss of generality, let $Y_1$ be constructed by `Cover` before $Y_2$. Note that since $w \in Y_1$, then $w \in \Psi(Z)$ for $Y_1$. When `Cover` constructs $Y_2$, it picks some center vertex $v$ and invokes Procedure `Cluster`. We consider three cases: either $w$ is $v$, or

$w$ is in the $r$-neighborhood of $v$, or $w$ is not in the $r$-neighborhood of $v$. First, $w$ cannot be $v$, since $w$ is in $\Psi(Y)$ for $Y_1$, $\Psi(Y) \subseteq \Psi(Z)$, and thus Procedure `Cover` has removed $w$ from the set $U$ of vertices from which it chooses centers $v$. In fact, since $v$ is outside $\Psi(Z)$ for all previous $Y$, its entire $r$-neighborhood cannot contain anything in a previous $Y$, by Lemma 4.2. Thus $w$ is not in the $r$-neighborhood of $v$, and so it will not be placed into $Y$ when `Cluster` is initialized. As `Cluster` grows cluster $Y_2$, by construction, it will only place vertices $w$ into $Y_2$ that it first places in the $Z$ layer. However, members of previous $Y$ are barred from entering into any subsequent $Z$ layer, by construction. □

LEMMA 4.4. *An $\Omega(|R|^{-1/\beta})$ fraction of the vertices whose $r$-neighborhood still remains to be covered (i.e., vertices in $R$) have their $r$-neighborhoods covered each time Algorithm* `All_Cover` *calls Procedure* `Cover`. *Thus Procedure* `Cover` *is invoked $O(\beta n^{1/\beta})$ times.*

*Proof.* The vertices that lie in $\Psi(Y)$ for some cluster $Y$ are the vertices that have their $r$-neighborhoods covered. The stopping condition guarantees that for each cluster, an $\Omega(|R|^{-1/\beta})$ fraction of the vertices in $\Psi(Z)$ (in $R$) lie in $\Psi(Y)$. Since every vertex in $R$ is in some $\Psi(Z)$ (by the termination condition for Procedure `Cover`), and since the $Y$ sets, and hence the $\Psi(Y)$ sets, are all disjoint (by Lemma 4.3), summing over all clusters produced in one invocation of `Cover` gives the result (as $|R| \leq n$). Hence, after $O(\beta n^{1/\beta})$ iterations of Procedure `Cover`, all vertices have their $r$-neighborhoods covered, and Algorithm `All_Cover` returns the desired cover. □

We can now complete the correctness proof.

THEOREM 4.5. *Given a graph $G = (V, \mathcal{E})$ (where $|V| = n, |\mathcal{E}| = E$) and integer parameters $\beta, r \geq 1$, Algorithm* `All_Cover` *constructs a sparse neighborhood cover.*

*Proof.* By Lemma 4.4, Algorithm `All_Cover` succeeds in covering all $r$-neighborhoods after $O(\beta n^{1/\beta})$ iterations of Procedure `Cover`. All clusters in the cover are constructed by Procedure `Cluster`, and hence have low diameter, by Lemma 4.1. Finally, by Lemma 4.3, each vertex appears in at most one set for each invocation of Procedure `Cover`, and by Lemma 4.4, Procedure `Cover` is invoked at most $O(\beta n^{1/\beta})$ times. Thus each vertex is in at most $O(\beta n^{1/\beta})$ sets, and hence the cover is sparse. □

**4.2. Complexity analysis.** We compute the cost of checking the stopping conditions and then performing the appropriate BFS explorations to construct $\mathcal{Y}$. The stopping conditions need to be checked at most $O(\beta)$ times for each cluster, since the conditions will be met within $O(\beta)$ iterations. To check the stopping conditions, we count vertices in $Z$, for all $Z \in \mathcal{Z}$. Recall that one of the stopping conditions insures that for all $Z$ the number of vertices in $Z$ is less than $O(n^{1/\beta})$ times the number of vertices in $Y$. Therefore, $O(\sum_{Z \in \mathcal{Z}} |Z|) = O(n^{1/\beta} \sum_{Y \in \mathcal{Y}} |Y|) = O(n^{1+1/\beta})$, since the sets $Y$ are disjoint. Thus, the cost of checking the stopping conditions to construct $\mathcal{Y}$ is $O(\beta n^{1+1/\beta})$.

Now we turn to estimating the cost of performing the BFSs. Ignore for the moment the cost of the BFS exploration of the final layer $Z \setminus \Psi(Z)$ for each cluster. Then for the construction of $\mathcal{Y}$, each vertex participates at most twice in BFSs: each edge is placed in at most one set $Y$, and prior to being placed in $Y$, it could lie in at most one $\Psi(Z)$. Thus, without the $Z$ layer, each edge has been examined at most twice in the construction of $\mathcal{Y}$.

Now consider the complexity of examining edges in $Z$. Notice that when $r = 1$, this is trivial, since every edge in $Z \setminus \Psi(Z)$ has an endpoint in $\Psi(Z)$, and every vertex is in a unique $\Psi(Z)$. For $r > 1$, bounding the amount of work done in $Z$ is controlled

by means of the third stopping condition on the growth of individual clusters. In particular, if we have done too much work on exploring the layer $Z \setminus Y$, we are then obligated to grow $Y$ to contain $Z$.

Recall that the final stopping condition on clusters ensures $\lambda(Z)$ is less than or equal to $O(n^{1/\beta})$ times $\lambda(Y)$. Let $Z_Y$ denote the set $Z$ associated with cluster $Y$. Then this gives

$$\sum_{Y \in \mathcal{Y}} |Z_Y| \leq \sum_{Y \in \mathcal{Y}} O(n^{1/\beta})\lambda(Y) \leq O(n^{1/\beta})\lambda(G) \leq O(n^{1/\beta}E).$$

Constructing $\beta n^{1/\beta}$ subcollections $\mathcal{Y}$ to complete the cover $\mathcal{X}$, we thus obtain the following theorem.

THEOREM 4.6. *Algorithm* `All_Cover` *produces a sparse* $(\beta, r)$*-neighborhood cover in time* $O((E\beta + n\beta^2)n^{2/\beta})$.

Setting $\beta = \log n$, we obtain the following corollary.

COROLLARY 4.7. *Algorithm* `All_Cover` *produces a sparse* $(\log n, r)$*-neighborhood cover in time* $O(E \log n + n \log^2 n)$.

**5. Application: Approximating *k*-pairs shortest paths.** In this section we present an approximation algorithm for the $k$-pairs shortest paths problem on undirected graphs with nonnegative edge weights. We exhibit a tradeoff between the quality of the approximation and running time. Specifically, we obtain a $32\beta$ approximation to the $k$-pairs shortest paths problem in time $O((E\beta + n\beta^2)n^{2/\beta}\delta + k\beta n^{1/\beta} \log\log n)$, for integer $\beta \geq 1$. This means that we can achieve a better running time than known exact algorithms even for paths that are only a constant factor times the length of the shortest path. Further work in this direction is reported in [12].

For $\beta = \log n$ we get an $O(\log n)$ times optimal approximation algorithm for the problem that runs in $\tilde{O}(E + k)$ time. Finding approximate shortest paths for *all* pairs will take $\tilde{O}\left(E + n^2\right) = \tilde{O}\left(n^2\right)$ time.

**5.1. Sparse tree covers.** The explanation of the algorithm is made simpler through the notion of *sparse tree covers*. We give these definitions here for unit edge weights; the extension to nonnegative edge weights is straightforward, as explained later.

DEFINITION 5.1. *For an undirected graph* $G(V, \mathcal{E})$ *and integers* $\beta, r \geq 1$*, a* $(\beta, r)$-tree cover *is a collection* $\mathcal{F}_{\beta,r}$ *of trees in* $G$*, that satisfies the following properties.*

1. *Every tree* $F \in \mathcal{F}_{\beta,r}$ *has depth* $8\beta r$ *or less.*
2. *For every two vertices* $u, v \in V$ *whose distance in* $G$ *is* $r$ *or less, there exists a common tree* $F \in \mathcal{F}_{\beta,r}$*, containing both.*

*A* $(\beta, r)$*-tree cover is said to be* sparse*, if each vertex is in at most* $\beta n^{1/\beta}$ *sets.*

Note that it is easy to modify Algorithm `All_Cover` so that it produces a sparse tree cover, not just a sparse neighborhood cover. In fact, Procedure `Cluster` actually constructs a BFS spanning tree for each cluster it builds, so the main change necessary is to include these trees in the output. The resulting $(\beta, r)$-tree cover will have the property that each tree in it is a BFS spanning tree of a set in the corresponding $(\beta, r)$-neighborhood cover. The time complexity remains $O((E\beta + n\beta^2)n^{2/\beta})$, which is $O(E \log n + n \log^2 n)$ when we set $\beta = \log n$.

**5.2. The algorithm.** We first give our algorithm for approximating $k$-pairs shortest paths with unit edge weights, and later explain how to extend this to the case when the edge weights are nonnegative.

**Preprocessing.** Let $\delta = \lceil \log \mathrm{Diam}(G) \rceil$. For every level $1 \leq i \leq \delta$, construct a sparse tree cover $\mathcal{F}_{\beta,2^i}$ for $G$, and number the trees in the cover. For every vertex $v$, and for every level $i$, store (in order of increasing tree ID) a list of all trees $F \in \mathcal{F}_{\beta,2^i}$ that contain $v$.

**Query response.** For any given pair of vertices $u, v$, for which a shortest path is sought, do the following.

Perform a binary search over the levels $1 \leq i \leq \delta$: for a given level $i$, if there exists a tree $F \in \mathcal{F}_{\beta,2^i}$ that contains both $u$ and $v$, then restrict the search to lower values of $i$; otherwise, restrict the search to higher values of $i$.

The binary search will produce a level $J$ such that $2^{J-1} \leq d(u,v) \leq 2^J 16\beta$, and a tree $F \in \mathcal{F}_{\beta,2^J}$ that contains both $u$ and $v$.

Return the value $16\beta 2^J$ as the approximate distance between $u$ and $v$. (For the ESP version of the problem, also return the unique path connecting a pair $u, v$ in the common tree $F$ as the approximate shortest path between $u$ and $v$.)

**The weighted case.** The algorithms for sparse neighborhood covers presented in section 3 can easily be modified to produce a *weighted* tree cover by forming shortest path trees rather than BFS trees. The algorithm presented in this section can in turn be trivially modified to handle weights. As we will see below, the running time of our algorithms remains asymptotically unchanged.

**5.3. Analysis.** First we wish to prove that our algorithm is an $O(\beta)$-approximation algorithm for the shortest paths problem.

LEMMA 5.2. *For each of the $k$ pairs of vertices given, our algorithm returns a path length between them within $32\beta$ times the length of the shortest path.*

*Proof.* We will argue this for a given pair $u, v$. Note that by Definition 5.1, if $\mathrm{dist}(u,v) \leq 2^i$, then the tree cover of level $i$ has a tree containing both $u$ and $v$. Since $J$ is the minimum level $i$ for which this is so, it must be that $2^{J-1} < \mathrm{dist}(u,v) \leq 2^J$. Also by Definition 5.1, the common tree has depth at most $8\beta \cdot 2^J < 16\beta \cdot \mathrm{dist}(u,v)$. That is, our algorithm returns the maximum length of the unique path connecting $u$ and $v$ in the common tree: $16\beta \cdot 2^J < 32\beta \cdot \mathrm{dist}(u,v)$. ∎

Now we address the running time of our algorithm.

LEMMA 5.3. *Our algorithm for approximating $k$-pairs shortest paths takes*

$$O((E\beta + n\beta^2)n^{2/\beta}\delta + k(\beta n^{1/\beta} \cdot \log \delta))$$

*time. For the ESP version, the complexity increases (inherently) by the total length of the paths.*

*Proof.* We have already noted that the algorithm for sparse neighborhood covers in this paper can produce the tree cover as it goes along at no additional payment in time. As we actually set up a data structure consisting of $\delta$ different tree covers, one for each level, the time bound for setting up the data structure is $O((E\beta + n\beta^2)n^{2/\beta}\delta)$ by Theorem 4.6.

As for the bound on a single query, the binary search for finding the right level requires checking $O(\log \delta)$ levels, and for a given level $i$, a common tree $F$ can be found in $\beta n^{1/\beta}$ time by searching the ordered lists of tree IDs to which the two vertices belong. (Recall that by the sparsity property, every vertex belongs to at most $\beta n^{1/\beta}$ different trees in the $i$th-level tree cover $\mathcal{F}_{\beta,2^i}$.)

For the *ESP* version, retrieving the unique path connecting a pair $u, v$ in the common tree $F$ takes time proportional to the length of that path. ∎

Lemmas 5.2 and 5.3 yield the following theorem.

THEOREM 5.4. *Our algorithm for approximating k-pairs shortest paths takes* $O((E\beta + n\beta^2)n^{2/\beta}\delta + k(\beta n^{1/\beta} \cdot \log \delta))$ *time, and returns a solution that is within* $O(\beta)$ *of the exact one.*

Setting $\beta = \log n$, we get the following corollary.

COROLLARY 5.5. *Our algorithm provides an* $O(\log n)$ *approximation to the k-pairs shortest paths problem in* $O((E \log n + n \log^2 n)\delta + k(\log n \cdot \log \delta))$ *time.*

For the *all*-pairs shortest paths problem we get the following corollary.

COROLLARY 5.6. *Our algorithm for approximating* all*-pairs shortest paths takes* $O((E\beta + n\beta^2)n^{2/\beta}\delta + n^2(\beta n^{1/\beta} \cdot \log \delta))$ *time, and returns a solution that is within* $O(\beta)$ *of the exact one.*

And again, setting $\beta = \log n$, we have the following corollary.

COROLLARY 5.7. *Our algorithm provides an* $O(\log n)$ *approximation to the all-pairs shortest paths problem in* $O((E \log n + n \log^2 n)\delta + n^2(\log n \cdot \log \delta))$ *time.*

**6. Discussion.** We close the paper by discussing two final points. First, the construction algorithm described in this paper can be adapted to work in a (synchronous or asynchronous) distributed network. The asynchronous implementation requires additional techniques in order to guarantee maintaining a bound of $\tilde{O}(1)$ messages per edge on the message complexity. Roughly speaking, a synchronous distributed variant of the cover construction algorithm needs to be bootstrapped concurrently with the synchronizer protocol of [8] in a mutually recursive fashion. This distributed implementation will be described in full elsewhere; concise description of the essential details can be found in [8, 7, 6]. Let us comment that this distributed algorithm implies similar improvements in message complexity to a variety of distributed applications that make use of sparse covers, including adaptive routing schemes and a distributed *from-scratch* BFS and network synchronizer construction.

Second, it is worth mentioning that there exists a somewhat weaker notion of neighborhood covers: a small *average degree cover* [23, 9], where the *average* cluster overlaps over all vertices is small. In contrast, our neighborhood cover algorithm produces a cover with small *maximum* cluster that overlaps over all vertices. The average degree cover problem is known to be significantly easier [23, 9]; in fact, the maximum degree cover algorithm in this paper can be thought of as constructed through a logarithmic number of iterations, each producing an average degree cover in the "remaining graph."

Average degree covers are not sufficient for an efficient solution to the $k$-pairs shortest paths problem, because the "imbalance" (of high and low overlaps) cannot be amortized over many queries. (Unlike the algorithms in this paper, other constructions of average degree covers [23, 12] cannot be run for multiple iterations to produce a small maximum degree cover.)

However, as pointed out in [1, 12], in the special case of the *all*-pairs shortest paths application, an average degree cover suffices and leads to a better tradeoff between running time and approximation; namely, the constant of approximation can be improved. Specifically, the approximation constant of $32\beta$ derived earlier can be improved in this case to $4\beta$ by running our sparse neighborhood cover algorithm and taking the $2r$-neighborhoods of the sets constructed in one iteration, thereby producing a sparse average degree cover, and by eliminating two of the stopping conditions of the algorithm. Cohen [12] observed that the extra stopping condition can be removed in this case, and she also introduced a randomized parallel algorithm for approximate all-pairs shortest paths. The randomized algorithm constructs *pairwise covers* along the lines of [21]. Pairwise covers satisfy the weaker condition that any

pair of points at distance $\leq r$ lie together in the same cluster (but 3 points in the same $r$-neighborhood may not all appear together in any one cluster).

**Acknowledgments.** We thank Yehuda Afek, Edith Cohen, Tom Leighton, and Moti Ricklin for stimulating discussions.

## REFERENCES

[1] Y. Afek and M. Ricklin, *Sparser: A paradigm for running distributed algorithms*, in Proc. 6th Workshop on Distributed Algorithms, Springer-Verlag, New York, 1992, pp. 1–10.

[2] R. K. Ahuja, K. Melhourn, J. B. Orlin, and R. E. Tarjan, *Faster Algorithms for the Shortest Path Problem*, Technical Report 193, MIT Operations Research Center, Cambridge, MA, 1988.

[3] N. Alon, Z. Galil, and O. Margalit, *On the exponent of the all-pairs shortest path problem*, in Proc. 32nd IEEE Symp. on Foundations of Computer Science, IEEE, Piscataway, NJ, 1991.

[4] N. Alon, Z. Galil, O. Margalit, and M. Naor, *Witnesses for boolean matrix multiplication and for shortest paths*, In Proc. 33rd IEEE Symp. on Foundations of Computer Science, IEEE, Piscataway, NJ, 1992.

[5] B. Awerbuch, A. Baratz, and D. Peleg, *Efficient Broadcast and Light-Weight Spanners*, Unpublished manuscript, 1991.

[6] B. Awerbuch, B. Patt-Shamir, D. Peleg, and M. Saks, *Adapting to asynchronous dynamic networks with polylogarithmic overhead*, in Proc. 24th ACM Symp. on Theory of Computing, ACM, New York, 1992, pp. 557–570.

[7] B. Awerbuch and D. Peleg, *Efficient Distributed Construction of Sparse Covers*, Technical Report CS90-17, The Weizmann Institute, Rehovot, Israel, 1990.

[8] B. Awerbuch and D. Peleg, *Network synchronization with polylogarithmic overhead*, in 31st IEEE Symp. on Foundations of Computer Science, IEEE, Piscataway, NJ, 1990, pp. 514–522.

[9] B. Awerbuch and D. Peleg, *Sparse partitions*, in 31st IEEE Symp. on Foundations of Computer Science, IEEE, Piscataway, NJ, 1990, pp. 503–513.

[10] P. A. Bloniarz, *A Shortest Path Algorithm with Expected Time* $o(n^2 \log n \log^* n)$, Technical Report 80-3, Dept. of Computer Science, State University of Albany, New York, 1980.

[11] B. Chandra, G. Das, G. Narasimhan, and J. Soares, *New sparseness results on graph spanners*, in Proc. 8th ACM Symp. on Computational Geometry, ACM, New York, 1992.

[12] E. Cohen, *Fast algorithms for constructing* $t$*-spanners and paths with stretch* $t$, in Proc. 34th IEEE Symp. on Foundations of Computer Science, IEEE, Piscataway, NJ, 1993, pp. 648–658.

[13] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press/McGraw-Hill, New York, 1990.

[14] M. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. Assoc. Comput. Mach., 34 (1987), pp. 596–615.

[15] A. M. Frieze and G. R. Grimmet, *The shortest-path problem for graphs with random arc-lengths*, Disc. Appl. Math., 10 (1985), pp. 57–77.

[16] H. Gabow, *Scaling algorithms for network problems*, J. Comput. System Sci., 31 (1985), pp. 148–168.

[17] H. Jakobsson, *Mixed-approach algorithms for transitive closure*, in Proc. 10th ACM Symp. on Principles of Database Systems, ACM, New York, 1991.

[18] D. Johnson, *Efficient algorithms for shortest paths in sparse networks*, J. Assoc. Comput. Mach., 24 (1977), pp. 1–13.

[19] D. R. Karger, D. Koller, and S. J. Phillips, *Finding the hidden path: Time bounds for all-pairs shortest paths*, in Proc. 32nd IEEE Symp. on Foundations of Computer Science, IEEE, Piscataway, NJ, 1991.

[20] P. N. Klein, *A Parallel Randomized Approximation Scheme for Shortest Paths*, Technical Report CS-91-56, Brown University, Providence, RI, 1991.

[21] N. Linial and M. Saks, *Decomposing graphs into regions of small diameter*, in Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms, SIAM, Philadelphia, 1991, pp. 320–330.

[22] C. C. McGeoch, *A New All-Pairs Shortest-Path Algorithm*, Technical Report 91-30, DIMACS, New Brunswick, NJ, 1991.

[23] D. Peleg, *Sparse Graph Partitions*, Technical Report CS89-01, The Weizmann Institute, Rehovot, Israel, 1989.

[24] S. Rao, *Finding small edge cuts in planar graphs*, in Proc. 24th ACM Symp. on Theory of Computing, ACM, New York, 1992, pp. 229–240.
[25] R. Seidel, *On the all-pairs-shortest-path problem*, in Proc. 24th ACM Symp. on Theory of Computing, ACM, New York, 1992, pp. 745–749.
[26] P. M. Spira, *A new algorithm for finding all shortest paths in a graph of positive arcs in average time* $O(n^2 \log^2 n)$, SIAM J. Comput., 2 (1973), pp. 28–32.
[27] J. D. Ullman and M. Yannakakis, *High-probability parallel transitive closure algorithms*, SIAM J. Comput., 20 (1991), pp. 100–125.