Fast algorithms for constructing t-spanners and paths with stretch t

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

Revised: December, 1994

Abstract

[1]The distance between two vertices in a weighted graph is the weight of a minimum-weight path between them (where the weight of a path is the sum of the weights of the edges in the path). A path has *stretch* t if its weight is at most t times the distance between its end points. We consider an undirected graph $G = (V, E)$ with nonnegative weights and present algorithms that compute paths with stretch $2 \le t \le \log n$. We present a $\tilde{O}((m + k)n^{(2+\epsilon)/t})$ time randomized algorithm that finds paths between k specified pairs of vertices and a $\tilde{O}((m+ns)n^{2(1+\log_n m+\epsilon)/t})$ deterministic algorithm that finds paths from s specified sources to all other vertices (for any fixed $\epsilon > 0$), where $n = |V|$ and $m = |E|$. This improves significantly over the slower $\tilde{O}(\min\{k, n\}m)$ exact shortest paths algorithms and a previous $\tilde{O}(mn^{64/t} + kn^{32/t})$ time algorithm by Awerbuch et al. A t-spanner of a graph G is a set of weighted edges on the vertices of G such that distances in the spanner are not smaller and within a factor of t from the corresponding distances in G. Previous work was concerned with bounding the size and efficiently constructing t-spanners. We construct t-spanners of size $\tilde{O}(n^{1+(2+\epsilon)/t})$ in $\tilde{O}(mn^{(2+\epsilon)/t})$ expected time (for any fixed $\epsilon > 0$), what constitutes a faster construction (by a factor of $n^{3+2/t}/m$) of sparser spanners than was previously attainable. We also provide efficient parallel constructions. Our algorithms are based on pairwise-covers and a novel approach to construct them efficiently.

[1]An extended abstract appeared in: Proc. 34th IEEE Annual Symposium on Foundations of Computer Science, 1993.

1. Introduction

The shortest-path problem amounts to finding paths of minimum weight between specified pairs of vertices on a weighted input graph. We consider the relaxed version of computing paths with *stretch* t, where the ratio between the weight of the path and the minimum-weight path connecting the same two vertices is at most t. We provide algorithms that compute paths of stretch t (typically $2 < t < \log n$) on weighted undirected graphs much more efficiently than known previously. A related concept are t-spanners of weighted graphs. A t-spanner is a weighted graph on the same set of vertices such that the distances between pairs of vertices in the spanner are not smaller and within a factor of t of the corresponding distances on the original graph. Numerous recent works were concerned with bounding the size of t-spanners, constructing them efficiently, and applying them to other problems (see e.g., Peleg and Ullman [11], Peleg and Schäffer [10], Althöfer et al. [1], and Chandra et al. [4]). We obtain significantly more efficient parallel and sequential constructions of sparse t-spanners along with improved upper bounds on their size. We note that our stretch-t paths algorithm essentially computes a sparse t-spanner and produces paths that consist of spanner edges. Our algorithms are favorable in scenarios where one needs to find short paths but has severe limitations on resources (e.g., time or number of processors) that prohibit the use of shortest-paths algorithms, or when there is a flat high cost associated with each edge that is being used for some short path, and hence, it is desirable to find a set of short paths that utilizes only a small subset of the edges (note that shortest-paths may require the use of all the edges). In addition, our algorithms are suited to find short paths on-line where we allow a small amount of preprocessing and supplied on-line with query pairs of vertices. Furthermore, they can be adapted to find short paths on-line on dynamic networks, where weighted edges are inserted or deleted, much faster than known shortest-paths algorithms.

We consider undirected graphs $G = (V, E)$ where V is the set of vertices and E the set of edges. Associated with the edges are positive weights $w : E \to R_+$. We denote $n = |V|$ and $m = |E|$. The weight of a path is the sum of the weights of the edges on the path. A *shortest path* between $\{u_1, u_2\} \subset V$ is a path of minimum weight among all paths connecting u_1 and u_2. The *distance* between $\{u_1, u_2\} \subset V$, denoted $\mathrm{dist}\{u_1, u_2\}$, is the weight of the respective shortest path. A path between $\{u_1, u_2\} \subset V$ has *stretch* t if the ratio between the weight of the path and the distance between $\{u_1, u_2\}$ is at most t. We use the elided asymptotic notation to eliminate polylogarithmic factors in n, e.g., $\tilde{O}(f) \equiv O(f \operatorname{polylog} n)$. The parallel algorithms and resource bounds discussed in this paper are for the EREW PRAM model.

1.1. Our results

We obtain the following results. Consider a weighted graph G, a set of query pairs $Q \subset V \times V$, and a stretch factor $2 < t = O(\log n)$. We present bounds for computing for all $(v, u) \in Q$, stretched distances $\tilde{\mathrm{dist}}\{v, u\}$ such that

$$\mathrm{dist}\{v, u\} \le \tilde{\mathrm{dist}}\{v, u\} \le t\,\mathrm{dist}\{v, u\} ,$$

with the following tradeoffs between the stretch t, the time, and the work performed.

- Deterministically:

i. When Q contains all the pairs consisting of one vertex from a set S of sources and any other vertex.
By a sequential algorithm, for any fixed $\epsilon > 0$ and t such that $t/((2+\epsilon)(1+\log_n m))$ is integral, in: (see Theorem 6.6)
$O(n^{(2+\epsilon)(1+\log_n m)/t}(m \log n + |S| n \log n))$ time.

- With probability $1 - O(1/\operatorname{poly}(n))$

ii. By a sequential algorithm, for any fixed $\epsilon > 0$ and t such that $t/(2+\epsilon)$ is integral, in: (see Section 6)
$O((m + |Q|)n^{(2+\epsilon)/t} t \log^2 n)$ time.

iii. For a fixed $0 \leq \delta \leq 1$, any fixed $\epsilon > 0$, and $\beta = O(\log n)$: (see Theorem 2.8)
$t = (2\beta)^{\lceil 1/\delta \rceil}(1+\epsilon)$ in:
$O(n^{\delta}\beta^2 \log n)$ time using $\tilde{O}(n^{1+2/\beta} + (m+|Q|)n^{1/\beta})$ work.

In an on-line problem, where we perform some preprocessing and the query pairs (or query sources) are given on-line, the algorithm in i needs $O(n^{(2+\epsilon)(1+\log_n m)/t} m \log n)$ preprocessing time and $O(n^{(2+\epsilon)(1+\log_n m)/t} n \log n)$ time per source, and the algorithm in ii needs $O(mn^{(2+\epsilon)/t} t \log^2 n)$ preprocessing time and $O(n^{(2+\epsilon)/t} t \log^2 n)$ time per query pair.

Graph spanners. We construct t-spanners of size (number of edges) $\tilde{O}(n^{1+(2+\epsilon)/t})$ (for any $\epsilon > 0$ and t such that $t/(2+\epsilon)$ is integral). These spanners can be constructed by a randomized algorithm that runs in $\tilde{O}(mn^{(2+\epsilon)/t})$ time. We present a deterministic algorithm that finds t-spanners of size $O(n^{1+(2+\epsilon)(1+\log_n m)/t})$ (for any $\epsilon > 0$ and t such that $t/((2+\epsilon)(1+\log_n m))$ is integral) and runs in $O(mn^{(2+\epsilon)(1+\log_n m)/t})$ time. We show that in parallel, t-spanners of size $O(n^{1+(2+\epsilon)/t})$ (for any $\epsilon > 0$ and t such that $t/(2+\epsilon/2)$ is integral) can be constructed in $O(R\beta^2 \log^2 n)$ expected time using $O(n^{1/\beta} m \beta \log^2 n)$ work, or alternatively, in $O(\beta \log^3 n)$ expected time using $O(n^3 \beta \log n)$ work, where $R = w_{\max}/w_{\min}$ and $\beta = t/(2+\epsilon/2)$.

1.2. Previous work

Peleg and Schäffer[10] proved the following bounds on the size of spanners of unweighted graphs. They showed that for infinitely many values of n, there exist n-vertex graphs such that all their t-spanners contain $\Omega(n^{1+1/t})$ edges. They also gave a polynomial algorithm that computes t-spanners with $O(n^{1+4/t})$ edges. For weighted graphs, Althöfer et al. [1] gave a simple greedy algorithm that constructs t-spanners of size $O(n^{1+2/(t-1)})$. In a subsequent paper, Chandra et al. [4] showed that the greedy algorithm of [1] has an $O(n^{3+4/(t-1)})$ time implementation. Hence, we tighten the known upper bound on the size of t-spanners and present a significantly more efficient sequential and parallel constructions than previously known. The key to achieving these improvements is the fact that all previous constructions of t-spanners were based on solving many instances of the exact shortest-paths problem, where in this paper we avoid that.

The fastest known sequential algorithm for computing exact shortest paths from a set of sources S to all other vertices requires $\tilde{O}(|S|m)$ time using weighted BFS (see, e.g. [7]). Awerbuch, Berger, Cowen, and Peleg [2] presented a faster algorithm for computing paths of stretch $t > 64$. They showed that paths with stretch $64 \leq t = O(\log n)$ between k pairs of vertices can be obtained in $\tilde{O}((mt+nt^2)n^{64/t}+ktn^{32/t})$ time. Hence, in particular, paths with stretch t from a set of sources S to all other vertices can be obtained in $\tilde{O}((mt+nt^2)n^{64/t} + |S|tn^{1+32/t})$ time. Note that when $t = O(\log n)$ the algorithm produces paths with stretch $O(\log n)$ and runs in time $\tilde{O}(m + |S|n)$.

The best known parallel shortest path algorithms generally perform significantly more work (product of time and number of processors) than their sequential counterparts. The best known work bound for an $\mathcal{NC}$ algorithm for the (weighted) single-source shortest-paths problem is $\tilde{O}(n^3)$ (the algorithm essentially solves the more general all-pairs problem). Other algorithms exhibit tradeoffs between work and time. Spencer [12] presented an algorithm for the single-source problem that runs in $\tilde{O}(\delta)$ time using $\tilde{O}(n^3/\delta^2+m)$ work, where the ratio of the largest edge-weight to the smallest edge-weight is polynomial in n. The author gave an algorithm that computes shortest paths from $|S|$ sources in $\tilde{O}(\delta)$ time using $O(|S|n^2+n^3/\delta^3)$ work [5]. The algorithm computes shortest paths if the ratio of the largest edge-weight to the smallest edge-weight is polynomial in n and otherwise can be adapted to find paths whose weights are within a factor of $(1 + 1/\operatorname{poly}(n))$ from the respective distances. In addition, the author showed that paths with stretch $(1 + \epsilon)$ (for a fixed ϵ) can be computed in $\tilde{O}(\delta)$ time using $\tilde{O}(|S|(n^2/\delta + m) + n^3/t^2)$ work. Another work-time tradeoff for the single-source shortest paths problem was given by Klein and Sairam [9] who presented a randomized algorithm that finds paths with stretch $(1+\epsilon)$ (for a fixed ϵ) in $\tilde{O}(n^{0.5})$ time using $\tilde{O}(mn^{0.5})$ work.

Note that the tradeoffs obtained by our sequential randomized algorithm (see ii) improve over the algorithm of Awerbuch et al. [2]. The tradeoffs obtained by our deterministic sequential algorithm (see i) improve over the algorithm of Awerbuch et al. for multi-source short-paths problems. The improvements reduce the time bound significantly and are particularly meaningful in the more interesting range where the stretch is small. As a concrete example consider computing paths between $O(m)$ arbitrary pairs of vertices. The best shortest-paths algorithm for this problem runs in time $\tilde{O}(mn)$ (essentially solves the all-pairs problem). Our randomized algorithm, in time $\tilde{O}(mn^{0.5})$ finds paths with stretch 4. The Awerbuch et al. algorithm, can not guarantee paths of stretch 4 and in time $\tilde{O}(mn^{0.5})$ computes paths with stretch 128. Implications of our parallel tradeoffs are that for any fixed $\epsilon > 0$, paths from s sources to all other vertices with a fixed constant stretch can be computed in $O(n^\epsilon)$ time using $O((m + sn)n^\epsilon)$ work and with a polylogarithmic stretch, in $O(n^\epsilon)$ time using $\tilde{O}(m + sn)$ work.

1.3. Overview

The main tool that we use to compute spanners and paths of stretch t are *pairwise covers* of graphs and efficient novel constructions of these covers. Pairwise covers are a modification of sparse neighborhood covers (see Awerbuch and Peleg [3]). Sparse neighborhoods covers were employed by Awerbuch et al. [2] to obtain a sequential algorithm for paths of a specified stretch. Pairwise covers turn out to be a more correct notion for computing paths of stretch t. Our constructions allow for significantly improved sequential bounds. In addition, we present parallel algorithms based on efficient parallel cover constructions. The construction of neighborhood covers in [2] seems to be inherently sequential.

A pairwise cover of a graph is defined with respect to parameters W and β. It consists of a collection of subsets of vertices (*clusters*) with the following properties: The sum of the sizes of the clusters is $\tilde{O}(n^{1+1/\beta})$, the maximal distance between any pair of vertices which are in the same cluster is at most $2\beta W$, and for every pair of vertices of distance at most W from each other there exists at least one cluster in the cover that contains both vertices. The value of β determines a tradeoff between the maximum diameter of a cluster (that we would like to be as small a multiple of W as possible), and the total size of the cover (that we would like to be as close to $O(n)$ as possible). For any $\beta \geq 2$ and a fixed $\epsilon > 0$, a logarithmic number of pairwise covers for different values of W can be used to compute paths with stretch $t = 2(1+\epsilon)\beta$. The work performed depends on the size of the cover. To achieve efficient parallel constructions of pairwise covers, we refine the definition to be with respect to an additional "path size" parameter ℓ. For a parameter $1 \leq \ell \leq n$, the ℓ-limited distance between a pair of vertices is the weight of the minimum-weight path connecting them among paths consisting of at most ℓ edges (this terminology is from [9]). In pairwise covers with parameter ℓ, we have a relaxed condition that pairs of vertices where the ℓ-limited distance between them is at most W must be both contained in at least one cluster. This relaxed requirement allows us to construct pairwise covers efficiently in parallel in time $\tilde{O}(\ell)$. The most technical and novel part of this paper is the construction of pairwise covers. It is also the only place where randomization is used.

We sketch our t-spanner constructions. For any $\beta \geq 2$ and a fixed $\epsilon > 0$, we construct a logarithmic number of pairwise covers for different values of W. Our construction of pairwise covers has the property that for every cluster, the algorithm computes a shortest path tree rooted at a vertex of the cluster such that distances from the root to all other vertices in the cluster are at most β. Hence, distances on the tree between any two vertices in the cluster are bounded by 2β. Roughly, for $t = 2(1+\epsilon)\beta$, the algorithm constructs a t-spanner by considering each cluster in each of the covers and augmenting the spanner with the edges of the respective shortest-paths tree. Our spanner contains a path of stretch t between every pair of vertices, We can obtain t-spanners with the additional property of size-2 paths if the property that the spanner is a subgraph of the original graph is compromised. This can be achieved if for each cluster, instead of adding the edges of the tree to the spanner, we add edges from the root to all other vertices in the cluster with appropriate weights. For every pair of vertices, our spanner contains a path of stretch t that uses at most 2 edges. We comment that the sparse neighborhood covers constructed by Awerbuch et al. [2] can similarly be used to obtain spanners but the resulting spanners have very large size compared with previous spanner algorithms.

The paper is organized as follows. In Section 2 we define pairwise covers, state the resource bounds for computing them, and employ them to compute stretch-t paths. In Section 3 we discuss the computation of a pairwise cover with respect to parameters W and ℓ. We reduce the problem to finding a cover with respect to a set of integral weights and parameters $\ell' = n$ and $W' = O(\ell)$. In Section 4 we discuss computing neighborhoods of vertices up to specified distances using work linear in the size of the neighborhood. Sections 5 and 6 contain the most novel part of the paper. In Section 5 we present a parallel algorithm for computing a pairwise cover in integral weighted graphs. The running time of the algorithm is linearly dependent on W and the work performed is roughly the sum over the clusters of the number of edges incident at cluster vertices. The neighborhoods of Section 4 are used in Section 5 as the clusters constituting a cover. In Section 6 we discuss sequential stretch-t paths algorithms. A randomized sequential algorithm is obtained as a simplified version of the parallel randomized algorithm introduced in previous sections. We note that for the sequential randomized algorithm Sections 2.3, 2.4, 3, and 4 may be skipped. We also present a deterministic algorithm that is based on deterministic construction of covers that is fairly different from the randomized construction used in Section 5. Section 7 is concerned with applications to graph spanners. In Section 8 we discuss computing stretch-t distances on dynamic networks, where edges are inserted and deleted. In Section 9 we discuss issues and open problems that arise from this work.

We remark that although the algorithms presented in this paper compute distances, a concise representation of paths with corresponding weights can be produced within the same resource bounds.

Notation. Consider an undirected graph $G = (V, E)$, with positive weights $w : E \to R_+$ associated with the edges. The *size* of a path is the number of edges in the path. For $\ell \in \mathcal{N}$, an ℓ-limited path is a path of size at most ℓ. We use the terms *shortest path* and *minimum-weight* path interchangeably for a path of minimum weight. For a subset of edges $E' \subseteq E$, we denote $w(E') = \sum_{e \in E'} w(e)$. For a subset of vertices $V' \subseteq V$, we denote by $\mathcal{E}(V') = \{(u_1, u_2) \in E | u_1 \in V' \wedge u_2 \in V'\}$ the edges in the subgraph induced by V'. For a subset of edges $E' \subseteq E$, two vertices $\{u_1, u_2\} \subseteq V$, an integer $1 \leq \ell \leq n$, and a weight function $\hat{w} : E' \to R_+$, we denote by $\mathrm{dist}^{\ell}_{E',\hat{w}}\{u_1, u_2\}$ the weight, according to $\hat{w}$, of the minimum weight path in E' between u_1 and u_2 that consists of at most ℓ edges. (When ℓ is omitted, presume $\ell = n$, and when E' is omitted, presume $E' = E$. Generally, when the weights $\hat{w}$ are clear from the context they are omitted from the subscript.) For a subset $V' \subseteq V$, a vertex $v \in V'$, an integer $1 \leq \ell \leq n$, and a scalar $W \in R_+$, denote by $N^{\ell}_W(V', v) \subset V'$ the set of all the vertices $u \in V'$ such that there exists a path from u to v of size at most ℓ and weight at most W in the subgraph induced by V'. (When $V' = V$, we omit the first parameter. If ℓ is omitted, presume $\ell = n$.) When we state that an algorithm terminates with probability of success $1 - O(1/\operatorname{poly}(n))$, it mean that we can replace $\operatorname{poly}(n)$ by a polynomial of any desired degree by adjusting constant factors in the resource bounds of the algorithm. We employ the argument that if algorithms with probability of success $1 - O(1/\operatorname{poly}(n))$ are called as subroutines a polynomial number of times, then the probability that all of them terminate successfully is $1 - O(1/\operatorname{poly}(n))$.

2. Using pairwise covers for stretched distances

In this section we define pairwise covers and employ them to compute stretch-t paths. We note that in the context of shortest path computations, the assumption that the weights are strictly positive does not limit the generality since (i) edges of zero weight can be contracted (the contractions can be performed well within the resource bounds of the algorithms presented here by using a connected components algorithm), and (ii) edges of negative weight induce negative cycles (by traversing the edge back and forth) that implies that all the distances between vertices that are in the same connected component as the negative weight edge are unbounded from below.

Remark 2.1 We make the following assumption throughout the paper and argue that it does not limit generality. We assume that the ratio of the largest to smallest edge weight in a graph is $O(\text{poly}(n))$. (That is, $\max_{e \in E} w(e)/\min_{e \in E} w(e) = O(\text{poly}(n))$.) We justify this assumption by using the following reduction of Klein and Sairam [9] (see also [5]) when the weights are general. Klein and Sairam had shown how to compute for a weighted input graph G a collection of weighted graphs with a total number of vertices $O(n \log n)$ and total size $O(m \log n)$ such that (i) in each of the graphs in the collection, the ratio of the largest to smallest edge weights is $O(\text{poly}(n))$ and (ii) a shortest path problem on the original graph can be translated efficiently to a shortest path problem on the graphs of the collection. A path in G obtained this way is guaranteed only to have stretch $(1 + 1/\text{poly}(n))$ but since in this paper we obtain paths of stretch no better than 2, we can safely allow that. The computation involved in obtaining this collection amounts to applying a connected components algorithm and is well within the resource bounds of the algorithms presented here.

This section is arranged as follows. In Subsection 2.1 we define pairwise covers. A pairwise cover of a graph is, roughly, a collection of subsets of vertices (clusters) defined with respect to a scalar $W \in R_+$, an integer ℓ (the *path size* of the cover), and a parameter β. The clusters are such that every pair of vertices with ℓ-limited distance of at most W must be contained in at least one cluster. The parameter β determines a tradeoff between the maximum diameter of a cluster and the size of the cover. In later sections we present algorithms for computing a pairwise cover. We shall see that the parallel running time depends linearly on the path size ℓ, and hence we would like to use covers with small path sizes. In Subsection 2.2 we present an algorithm that computes stretched ℓ-limited distances from a set of query vertices to all other vertices. The algorithm utilizes a logarithmic number of pairwise covers with path size ℓ but with different values for the parameter W. Recall that the time bound of our cover algorithm depends linearly on the path size ℓ, and hence we would like to avoid using the algorithm for large values of ℓ. Unfortunately, however, ℓ-limited distances for $\ell < n$ may not yield any information about the actual distances in the graph. To overcome this difficulty we seek a method to replace the current set of edges E by a set of weighted edges E' such that for some $\ell << n$, the ℓ-limited distances in E' are within some factor of the distances with respect to E. The following definition formalizes this concept:

Definition 2.2 Consider a graph $G = (V, E)$ with weights $w : E \to R_+$, and a set $E' \subset V \times V$ of edges with weights $w' : E' \to R_+$. We say that the weighted graph (V, E') (ℓ, t)-*approximates* G if for every pair of vertices $\{u_1, u_2\} \subset V$, the following inequalities hold:

$$\text{dist}_E\{u_1, u_2\} \le \text{dist}_{E'}\{u_1, u_2\} \le \text{dist}^{\ell}_{E'}\{u_1, u_2\} \le t\text{dist}_E\{u_1, u_2\} \ .$$

We refer to the parameter ℓ as the *path size* of the approximation and to the parameter t as the *stretch*. Note that since minimum weight paths are of size at most n, any graph $(n, 1)$-approximates itself. In Subsection 2.3 we introduce an algorithm that for an input graph G, produces a graph G' that (ℓ, t)-approximates G for some $\ell << n$. The algorithm utilizes pairwise covers with typical path size of n^{μ} (for small $\mu > 0$) and exhibits a tradeoff between the magnitude of μ and the stretch factor. In Subsection 2.4 we combine the algorithms of Subsections 2.2 and 2.3 to obtain stretched distances: Initially we apply the algorithm of Subsection 2.3 to the graph G and obtain an approximation G' of G. Subsequently, we apply the algorithm of Subsection 2.2 to G'.

2.1. Pairwise covers

Pairwise covers are a modification of sparse neighborhood covers that were introduced by Awerbuch and Peleg [3]. Sparse neighborhood covers were employed by Awerbuch et al. [2] for stretch-t path computations.

Definition 2.3 [**Pairwise cover**] Consider a graph $G = (V, E)$ with weights $w : E \to R_+$, integers $1 \le \ell \le n$ and $\beta = O(\log n)$, and scalars $w_r \in R_+$ and $0 \le \hat{\epsilon} < 1/2$. A *pairwise* $(\ell, \beta, w_r, \hat{\epsilon})$*-cover* (for brevity, $(\ell, \beta, w_r, \hat{\epsilon})$*-cover*) of G is a collection of sets of vertices $X_1, \ldots, X_k$ (*clusters*) and vertices $v_1, \ldots, v_k$ (where for $1 \le i \le k$, $v_i \in X_i$ is the *center* of the cluster X_i) such that

i. $\forall \{u, v\} \subset V$, such that $\text{dist}^{\ell}\{u, v\} \le w_r/(1 + \hat{\epsilon})$, $\exists i$ such that $\{u, v\} \subset X_i$,

ii. $\forall i$, $\forall u \in X_i$, $\text{dist}\{v_i, u\} \le \beta w_r$, and

iii. $\forall v \in V$, $|\{1 \le i \le k | v \in X_i\}| = O(n^{1/\beta}\beta \log n)$. (Each vertex belongs to $O(n^{1/\beta}\beta \log n)$ clusters).

In Sections 3–5 we present an algorithm that computes an $(\ell, \beta, w_r, \hat{\epsilon})$-cover within the following bounds:

Theorem 2.4 *An $(\ell, \beta, w_r, \hat{\epsilon})$-cover of G can be computed with probability $1 - O(1/\operatorname{poly}(n))$, in $O(\ell\hat{\epsilon}^{-1} \log n\beta^2)$ time using $O(n^{1/\beta} m\hat{\epsilon}(\log n)/(\ell\beta))$ processors. If we allow $O(\ell\hat{\epsilon}^{-1} n^{\alpha}\beta^2)$ time (for a fixed $\alpha > 0$) the cover algorithm uses $O(\beta n^{1/\beta} m \log n)$ work.*

Remark 2.5 We remark that a simpler sequential version of the algorithm produces an $(n, \beta, w_r, 0)$-cover in $O(mn^{1/\beta}\beta \log n)$ time. The sequential version is immediate after presenting the parallel algorithm, and hence, it is omitted.

The probabilistic behavior manifests itself either in not terminating within the stated resource bounds or by producing a set of clusters for which property iii of Definition 2.3 does not hold. Note that properties i and ii always hold and property iii can be easily verified. Hence, we can assume that if the algorithm terminates within the stated resource bounds it returns a valid cover. The algorithm can be modified to compute for each cluster a shortest-paths tree rooted at the center vertex. The tree distances from the center to other vertices in the cluster are at most βw_r.

2.2. Obtaining stretched limited distances

The inputs to following algorithm are a graph G with weights $w : E \to R_+$, integers $1 \leq \ell \leq n$ and $\beta \geq 1$, a scalar $0 \leq \hat{\epsilon} \leq 1/2$, and a set of query pairs $Q \subset V \times V$. The algorithm computes for every $\{v, u\} \in Q$, a stretched distance $\tilde{\text{dist}}\{v, u\}$ such that:

$$\text{dist}_E\{v,u\} \leq \tilde{\text{dist}}\{v,u\} \leq 2\beta(1+\hat{\epsilon})^2 \text{dist}_E^\ell\{v,u\} \ .$$

We remark that the algorithm employs pairwise covers in a similar manner to the use of sparse neighborhood covers made by the Awerbuch et al. [2] algorithm.

Algorithm 2.6 [Compute distances]

i. $w_{\min} \leftarrow \min_{e\in E} w(e)$
   $w_{\max} \leftarrow \max_{e\in E} w(e)$
   $r \leftarrow \lceil \log_{1+\hat{\epsilon}}(n w_{\max}/w_{\min}) \rceil$
   For $i = 0, \ldots, r$: $w_i \leftarrow w_{\min}(1+\hat{\epsilon})^i$

ii. For $i = 0, \ldots, r$ do (in parallel):

   (1) Compute $(\ell, \beta, w_i, \hat{\epsilon})$-cover χ_i
   (2) For each vertex $v \in V$, $x(i,v) \leftarrow \{X \in \chi_i | v \in X\}$.
   (The sets $x(i,v)$ are sorted lists of indexes of clusters.)

iii. For all $\{v,u\} \in Q$: $\tilde{\text{dist}}\{v,u\} \leftarrow 2\beta w_i$,
   where $i \leftarrow \min\{j | x(j,u) \wedge x(j,v) \neq \emptyset\}$.

Correctness. Consider a pair of vertices $\{v,u\} \in Q$. Let $i = \min\{j | x(j,u) \wedge x(j,v) \neq \emptyset\}$. It suffices to show that

i. $\text{dist}_E\{v,u\} \leq 2\beta w_i = \tilde{\text{dist}}_E\{v,u\}$ and

ii. $\tilde{\text{dist}}_E\{v,u\}/(2\beta(1+\hat{\epsilon})^2) = w_i/(1+\hat{\epsilon})^2 \leq \text{dist}_E^\ell\{v,u\}$.

Let $X \in \chi_i$ be such that $\{u, v\} \subset X$. (The existence of such an X is immediate from the definition of i.) The first inequality follows from property ii of a cover (see Definition 2.3). We prove that the second inequality holds. If $i = 0$, the proof is immediate since for any two vertices $\{u_1, u_2\}$, $\text{dist}^\ell_E\{u_1, u_2\} \geq \text{dist}_E\{u_1, u_2\} \geq w_0$. It follows from property i of a cover that if some j is such that $\text{dist}^\ell_E\{v, u\} \leq w_j/(1+\hat{\epsilon})$ then there exists $X \in \chi_j$ for which $\{v, u\} \subset X$, and hence, $x(j, u) \wedge x(j, v) \neq \emptyset$. For every $j < i$, $x(j, u) \wedge x(j, v) \neq \emptyset$. Therefore, $\text{dist}^\ell_E\{v, u\} > w_j/(1+\hat{\epsilon})$. Thus, taking $j = i-1$ we get $\text{dist}^\ell_E\{v, u\} > w_{i-1}/(1+\hat{\epsilon}) = w_i/(1+\hat{\epsilon})^2$.

Complexity. Recall (see Remark 2.1) that we assume $w_{\max}/w_{\min} = O(\log n)$ and therefore $r = O(\log n/\log(1+\hat{\epsilon})) = O(\hat{\epsilon}^{-1}\log n)$. Consider performing step ii for one value $0 \leq i \leq r$. It follows from Theorem 2.4 that with probability $1 - O(1/\text{poly}(n))$, the construction of the cover χ_i can be performed in $O(\ell\hat{\epsilon}^{-1}\log n\beta^2)$ time using $O(mn^{1/\beta}\hat{\epsilon}/(\ell\beta))$ processors. (Thus, using $O(mn^{1/\beta}\beta\log n)$ work.) Hence, with probability $(1-1/\text{poly}(n))^r = 1 - O(1/\text{poly}(n))$, the computation of all the covers $\chi_1, \ldots, \chi_r$ terminates within $O(\ell\hat{\epsilon}^{-1}\log n\beta^2)$ time and $O(\hat{\epsilon}^{-1}mn^{1/\beta}\beta\log^2 n)$ work. It follows from standard PRAM techniques that the construction of the sets $x(i, v)$ (for all $v \in V$), can be performed in $T = \Omega(\log n)$ time using $O(\sum_{X\in\chi_i}|X|(\log n)/T) = O(n^{1+1/\beta}\beta\log^2 n/T)$ processors, or when allowing $\Omega(n^\alpha)$ time for a fixed $\alpha > 0$, using $O(n^{1+1/\beta}\beta\log n)$ work. (Since property iii of Definition 2.3 implies that $\sum_{X\in\chi_i}|X| = O(n^{1+1/\beta}\beta\log n)$.) The computation of the stretched distances (Step iii) can be performed in $T = \Omega(r)$ time using $O(\sum_{\{v,u\}\in Q}\sum_i(|x(i,u)| + |x(i,v)|)/T) = O(\hat{\epsilon}^{-1}|Q|n^{1/\beta}\beta\log^2 n/T)$ processors (since property iii asserts that for all $v \in V$ and $1 \leq i \leq r$, $|x(i, v)| \leq n^{1/\beta}\beta\log n$). Hence, Algorithm 2.6 can be implemented to run in $O(\ell\hat{\epsilon}^{-1}\log n\beta^2)$ time using $O((m+|Q|)n^{1/\beta}\hat{\epsilon}^{-1}\beta\log^3 n)$ work and have probability of success $1 - O(1/\text{poly}(n))$. (or when allowing $\Omega(\ell\hat{\epsilon}^{-1}n^\alpha\beta^2)$ time using $O((m+|Q|)n^{1/\beta}\hat{\epsilon}^{-1}\beta\log^3 n)$ work.)

2.3. Obtaining approximations with small path size

In this subsection we present an algorithm that produces approximations (in the sense of Definition 2.2) with small path sizes for input graphs. The algorithm inputs a weighted graph $G = (V, E)$, with weights $w : E \to R_+$, integers $1 \leq \ell' \leq n$ and $\beta \geq 1$, and a scalar $0 \leq \hat{\epsilon} \leq 1/2$. The algorithm computes a set of edges $E' \subset V \times V$ with weights $w' : E \to R_+$ such that

i. $|E'| = O(\hat{\epsilon}^{-1}n^{1+1/\beta}\beta\log^2 n)$, and

ii. for every $\ell' \leq \ell \leq n$ and every pair of vertices $\{u_1, u_2\} \subset V$ the following holds:

$$\text{dist}_E\{u_1, u_2\} \leq \text{dist}_{E'}\{u_1, u_2\} \leq$$

$$\text{dist}^{2\lceil \ell/\ell' \rceil}_{E'}\{u_1, u_2\} \leq 2\beta(1+\hat{\epsilon})^2\text{dist}^\ell_E\{u_1, u_2\} .$$

Hence, the graph (V, E') is a $(2\lceil n/\ell' \rceil, 2\beta(1+\hat{\epsilon})^2)$-approximation of G. Furthermore, if $G = (V, E)$ is an (ℓ, t)-approximation of some graph $(V, \hat{E})$ for some parameters ℓ and t, then the graph (V, E') produced by the algorithm is a $(2\lceil \ell/\ell' \rceil, 2t\beta(1+\hat{\epsilon})^2)$-approximation of $(V, \hat{E})$. We first present the algorithm and analyze its resource bounds. Later on we study and analyze the resource bounds of repeated applications of the algorithm.

Algorithm 2.7 [Approximation with small path size]

i. $w_{\min} \leftarrow \min_{e \in E} w(e)$
$w_{\max} \leftarrow \max_{e \in E} w(e)$
$r \leftarrow \lceil \log_{1+\hat{\epsilon}}(n w_{\max}/w_{\min}) \rceil$
For $i = 0, \ldots, r$: $w_i \leftarrow w_{\min}(1+\hat{\epsilon})^i$
$E' \leftarrow \emptyset$

ii. For $i = 0, \ldots, r$ do (in parallel):

(1) Compute $(\ell', \beta, w_i, \hat{\epsilon})$-cover χ_i

(2) For each cluster $X \in \chi_i$ with center $v \in X$:
add to E' edges of weight $w_i\beta$ from v to all other vertices in X:
$E' \leftarrow E' \bigcup_{X \in \chi_i} \{(v, u) | u \in X\}$
$\forall u \in X, w'((v, u)) \leftarrow w_i\beta$

Correctness. We first bound the size of E'. Observe that when executing step ii for some value $0 \leq i \leq r$, at most $\sum_{X \in \chi_i} |X - 1|$ edges are produced for E'. Hence, it follows from property iii of Definition 2.3 and the assumption that $r = O(\hat{\epsilon}^{-1} \log n)$ (see Subsection 2.2) that when the algorithm terminates $|E'| = O(\hat{\epsilon}^{-1} n^{1+1/\beta} \beta \log^2 n)$.
It remains to show that distances in (V, E^*) comply with the stated inequalities. It follows from the construction of E' and property ii of Definition 2.3 that if $e = (u_1, u_2) \in E'$ then $w'(e) \geq \mathrm{dist}_E\{u_1, u_2\}$. Hence, for every pair of vertices $\{u_1, u_2\} \subset V$, $\mathrm{dist}_E\{u_1, u_2\} \leq \mathrm{dist}_{E'}\{u_1, u_2\}$. (Thus, the leftmost inequality holds.) We show that the rightmost inequality holds. Consider a pair of vertices $\{u_1, u_2\} \subset V$ and let $D = \mathrm{dist}_E^{\ell'}\{u_1, u_2\}$. We show that there exists a path p of size at most 2 between u_1 and u_2 in E' such that $w'(p) \leq 2\beta(1+\hat{\epsilon})^2 D$. Let $i = \min\{j | \exists X \in \chi_j \text{ s.t. } \{u_1, u_2\} \subset X\}$. We have $D > w_i/(1+\hat{\epsilon})^2$ (by an argument similar to the one given in the correctness analysis of Algorithm 2.6). Let $X \in \chi_i$ be such that $\{u_1, u_2\} \subset X$ and let v be the center of X. We assume (without loss of generality) that if $v \in \{u_1, u_2\}$ then $v = u_2$. By definition of E', the edges $e_1 = (v, u_1)$ and (if $u_2 \neq v$) $e_2 = (v, u_2)$ are contained in E' and are such that $w'(e_1) = w'(e_2) = \beta w_i$. Hence (if $u_2 \neq v$), the edges e_1 and e_2 constitute a path p of size 2 between u_1 and u_2 in E' of weight $w'(p) = 2\beta w_i$. If $u_2 = v$, then the edge $e_1 = (u_1, u_2) \in E'$ constitutes a path p of size 1 and weight $w'(e_1) = \beta w_i$ between u_1 and u_2. Since $D > w_i/(1+\hat{\epsilon})^2$, we conclude that $w'(p) \leq 2\beta(1+\hat{\epsilon})^2 D$.
Consider a path p of size at most ℓ in E. To conclude the correctness proof, it suffices to show that there exists a path p' of size at most $2\lceil \ell/\ell' \rceil$ in E' such that $w'(p') \leq 2\beta(1+\hat{\epsilon})^2 w(p)$. Treat p as a list of edges and consider a partition of p to $k \leq \lceil \ell/\ell' \rceil$ segments $p_1, \ldots, p_k$, each of size at most ℓ'. By the argument given above, for each segment p_i $(1 \leq i \leq k)$ there

is a corresponding path p_i' in E' of size at most 2 between the end vertices of p_i, such that $w'(p_i') \leq 2\beta(1+\hat{\epsilon})^2 w(p_i)$. Denote $p' = \bigcup_{i=1}^k p_i'$. Note that the size of p' is at most $2k \leq 2\lceil \ell/\ell' \rceil$ and that

$$w'(p') = \sum_{i=1}^{k} w'(p_i') \leq 2\beta(1+\hat{\epsilon})^2 \sum_{i=1}^{k} w(p_i) = w(p) \; .$$

Complexity. The complexity of the algorithm is dominated by the computation of the covers at step ii. The algorithm performs step ii in parallel for $r = O(\hat{\epsilon}^{-1} \log n)$ values of i. Hence, it follows from Theorem 2.4 that Algorithm 2.7 can be implemented to run with probability of success $1 - O(1/\operatorname{poly}(n))$ in $O(\ell' \hat{\epsilon}^{-1} \log n \beta^2)$ time using $O(\hat{\epsilon}^{-1} m n^{1/\beta} \beta \log^3 n)$ work. (or, alternatively, in $\Omega(\ell' \hat{\epsilon}^{-1} n^{\alpha} \beta^2)$ time using $O(\hat{\epsilon}^{-1} m n^{1/\beta} \beta \log^2 n)$ work.)

Repeated applications of Algorithm 2.7. Algorithm 2.7 can be applied repeatedly to obtain approximations of smaller path size but larger stretch. In each run, the algorithm is applied to the set of edges produced by the previous run. Suppose a graph (V, E) is an (ℓ, t)-approximation of some graph $(V, \hat{E})$. Consider k (for meaningful results we always have $k = o(\log n)$) repeated applications of Algorithm 2.7 starting with (V, E). Denote by E_i the set of edges computed by the ith run of the algorithm (where $E_0 \equiv E$). The graph (V, E_i) is the input for the $i+1$st run of the algorithm. Note that for $i > 0$, $|E_i| = O(\hat{\epsilon}^{-1} n^{1+1/\beta} \beta \log^2 n)$. Hence, the resulting resource bounds for k repeated applications of the algorithm are $O(k\ell' \hat{\epsilon}^{-1} \log n \beta^2)$ time using $O(\hat{\epsilon}^{-1} m n^{1/\beta} \beta \log^2 n + k \hat{\epsilon}^{-2} n^{1+2/\beta} \beta^2 \log^4 n)$ work. The set E_k produced by the kth run of the algorithm is such that (V, E_k) constitutes a $(2^k \lceil \ell/(\ell')^k \rceil, t(1+\hat{\epsilon})^{2k}(2\beta)^k)$-approximation of $(V, \hat{E})$. We typically use the above where $\hat{\epsilon}$ and k are fixed constants.
We remark that under some conditions we can get further improvements on the above bounds. Note that the work bound stated above may have an imbalance between the two terms. If $m >> n^{1+1/\beta}$ we can "afford" smaller values of β and within the same resource bounds achieve smaller stretch. Suppose we use for the parameter β the values β_1 and $\beta_2 = 1/(\log_n m - 1)$ alternatingly in repeated applications of the algorithm. In the first run, that is applied to the graph (V, E), we use β_1. We assume that k and $\hat{\epsilon}$ are fixed. It follows that the algorithm runs in $O(\ell' \log n(\beta_1^2 + \beta_2^2))$ time using $O(mn^{1/\beta_1} \log^4 n)$ work. The graph (V, E_k) is a $(2^k \lceil \ell/(\ell')^k \rceil, t 2^k \beta_1^{\lceil k/2 \rceil} \beta_2^{\lfloor k/2 \rfloor}(1+\hat{\epsilon})^{2k})$-approximation of $(V, \hat{E})$.

2.4. Computing stretched distances

Observe that any weighted graph is an $(n, 1)$-approximation of itself. We can obtain stretch-t paths in a graph (V, E) by (i) applying Algorithm 2.7 repeatedly to obtain an approximation (V, E') of (V, E) with small path size, and (ii) applying Algorithm 2.6 to (V, E').

Theorem 2.8 *For a weighted graph $G = (V, E)$, a set of query pairs $Q \subset V \times V$, a scalar $\beta \geq 1$, and fixed $0 \leq \epsilon \leq 1$ and $0 \leq \delta \leq 1$, we can obtain paths with stretch*

$$(2\beta)^{\lceil 1/\delta \rceil}(1+\epsilon)$$

between all pairs of vertices in Q in

$$O(n^\delta \log n \beta^2) \textit{ time using } O(n^{1+2/\beta}\beta^2 \log^4 n + (m + |Q|)n^{1/\beta}\beta \log^2 n) \textit{ work}$$

with probability of success $1 - O(1/\operatorname{poly}(n))$.

Proof: We choose $k = \lceil 1/\delta \rceil - 1$ and $\hat{\epsilon} < (1+\epsilon)^{(-2\lceil 1/\delta \rceil)} - 1$. We apply k repeated application of Algorithm 2.7 to (V, E) with $\ell' = 2n^\delta$. Denote by (V, E') the resulting graph (obtained with probability $1 - O(1/\operatorname{poly}(n))$. It follows (see Subsection 2.3) that (V, E') constitutes an approximation of (V, E) with path size $\lceil n^\delta \rceil$ and stretch

$$(2\beta(1 + \hat{\epsilon})^2)^k$$

The graph (V, E') is obtained in $O(n^\delta \log n \beta^2)$ time using $O(n^{1+2/\beta}\beta^2 \log^4 n + mn^{1/\beta}\beta \log^2 n)$ work. We apply Algorithm 2.6 to (V, E') with parameter $\ell = \lceil n^\delta \rceil$ and query set Q. We return the stretched distances found by Algorithm 2.6. It is easy to verify that these distances are within a factor of

$$(2\beta(1 + \hat{\epsilon})^2)^{\lceil 1/\delta \rceil} \le (2\beta)^{\lceil 1/\delta \rceil}(1 + \epsilon)$$

of the corresponding shortest distances in G. The resource bounds of the application of Algorithm 2.6 are $O(n^\delta \log n \beta^2)$ time using $O((|E'| + |Q|)n^{1/\beta}\beta \log^2 n)$ work. By combining the above we obtain the stretch and resource bounds claimed in the statement of the theorem. ∎

It follows that for any fixed $\epsilon > 0$, by choosing $\delta < \epsilon$ and $\beta > 2/\epsilon$, paths with constant stretch can be obtained in $O(n^\epsilon)$ time using $O(n^\epsilon(|Q| + n))$ work. In [5] the author presented an algorithm that for an input graph $G = (V, E)$ and parameter $\mu \ge 0.5$, produces a graph that is a $(6n^{1-\mu} \log^3 n, 1 + 1/\operatorname{poly}(n))$-approximation of G. The graph obtained has $O(m + n^{1+\mu} \log n)$ edges and the algorithm uses $O(n^{1+2\mu} \log n)$ work and runs in $\Omega(\log^2 n \log^* n)$ time, We utilize this to improve on the tradeoffs stated in Theorem 2.8. We obtain the following:

Theorem 2.9 *For* $0.5 \le \mu \le 1$, *and fixed constants* $0 \le \delta \le 1$ *and* $\epsilon > 0$, *we can obtain (with probability* $1 - O(1/\operatorname{poly}(n))$*) paths with stretch*

$$(2\beta)^{\lceil (1-\mu)/\delta \rceil}(1 + \epsilon)$$

between all pairs of vertices in a given set $Q \subset V \times V$ *in* $O(n^\delta \log n \beta^2)$ *time using*

$$O(n^{1+2\mu} \log n + n^{1+2/\beta}\beta^2 \log^4 n + (m + n^{1+\mu} \log n + |Q|)n^{1/\beta}\beta \log^2 n)$$

work.

Proof: We use a simple modification of the proof of Theorem 2.8 where we start with the graph G obtained by applying the algorithm of [5] with parameter μ. ∎

3. Computing an $(\ell, \beta, w_r, \hat{\epsilon})$-cover

In this section we reduce the problem of finding an $(\ell, \beta, w_r, \hat{\epsilon})$-cover in a graph $G = (V, E)$ with weights $w : E \rightarrow R_+$ to a problem of finding a $(\beta, 2\ell\hat{\epsilon}^{-1})$-cover in (V, E) with respect to some integral weights $\bar{w} : E \rightarrow \mathcal{N}$. We define (β, W)-cover of a weighted graph to be an $(n, \beta, W, 0)$-cover of the graph (see Definition 2.3). The weights $\bar{w}$ are obtained by scaling and rounding up the weights w. The reduction is based on a technique and ideas from Klein and Sairam [9]. The weights $\bar{w}$ are computed as follows:

Algorithm 3.1 [Generate integral weights $\bar{w}$]

i. $\pi \leftarrow 2\ell\hat{\epsilon}^{-1}/w_r$

ii. For all $e \in E$, let $\bar{w}(e) \leftarrow \lceil \pi w(e) \rceil$ $(e \in E)$

Complexity. Using standard PRAM techniques, Algorithm 3.1 can be performed in $O(\log n)$ time using $O(m)$ work.

The weights $\bar{w} : E \rightarrow \mathcal{N}$ have the following properties:

Proposition 3.2 *For every pair of vertices, $\{u_1, u_2\} \subset V$,*

i. $\pi \mathrm{dist}^{\ell}_{E,w}\{u_1, u_2\} \geq \mathrm{dist}_{E,\bar{w}}\{u_1, u_2\} - \ell$, *and*

ii. $\pi \mathrm{dist}_{E,w}\{u_1, u_2\} \leq \mathrm{dist}_{E,\bar{w}}\{u_1, u_2\}$.

Proof: Observe that for every path p of size $k = |p|$,

$$\bar{w}(p) - k \leq \pi w(p) \leq \bar{w}(p) \ .$$

Part ii is immediate from the right inequality. We apply the left inequality to prove part i. Let $\hat{p}$ be the path that minimizes $w(\hat{p})$ among all paths of size at most ℓ between u_1 and u_2. We have

$$\pi \mathrm{dist}^{\ell}_{E,w}\{u_1, u_2\} = \pi w(\hat{p}) \geq \bar{w}(\hat{p}) - |p| \geq \mathrm{dist}_{E,\bar{w}}\{u_1, u_2\} - \ell \ .$$

∎

Definition 3.3 [(β, W)**-covers of graphs**] For a given graph $G = (V, E)$ with integral weights $w : E \rightarrow \mathcal{N}$, a scalar $W \geq 1$ and an integer $\beta = O(\log n)$, a (β, W)-cover of G is a collection of sets of vertices $X_1, \ldots, X_k$ (clusters) and vertices $v_1, \ldots, v_k$ where $v_i \in X_i$ (centers) such that

i. $\forall \{u, v\} \subset V$ such that $\mathrm{dist}\{u, v\} \leq W$, $\exists i$ such that $\{u, v\} \subset X_i$,

ii. $\forall i$, for all $u \in X_i$, $\mathrm{dist}\{v_i, u\} \leq \beta W$, and

iii. for every $v \in V$, $|\{1 \le i \le k | v \in X_i\}| = O(n^{1/\beta}\beta \log n)$.

In Section 5 we present a randomized algorithm that computes a (β, W)-cover of an input graph G with integral edge-weights. The algorithm runs in $O(W\beta^2 \log n)$ time using $O(n^{1/\beta}m/(W\beta))$ processors, and computes a cover with probability $1 - O(1/\operatorname{poly}(n))$.

We discuss computing an $(\ell, \beta, w_r, \hat{\epsilon})$-cover in a graph $G = (V, E)$ with weights $w : E \to R_+$. Consider applying Algorithm 3.1 to obtain the weights $\bar{w} : E \to \mathcal{N}$. Define $W = \pi w_r = 2\ell\hat{\epsilon}^{-1}$. Let $X_1, \ldots, X_k$ with corresponding centers $v_1, \ldots, v_k$ be a (β, W)-cover of (V, E) with respect to the weights $\bar{w}$.

Proposition 3.4 *The sets $X_1, \ldots, X_k$ with corresponding centers $v_1, \ldots, v_k$ constitute an $(\ell, \beta, w_r, \hat{\epsilon})$-cover of G.*

Proof: We prove that condition i of Definition 2.3 holds for $X_1, \ldots, X_k$. Consider a pair of vertices $\{u_1, u_2\} \subset V$ such that $\operatorname{dist}^{\ell}_{E,w}\{u_1, u_2\} \le w_r/(1 + \hat{\epsilon})$. It follows from Proposition 3.2 that

$$\operatorname{dist}_{E,\bar{w}}\{u_1, u_2\} \le \pi \operatorname{dist}^{\ell}_{E,w}\{u_1, u_2\} + \ell \le W(1/(1 + \hat{\epsilon}) + \hat{\epsilon}/2) \le W \ .$$

Hence, from condition i of Definition 3.3, for some i, $\{u_1, u_2\} \subset X_i$. We show that condition ii of Definition 2.3 holds. Consider $1 \le i \le k$, and $u \in X_i$. It follows from Proposition 3.2 that $\pi \operatorname{dist}_{E,w}\{v_i, u\} \le \operatorname{dist}_{E,\bar{w}}\{v_i, u\}$. Condition ii of Definition 3.3 asserts that $\operatorname{dist}_{E,\bar{w}}\{v_i, u\} \le \beta W \le \pi\beta w_r$. Hence, $\operatorname{dist}_{E,w}\{v_i, u\} \le \beta w_r$ and thus condition ii holds. The validity of condition iii is immediate from condition iii of Definition 3.3. ∎

Proposition 3.4 asserts that an $(\ell, \beta, w_r, \hat{\epsilon})$-cover of G with weights w can be computed within the same resource bounds as a $(\beta, 2\ell\hat{\epsilon}^{-1})$-cover of G with weights $\bar{w}$. The proof of Theorem 2.4 follows using the bounds for computing a (β, W)-cover stated above.

4. Computing neighborhoods

In this section we present a parallel algorithm for computing neighborhoods of vertices in a graph $G = (V, E)$ with integral weights $w : E \to \mathcal{N}$. The k-neighborhood of a vertex v, $N_k(v)$ is the set of vertices of distance at most k from v. The time depends linearly on k (the radius of the neighborhood) and the work performed is linear in $\mathcal{E}(N_k(v))$, the number of edges between vertices in the neighborhood.

We assume that the input is such that at each vertex, the incident edges are sorted into a list of "buckets" according to increasing weights. (Note that this can be achieved in $O(\log^2 n)$ time and either $O(m \log n)$ work using comparison based sorting or $O(m)$ work, since weights are integral and for the purposes of this paper it suffices to consider weights that are of size at most $W = O(\hat{\epsilon}^{-1}n)$.) We prove the following.

Proposition 4.1 *For a set of integers $k_1, \ldots, k_r$, and a set of vertices $s_1, \ldots, s_r$, the computation of $N_{k_i}(s_i)$ for $i = 1 \ldots, r$ can be performed using p processors in time*

$$O((\max_i k_i \log n) + \sum_{i=1}^{r} |\mathcal{E}(N_{k_i}(s_i))|(\log n)/p) \ .$$

Or, alternatively, for any fixed $\alpha > 0$, in time $\Omega(\max_i k_i n^\alpha)$ and work $O(\sum_{i=1}^{r} |\mathcal{E}(N_{k_i}(s_i))|)$.

The algorithm essentially amounts to performing, in parallel for s_i ($1 \leq i \leq r$), a weighted parallel BFS computation from s_i while considering only paths of weight at most k_i.

We present a weighted parallel BFS algorithm from a vertex s in G. The algorithm is a straightforward generalization of a parallelization of the standard sequential BFS algorithm used by Ullman and Yannakakis [13]. A weighted version of the parallel BFS algorithm was also used by Klein and Sairam [9].

The algorithm consists of iterations, where in the ith iteration ($i \geq 0$), the algorithm computes the set $\text{level}_i(s) \subset V$ of all vertices of distance i from s. The algorithm terminates when a stopping condition is met.

Algorithm 4.2 [Weighted Parallel BFS from s]

i. $\text{level}_0(s) \leftarrow \{s\}$
   $F = \{s\}$
   $i \leftarrow 0$

ii. **Repeat:**
   $i \leftarrow i + 1$
   $\text{level}_i(s) \leftarrow \{v \in V \setminus F | \exists j < i \exists u \in \text{level}_j(s) \text{ s.t. } (v, u) \in E \wedge w(v, u) = i - j\}$
   $F = F \cup \text{level}_i(s)$
   Until: stopping condition is satisfied

Correctness. It is easy to verify that after the ith iteration,

$$\text{level}_i(s) = \{u \in V | \text{dist}_E\{s, u\} = i\} \ .$$

Complexity. We outline an EREW PRAM implementation of the algorithm and analyze the resulting resource bounds.

We assume that at each vertex $u \in V$ there is a list of "buckets" containing a partition of the vertices adjacent to u. The partitioning of adjacent vertices to buckets is done by sorting the edges incident at u. Adjacent vertices are placed in buckets according to the weight of the edge to the vertex u. The buckets are maintained in a list, according to increasing weights, such that each bucket has a pointer to the nonempty bucket of next larger weight. For a vertex u and integer j, we denote by $B(u, j)$ the bucket of adjacent vertices to u through edges of weight ℓ. The algorithm also maintains a partition of the vertices in F as a list of sets A_j $(j > 0)$. The partition is modified in the course of the algorithm. At the termination of iteration i, a vertex $u \in \text{level}_j(s)$ (for $j < i$) is placed in the set A_{c_u+j}, where c_u is the smallest weight of an edge incident at u such that $c_u + j > i$. Note that at iteration i it suffices to examine vertices in A_{i+1} to determine the $i+1$st level.

We specify the actions of the algorithm in the ith iteration. The algorithm examines (in parallel) the vertices in A_i. For each $u \in A_i \cap \text{level}_j(s)$, the algorithm scans the vertices in $B(u, i-j)$. For each $u' \in B(u, i-j)$ the algorithm does as follows. If $u' \notin F$, it is placed in $\text{level}_i(s)$. The vertex u' is placed in a set $A_{i'}$, where $i' = i + \min\{j' | B(u', j') \neq \emptyset\}$. The vertex u is replaced in a different set $A_{i''}$ $(i'' > i)$, according to the weight of the next nonempty bucket on its bucket list $i'' = i + \min\{j' > i - j | B(u, j') \neq \emptyset\}$. It follows from standard EREW PRAM techniques that the ith iteration can be performed in $O(\log n)$ time using $\sum_{j<i} \sum_{u \in \text{level}_j(s)} |B(u, i-j)| \log n$ work. Or alternatively, in $\Omega(n^\alpha)$ time using $\sum_{j<i} \sum_{u \in \text{level}_j(s)} |B(u, i-j)|$ work.

Consider performing instances of the above computation in parallel for a set of sources $s_1, \ldots, s_r$, where the computation for a source s_i is done for up to $k_i \leq \ell$ $(1 \leq i \leq r)$ iterations. The ith iterations (for $0 \leq i \leq \ell$) of all instances are performed in parallel. Hence, with p available processors, the ith iterations of all instances can be performed in

$$O(\log n + \sum_{h=1}^{r} \sum_{j<i} \sum_{u \in \text{level}_j(s_h)} |B(u, i-j)|(\log n)/p)$$

time or alternatively in

$$O(n^\alpha + \sum_{h=1}^{r} \sum_{j<i} \sum_{u \in \text{level}_j(s_h)} |B(u, i-j)|/p)$$

time. For $1 \leq j \leq r$, denote by F_j the set F generated by the instance of the algorithm for the source s_j. Note that for all $1 \leq h \leq r$,

$$\sum_{i \leq r} \sum_{j<i} \sum_{u \in \text{level}_j(s_h)} |B(u, i-j)| = 2|\mathcal{E}(F_h)| \ .$$

It follows that ℓ iteration of Algorithm 4.2 can be performed in time

$$O(\ell \log n + \sum_{s \in S} |\mathcal{E}(F_s)|(\log n)/p) \ .$$

(or alternatively in time $O(\ell n^\alpha)$ and work $O(\sum_{s \in S} |\mathcal{E}(F_s)|/p)$.) This concludes the proof of Proposition 4.1.

Remark 4.3 [Computing neighborhoods sequentially] Consider a graph with weights $w : E \to R_+$, a scalar w_r, and a vertex s. The goal is to compute $N_{kw_r}(s)$ where k is the first j such that $N_{jw_r}(s)$ satisfies some desired easy-to-check property. Assume that G is represented in a way that the edges incident at each vertex can be accessed in order according to increasing weight. A simple modification of the sequential BFS algorithm can compute $N_{kw_r}(s)$ in $|\mathcal{E}(N_{k_i w_r}(s_i))|$ time. The algorithm uses $O(k+n)$ additional storage for maintaining a partition of the vertices into sets A_j $(1 \leq j \leq k)$. The latter partition is used in a similar fashion to the partition maintained by the parallel algorithm stated above.

5. Computing a (β, W)-cover

In this section we present and analyze an algorithm that computes a (β, W)-cover χ for an input graph $G = (V, E)$ with integral weights $w : E \rightarrow \mathcal{N}$.

Definition 5.1 For $V' \subset V$, $v \in V'$, and an integer $k \geq 0$, define

i. $\text{core}(v, k, V') = N_{kW}(V', v)$, and

ii. $\text{cluster}(v, k, V') = N_{(k+1)W}(V', v)$.

The following proposition states some easy to verify properties.

Proposition 5.2 *i. For $V' \subset V$, $k \in \mathcal{N}$, and $\{v_1, v_2\} \subset V'$: $v_1 \in \text{core}(v_2, k, V')$ if and only if $v_2 \in \text{core}(v_1, k, V')$. Similarly, $v_1 \in \text{cluster}(v_2, k, V')$ if and only if $v_2 \in \text{cluster}(v_1, k, V')$.*

ii. For $V' \subset V$, $k \in \mathcal{N}$, and $u \in \text{cluster}(v, k, V')$, $\text{dist}_E\{v, u\} \leq \text{dist}_{\mathcal{E}(V')}\{v, u\} \leq (k+1)W$.

iii. For integers $k_2 < k_1$, subsets $V_2 \subseteq V_1 \subseteq V$, and $v \in V_2$:

$$\text{core}(v, k_2, V_2) \subseteq \text{cluster}(v, k_2, V_2) \subseteq \text{core}(v, k_1, V_1) \subseteq \text{cluster}(v, k_1, V_1) \ .$$

Algorithm 5.3 [Compute a (β, W)-cover χ]

i. $V_1 \leftarrow V$, $\chi \leftarrow \emptyset$

ii. For $i = 1, \ldots, \beta$:

(1) Choose uniformly at random a set $S_i \subset V_i$ of size $\lceil C n^{i/\beta} |V_i| (\log n)/n \rceil$
($C \geq 1$ is some constant, where the value of C determines the probability that the algorithm terminates correctly.)
If $|V_i| < \lceil C n^{i/\beta} |V_i| (\log n)/n \rceil$ choose $S_i \leftarrow V_i$
Let $k_i \leftarrow \beta - i$

(2) Perform, in parallel for $v \in S_i$, a computation of $\text{core}(v, k_i, V_i)$, $\text{cluster}(v, k_i, V_i)$. (Use the algorithm of Section 4.)

(3) $\chi \leftarrow \chi \cup \{\text{cluster}(v, k_i, V_i) | v \in S_i\}$

(4) $V_{i+1} \leftarrow V_i \setminus \bigcup_{v \in S_i} \text{core}(v, k_i, V_i)$
If $V_{i+1} = \emptyset$, stop.

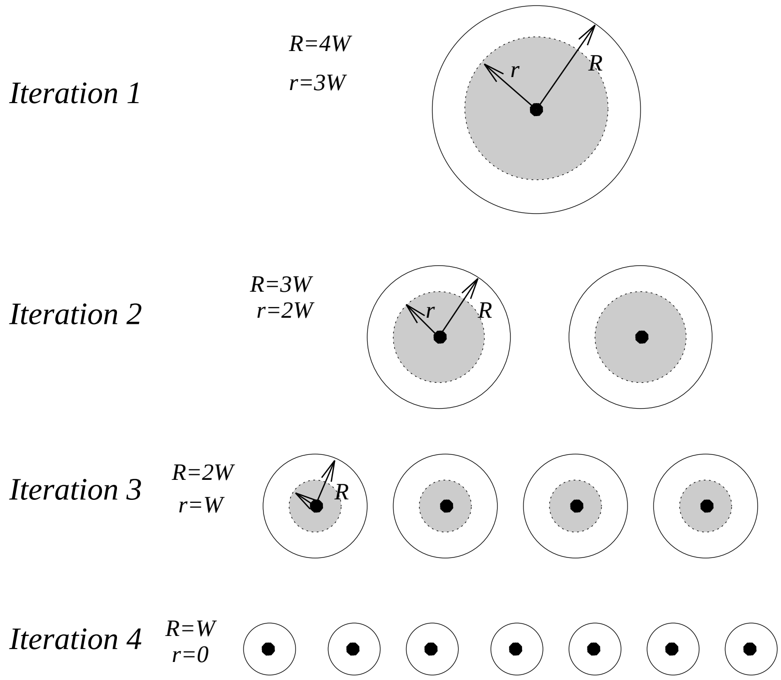

Figure 1: Clusters produced by the algorithm with $\beta = 4$

Figure 1 illustrates the clusters produced by the algorithm. In the first iteration, we select a small number of neighborhoods with a large radius. In following iterations, we select larger number of neighborhoods with smaller radiuses. The algorithm is such that with high probability, these neighborhoods do not contain many vertices.

Note that the condition $|V_i| < \lceil Cn^{i/\beta}|V_i|(\log n)/n\rceil$ must occur for some i. When $|V_i| < \lceil Cn^{i/\beta}|V_i|(\log n)/n\rceil$, the current iteration is last. Since $S_i = V_i$, for all $v \in V_i$, $v \in \mathrm{core}(v, k_i, V_i)$ and hence $V_{i+1} = \emptyset$. Therefore, we have $V \equiv V_1 \supset V_2 \supset \cdots \supset V_t = \emptyset$, where $i = t-1$ in the last iteration.

In the remaining part of this section we prove the following:

Theorem 5.4 *With probability* $1 - O(1/\operatorname{poly}(n))$, *Algorithm 5.3 is*

i. correct, that is, χ *constitutes a* (β, W)*-cover of* G, *and*

ii. can be implemented to run in $O(W\beta^2 \log n)$ *time using* $O(n^{1/\beta}m/(W\beta))$ *processors. (Hence, performing* $O(n^{1/\beta}m\beta \log n)$ *work.)*

Proposition 5.5 *For every pair of vertices* $\{u_1, u_2\} \subset V$, *if* $\mathrm{dist}_E\{u_1, u_2\} \le W$, *then* $\exists X \in \chi$ *such that* $\{u_1, u_2\} \subset X$.

Proof: Let $\{u_1, u_2\} \subset V$ be such that $\mathrm{dist}_E\{u_1, u_2\} \le W$ and let p be a path in (V, E) of weight at most W between u_1 and u_2. During this proof we interchangeably refer to p as the path as a whole or as the set of vertices of p. Since $\{u_1, u_2\} \subset p$, it suffices to show that $p \subset X$ for some $X \in \chi$. Let j be such that $p \subset V_j$ but $p \not\subset V_{j+1}$. Note that j is well defined since the sets V_i $(i \geq 1)$ are monotonically decreasing and include the set of all vertices and the empty set. Denote by χ_j the assignment of χ at the end of the jth iteration. We show that $\exists X \in \chi_j$ such that $p \subset X$. Let $u \in p$ be such that $p \notin V_{j+1}$. It follows from the algorithm that $u \in \text{core}(s, k_j, V_j)$ for some $s \in S_j$. Hence, from the definition, $N_W(V_j, u) \subset \text{cluster}(s, k_j, V_j)$. Since p is of weight at most W and $p \subset V_j$, it follows that $p \subset N_W(V_j, u)$. Hence,

$$\{u_1, u_2\} \subset p \subset N_W(V_j, u) \subset \text{cluster}(s, k_j, V_j) \in \chi \ .$$

∎

Proposition 5.6 *For every $v \in V$, and $1 \leq i \leq \beta$,*

$$\text{core}(v, k_{i+1}, V_{i+1}) \subseteq \text{cluster}(v, k_{i+1}, V_{i+1}) \subseteq \text{core}(v, k_i, V_i) \subseteq \text{cluster}(v, k_i, V_i) \ .$$

Proof: Immediate from Proposition 5.2 part iii. ∎

Proposition 5.7 *With probability $1 - O(1/\operatorname{poly}(n))$, for $j > i$*

$$\forall v \in V_j, |\text{cluster}(v, k_j, V_j)| < n^{1-i/\beta} \ .$$

Proof: If i is the last iteration, $V_{i+1} = \emptyset$ and the proposition follows. Otherwise, $|S_i| = \Omega(n^{i/\beta}|V_i|/n \log n)$. Since $\text{cluster}(v, k_j, V_j) \subset \text{core}(v, k_i, V_i)$ (see Proposition 5.6) and $V_j \subset V_i$,

$$\{v \in V_j |\ |\text{cluster}(v, k_j, V_j)| \geq n^{1-i/\beta}\} \subset \{v \in V_i |\ |\text{core}(v, k_i, V_i)| \geq n^{1-i/\beta}\} \ .$$

It is therefore sufficient to prove that with probability $1 - O(1/\operatorname{poly}(n))$,

$$\{v \in V_i |\ |\text{core}(v, k_i, V_i)| \geq n^{1-i/\beta}\} \cap V_{i+1} = \emptyset \ .$$

Consider a vertex $v \in V_i$ such that $|\text{core}(v, k_i, V_i)| \geq n^{1-i/\beta}$. We prove that with probability $1 - O(1/\operatorname{poly}(n))$, $v \in \bigcup_{s \in S_i} \text{core}(s, k_i, V_i)$ (and hence, using part i of Proposition 5.2, $v \notin V_{i+1}$). Since the elements of S_i are chosen uniformly at random, for $s \in S_i$ we have

$$\text{Prob}\{v \in \text{core}(s, k_i, V_i)\} = |\text{core}(v, k_i, V_i)|/|V_i| \geq n^{1-i/\beta}/|V_i| \ .$$

Hence, since the elements of S_i are chosen independently,

$$\text{Prob}\{v \in V_{i+1}\} = \text{Prob}\left\{v \notin \textstyle\bigcup_{s \in S_i} \text{core}(s, k_i, V_i)\right\} \leq (1 - n^{1-i/\beta}/|V_i|)^{|S_i|}$$
$$= (1 - n^{1-i/\beta}/|V_i|)^{\Omega(|V_i| n^{i/\beta - 1} \log n)} = O(1/\operatorname{poly}(n)) \ .$$

To conclude the proof note that

$$\text{Prob}\left\{\{v \in V_i |\ |\text{core}(v, k_i, V_i)| \geq n^{1-i/\beta}\} \cap V_{i+1} \neq \emptyset\right\}$$
$$\leq \textstyle\sum_{v \in V_i} \text{Prob}\{|\text{core}(v, k_i, V_i)| \geq n^{1-i/\beta} \wedge v \in V_{i+1}\} \leq |V_i|/\operatorname{poly}(n) \leq O(1/\operatorname{poly}(n)) \ .$$

∎

Proposition 5.8 *With probability* $1 - O(1/\operatorname{poly}(n))$*, for all* $e \in E$ *and all* $v \in V$*,*

$$\begin{aligned} |\{X \in \chi | e \in \mathcal{E}(X)\}| &= O(n^{1/\beta}\beta \log n) \text{ and} \\ |\{X \in \chi | v \in X\}| &= O(n^{1/\beta}\beta \log n) \ . \end{aligned}$$

Proof: The claim for edges follows from the claim for vertices. Consider the ith iteration and a vertex v. It follows from Proposition 5.7 that with probability $1 - O(1/\operatorname{poly}(n))$,

$$\text{for all } u \in V_i, \ \ |\text{cluster}(u, k_i, V_i)| \leq n^{1-(i-1)/\beta} \ .$$

Using Proposition 5.2 part i we have that for $u \in V_i$, $v \in \text{cluster}(u, k_i, V_i)$ if and only if $u \in \text{cluster}(v, k_i, V_i)$. Hence, for any $u \in V_i$ chosen uniformly at random

$$\text{Prob}\{v \in \text{cluster}(u, k_i, V_i)\} \leq |\text{cluster}(v, k_i, V_i)|/|V_i| \leq n^{1-(i-1)/\beta}/|V_i| \ .$$

Therefore, the expected number of vertices $s \in S_i$ for which $v \in \text{cluster}(s, k_i, V_i)$ is bounded by $|S_i| n^{1-(i-1)/\beta}/|V_i| = O(n^{1/\beta} \log n)$. By applying Chernoff bound we obtain

$$\text{Prob}\{|\{s \in S_i | v \in \text{cluster}(s, k_i, V_i)\}| = O(n^{1/\beta} \log n)\} \geq 1 - O(1/\operatorname{poly}(n)) \ .$$

Since $\beta = O(\log n) = O(\operatorname{poly} n)$, it follows that for all $v \in V$,

$$\text{Prob}\{\sum_{i=1}^{\beta} |\{s \in S_i | v \in \text{cluster}(s, k_i, V_i)\}| = O(\beta n^{1/\beta} \log n)\} \geq 1 - O(1/\operatorname{poly}(n)) \ .$$

Since $|V| = O(n)$, we can deduce that with probability $1 - O(1/\operatorname{poly}(n))$,

$$\forall v \in V, \sum_i |\{s \in S_i | v \in \text{cluster}(s, k_i, V_i)\}| = O(n^{1/\beta}\beta \log n) \ .$$

∎

Corollary 5.9 *With probability* $1 - O(1/\operatorname{poly}(n))$*,*

$$\begin{aligned} \sum_{X \in \chi} |\mathcal{E}(X)| &= O(n^{1/\beta} m \beta \log n) \text{ and} \\ \sum_{X \in \chi} |X| &= O(n^{1+1/\beta}\beta \log n) \ . \end{aligned}$$

Proof: Note that

$$\sum_{X \in \chi} |\mathcal{E}(X)| = \sum_{e \in E} |\{X \in \chi | e \in X\}| \text{ and } \sum_{X \in \chi} |X| = \sum_{v \in V} |\{X \in \chi | v \in X\}| \ .$$

The proof follows using Proposition 5.8. ∎

We conclude the proof of Theorem 5.4. To prove the correctness of Algorithm 5.3 we verify that with probability $1 - O(1/\operatorname{poly}(n))$, χ constitutes a (β, W)-cover of G. We show that χ satisfies properties i and ii of Definition 3.3 and with probability $1 - O(1/\operatorname{poly}(n))$,

χ satisfies property iii of the definition. Proposition 5.5 establishes that property i holds for χ. The validity of property ii follows from the fact that for all i, $k_i \leq \beta - 1$. Proposition 5.8 asserts that χ satisfies Property iii (with probability $1 - O(1/\operatorname{poly}(n))$).

We furnish resource bounds for Algorithm 5.3. We perform the computation of clusters in step ii2 using the algorithm of Section 4. The latter algorithm assumes that the graph is represented in a certain format (see Section 4). This representation allowed us to compute clusters with work proportional to the combined size of the clusters computed (instead of having to look at all the edges in the graph). We remark that this representation can be easily generated and revised (for the subgraphs induced by V_i where $i > 0$) within the resource bounds of the algorithm. Consider a single iteration of the algorithm. The computation in step ii2 dominates the complexity. It follows from Proposition 4.1 that the computation of step ii2 in the ith iteration can be performed in

$$O(W\beta \log n + \sum_{s \in S_i} |\mathcal{E}(\mathrm{cluster}(s, k_i, V_i))|(\log n)/p)$$

time using p processors (and in $O(\sum_{s \in S_i} |\mathcal{E}(\mathrm{cluster}(s, k_i, V_i))|)$ work, when allowing $O(W\beta n^{\alpha})$ time.) Hence, since there are at most β iterations, the algorithm can be performed in

$$O(W\beta^2 \log n + \sum_{x \in \chi} |\mathcal{E}(X)|(\log n)/p)$$

time using p processors. By substituting the bound from Corollary 5.9 we deduce that with probability $1 - O(1/\operatorname{poly}(n))$, the algorithm can be performed in $O(W\beta^2 \log n)$ time using $p = O(n^{1/\beta} m(\log n)/(W\beta))$ processors. The work performed by the algorithm is $O(n^{1/\beta} m\beta \log^2 n)$. When allowing $O(W\beta^2 n^{\alpha})$ time (for fixed α), the work is reduced to $O(n^{1/\beta} m\beta \log n)$.

6. Sequential algorithms

This section is concerned with sequential stretch-t paths algorithms. We first discuss a sequential version of the randomized algorithm presented in previous sections. In the main part of this section we present a deterministic algorithm. Our sequential algorithms are both simpler than the previous Awerbuch et al. [2] algorithm and in addition significantly improve the tradeoffs between the stretch and the running time.

6.1. A randomized sequential algorithm

In Subsection 2.2 we presented a parallel algorithm (Algorithm 2.6) that computes stretched ℓ-limited distances, where $1 \leq \ell \leq n$. Hence, when $\ell = n$, the algorithm produces stretch-t paths. The resulting algorithm inputs a graph G, with weights $w : E \rightarrow R_+$, integer $\beta \geq 1$, a scalar $0 \leq \hat{\epsilon} \leq 1/2$, and a set of query pairs $Q \subset V \times V$. The algorithm runs in time $O((m + |Q|)n^{1/\beta}\beta \log^2 n)$ and with probability $1 - O(1/\operatorname{poly}(n))$ computes for all $\{v, u\} \in Q$, stretched distance $\tilde{\mathrm{dist}}\{v, u\}$ such that:

$$\mathrm{dist}_E\{v, u\} \leq \tilde{\mathrm{dist}}\{v, u\} \leq 2\beta(1 + \hat{\epsilon})^2 \mathrm{dist}_E\{v, u\} \ .$$

A simplified sequential algorithm with slightly worse time bounds and slightly better stretch can be obtained by avoiding the reduction to small integral weights performed in Section 3. Small integral weights are necessary for efficient parallel computation of neighborhoods, but are not crucial in the sequential version. The simplified algorithm applies a sequential version of Algorithm 2.6 and uses $(n, \beta, w_i, 0)$-covers instead of $(\ell, \beta, w_i, \hat{\epsilon})$-covers (for $i = 1, \ldots, r$). $(n, \beta, w_i, 0)$-covers can be computed directly by Algorithm 5.3. In the implementation of Algorithm 5.3 we disregard the assumption that the weights are integral and utilize a slightly modified version of Dijkstra's algorithm to compute neighborhoods. If edges are initially sorted according to their weight at the vertices, we can compute a neighborhood with V' vertices in time $O(|\mathcal{E}(V')| + |V'| \log |V'|)$ (using Fibonnaci heaps). The simplified algorithm inputs a graph G, with weights $w : E \to R_+$, integer $\beta \geq 1$, a scalar $0 \leq \hat{\epsilon} \leq 1/2$, and a set of query pairs $Q \subset V \times V$. The algorithm runs in time $O((m + n \log n + |Q|)n^{1/\beta} \beta \log^2 n)$ and with probability $1 - O(1/\operatorname{poly}(n))$ computes for all $\{v, u\} \in Q$, stretched distance $\tilde{\operatorname{dist}}\{v, u\}$ such that:

$$\operatorname{dist}_E\{v, u\} \leq \tilde{\operatorname{dist}}\{v, u\} \leq 2\beta(1 + \hat{\epsilon})\operatorname{dist}_E\{v, u\} \ .$$

6.2. The deterministic algorithm

We present a deterministic stretch-t paths algorithm. For a weighted graph $G = (V, E)$ and a set of sources $S \subset V$, the algorithm computes stretched distances for all pairs of vertices in $S \times V$. For an integer β and any fixed $\epsilon > 0$, the algorithm obtains stretched distances to within a factor of $2(1 + \log_n m)\beta(1 + \epsilon)$ using $\tilde{O}(n^{1/\beta}(m + n|S|))$ time. For the all-pairs problem (when $|S| = n$), the running time is $\tilde{O}(n^{2+1/\beta})$. A modification of the algorithm achieves stretch $4\beta(1 + \epsilon)$ using $\tilde{O}(n^{2+1/\beta})$ time. For dense graphs $\log_n m \approx 2$, and hence the modification yields improved stretch (by a factor of 2/3) within comparable time bounds. In contrast, the Awerbuch et al. [2] algorithm obtains stretch 32β using $\tilde{O}(n^{2/\beta} m + n^{1+1/\beta}|S|)$ time. We remark, however, that the Awerbuch et al. [2] algorithm is more general since stretched distances between any k specified pairs of vertices can be obtained in $\tilde{O}(n^{2/\beta} m + n^{1/\beta} k)$ time.

We use the following definition for pairwise covers. Note that it differs from Definition 2.3 that was used for the randomized algorithm.

Definition 6.1 Consider a graph $G = (V, E)$ with weights $w : E \to R_+$, an integer $\beta = O(\log n)$, and a scalar $w_r \in R_+$. A *pairwise* (β, w_r)*-cover* of G is a collection of sets of vertices $X_1, \ldots, X_k$ (*clusters*) and vertices $v_1, \ldots, v_k$, where $v_i \in X_i$ is the *center* of X_i, such that

i. $\forall \{u, v\} \subset V$ such that $\operatorname{dist}\{u, v\} \leq w_r$ $\exists i$ such that $\{u, v\} \subset X_i$,

ii. $\forall i, \forall u \in X_i$, $\operatorname{dist}\{v_i, u\} \leq (1 + \log_n m)\beta w_r$, and

iii. $\sum_{i=1}^{k} |X_i| = O(n^{1+1/\beta})$.

We claim that a (β, w_r)-cover of G can be computed in $O(mn^{1/\beta})$ time. We first discuss the structure of the clusters we choose to constitute the cover.

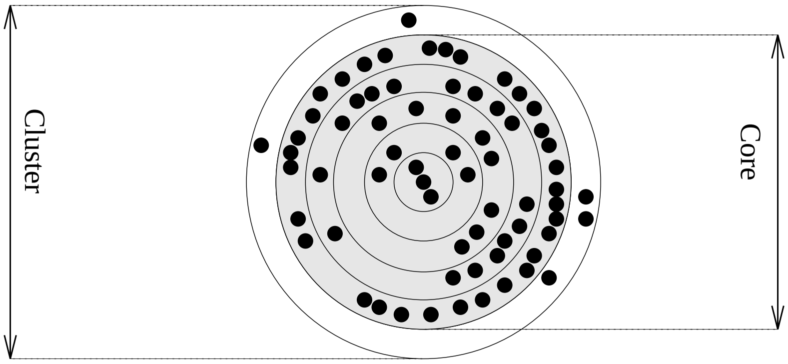

Figure 2: Example of a cluster and respective core for $n^{1/\beta} < 2$

6.3. Structure of clusters

We consider a graph $G = (V, E)$ with weights $w : E \to R_+$ and parameters β and $w_r \in R_+$. We discuss the structure and properties of subsets of vertices we call *clusters*. We use carefully chosen clusters as the building blocks of a (β, w_r)-cover.

A cluster is a subset of vertices that is defined with respect to two parameters: a subset of vertices $V' \subset V$ and a vertex $v \in V'$ (the *center* of the cluster). The cluster comprises a neighborhood of the center v in the graph $(V', \mathcal{E}(V'))$ up to a certain distance (that is an integral multiple of w_r). The *core* of a cluster is the subset of cluster vertices whose neighborhoods up to distance w_r in $(V', \mathcal{E}(V'))$ are contained in the cluster. For the parameters (v, V'), we define core(v, V') and cluster(v, V') to be the sets obtained by performing the following computation.

i. cluster$(v) \leftarrow \{v\}$

ii. $i \leftarrow 1$

iii. **Repeat:**

  (1) core$(v) \leftarrow$ cluster(v)

  (2) cluster$(v) \leftarrow \{u \in V' | \mathrm{dist}_{\mathcal{E}(V')}\{v, u\} \le i w_r\}$

  (3) $i \leftarrow i + 1$

iv. **Until:** $|\mathrm{cluster}(v)| \le n^{1/\beta}(|\mathrm{core}(v)| + 1) \wedge$
$|\mathcal{E}(\mathrm{cluster}(v))| \le n^{1/\beta}(|\mathcal{E}(\mathrm{core}(v))| + 1)$

Figure 2 illustrated a cluster. The shells corresponds to neighborhoods of the center that are integral multiples of w_r. In the example, $n^{1/\beta} < 2$. Hence the cluster stops growing when annexing the next shell does not double the number of vertices. The external shell defines the cluster and the previous shell defines the core.

Proposition 6.2 *At termination,* $i \le (1 + \log_n m)\beta - 1$.

Proof: Note that the size of at least one of cluster(v) or $\mathcal{E}(\text{cluster}(v))$ increases by more than a factor of $n^{1/\beta}$ in every iteration except for the last one. Observe that $|\text{cluster}(v)| \le |V'| \le n$ and $|\mathcal{E}(\text{core})| \le m$. Thus, $|\text{cluster}(v)|$ can increase by a factor of more than $n^{1/\beta}$ at most $\beta - 1$ times and $|\mathcal{E}(\text{cluster}(v))|$ can increase by a factor of more than $n^{1/\beta}$ at most $\beta \log_n m - 1$ times. Hence, the stopping condition occurs within $(1 + \log_n m)\beta - 1$ iterations. ∎

Consider a subset $V' \subset V$ and $v \in V'$.

Corollary 6.3 *For all* $u \in \text{cluster}(v, V')$,

$$\text{dist}_E\{u, v\} \le \text{dist}_{\mathcal{E}(V')}\{u, v\} \le (1 + \log_n m)\beta w_r \ .$$

The following proposition is immediate.

Proposition 6.4 *i. For all* $u \in \text{core}(v, V')$, $N_W(V', u) \subset \text{cluster}(v, V')$.

ii. $|\text{cluster}(v, V')| \le n^{1/\beta}(|\text{core}(v, V')| + 1)$

iii. $|\mathcal{E}(\text{cluster}(v, V'))| \le n^{1/\beta}(|\mathcal{E}(\text{core}(v, V'))| + 1)$

We discuss the construction of clusters. We assume that the edges incident at each vertex are initially sorted according to their weight. Remark 4.3 asserts that a simple modification of the standard BFS algorithm yields the following bound: for all $V' \subset V$ and $v \in V'$, the sets cluster(v, V') and core(v, V') can be computed in $O(|\mathcal{E}(\text{cluster}(v, V'))|)$ time.

6.4. The cover algorithm

The following algorithm produces a pairwise cover.

Algorithm 6.5 [Compute a (β, w_r)-cover]

i. $V' \leftarrow V$, $\chi \leftarrow \emptyset$

- **Main Loop:**

ii. Choose $v \in V'$
Compute cluster(V', v) and core(V', v).
$\chi \leftarrow \chi \cup \{\text{cluster}(V', v)\}$
$V' \leftarrow V' \setminus \text{core}(V', v)$
If $V' \ne \emptyset$, go to step ii

We show that χ is indeed a (β, w_r)-cover as in Definition 6.1. Property ii is immediate from Corollary 6.3. Property iii follows from Proposition 6.4 and the fact that the cores of the clusters in χ are disjoint. In order to prove that Property i holds, consider a pair of vertices $\{v_1, v_2\} \subset V$ such that $\text{dist}_E\{v_1, v_2\} \le w_r$. Let p be a set of vertices constituting a path between v_1 and v_2 of weight at most w_r. Consider the first iteration where $p \cap \mathrm{core}(V', v) \neq \emptyset$ (V' and v denote the respective assignments of values to V' and v during this iteration). It follows that $p \subset V'$. By definition, cluster(V', v) contains all vertices of distance at most w_r from core(V', v) in $\mathcal{E}(V')$. Hence, since $\mathcal{E}(p) \subset \mathcal{E}(V')$ and $p \cap \mathrm{core}(V', v) \neq \emptyset$, we have $p \subset \mathrm{cluster}(V', v)$. Therefore, $\{v_1, v_2\} \subset p \subset \mathrm{cluster}(V', v)$. The running time of the algorithm is dominated by the construction of the clusters. We assume that at each vertex the incident edges are sorted according to weight. We argued above that the computation of a cluster $X \in \chi$ can be performed in $O(|\mathcal{E}(X)|)$ time. Hence, using property iii of Definition 6.1, we obtain the following bound on the running time

$$O(\sum_{X \in \chi} |\mathcal{E}(X)|) = O(n^{1/\beta} m) .$$

6.5. Computing stretched distances

A cover as in Definition 6.1 can be used in a similar manner to Algorithm 2.6 of Subsection 2.2 to compute stretched distances from a set of sources $S \subset V$ to all other vertices.

Theorem 6.6 *For a weighted graph G, an integer $\beta \geq 1$, a fixed $\epsilon \geq 0$, and a set of query sources $S \subset V$, the following algorithm computes for all $v \in S$ and for all $u \in V$, stretched distance* $\tilde{\mathrm{dist}}(v, u)$ *such that:*

$$\mathrm{dist}_E(v, u) \leq \tilde{\mathrm{dist}}(v, u) \leq 2\beta(1 + \log_n m)(1 + \epsilon)\mathrm{dist}_E(v, u) .$$

in $O(n^{1/\beta} m \log n + |S| n^{1+1/\beta} \log n)$ time.

Algorithm 6.7 [Compute distances]

i. $w_{\min} \leftarrow \min_{e \in E} w(e)$
$w_{\max} \leftarrow \max_{e \in E} w(e)$
$r \leftarrow \lceil \log_{1+\hat{\epsilon}}(n w_{\max}/w_{\min}) \rceil$
For $i = 0, \ldots, r$: $w_i \leftarrow w_{\min}(1 + \hat{\epsilon})^i$

ii. For $i = 0, \ldots, r$ do (in parallel):

(1) Compute (β, w_i)-cover χ_i

(2) For each vertex $v \in S$, $x(i, v) \leftarrow \bigcup\{X \in \chi_i | v \in X\}$.

iii. For all $v \in S$:
For all $u \in V$:
$\tilde{\mathrm{dist}}(v, u) \leftarrow 2(1 + \log_n m)\beta w_i$, where $i \leftarrow \min\{j | u \in x(j, v)\}$.

The correctness follows from arguments similar to the ones given in Subsection 2.2. The computation amounts to firstly computing $r = O(\epsilon^{-1}\log n)$ covers for different values of w_r, and secondly, for every source $v \in S$ computing $x(j,v)$ for $1 \le j \le r$. The computation per source can be performed in time linear in the sum of the sizes of the covers. Recall that the computation of covers using Algorithm 6.5 assumes that the edges are sorted according to weight at vertices. Hence, the running time is $O(m\log n)$ for the sorting of edges, $O(\epsilon^{-1}n^{1/\beta}m\log n)$ for computing the covers, and $O(n^{1+1/\beta}\epsilon^{-1}\log n)$ time per source.

Remark 6.8 Consider relaxing the termination conditions in the definition of clusters and not requiring that $|\mathcal{E}(\text{cluster}(v))| \le n^{1/\beta}(|\mathcal{E}(\text{core}(v))|+1)$. We obtain improved stretch (the diameter of cluster is shorter by a factor of $(1+\log_n m)$ but worse time bounds. The tradeoffs for stretched all-pairs distances in dense graphs improve. We obtain an $O(n^{2+2/\beta}\log n)$ time algorithm for computing stretched distances to within a factor of 2β between all pairs of vertices.

7. Applications to graph spanners

Consider a weighted graph $G = (V, E)$. A t-spanner of G is a weighted graph (V, E') such that for all pairs $\{u_1, u_2\} \subset V$ we have

$$\text{dist}_E\{u_1,u_2\} \le \text{dist}_{E'}\{u_1,u_2\} \le t\,\text{dist}_E\{u_1,u_2\}\ .$$

We utilize our constructions of pairwise covers to obtain t-spanners for weighted graphs. We present a randomized construction of spanners of size (number of edges) $O(n^{1+(2+\epsilon)/t})$ in time $O(mn^{(2+\epsilon)/t})$ (for any $\epsilon > 0$, and t such that $t/(2+\epsilon)$ is integral). We give a deterministic algorithm that finds t-spanners of size $O(n^{1+(2+\epsilon)(1+\log_n m)/t})$ and runs in $O(mn^{(2+\epsilon)(1+\log_n m)/t})$ time (for any $\epsilon > 0$, and t such that $t/((1+\log_n m)(2+\epsilon))$ is integral). In addition, we discuss parallel constructions of spanners.

7.1. Computing spanners sequentially

We show how to obtain sparse (small size) spanners. Our spanners are constructed from a collection of pairwise covers (see Definitions 3.3 and 6.1). Let $R = w_{\max}/w_{\min}$. Using considerations similar to Remark 2.1, we may assume that $R = O(\text{poly}(n))$. Let $r = \lceil \log_{1+\epsilon} R\rceil$. It follows that $r = O(\epsilon^{-1}\log n)$. Let $\beta = t/(2+\epsilon/2)$ if the randomized construction of covers is used (see Definition 3.3) and let $\beta = t/((1+\log_n m)(2+\epsilon/2))$ if the deterministic construction is used (see Definition 6.1). For each $0 \le i \le r$, let χ_i be a (β, w_i)-cover of G , where $w_i = w_{min}(1+\epsilon)^i$. The t-spanner $S \subset E$ is constructed as follows. For each $0 \le i \le r$ and each cluster $X \in \chi_i$, add to S the edges of the partial BFS tree used to construct X. It follows immediately from the properties of our construction of covers, that for every edge $e \in E$, there is a path in S of weight at most $tw(e)$ connecting the end points of e. Hence, for every path $p \subset E$, there is a path in S of weight at most $tw(p)$. Therefore, S indeed comprises a t-spanner. To bound the size of the spanner S, note that the number of edges in

S is bounded by $\sum_{i=0}^{r} \sum_{X \in \chi_i} (|X| - 1)$. Hence, it follows from the properties of covers that $|S| = O(rn^{1+1/\beta} \beta \log n)$. To obtain a time bound for the spanner algorithm, note that it amounts to computing r covers. The time bounds for computing a single cover are $O(mn^{1/\beta})$ time for the deterministic cover algorithm and $O(mn^{1/\beta} \log n)$ time for the randomized cover algorithm.

Additional properties of our spanners. The spanner S consists of a collection of subtrees $\mathcal{T}$ of the graph G. The following stronger statement holds: for any two vertices $\{u_1, u_2\} \subset V$, the minimum over all the trees $T \in \mathcal{T}$ such that $\{u_1, u_2\} \subset T$ of the distance between u_1 and u_2 in T, is within a factor t of $\text{dist}_E\{u_1, u_2\}$. Furthermore, if the covers produced by the randomized construction are used, then every vertex is contained in at most $O(rn^{1/\beta} \beta \log n)$ subtrees in $\mathcal{T}$. Consider a variant where we do not require the spanner S to consist of edges from E, but we want limited distances on the spanner (using small size paths) to be within a small factor of distances on the graph. Construct a spanner S' as follows. Take $R = nw_{\max}/w_{\min}$ and $r = \lceil \log_{1+\epsilon} R \rceil = O(\epsilon^{-1} \log n)$. For each $0 \le i \le r$, let χ_i be a (β, w_i)-cover of G , where $w_i = w_{min}(1+\epsilon)^i$. For each $0 \le i \le r$, each cluster $X \in \chi_i$ with respective center $u(X)$, and each $v \in X$, include in S' an edge $(u(X), v)$ of weight equals to the distance from $u(X)$ to v in X. The spanner S' has a stronger property that the 2-limited distances in S' are within a factor t of the respective distances in G.

7.2. Computing spanners in parallel

We provide three methods for computing spanners in parallel. In this subsection we assume that $\beta = t/(2 + \epsilon/2)$ and use Definition 3.3 for covers.

Straightforward parallel implementation.. We describe a parallel implementation of the sequential algorithm above that has the same work bound, and runs in time linearly dependent on $R = w_{\max}/w_{\min}$. Hence, this implementation is of interest when R is sufficiently small. In particular, the algorithm is in $\mathcal{NC}$ for unweighted graphs. Note that for the covers computed by the spanner algorithm we have $w_{\min} \le w_i \le (1+\epsilon) w_{\max}$ $(0 \le i \le r)$. By definition, for any cluster $X \in \chi_i$, the distance from the center of X to any vertex $v \in X$ is at most βw_i, and hence, the partial BFS tree generated while computing X has path size of $O(R\beta)$. Consider an application of Algorithm 5.3 to compute χ_i, where in Step ii2 we compute clusters by a weighted parallel BFS algorithm (see Section 4). It follows that the cover algorithm runs in expected time $O(R\beta^2 \log n)$ and performs $O(n^{1/\beta} m \beta \log n)$ work. Hence, t-spanners of size $O(n^{1+(2+\epsilon)/t})$ can be obtained in $O(R\beta^2 \log^2 n)$ expected time using $O(n^{1/\beta} m \beta \log^2 n)$ work.

Using (exact) all-pairs shortest paths. We showed above that computing a spanner amounts to concurrently computing $r = O(\log n)$ covers. Consider a single application of Algorithm 5.3 to compute a cover. The complexity of the algorithm is dominated by step ii2. Note that step ii2 can be performed (with worse work bounds) as follows. First compute all-pairs distances on the graph induced by the vertices V_i. The sets $\mathrm{core}(v, k_i, V_i)$ and $\mathrm{cluster}(v, k_i, V_i)$ for $v \in S_i$ can be easily generated using the all-pairs distances table. The parallel all-pairs shortest paths computation can be performed in $O(\log^2 n)$ time with $O(n^3)$ work on an EREW PRAM. The generation of the sets $\mathrm{core}(v, k_i, V_i)$ and $\mathrm{cluster}(v, k_i, V_i)$ for $v \in S_i$ can be performed within the same time bounds using work proportional to the sum of the sizes of these sets. Note that the resource bounds of the all-pairs shortest paths computation dominate all other steps. A shortest paths computation is performed in each one of the β iterations of Algorithm 5.3. It follows that a pairwise cover can be computed in $O(\beta \log^2 n)$ time using $O(\beta n^3)$ work. Hence, t-spanners of size $O(n^{1+(2+\epsilon)/t})$ can be obtained in $O(\beta \log^3 n)$ expected time using $O(n^3 \beta \log n)$ work. We remark that although the work bound is worse than the sequential time bound achieved above, it is better than even the sequential time bounds of previous algorithms.

Tradeoffs between time and stretch. This method is based on the same technique used to obtain a fast parallel stretch-t paths algorithm (see Subsection 2.3). We sketch the algorithm. Consider some parameter $\ell' < R$, and let $k = \lceil \log_{\ell'/2} R \rceil$ We repeatedly apply k times a version of Algorithm 2.7, with parameter ℓ', where in step i we use $r \leftarrow \lceil \log_{1+\hat{\epsilon}} R \rceil$. The above guarantees that if $\mathrm{dist}_E\{u_1, u_2\} \leq w_{\max}$ then for all $\ell' \leq \ell$,

$$\mathrm{dist}_{E'}^{2\lceil \ell/\ell' \rceil}\{u_1, u_2\} \leq 2\beta(1+\hat{\epsilon})^2 \mathrm{dist}_E^{\ell}\{u_1, u_2\} \ ,$$

where E' is the set of edges obtained by applying the algorithm to (V, E). Note that the running time is $\tilde{O}(k\ell')$. We consider the BFS trees obtained by the covers computed at the last repetition of the algorithm and "expand" them back to consist of original edges. The resulting spanners are t^k-spanners, but their size and the work performed is the same as for the sequential t-spanner algorithm.

8. Stretch-t paths on dynamic networks

The dynamic shortest-paths problem amounts to resolving on-line distance queries between pairs of vertices when weighted edges (and possibly new vertices) are inserted to or deleted from the graph. Unfortunately, even when only insertions are permitted, the currently best known solution amounts to the brute force method of applying Dijkstra's $O(m + n \log n)$ time algorithm to resolve each distance query. The pairwise covers and stretch-t paths algorithms introduced here enable us to obtain more efficient algorithms for dynamic stretch-t distances. We consider a dynamic stretch-t paths problem where we are allowed to insert edges (while possibly introducing new vertices), delete recently-inserted edges, and perform stretch-t distance queries. Another relevant setting is when some large graph is fixed, but we allow insertions to and deletions from a small set of new edges. The distance queries are performed with respect to the augmented graph.

Consider the sequential version of Algorithm 2.6 (see Subsection 6.1). In the first part of the algorithm, it produces a set of pairwise covers as to be able to answer on-line distance queries. The first part takes $\tilde{O}(mn^{(2+\epsilon)/t})$ time. Consequently, in the second part of the algorithm, each distance query takes $\tilde{O}(n^{(2+\epsilon)/t})$ time. Our dynamic algorithms utilize Algorithm 2.6. Note that since Algorithm 2.6 is randomized, the time bounds mentioned below are for the expected running time.

We first consider the following problem: The graph G is fixed and there is a dynamic set of new weighted edges I. Some of the edges in I may introduce new vertices. We denote by $V(I)$ the vertices in I. The operations allowed are to insert edges to I, delete edges from I, and ask stretch-t distance queries between vertices in $V \cup V(I)$, with respect to the graph $(V \cup V(I), E \cup I)$. We sketch an algorithm for this problem and analyze its performance. The initialization step is to apply the first part of Algorithm 2.6 to the fixed graph G, and generate a collection of pairwise covers. The algorithm dynamically maintains a graph $G_I = (V_I, E_I)$ as follows. $V_I = V(I) \cup C$, where C is a collection of all centers of clusters that contain vertices from $V(I)$. The set E_I contains I and for each occurrence of a vertex $v \in V(I)$ in a cluster X with center $c \in C$, E_I contains the edge (v, c) weighted by the radius of X. Recall (see Definition 2.3) that each vertex $v \in V$ belongs to $\tilde{O}(n^{(2+\epsilon)/t})$ clusters. Hence, $|C| = \tilde{O}(|V(I)|n^{(2+\epsilon)/t})$, and $E_I \setminus I = \tilde{O}(|V(I)|n^{(2+\epsilon)/t})$. It is easy to see that an insertion or deletion of an edge takes $O(1)$ time, if $V(I)$ is not modified, and $\tilde{O}(n^{(2+\epsilon)/t})$ time otherwise. To perform a stretch-t distance query for a pair $\{u_1, u_2\} \subset V \cup V(I)$, we go as follows. We utilize the second part of Algorithm 2.6, applied to the set of pairwise covers produced earlier, to compute stretch-t distances on G from u_1 and from u_2 to every vertex in $V(I) \cap V$. We perform a single-source shortest-paths computation, rooted at u_1, on G_I augmented as follows: by the vertices $\{u_1, u_2\}$ and edges between $\{u_1, u_2\}$ and $V(I) \cap V$, weighted by the stretch-t G-distances computed above. The answer to the query is the distance between u_1 and u_2 in the augmented G_I. It is easy to verify that the distance between u_1 and u_2 in the augmented G_I is within a factor of t of the distance in $(V \cup V(I), E \cup I)$. We bound the time required for performing a query. The stretch-t distances computations in G can be performed in time $\tilde{O}(|V(I) \cap V|n^{(2+\epsilon)/t})$. The single-source shortest paths computation take time $\tilde{O}(|V(I)| + |E_I||) = \tilde{O}(|I| + |V(I)|n^{(2+\epsilon)/t})$ (using, e.g., Dijkstra's algorithm). Hence, the total query time is $\tilde{O}(|I|n^{(2+\epsilon)/t})$. We remark that the time is improved when $|V(I)| \ll |I|$.

Consider now the following problem. Let k be a parameter. The weighted edges are maintained in a list M. The allowed operations are to insert edges to the head of M, to delete edges that are of distance at most k from the head of M, and to ask t-stretch distance queries. We sketch an algorithm that operates in stages. Each stage amounts to an instance of the problem discussed in the previous paragraph. In the beginning of each stage, we consider a suffix of M, denoted by M' that contains all edges in M but the initial $2k$ edges. The set M' constitutes the fixed graph, and we initialize $I = M \setminus M'$. Within a stage, the insertions, deletions, and distance queries are handles as sketched in the previous paragraph. The current stage terminates (and a new stage begins) when there is a request to delete an edge in M', or when $|M \setminus M'| \geq 4k$. It is easy to see that each stage consists of at least k deletions or insertions. Using standard dynamic algorithms methods, we can amortize the cost of the operations such that insertions or deletions take $\tilde{O}((k + m/k)n^{(2+\epsilon)/t})$ time and queries take $\tilde{O}(kn^{(2+\epsilon)/t})$ time. Note that when $k = m^{0.5}$, each operation takes $\tilde{O}(m^{0.5}n^{(2+\epsilon)/t})$ time. In general, k can be determined according to the ratio of edge insertions/deletions and queries.

9. Concluding remarks

We discuss some issues and open problem that arise from this work.

We remark that stretch-n paths can be obtained using a minimum spanning tree (MST). For a pair of vertices, consider the MST path p between them and the heaviest edge e on this path. It is easy to see that the distance between the vertices is bounded from below by $w(e)$, and bounded from above by $w(p) \leq nw(e)$. An MST can be computed sequentially in $O(m \log n)$ time and in parallel in $\mathcal{NC}$ using $\tilde{O}(m)$ work.

The techniques used in this paper and in the previous work of Awerbuch et al. [2] do not seem to generalize to directed graphs. A natural question regards obtaining comparable work-stretch tradeoffs for directed graphs. We conjecture that for directed graphs, obtaining stretch-t paths where t is constant or polylogarithmic is nearly as hard as finding shortest-paths.

A result due to Karger, Koller, and Phillips [8] provides some indication that for directed graphs, either a comparable time-stretch tradeoffs are not obtainable or stretch-t paths are indeed strictly easier to find that shortest-paths. Karger et al. proved a lower bound of $\Omega(mn)$ on the amount of work required by path-comparison-based algorithm for all-pairs shortest-paths computations. Note, however, that this lower bound is applicable for a restricted class of algorithms (that does not include the algorithms presented in this paper) and is applicable only to the all-pairs problem. Karger conjectured that the lower bound is likely to hold for undirected graphs as well [Karger, personal communication].

We point out some additional properties of the pairwise covers constructed in this paper that were not needed for the applications used here. The first property of a (β, w)-cover stated that if two vertices $\{u_1, u_2\} \subset V$ have distance at most w, then at least one cluster in the cover contains all the vertices on some path of weight w between u_1 and u_2. It is easy to prove the stronger property that at least one cluster must contain *all* paths of weight at most w between u_1 and u_2. Moreover, the (β, w)-covers introduced here actually constitute neighborhood covers: For every vertex $u \in V$, the $w/2$ neighborhood of u must be contained in at least one cluster. Recall that pairwise-covers were introduced here as a more correct (in the context of short paths computations) and constructible in parallel alternative for the neighborhood covers of Awerbuch and Peleg [3]. This indicates that our constructions may have other applications.

We presented efficient algorithms to produce sparse spanners. Some of the previous work on spanners (see Chandra et al. [4]) is concerned with finding spanners that are not only sparse but in addition have small weight (that is, within a small factor of the weight of the minimum spanning tree). It is likely that our methods can be extended to efficiently generate sparse spanners with small weight.

We presented parallel algorithms that compute stretch-t paths, where t is a constant or polylogarithmic, while performing significantly less work than exact shortest-paths algorithms. This is notable because typically, parallel shortest paths algorithms perform much more work than their sequential counterparts. A particular example is the single-source shortest-paths problem on graphs with nonnegative weights. It is solved sequentially in

$\tilde{O}(m)$ time but the currently best known work bound of a polylog time algorithm is $O(n^3)$. We remark that the author, in a later work [6], that employs our parallel pairwise cover constructions, presented a $\tilde{O}(mn^{\epsilon_0} + s(m + n^{1+\epsilon_0}))$ work polylog-time randomized algorithm that computes paths within $(1 + O(1/\operatorname{polylog} n))$ of shortest from s source nodes to all other nodes in weighted undirected networks (for any fixed $\epsilon_0 > 0$). Another implication of the work in [6] is efficiently parallelizing the randomized t-spanners, stretch-t paths, and pairwise covers algorithms presented in the current paper. Namely, obtaining polylogarithmic time parallel algorithms that perform work comparable to the respective sequential running times presented here and an additional additive term of $O(mn^{\epsilon_0})$ (where $\epsilon_0 > 0$ is any fixed constant).

Acknowledgments. The author would like to thank Lenore Cowen for presenting the Awerbuch et al. [2] work at Bell Labs and discussing it and the current work with the author, to thank Lenore Cowen, David Johnson, and Alex Wang for suggestions that improved the write-up, and Andrew Goldberg, Monika Rauch, and Peter Shor for useful discussions and bibliographic pointers.

References.

[1] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9, 1993.

[2] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Near-linear cost sequential and distributed constructions of sparse neighborhood covers. In *Proc. 34th IEEE Annual Symposium on Foundations of Computer Science*, pages 638–647. IEEE, 1993.

[3] B. Awerbuch and D. Peleg. Sparse partitions. In *Proc. 31st IEEE Annual Symposium on Foundations of Computer Science*, pages 503–513. IEEE, 1990.

[4] B. Chandra, D. Gautam, G. Narasinhan, and J. Soares. New sparseness results on graph spanners. In *Proc. 8th Annual ACM Symposium on Computational Geometry*, pages 192–201. ACM, 1992.

[5] E. Cohen. Using selective path-doubling for parallel shortest-path computations. In *Proc. of the 2nd Israeli Symposium on the Theory of Computing and Systems*, pages 78–87. IEEE, 1993. full version submitted.

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

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

[8] D. R. Karger, D. Koller, and S. Phillips. Finding the hidden path: time bounds for all-pairs shortest paths. In *Proc. 32nd IEEE Annual Symposium on Foundations of Computer Science*, pages 560–568. IEEE, 1991.

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

[10] D. Peleg and A. A. Schäffer. Graph spanners. *J. Graph Theory*, 13:99–116, 1989.

[11] D. Peleg and J. D. Ullman. An optimal synchronizer for the hypercube. *SIAM J. Comput.*, 18:740–747, 1989.

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

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