SIAM J. COMPUT.
Vol. 48, No. 4, pp. 1436–1480

HOPSETS WITH CONSTANT HOPBOUND, AND APPLICATIONS TO APPROXIMATE SHORTEST PATHS[*]

MICHAEL ELKIN[†] AND OFER NEIMAN[†]

Abstract. A (β,ϵ)-hopset for a weighted undirected n-vertex graph $G=(V,E)$ is a set of edges, whose addition to the graph guarantees that every pair of vertices has a path between them that contains at most β edges, whose length is within $1+\epsilon$ of the shortest path. In her seminal paper, Cohen [*J. ACM,* 47 (2000), pp. 132–166] introduced the notion of hopsets in the context of parallel computation of approximate shortest paths, and since then it has found numerous applications in various settings, such as dynamic graph algorithms, distributed computing, and the streaming model. Cohen [*J. ACM,* 47 (2000), pp. 132–166] devised efficient algorithms for constructing hopsets with *polylogarithmic* in n number of hops. Her constructions remain the state of the art since the publication of her paper in the proceedings of STOC'94, i.e., for more than two decades. In this paper we exhibit the first construction of sparse hopsets with a *constant number of hops.* We also find efficient algorithms for hopsets in various computational settings, improving the best-known constructions. Generally, our hopsets strictly outperform the hopsets of [*J. ACM,* 47 (2000), pp. 132–166] in terms of both their parameters and the resources required to construct them. We demonstrate the applicability of our results for the fundamental problem of computing approximate shortest paths from s sources. Our results improve the running time for this problem in the parallel, distributed, and streaming models for a vast range of s.

Key words. hopset, shortest path, graph algorithms

AMS subject classification. 68W99

DOI. 10.1137/18M1166791

1. Introduction.

1.1. Hopsets, setting, and main results.

We are given an n-vertex weighted undirected graph $G=(V,E,\omega)$. Consider another graph $G_H=(V,H,\omega_H)$ on the same vertex set V. Define the union graph $G'=G\cup G_H$, that is, $G'=(V,E'=E\cup H,\omega')$, where $\omega'(e)=\omega_H(e)$ for $e\in H$, and $\omega'(e)=\omega(e)$ for $e\in E\setminus H$. For a positive integer parameter β, and a pair $u,v\in V$ of distinct vertices, a *β-limited distance* between u and v in G', denoted $d^{(\beta)}_{G'}(u,v)$, is the length of the shortest u-v path in G' that contains at most β edges (also known as *hops*). For a parameter $\epsilon>0$, and a positive integer β as above, a graph $G_H=(V,H,\omega_H)$ is called a *(β,ϵ)-hopset* for the graph G if for every pair $u,v\in V$ of vertices, we have $d_G(u,v)\le d^{(\beta)}_{G'}(u,v)\le(1+\epsilon)\cdot d_G(u,v)$. (Here $d_G(u,v)$ stands for the distance between u and v in G.) We often refer to the edge set H of G_H as the *hopset.* The parameter β is called the *hopbound* of the hopset.

Hopsets are a fundamental graph-algorithmic construct. They are extremely useful for computing approximate shortest paths and distances and for routing problems in numerous computational settings, in which computing shortest paths with a limited number of hops is significantly easier than computing shortest paths with no limitation on the number of hops. A partial list of these settings includes distributed,

[*]Received by the editors January 23, 2018; accepted for publication (in revised form) June 6, 2019; published electronically August 29, 2019. A preliminary version of this paper [EN16a] appeared in FOCS'16.
https://doi.org/10.1137/18M1166791
Funding: The first author was supported by ISF grant (724/15). The second author was supported in part by ISF grant (523/12) and BSF grant 2015813.

[†]Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel (elkinm@cs.bgu.ac.il, neimano@cs.bgu.ac.il).

parallel, streaming, and centralized dynamic models.

Hopsets were explicitly introduced in Cohen's seminal paper [Coh00], which first appeared in the proceedings of STOC'94. Implicit constructions of hopsets had already been given in the beginning of the 1990s by Ullman and Yannakakis [UY91], Klein and Subramanian [KS97], Cohen [Coh97], and Shi and Spencer [SS99]. Cohen [Coh00] showed that for any parameters $\epsilon > 0$ and $\kappa = 1, 2, \ldots,$ and any n-vertex graph G, there exists a (β, ϵ)-hopset H with $|H| = \tilde{O}(n^{1+1/\kappa})$ edges,[1] where the hopbound β is polylogarithmic in n. Specifically, it is given by $\beta = (\frac{\log n}{\epsilon})^{O(\log \kappa)}$. Algorithmically, she showed that given an additional parameter $\rho > 0$, (β, ϵ)-hopsets with

$$\beta_{Coh} = \left(\frac{\log n}{\epsilon}\right)^{O((\log \kappa)/\rho)} \tag{1}$$

can be computed in $O(|E| \cdot n^{\rho})$ time in the centralized model of computation, and in $O(\beta) \cdot \text{polylog}(n)$ PRAM time, with $O(|E| \cdot n^{\rho})$ work. She used these hopsets' constructions to devise efficient parallel algorithms for computing $S \times V$ $(1+\epsilon)$-approximate shortest paths (henceforth, $(1+\epsilon)$-ASP). Until now, her results for these problems were the state of the art in this context for over two decades.

Despite being a major breakthrough, Cohen's hopsets leave much to be desired. Indeed, until very recently, the only general lower bound applicable to them is that of [CG06] (based on [Yao82, AS87]), asserting that there exist n-vertex graphs for which any (β, ϵ)-hopset for $0 \le \epsilon < 1$ requires $\Omega(n \cdot \log^{(\lfloor \beta/2 \rfloor)} n)$ edges, where $\log^{(t)} n$ stands for a t-iterated logarithm. Cohen [Coh00] herself wrote the following in the introduction of her paper):

> One intriguing issue is the *existence* question of sparse hopsets with certain attributes. In addition, we would like to construct them efficiently. [Italics in original.]

The same motif repeats itself in the concluding section of her paper, where she writes as follows:

> We find the existence of good hopsets to be an intriguing research problem on its own right.

Following Cohen's work, numerous additional applications of hopsets were discovered, and some new constructions of hopsets were presented. Most notably, Bernstein [Ber09] and Henzinger, Krinninger, and Nanongkai [HKN14] devised new constructions of hopsets and used them for maintaining approximate shortest paths in dynamic centralized setting. Nanongkai [Nan14] and Henzinger, Krinninger, and Nanongkai [HKN16] used hopsets for computing approximate shortest paths in the distributed and streaming settings. Lenzen and Patt-Shamir [LP15] and the authors of the current paper (and its preliminary version [EN16a]) used them for compact routing. Miller et al. [MPVX15] devised new constructions of hopsets and used them for approximate shortest paths in the PRAM setting. The (β, ϵ)-hopset of [Ber09] has hopbound $\beta = O((1/\epsilon)^{\kappa} \cdot \log n)$, and size $O(\kappa \cdot n^{1+1/\kappa} \cdot \log \Lambda)$, for any integer parameter $\kappa \ge 1$, where Λ is the aspect ratio of the graph.[2] The hopsets of [HKN14, HKN16] have hopbound $2^{\tilde{O}_{\epsilon}(\sqrt{\log n})}$ and size $n \cdot 2^{\tilde{O}(\sqrt{\log n})} \cdot \log \Lambda$. The hopsets of Miller et al. [MPVX15] have hopbound at least $\Omega(n^{\alpha})$, for a constant $\alpha > 0$, and linear size.

[1]The notation $\tilde{O}(f(n))$ stands for $O(f(n) \cdot \log^{O(1)} f(n))$.

[2]The aspect ratio of a graph G is defined by the ratio of the largest distance to the smallest distance in G.

TABLE 1

Comparison between (β, ϵ)-hopsets (neglecting the dependency on ϵ). We note that the hopsets of [Ber09, HKN14, HKN16, MPVX15] *were designed for certain computational models (i.e., dynamic, streaming, distributed). The second row of our results follows from the first by setting $\kappa = \log n$.*

Reference	Size	Hopbound	Run-time
[Ber09]	$O(\kappa \cdot n^{1+1/\kappa} \cdot \log \Lambda)$	$O(2^\kappa \cdot \log n)$	
[HKN14, HKN16]	$n \cdot 2^{\tilde{O}(\sqrt{\log n})} \cdot \log \Lambda$	$2^{\tilde{O}(\sqrt{\log n})}$	
[MPVX15]	$O(n)$	n^α, $(\alpha = \Omega(1))$	
[Coh00]	$O(n^{1+\frac{1}{\kappa}} \cdot \log n)$	$(\log n)^{O(\frac{\log \kappa}{\rho})}$	$\|E\| \cdot n^\rho$
This paper	$O(n^{1+\frac{1}{\kappa}} \cdot \log n)$	$\left(\log \kappa + \frac{1}{\rho}\right)^{\log \kappa + \frac{1}{\rho} + O(1)}$	$\|E\| \cdot n^\rho$
	$O(n \log n)$	$(\log \log n + 1/\rho)^{\log \log n + O(1/\rho)}$	$\|E\| \cdot n^\rho$

We will discuss the results of [Ber09, HKN14, Nan14, HKN16, MPVX15] in greater detail in what follows. However, all of these results fail to address the fundamental challenge of Cohen [Coh00] concerning the existence and efficient constructability of hopsets that are strictly and substantially superior to those devised in [Coh00]. In this paper we build such hopsets. Specifically, for any $\epsilon > 0$, $\kappa = 1, 2 \ldots$, and any n-vertex graph $G = (V, E)$, we show that there exists a (β, ϵ)-hopset with $\beta = (\frac{\log \kappa}{\epsilon})^{\log \kappa}$, and $O(n^{1+1/\kappa} \cdot \log n)$ edges. Hence, these hopsets *simultaneously* exhibit arbitrarily small constant approximation factor $1 + \epsilon$, arbitrarily close to 1 constant exponent $1 + 1/\kappa$ of the hopset's size, and *constant* hopbound β. In all previous hopsets' constructions, the hopbound was at least *polylogarithmic* in n, in all regimes. Moreover, we devise efficient algorithms to build our hopsets in various computational models. Specifically, given a parameter $\rho > 0$ that controls the running time, our centralized algorithm constructs a (β, ϵ)-hopset with $O(n^{1+1/\kappa} \cdot \log n)$ edges in expected $O(|E| \cdot n^\rho)$ time, with

$$\beta = O\left(\frac{1}{\epsilon} \cdot (\log \kappa + 1/\rho)\right)^{\log \kappa + 1/\rho + O(1)} . \tag{2}$$

Again, these hopsets simultaneously exhibit arbitrarily small constant approximation $1 + \epsilon$, arbitrarily close to 1 hopset's size exponent $1 + 1/\kappa$, arbitrarily close to $O(|E|)$ running time (in the sense of $O(|E| \cdot n^\rho)$ for an arbitrarily small constant $\rho > 0$), and still the hopbound β of the constructed hopset remains *constant*!

After the publication of a preliminary version of this work, [ABP16] showed the following lower bound. For any positive integer k and $\epsilon > 1/n^{o(1)}$, any (β, ϵ)-hopset with $n^{1+1/(2^k-1)-\delta}$ edges, for $\delta > 0$, has $\beta = \Omega(1/(2^k \epsilon))^k$. Setting $\kappa = 2^k - 1$, this implies that the dependence on ϵ in our construction cannot be improved by more than an $O(1/k)$ term in the exponent.

As mentioned above, in [Coh00] (the previous state of the art), with the same approximation factor, hopset size, and running time, the hopbound behaves as given in (1). Hence our result is stronger than that of [Coh00] in a number of ways. First, the hopbound β_{Coh} is at least polylogarithmic, while ours is constant. Second, the exponent $O((\log \kappa)/\rho)$ of β_{Coh} is substantially larger than the exponent $\log \kappa + 1/\rho + O(1)$ in our β. The parameters of our hopset strictly dominate those of the hopsets of [Ber09, HKN14, HKN16]. The hopset construction of [MPVX15] is incomparable to ours, as our hopbound is much better than $n^{\Omega(1)}$ of [MPVX15], but our hopset's size is at least $\Omega(n \log n)$, while the hopset of [MPVX15] has linear size. See Table 1 for a concise comparison of existing hopsets' constructions.

1.2. Hopsets in parallel, streaming, and distributed models. We also devise efficient parallel, distributed, and streaming algorithms for constructing hopsets with constant hopbound.

1.2.1. Hopsets in the streaming model. In the streaming model, the only previously known algorithm for constructing hopsets was that of [HKN16]. Using $2^{\tilde{O}_\epsilon(\sqrt{\log n})} \cdot \log \Lambda$ passes over the stream, and $n \cdot 2^{\tilde{O}(\sqrt{\log n})} \cdot \log \Lambda$ space, their algorithm produces a hopset with hopbound $\beta = 2^{\tilde{O}_\epsilon(\sqrt{\log n})}$ and size $n \cdot 2^{\tilde{O}(\sqrt{\log n})} \cdot \log \Lambda$. Our streaming algorithm constructs a hopset using $n^{O(\rho)}\beta \cdot \log^2 n$ passes over the stream, with

$$\beta = O\left(\frac{\kappa(\log \kappa + 1/\rho)}{\epsilon}\right)^{\log \kappa + 1/\rho}; \tag{3}$$

i.e., it is independent of n. The expected size of the hopset is $O(n^{1+1/\kappa} \cdot \log n)$, and it uses space $O(n^{1+1/\kappa} \cdot \log^2 n \cdot \beta)$. Also, by setting $\kappa = \Theta(\log n)$, $\rho = \sqrt{\frac{\log \log n}{\log n}}$,[3] our result strictly dominates that of [HKN14, HKN16]; the hopbound and number of passes are essentially the same, while our space usage and hopset size are significantly better.

1.2.2. Hopsets in the PRAM model. In the PRAM model, multiple processors are connected to a single memory block, and the operations are performed in parallel by these processors. We will be concerned mostly with the Exclusive Read Exclusive Write (EREW) PRAM model, which allows only a single processor to access any memory cell at any given round. The *running time* is measured by the number of rounds, and the *work* is measured by the number of processors.

In this model, Klein and Subramanian [KS97] and Shi and Spencer [SS99] (implicitly) devised algorithms for constructing exact ($\epsilon = 0$) hopsets with hopbound $\beta = O(\sqrt{n})$ of linear size $O(n)$, with running time $O(\sqrt{n} \cdot \log n)$, and with $O(|E| \cdot \sqrt{n})$ work. Cohen [Coh00] constructed (β, ϵ)-hopsets with size $n^{1+1/\kappa} \cdot (\log n)^{O((\log \kappa)/\rho)}$, with hopbound β_{Coh} given by (1), in time $(\frac{\log n}{\epsilon})^{O((\log \kappa)/\rho)}$, using $O(|E| \cdot n^\rho)$ work. Her κ and ρ are restricted by $\kappa, 1/\rho = O(\log \log n)$, and thus the resulting hopset is never sparser than $n \cdot 2^{O(\frac{\log n}{\log \log n})}$.

Miller et al. [MPVX15] devised two constructions of linear-size (β, ϵ)-hopsets but with very large β. One has $\beta = O_\epsilon(n^{\frac{4+\alpha}{4+2\alpha}})$, and running time given by the same expression, and work $O(|E| \cdot \log^{3+\alpha} n)$, for a parameter $\alpha \geq 0$. Another has $\beta = n^\alpha$, for a constant α, and running time given by the same expression, and work $O(|E| \cdot \log^{O(1/\alpha)} n)$.

Our algorithm has two regimes. In the first regime it constructs (β, ϵ)-hopsets with $\beta = (\frac{\log n}{\epsilon})^{\log \kappa + 1/\rho + O(1)}$, with expected size $O(n^{1+1/\kappa} \cdot \log n)$, in time $(\frac{\log n}{\epsilon})^{\log \kappa + 1/\rho + O(1)}$, using $O(|E| \cdot n^\rho)$ work. This result strictly improves upon Cohen's hopset [Coh00], as the exponent of β and of the running time in the latter is $O((\log \kappa)/\rho)$, instead of $\log \kappa + 1/\rho + O(1)$ as in our case. Also, the size of our hopset is smaller than that of [Coh00] by a factor of $O(\log n)^{O((\log \kappa)/\rho)}$.

In the second regime, our PRAM algorithm computes a hopset with constant (i.e., independent of n) hopbound β but in larger parallel time. See Table 2 for a concise comparison of available PRAM algorithms.

[3]Note that for such a choice of ρ, it is best to take κ as large as possible to reduce the hopset size, since it can only affect the constant factor in the exponent of β.

Table 2

Comparison between (β,ϵ)-hopsets in the PRAM model (neglecting the dependency on ϵ). The hopsets of [KS97, SS99] *provide exact distances. The third row of our results follows from the second one by setting $\kappa = \log n$.*

Reference	Size	β = hopbound	Time	Work
[KS97, SS99]	$O(n)$	$O(\sqrt{n})$	$O(\sqrt{n}\log n)$	$O(\|E\| \cdot \sqrt{n})$
[MPVX15]	$O(n)$	$O(n^{\frac{4+\alpha}{4+2\alpha}})$	$O(n^{\frac{4+\alpha}{4+2\alpha}})$	$O(\|E\| \cdot \log^{3+\alpha} n)$
	$O(n)$	$O(n^{\alpha})$ $(\alpha \geq \Omega(1))$	$O(n^{\alpha})$	$O(\|E\| \cdot \log^{O(1/\alpha)} n)$
[Coh00]	$n^{1+1/\kappa} \cdot (\log n)^{O(\frac{\log \kappa}{\rho})}$	$(\log n)^{O(\frac{\log \kappa}{\rho})}$	$(\log n)^{O(\frac{\log \kappa}{\rho})}$	$O(\|E\| \cdot n^{\rho})$
This paper	$O(n^{1+\frac{1}{\kappa}} \cdot \log n)$	$(\log n)^{\log \kappa + \frac{1}{\rho} + O(1)}$	$(\log n)^{\log \kappa + \frac{1}{\rho} + O(1)}$	$O(\|E\| \cdot n^{\rho})$
	$O(n^{1+\frac{1}{\kappa}} \cdot \log n)$	$\left(\frac{\log \kappa + \frac{1}{\rho}}{\zeta}\right)^{\log \kappa + O(\frac{1}{\rho})}$	$O(n^{\zeta}) \cdot \beta$	$O(\|E\| \cdot n^{\rho+\zeta})$
	$O(n \cdot \log n)$	$\left(\frac{\log\log n + 1/\rho}{\zeta}\right)^{\log\log n + O(1/\rho)}$	$O(n^{\zeta}) \cdot \beta$	$O(\|E\| \cdot n^{\rho+\zeta})$

1.3. Hopsets in distributed models.

There are two distributed models in which hopsets are studied in the literature [HKN14, Nan14, HKN16, LP15, EN16b]. These are the Congested Clique model and the CONGEST model. In both models every vertex of an n-vertex graph $G = (V, E)$ hosts a processor, and the processors communicate with one another in discrete rounds via short messages. Each message is allowed to contain the identity of a vertex or an edge, and an edge weight, or anything else of no larger (up to a fixed constant factor) size.[4] On each round each vertex can send possibly different[5] messages to its neighbors. The local computation is assumed to require zero time, and we are interested in algorithms that run for as few rounds as possible. (The number of rounds is called the *running time*.) In the Congested Clique model, we assume that all vertices are interconnected via direct edges, but there might be some other weighted undirected graph $G' = (V, E', \omega)$, $E' \subseteq E = \binom{V}{2}$, embedded in the clique G, for which we want to compute a hopset. In the CONGEST model, every vertex can send messages only to its G-neighbors, but we also assume that there is an embedded "virtual" graph $G' = (V', E', \omega)$, $V' \subseteq V$, known locally to the vertices. (At the beginning of the computation, every vertex $u \in V$ knows whether $u \in V'$, and if this is the case, then every vertex also knows the identities of its G'-neighbors.) We remark that the assumption of embedded graph G' in the CONGEST model appears in previous papers on computing hopsets in distributed setting, that is, in [HKN14, Nan14, HKN16, LP15, EN16b]. This assumption is motivated by distributed applications of hopsets, i.e., approximate shortest paths computation, distance estimation, and routing, which require a hopset for a virtual graph embedded in the underlying network in the above way.

Henzinger, Krinninger, and Nanongkai [HKN16] devised an algorithm for constructing hopsets in the Congested Clique model. Their hopset has hopbound $\beta = 2^{\tilde{O}_\epsilon(\sqrt{\log n})}$ and size $n \cdot 2^{\tilde{O}(\sqrt{\log n})} \cdot \log \Lambda$, where Λ is the aspect ratio of the embedded graph. The running time of their algorithm is $2^{\tilde{O}_\epsilon(\sqrt{\log n})} \cdot \log \Lambda$.

Our algorithm, for parameters $\epsilon > 0$, $\rho > 0$, $\kappa = 2, 3, \ldots$, computes a hopset with

$$\beta = O\left(\frac{\log \kappa + 1/\rho}{\epsilon \cdot \rho}\right)^{\log \kappa + O(1/\rho)}, \tag{4}$$

[4]Typically, in the CONGEST model, only messages of size $O(\log n)$ bits are allowed, but edge weights are restricted to be at most polynomial in n. Our definition is geared toward capturing a more general situation, when there is no restriction on the aspect ratio. Hence results achieved in our more general model are more general than previous ones.

[5]In the *Broadcast Congested Clique* and *Broadcast CONGEST* models, these messages must be identical.

with expected size $O(n^{1+1/\kappa} \cdot \log n)$, in $O(n^\rho \cdot \beta^2)$ rounds.[6]

Comparing our result to that of [HKN16], we first note that our hopset achieves a constant (i.e., independent of n) hopbound. Second, by setting $\kappa = \Theta(\log n)$, $\rho = \sqrt{\frac{\log\log n}{\log n}}$, we can have our hopbound and running time equal to $2^{\tilde{O}_\epsilon(\sqrt{\log n})}$, i.e., roughly the same as, but in fact slightly better than, the respective bounds of [HKN16]. Our hopset's size then becomes $O(n \cdot \log n)$, i.e., much closer to linear than $n \cdot 2^{\tilde{O}(\sqrt{\log n})}$ of the hopset of [HKN16].

The situation is similar in the CONGEST model. Denote by $m = |V'|$ the size of the vertex set of the embedded graph G'. The algorithm of [HKN16] computes a hopset with the same hopbound and size as in the Congested Clique model (with n replaced by m), and it does so in $(D + m) \cdot 2^{\tilde{O}_\epsilon(\sqrt{\log m})} \cdot \log \Lambda$ time, where Λ is the aspect ratio of G', and D is the hop-diameter of G.[7] Our algorithm computes in $O((D + m^{1+\rho}) \cdot \beta \cdot m^\rho)$ time a hopset with a (constant) hopbound as given by (4) and with expected size $O(m^{1+1/\kappa} \cdot \log m)$. (See Corollary 4.10, which in fact gives stronger, but more complicated, bounds.) Again, our hopset can have constant hopbound, while that of [HKN16] is $2^{\tilde{O}_\epsilon(\sqrt{\log m})}$. Also, by setting $\kappa = \Theta(\log m)$, $\rho = \sqrt{\frac{\log\log m}{\log m}}$, we obtain a result which strictly dominates that of [HKN16].[8]

1.4. Applications. Our algorithms for constructing hopsets also give rise to improved algorithms for the problems of computing $(1+\epsilon)$-approximate shortest distances (henceforth, $(1+\epsilon)$-ASD) and paths (henceforth, $(1+\epsilon)$-ASP). In all settings, we consider a subset $S \subseteq V$ of origins, and we are interested in distance estimates or in approximate shortest paths for pairs in $S \times V$. Denote $s = |S|$.

Our PRAM algorithm for the $(1 + \epsilon)$-ASP problem has running time $O(\frac{\log n}{\epsilon})^{\log \kappa + 1/\rho + 1}$ and uses $O(|E| \cdot (n^\rho + s))$ work. Cohen's algorithm [Coh00] for the same problem has (parallel) running time $O(\frac{\log n}{\epsilon})^{O((\log \kappa)/\rho)}$ and has the same work complexity as our algorithm. Hence, both our algorithm and Cohen's algorithm achieve polylogarithmic time and near-optimal work complexity, but the exponent $\log \kappa + 1/\rho + 1$ of the logarithm in our result is significantly smaller than in Cohen's. The latter is $O((\log \kappa)/\rho)$.

In the Congested Clique model, Henzinger, Krinninger, and Nanongkai [HKN16] used hopsets to design an algorithm that computes single-source $(1 + \epsilon)$-ASP in $2^{\tilde{O}(\sqrt{\log n})}$ time. Applying their algorithm separately from each source results in time $s \cdot 2^{\tilde{O}(\sqrt{\log n})}$. For a comparison, our algorithm computes $S \times V$ $(1 + \epsilon)$-ASP for $s = n^{\Omega(1)}$, in $s \cdot (1/\epsilon)^{O(1)}$ time. We remark that an algorithm of Censor-Hillel et al. [CKK+15] computes all-pairs ASP in $O(n^{0.158})$ time. Hence our result here improves the state of the art for the range $n^{\Omega(1)} \leq s \leq o(n^{0.158})$.

In the distributed CONGEST model (see section 1.3 for its definition), the hopset-based algorithm of [HKN16] computes single-source $(1+\epsilon)$-ASP in $(D+\sqrt{n})\cdot 2^{\tilde{O}(\sqrt{\log n})}$.

[6]Our hopsets come in two different varieties. One variety consists of *ordinary* hopsets, and the other consists of *path-reporting* hopsets. The latter are used in applications in which the actual paths, and not just distance estimates, need to be reported. Generally, the parameters of path-reporting hopsets are slightly worse than those of their ordinary counterparts. All bounds that we provide in the introduction apply to both path-reporting and ordinary hopsets. Even stronger bounds for ordinary hopsets are given in the technical part of the paper.

[7]The hop-diameter of a graph is the maximum hop-distance between two vertices. The hop-distance between a pair u, v of vertices is the minimal number of hops in a path between them.

[8]Modulo some minor modifications (see section 4 for details), our results apply even in the more restricted Broadcast Congested Clique and Broadcast CONGEST models.

$\log \Lambda$ time. Using it naively for $S \times V$ $(1+\epsilon)$-ASP results in a running time of $(D + s \cdot \sqrt{n}) \cdot 2^{\tilde{O}(\sqrt{\log n})} \cdot \log \Lambda$. Using our hopsets we solve this problem in $(D + \sqrt{n \cdot s}) \cdot 2^{\tilde{O}(\sqrt{\log n})} \cdot \log \Lambda$ time. Whenever $s = n^{\Omega(1)}$, we use our hopset with different parameters, and our running time becomes $\tilde{O}(D + \sqrt{n \cdot s}) \cdot \log \Lambda$. We also remark that if large messages of size n^{ρ} are allowed for some constant $\rho > 0$, then we can compute *single-source* $(1+\epsilon)$-ASP in $\tilde{O}(D + \sqrt{n}) \cdot \log \Lambda$ time, and in the Congested Clique model we obtain *single-source* $(1+\epsilon)$-ASP in $\mathrm{polylog}(n) \cdot \log \Lambda$ time. Also, with minor modifications (see section 4 for details), these results apply to the Broadcast Congested Clique and Broadcast CONGEST model.

In the streaming model, Henzinger, Krinninger, and Nanongkai [HKN16] devised a single-source $(1+\epsilon)$-ASP streaming algorithm with $2^{\tilde{O}(\sqrt{\log n})} \cdot \log \Lambda$ passes that uses $n \cdot 2^{\tilde{O}(\sqrt{\log n})} \cdot \log \Lambda$ space. To the best of our knowledge, the best-known streaming $S \times V$ $(1+\epsilon)$-ASP algorithm with this space requirement is to run the algorithm of [HKN16] for each source separately, one after another. The resulting number of passes is $s \cdot 2^{\tilde{O}(\sqrt{\log n})} \cdot \log \Lambda$. Our algorithm for this problem builds a hopset, whose parameters depend on s. As a result, our algorithm has an improved number of passes, particularly when s is large (we also avoid the dependence on Λ). Our space usage is only $\tilde{O}(n)$ for $(1+\epsilon)$-ASD. Alternatively, by allowing larger space we can get $O(s)$ passes for any fixed $\epsilon > 0$ whenever $s = n^{\Omega(1)}$. See Theorem 5.8 for the precise results.

1.5. Subsequent work. After the publication of a preliminary version of our paper [EN16a], Becker et al. [BKKL16], using a completely different approach, devised an algorithm for single-source $(1+\epsilon)$-ASP in $\mathrm{polylog}(n) \cdot \log \Lambda$ time in the Broadcast Congested Clique model, and in $\tilde{O}(D + \sqrt{n}) \cdot \log \Lambda$ time in the Broadcast CONGEST model, using *short* messages. They also devised a streaming single-source $(1+\epsilon)$-ASP algorithm that uses $\tilde{O}(n) \cdot \log \Lambda$ space and $\mathrm{polylog}(n) \cdot \log \Lambda$ passes.

In another development subsequent to the conference version of our current work [EN16a], in [EN17] we, and, independently in [HP19], Huang and Pettie, devised improved constructions of hopsets with constant hopbound. Our constructions in the current paper are based on the superclustering-and-interconnection approach introduced by Elkin and Peleg [EP01, EP04] for building near-additive spanners. On the other hand, the subsequent constructions of [EN17, HP19] are based on Thorup and Zwick's constructions of near-additive spanners from [TZ06]. In particular, our hopset has size $\Omega(n \log n)$ for any setting of the parameters, while the hopsets of [EN17, HP19] can achieve linear size.

1.6. Overview of techniques. In this section we sketch the main ideas used in the hopsets' constructions of [Coh00], in [Ber09, HKN14, HKN16], and in our constructions.

Cohen's algorithm [Coh00] starts with constructing a pairwise cover $\mathcal{C}$ of the input graph [Coh93, ABCP93]. This is a collection of small-diameter clusters, with limited intersections, and such that for any path π of length at most W, for a parameter W, all vertices of π are clustered in the same cluster. For each cluster $C \in \mathcal{C}$, the algorithm inserts into the hopset a star $\{(r_C, u) \mid u \in C\}$ connecting the center r_C of C with every other vertex of C. In addition, it adds to the hopset edges connecting centers of large clusters with one another, and it recurses on small clusters.

This powerful approach has a number of limitations. First, the collection of star edges itself contains $O(\kappa \cdot n^{1+1/\kappa})$ edges, where κ is a parameter, which controls the hopset's size. Each level of the recursion increases the exponent of the number of edges in the hopset by roughly a factor of $\kappa \cdot n^{1/\kappa}$, and as a result, the hopset of [Coh00] cannot be very sparse. Second, each distance scale $[2^k, 2^{k+1}]$, $k = 0, 1, 2, \ldots$, requires a separate hopset, and as a result, a separate collection of covers. This increases the hopset's size even further, but in addition, a hopset of scale $k+1$ in Cohen's algorithm is computed using hopsets of all the lower scales. This results in accumulation of error; i.e., if the error incurred by each hopset computation is $1+\epsilon$, the approximation factor of the ultimate hopset becomes $(1+\epsilon)^{\log \Lambda}$. After rescaling $\epsilon' = \epsilon \log \Lambda$, one obtains a hopbound of roughly $(1/\epsilon)^{\ell} = O(\frac{\log \Lambda}{\epsilon'})^{\ell}$, where ℓ is the number of levels of the recursion. As a result, the hopbound in [Coh00] is at least polylogarithmic in n.

Another line of works [Ber09, HKN14, HKN16] is based on the distance oracles and emulators[9] of Thorup and Zwick [TZ01, TZ06]. They build a hierarchy of sampled sets $V = A_0 \supset A_1 \supset \cdots \supset A_{k-1} \supset A_k = \emptyset$, where for any $i = 1, \ldots, k-1$, each vertex $v \in A_{i-1}$ joins A_i independently at random with probability $n^{-1/k}$. For each vertex $v \in V$, one can define the *TZ cluster* $C(v)$ by $C(v) = \bigcup_{i=0}^{k-1} \{u \mid u \in A_i, d_G(u,v) < d_G(u, A_{i+1})\}$. Thorup and Zwick [TZ06] showed that for *unweighted* graphs, $H = \{(v,u) \mid u \in C(v)\}$ is a $(1+\epsilon, \beta)$-emulator with $O(k \cdot n^{1+1/k})$ edges, and $\beta = O(1/\epsilon)^k$. (Observe that β here is the additive error of the emulator. We use the same notation to emphasize a connection between constructions of hopsets and emulators, where the emulator's additive error corresponds to the hopbound of the hopset.) Bernstein and others [Ber09, HKN14, HKN16] showed that a closely related construction provides a hopset. Specifically, [Ber09] applied a truncated version of this construction on all distances scales to obtain a $(1+\epsilon, \beta)$-hopset with the same parameters as those of the emulator of [TZ06] (the size is multiplied by $\log \Lambda$). In the construction of [HKN16] (which required efficient distributed construction), the authors set $k = \Theta(\sqrt{\log n})$ and build TZ clusters with respect to $2^{\tilde{O}(\sqrt{\log n})}$-limited distances. This results in a so-called *restricted hopset*, i.e., a hopset H_1 that handles $2^{\tilde{O}(\sqrt{\log n})}$-limited distances. Consequently, all nearly shortest paths with N hops in G, for some N, now translate into nearly shortest paths (incurring an approximation factor of $1+\epsilon$ of H_1) with $\frac{N}{2^{\tilde{O}(\sqrt{\log n})}}$ hops in $G \cup H_1$. Nanongkai [Nan14] called this operation a *hop reduction*, as this essentially reduces the maximum number of hops from $n-1$ to $n/2^{\sqrt{\log n}}$. Then the hop reduction is repeated $\sqrt{\log n}$ times until a hopset for all distances is constructed.

This scheme appears to be incapable of providing very sparse hopsets, as just the invocation of Thorup and Zwick's algorithm with $\kappa = \Theta(\sqrt{\log n})$ gives $n \cdot 2^{\Omega(\sqrt{\log n})}$ edges. In addition, the repetitive application of hop reduction blows up the hopbound to $2^{\Omega(\sqrt{\log n})}$; i.e., the large hopbound appears to be inherent in this approach.

Our approach combines techniques from [EP04] for constructing $(1+\epsilon, \beta)$-spanners in unweighted graphs with those of [Coh00], and with a set of new ideas. To build their spanners, Elkin and Peleg [EP04] start by constructing an Awerbuch–Peleg partition $\mathcal{P} = \{C_1, \ldots, C_q\}$ [AP90] of the vertex set V into disjoint clusters of small diameter. (This partition satisfies an additional property, which is irrelevant to this discussion.) It then sets a distance threshold δ_1 and a degree threshold deg_1. Every cluster $C \in \mathcal{P}$ that has at least deg_1 unclustered clusters $C' \in \mathcal{P}$ in its δ_1-vicinity (i.e., the clusters of distance at most δ_1 from C) creates a supercluster that contains C and these clusters. (At the beginning all clusters are unclustered. Those that

[9]A graph $G' = (V', E', \omega')$ is called a $(1+\epsilon, \beta)$-*emulator* of an unweighted graph $G = (V, E)$, if $V \subseteq V'$, and for every pair of $u, v \in V$ of vertices, it holds that $d_{G'}(u,v) \le d_G(u,v) \le (1+\epsilon) d_G(u,v) + \beta$. If G' is a subgraph of G, then G' is called a $(1+\epsilon, \beta)$-*spanner* of G.

join a supercluster become clustered.) This *superclustering step* continues until no additional superclusters can be formed. All remaining unclustered clusters that are at pairwise distance at most δ_1 are now interconnected by shortest paths in the spanner. This is the *interconnection* step of the algorithm. Together the superclustering and interconnection steps form a single *phase* of the algorithm. Once the first phase is over, the same process (interleaving superclustering and interconnection) is repeated with new distance and degree thresholds δ_2 and deg_2, respectively, on the set of superclusters of the previous phase. The sequences $\delta_1, \delta_2, \ldots$ and $deg_1, deg_2, \ldots$ are set carefully to optimize the parameters of the resulting spanner.

The basic variant of our hopset construction considers each distance scale $[2^k, 2^{k+1}]$, $k = 0, 1, 2, \ldots$, separately (w.l.o.g we assume all weights are at least 1). Instead of the Awerbuch–Peleg partition, we use the partition $\mathcal{P} = \{\{v\} \mid v \in V\}$ into single vertices. We set the distance threshold δ_1 to roughly $2^k/\beta = 2^k/(1/\epsilon)^\ell$, where ℓ is the number of phases of the algorithm, and raise it by a factor of $1/\epsilon$ on every phase. The degree thresholds are also set differently from the way they were set in [EP04]. This is because, intuitively, the hopset contains fewer edges than the spanner, as the hopset can use a single edge, while a spanner needs to use an entire path. Hence the degree sequence that optimizes the hopset's size is different from the one that optimizes the spanner's size.

The superclustering and interconnection steps are also implemented in a different way than in [EP04] because of efficiency considerations. The algorithm of [EP04] is not particularly efficient, and there are no known efficient streaming, distributed, or parallel implementations of it.[10] On a high level, our superclustering and interconnection steps are conducted as follows. On phase i we sample clusters $C \in \mathcal{P}$ independently at random with probability $1/deg_i$. The sampled clusters create superclusters of radius δ_i around them. Then the unclustered clusters of $\mathcal{P}$ that are within distance $\delta_i/2$ from one another are interconnected by hopset edges. Note that here the superclustering distance threshold and the interconnection distance thresholds differ by a factor of 2. This ensures that all of these steps can be efficiently implemented. We also show that the overhead introduced by this factor to the resulting parameters of our hopset is insignificant.

Our superclustering-and-interconnection approach to constructing hopsets was not previously used in the literature [Coh00, Ber09, HKN14, Nan14, HKN16]. Rather it is adapted from [EP04]. The latter paper deals with nearly additive spanners for unweighted graphs. We believe that our main technical contribution consists of realizing that the technique of [EP04] can be instrumental for constructing drastically improved hopsets, and adapting that technique from the context of near-additive spanners for unweighted graphs to the context of hopsets for general graphs.

To construct a hopset for *all* scales, in the centralized setting we simply take the union of the single-scale hopsets. In parallel, distributed, and streaming settings, however, Dijkstra explorations for large scales could be too expensive. To remedy this, we rely on lower-scale hopsets for computing the current scale, as in Cohen's algorithm. On the other hand, a naive application of this approach results in polylogarithmic hopbound β. To achieve *constant* (i.e., independent of n) hopbound β, we compute

[10]The algorithms of [Elk01, EZ06] that construct $(1+\epsilon, \beta)$-spanners in distributed and streaming settings are not based on the superclustering-and-interconnection technique. Rather they are based on a completely different approach reminiscent of that in [Coh00], i.e., they build covers and recurse in small clusters.

in parallel hopsets for many different scales, using the same lower-scale hopset for distance computations. This results in a much smaller accumulation of error than in Cohen's scheme, but it requires a larger running time. (Roughly speaking, computing a scale-t hopset using scale-s hopset, for $t > s$, requires time proportional to 2^{t-s}.) We carefully balance this increase in running time with other parameters to optimize the attributes of our ultimate hopset.

Finally, we need to replace the logarithmic dependence on the aspect ratio Λ with a logarithmic dependence on n. Cohen's results [Coh00] do not have this dependence, as they rely on a PRAM reduction of Klein and Subramanian [KS97] that replaces the dependence on the aspect ratio Λ by n. However, Klein and Subramanian [KS97] (see also [Coh97] for another analysis) analyzed this reduction for single-source distance estimation, while in the hopset's case we need to apply it to all pairs. The distributed and streaming hopsets' constructions [Ber09, HKN14, HKN16, Nan14] all have a dependence on $\log \Lambda$.

We develop a new analysis of Klein and Subramanian reduction that applies to the hopsets' scenario (see section 4). We also show that the reduction can be efficiently implemented in distributed and streaming settings.

1.7. Structure of the paper. Our main construction of hopsets, along with the centralized algorithm implementing it, is given in section 3.1. Sections 3.2–3.6 provide implementations of this construction in various computational models. In section 4 we eliminate the dependence on the aspect ratio from most of our bounds. In section 5 we use our hopsets to devise faster algorithms for computing approximate shortest paths and distances.

2. Preliminaries.

Let $G = (V, E)$ be a weighted graph on n vertices with diameter Λ. We shall assume throughout that edge weights are positive integers. For $C \subseteq V$, $G(C)$ is the induced graph on C. Let d_G be the shortest path metric on G, and let $d_G^{(t)}$ be the t-limited distance; that is, for $u, v \in V$, $d_G^{(t)}(u, v)$ is the minimal length of a path between u, v that contains at most t edges (set $d_G^{(t)}(u, v) = \infty$ if there is no such a path). Note that $d_G^{(t)}$ is not a metric. Also, denote by $\Gamma_G(v)$ the set of neighbors of v. If the graph G can be understood from the context, the subscript G is omitted.

3. Hopsets.

3.1. A centralized construction. Let $G = (V, E)$ be a weighted graph on n vertices with diameter Λ; we assume throughout the paper that the minimal distance in G is 1. Fix parameters $\kappa \geq 2$, $0 < \epsilon < 1/10$, and $1/\kappa \leq \rho \leq 1/2$. (For technical reasons, if $\kappa > \log n / \log\log n$, we require that $\rho < 1/3$.) Recall that κ governs the sparsity of our hopset, ϵ governs its stretch, and ρ governs the running time. The parameter β, which governs the hopbound, will be determined later as a function of $n, \Lambda, \kappa, \rho, \epsilon$. We build separately a hopset H_k for every distance range $(2^k, 2^{k+1}]$ for $k \leq \log \Lambda$. We will call such a hopset H_k a *single-scale* hopset.

Denote $\hat{R} = 2^{k+1}$. For $\hat{R} \leq \beta = (1/\epsilon)^\ell$, where ℓ is the number of levels of the construction (to be determined), an empty hopset $H_k = \emptyset$ suffices for pairs of distance in $(\hat{R}/2, \hat{R}]$. (Indeed, for $d_G(u, v) \leq \hat{R} \leq \beta$, we have $d_{G \cup H_k}^{(\beta)}(u, v) = d_G^{(\beta)}(u, v) = d_G(u, v)$ for $H_k = \emptyset$.) Hence we assume that $k > \log \beta - 1$, i.e., $\hat{R} > \beta$.

The algorithm initializes the hopset H_k as an empty set and proceeds in phases. It starts with setting $\hat{\mathcal{P}}_0 = \{\{v\} \mid v \in V\}$ to be the partition of V into singleton clusters. The partition $\hat{\mathcal{P}}_0$ is the input of phase 0 of our algorithm. More generally, $\hat{\mathcal{P}}_i$ is the input of phase i, for every index i in a certain appropriate range, which we

Algorithm 1 $H_k =$ `Single-Scale Hopset`$(G = (V, E), \kappa, \rho, \epsilon, k)$.

```
1: $H_k = \emptyset$;
2: $\hat{\mathcal{P}}_0 = \{\{v\} \mid v \in V\}$;
3: for $v \in V$ do
4:     $r_{\{v\}} = v$;
5: end for
6: $R_0 = 0$;
7: $i_0 = \lfloor \log(\kappa\rho) \rfloor$; $i_1 = i_0 + \left\lceil \frac{\kappa+1}{\kappa\rho} \right\rceil - 2$; $\ell = i_1 + 1$;
8: $\alpha = \epsilon^{\ell} \cdot 2^{k+1}$;
9: for $i = 0, 1, \ldots, \ell - 1$ do
10:     $\delta_i = \alpha/\epsilon^i + 4R_i$;
11:     $R_{i+1} = R_i + \delta_i$;
12:     $deg_i = \begin{cases} n^{2^i/\kappa}, & i \le i_0, \\ n^{\rho} & \text{otherwise}; \end{cases}$
13:     $(\hat{\mathcal{P}}_{i+1}, \mathcal{U}_i) =$ superclustering$(G, \hat{\mathcal{P}}_i, deg_i, \delta_i)$;
14:     interconnection$(G, \mathcal{U}_i, \delta_i)$;
15: end for
16: interconnection$(G, \hat{\mathcal{P}}_\ell, \delta_\ell)$;
17: return $H_k$;
```

Algorithm 2 $(\mathcal{P}, \mathcal{U}) =$ `superclustering`$(G, \hat{\mathcal{P}}, deg, \delta)$.

```
1: $\mathcal{U} = \emptyset$;
2: Create $\mathcal{S}$ by sampling each cluster in $\hat{\mathcal{P}}$ into $\mathcal{S}$ independently with probability
   $1/deg$;
3: for each $C \in \mathcal{S}$ do
4:     $\hat{C} \leftarrow C$;
5:     $r_{\hat{C}} = r_C$;
6: end for
7: for each $C' \in \hat{\mathcal{P}}$ do
8:     Let $C \in \mathcal{S}$ be the cluster minimizing $d_G(r_C, r_{C'})$ (breaking ties arbitrarily);
9:     if $d_G(r_C, r_{C'}) \le \delta$ then
10:         $H_k \leftarrow H_k \cup \{(r_C, r_{C'})\}$;
11:         $\hat{C} \leftarrow \hat{C} \cup C'$;
12:     else
13:         $\mathcal{U} \leftarrow \mathcal{U} \cup \{C'\}$;
14:     end if
15: end for
16: $\mathcal{P} = \{\hat{C} \ : \ C \in \mathcal{S}\}$;
17: return $(\mathcal{P}, \mathcal{U})$;
```

will specify in what follows.

Throughout the algorithm, all clusters C that we construct will be centered at designated centers r_C. In particular, each singleton cluster $C = \{v\} \in \hat{\mathcal{P}}_0$ is centered at v. We define $Rad(C) = \max\{d_{G(C)}(r_C, v) \mid v \in C\}$ and $Rad(\hat{\mathcal{P}}_i) = \max_{C \in \hat{\mathcal{P}}_i}\{Rad(C)\}$.

The algorithm for a single-scale hopset is succinctly described in Algorithm 1. The

Algorithm 3 `interconnection`$(G,\mathcal{U},\delta)$.

```
1: for any two clusters $C, C' \in \mathcal{U}$ such that $d_G(r_C, r_{C'}) \leq \delta/2$ do
2:     $H_k \leftarrow H_k \cup \{(r_C, r_{C'})\}$;
3: end for
```

superclustering and interconnection steps appear in Algorithm 2 and Algorithm 3, respectively.

All phases of our algorithm, except the last phase, consist of two steps. Specifically, these are the *superclustering* and the *interconnection* steps. The last phase contains only the interconnection step, and the superclustering step is skipped. We also partition the phases into two *stages*. The first stage consists of phases $0, 1, \ldots, i_0 = \lfloor \log(\kappa\rho) \rfloor$, and the second stage consists of all the other phases $i_0 + 1, \ldots, i_1$, where $i_1 = i_0 + \lceil \frac{\kappa+1}{\kappa\rho} \rceil - 2$, except the last phase $\ell = i_1 + 1$. The last phase will be referred to as the *concluding phase*.

Each phase i accepts as input two parameters: the distance threshold parameter δ_i, which determines the range of the Dijkstra explorations, and the degree parameter deg_i, which determines the sampling probability. The Dijkstra explorations are used to discover distances from the clusters' centers. The difference between stages 1 and 2 is that in stage 1 the degree parameter grows exponentially, while in stage 2 it is fixed. The distance threshold parameter grows at the same steady rate (increases by a factor of $1/\epsilon$) throughout the algorithm.

The distance thresholds' sequence is given by $\alpha = \epsilon^\ell \cdot \hat{R}$, $\delta_i = \alpha(1/\epsilon)^i + 4R_i$, where $R_0 = 0$ and $R_{i+1} = \delta_i + R_i = \alpha(1/\epsilon)^i + 5R_i$ for $i \geq 0$. It follows that $R_1 = \alpha$, and by estimating the recurrence we obtain $R_i \leq 2 \cdot \alpha \cdot (1/\epsilon)^{i-1}$. The degree sequence in the first stage of the algorithm is given by $deg_i = n^{2^i/\kappa}$ for $i = 0, 1, \ldots, i_0$. We then use $deg_i = n^\rho$ in all subsequent phases $i_0 + 1, \ldots, i_1$. Finally, in phase $\ell = i_1 + 1$ we perform just the interconnection step. Note that $\ell \geq 2$ since $\rho \leq 1/2$.

Next we take a closer look at the execution of phase i, $i = 0, 1, 2, \ldots, \ell - 1$. At the beginning of the phase we have a collection $\hat{\mathcal{P}}_i$ of clusters, of radius $2\alpha \cdot (1/\epsilon)^{i-1}$ for $i \geq 1$, and radius 0 for $i = 0$. (It will be shown in Claim 3.1 that $Rad(\hat{\mathcal{P}}_i) \leq R_i \leq 2\alpha \cdot (1/\epsilon)^{i-1}$ for all $i = 0, 1, \ldots, \ell$.) Each of these clusters is now sampled with probability $1/deg_i$, independently at random (iar). The resulting set of sampled clusters is denoted $\mathcal{S}_i$. We then initiate a single Dijkstra exploration in G rooted at the set $Roots = \{r_C \mid C \in \mathcal{S}_i\}$ of cluster centers of sampled clusters. The Dijkstra exploration is conducted to depth δ_i. Let F_i denote the resulting forest.

Let $C' \in \hat{\mathcal{P}}_i \setminus \mathcal{S}_i$ be a cluster with center $r_{C'}$ reached by the exploration, and let r_C, for some cluster $C \in \mathcal{S}_i$, be the cluster center such that $r_{C'}$ belongs to the tree of F_i rooted at r_C. We then add an edge $(r_C, r_{C'})$ of weight $\omega(r_C, r_{C'}) = d_G(r_C, r_{C'})$ into the hopset H_k, which we are now constructing. A supercluster $\hat{C}$ rooted at $r_{\hat{C}} = r_C$ is now created. It contains all vertices of C and of clusters C' as above. This completes the description of the superclustering step. The resulting set $\hat{\mathcal{S}}_i$ of superclusters becomes the next level partition $\hat{\mathcal{P}}_{i+1}$, i.e., we set $\hat{\mathcal{P}}_{i+1} \leftarrow \hat{\mathcal{S}}_i$.

CLAIM 3.1. *Fix any cluster $C \in \hat{\mathcal{P}}_i$ with center r_C. Then for any $u \in C$ there is a path in H_k of at most i edges from r_C to u of length at most R_i.*

Proof. The proof is by induction on i. The base case $i = 0$ holds because C is a singleton. Assume it holds for i, and fix any $\hat{C} \in \hat{\mathcal{P}}_{i+1}$ and $u \in \hat{C}$. Recall that $\hat{C}$ consists of a sampled cluster $C \in \mathcal{S}_i$, and clusters $C' \in \hat{\mathcal{P}}_i$ for which the Dijkstra exploration to range δ_i from r_C reached their center $r_{C'}$. Assume $u \in C'$ (the case where $u \in C$ is simpler). Then by induction there is a path of length at most R_i from $r_{C'}$ to u in H_k of i hops, and by construction we added the edge $(r_C, r_{C'})$ of weight $d_G(r_C, r_{C'})$ into the hopset H_k. This implies a path of $i+1$ hops and length at most

$$\delta_i + R_i = R_{i+1} \ . \qquad \square$$

Let $\hat{\mathcal{U}}_i$ denote the set of $\hat{\mathcal{P}}_i$ clusters which were not superclustered into $\hat{\mathcal{S}}_i$ clusters. These clusters are involved in the interconnection step. Specifically, each of the cluster centers r_C, $C \in \hat{\mathcal{U}}_i$, now initiates a separate Dijkstra exploration to depth $\frac{1}{2}\delta_i = \frac{1}{2}\alpha \cdot (1/\epsilon)^i + 2R_i$. All the vertices within distance $\frac{1}{2}\delta_i$ from r_C are *visited* by this exploration. For any cluster center $r_{C'}$ of a cluster $C' \in \hat{\mathcal{U}}_i$ such that $r_{C'}$ was discovered by an exploration originated at r_C, we now insert an edge $(r_C, r_{C'})$ into the hopset and assign it weight $\omega(r_C, r_{C'}) = d_G(r_C, r_{C'})$. This completes the description of the interconnection step.

LEMMA 3.2. *For any vertex $v \in V$, the expected number of explorations that visit v at the interconnection step of phase $0 \le i \le i_1$ is at most deg_i.*

Proof. For $0 \le i \le i_1$, assume that there are t clusters of $\hat{\mathcal{P}}_i$ within distance $\delta_i/2$ from v. If at least one of them is sampled to $\mathcal{S}_i$, then no exploration will visit v (since in the superclustering phase the sampled center will explore to distance δ_i, and thus all these t clusters will be superclustered into some cluster of $\hat{\mathcal{S}}_i$). The probability that none of them is sampled is $(1 - 1/deg_i)^t$, in which case we get that t explorations visit v, so the expectation is $t \cdot (1 - 1/deg_i)^t \le deg_i$ for any l. (To see the last inequality, note that $t \cdot (1 - 1/deg_i)^t \le t \cdot e^{-t/deg_i}$, so it suffices to show that $t/deg_i \cdot e^{-t/deg_i} \le 1$. Taking log, it becomes $\ln(t/deg_i) - t/deg_i \le 0$, which holds for all t, deg_i.) $\square$

A similar argument yields the following lemma (which will be needed in later sections).

LEMMA 3.3. *For any constant $c > 1$, with high probability (whp) at least $1 - 1/n^{c-1}$, for every vertex $v \in V$, at least one among the $deg_i \cdot c \cdot \ln n$ closest cluster centers $r_{C'}$ with $C' \in \hat{\mathcal{P}}_i$ to v is sampled, i.e., satisfies $C' \in \mathcal{S}_i$.*

We analyze the number of clusters in collections $\hat{\mathcal{P}}_i$ in the following lemma.

LEMMA 3.4. *Assuming $n^\rho = \omega(1)$, whp, for every $i = 0, 1, \ldots, i_0 + 1$ we have*

$$|\hat{\mathcal{P}}_i| \ \le \ 2 \cdot n^{1 - \frac{2^i - 1}{\kappa}} \ , \tag{5}$$

and for $i = i_0 + 2, \ldots, i_1 + 1$,

$$|\hat{\mathcal{P}}_i| \ \le \ 2 \cdot n^{1 + 1/\kappa - (i - i_0)\rho} \ .$$

Proof. For the first assertion, the probability that a vertex $v \in V$ will be a center of a cluster in $\hat{\mathcal{P}}_i$ is $\prod_{j=0}^{i-1} 1/deg_j = n^{-(2^i - 1)/\kappa}$. Thus the expected size of $\hat{\mathcal{P}}_i$ is $n^{1-(2^i-1)/\kappa}$, and as these choices are made independently, by the Chernoff bound,

$$\mathbb{P}[|\hat{\mathcal{P}}_i| \ge 2\mathbb{E}[|\hat{\mathcal{P}}_i|]] \le exp\{-\Omega(\mathbb{E}[|\hat{\mathcal{P}}_i|])\} = exp\{-\Omega(n^{1 - \frac{2^i - 1}{\kappa}})\} \ .$$

Since for $\rho \le 1/2$ and $i \le i_0 + 1 = \lfloor \log \rho\kappa \rfloor + 1$, we have $n^{1 - \frac{2^i - 1}{\kappa}} \ge n^{1 - 2\rho + 1/\kappa} = \omega(\log n)$ (recall that when $\kappa > \log n / \log\log n$, then $\rho < 1/3$), we conclude that whp for all $0 \le i \le i_0 + 1$, $|\hat{\mathcal{P}}_i| \le 2n^{1 - \frac{2^i - 1}{\kappa}}$. In particular, $|\hat{\mathcal{P}}_{i_0+1}| = O(n^{1 - \rho + 1/\kappa})$.

For the second assertion, consider any $i \in [i_0+2, i_1+1]$; the expected size of $\hat{\mathcal{P}}_i$ is

$$\mathbb{E}[|\hat{\mathcal{P}}_i|] = n \cdot \prod_{j=0}^{i-1} 1/deg_j \le n^{1+1/\kappa-\rho-(i-1-i_0)\rho} = n^{1+1/\kappa-(i-i_0)\rho} \ .$$

Since $n^{1+1/\kappa-(i-i_0)\rho} \ge n^\rho$ for any $i \le i_1$, by Chernoff's bound with probability at least $1 - exp\{-\Omega(n^\rho)\}$ (which is $1-o(1)$ by our assumption on n^ρ), we have

$$|\hat{\mathcal{P}}_i| \ \le \ 2 \cdot n^{1+1/\kappa-(i-i_0)\rho}. \qquad \square$$

This lemma implies that whp,

$$(6) \qquad |\hat{\mathcal{P}}_{i_1+1}| \ \le \ O(n^{1+1/\kappa-(i_1+1-i_0)\rho}) \ = \ O(n^{1+1/\kappa-(\lceil\frac{\kappa+1}{\kappa\rho}\rceil-1)\rho}) \ = \ O(n^\rho) \ .$$

For the assumption of the lemma to hold, we will need to assume that $\rho \ge \frac{\log\log n}{2\log n}$, say. We will soon show that this assumption is valid in our setting.

The running time required to implement the single Dijkstra exploration in the superclustering of phase i is $O(|E| + n\log n)$, while in the interconnection step, by Lemma 3.2 every vertex is expected to be visited by at most deg_i explorations, so the expected running time of phase $0 \le i \le i_1$ is $O(|E|+n\log n)\cdot deg_i$. Recall that in the last phase i_1+1 there is no superclustering step, but as (6) implies, there are whp only $O(n^\rho)$ clusters, so each vertex will be visited at most $O(n^\rho)$ times. Thus the total expected running time is

$$\begin{aligned} O(|E|+n\log n)\cdot\left(\sum_{i=0}^{\ell-1}(deg_i) + n^\rho\right) &= O(|E|+n\log n)\cdot\left(\sum_{i=0}^{i_0}(n^{2^i/\kappa}) + (i_1-i_0)n^\rho\right) \\ &= O(|E|+n\log n)\cdot(n^{2^{i_0}/\kappa} + n^\rho/\rho) \\ &= O(|E|+n\log n)\cdot n^\rho/\rho \ . \end{aligned}$$

The size of the hopset H_k that was constructed by this algorithm is dominated by the number of edges inserted by the interconnection steps, since all the edges inserted at superclustering steps induce a forest. Due to Lemma 3.2, the expected number of edges inserted by the interconnection step of phase i is at most $O(|\hat{\mathcal{P}}_i| \cdot deg_i) = O(n^{1+1/\kappa})$, for $i \le i_0$, and $\sum_{i=i_0+1}^{\ell+1} O(|\hat{\mathcal{P}}_i|\cdot deg_i) = O(n^{1+1/\kappa})$ edges on the later phases. Hence overall $\mathbb{E}(|H_k|) = O(n^{1+1/\kappa}\cdot\log\kappa)$. We remark that the factor $\log\kappa$ can be eliminated from the hopset size by using a refined degree sequence, at the cost of increasing the number of phases by 1 (this will increase the exponent of β by 1). We elaborate on this in section 3.1.1. Then the number of edges contributed to the hopset H_k by all interconnection steps becomes $O(n^{1+1/\kappa})$.

Next we analyze the stretch and the hopbound of H_k. Write $H = H_k$. Observe that, by Claim 3.1, $Rad(\hat{\mathcal{U}}_0) = R_0 = 0$, and, for all $i \in [1,\ell]$, $Rad(\hat{\mathcal{U}}_i) \le Rad(\hat{\mathcal{P}}_i) \le R_i \le 2\alpha(1/\epsilon)^{i-1}$. Write $c = 2$. Note also that for any pair of distinct clusters $C, C' \in \hat{\mathcal{U}}_i$, for any i, which are at distance $d_G(C,C') \le \frac{1}{2}\alpha\cdot(1/\epsilon)^i$, it holds that $d_G(r_C, r_{C'}) \le d_G(C,C') + 2R_i \le \frac{1}{2}\alpha(1/\epsilon)^i + 2\cdot R_i = \frac{1}{2}\delta_i$. Hence for every pair of clusters C, C' as above, an edge $(r_C, r_{C'})$ of weight $\omega(r_C, r_{C'}) = d_G(r_C, r_{C'})$ belongs to the hopset.

Observe that $\hat{\mathcal{U}} = \bigcup_{i=0}^{\ell}\hat{\mathcal{U}}_i$ is a partition of G. This holds because the sets $\hat{\mathcal{U}}_i$ are pairwise disjoint (when a cluster is not superclustered it joins some $\hat{\mathcal{U}}_i$ and ceases to participate in the next steps) and because in the last step we take all the remaining clusters to $\hat{\mathcal{U}}_\ell = \hat{\hat{\mathcal{P}}}_\ell$. For any i, we denote $\hat{\mathcal{U}}^{(i)} = \bigcup_{j=0}^{i}\hat{\mathcal{U}}_j$.

LEMMA 3.5. *Let x, y be a pair of vertices with $d_G(x,y) \le \frac{1}{2}\alpha \cdot (1/\epsilon)^i$ and such that all vertices of a shortest path $\pi(x,y)$ in G between them are clustered in $\hat{\mathcal{U}}^{(i)}$ for some $i \le \ell$. Then it holds that*

$$d^{(h_i)}_{G\cup H}(x,y) \le d_G(x,y)(1+16c(i-1)\cdot\epsilon) + 8\cdot\alpha\cdot c\cdot(1/\epsilon)^{i-1} \ , \tag{7}$$

with h_i given by $h_0 = 1$, and $h_{i+1} = (h_i+1)(1/\epsilon+2)+2i+5$.

Proof. The proof is by induction on i. The base case is the case $i = 0$.

Base case. We assume $d_G(x,y) \le \frac{1}{2}\alpha$ and all vertices of $\pi(x,y)$ are clustered in $\hat{\mathcal{U}}_0$; then there is an edge (x,y) in H with $\omega(x,y) = d_G(x,y)$, and indeed,

$$d^{(h_0)}_{G\cup H}(x,y) = d_G(x,y) \le d_G(x,y)(1-16c\cdot\epsilon) + 8\cdot\alpha\cdot c\cdot(1/\epsilon)^{-1} \ .$$

Inductive step. We assume the assertion of the lemma for some index i, and prove it for $i+1$.

Consider first a pair u, v of vertices such that all vertices of $\pi(u,v)$ are clustered in $\hat{\mathcal{U}}^{(i)}$, for a fixed $i < \ell$, without any restriction on $d_G(u,v)$.

We partition $\pi(u,v)$ into segments $L_1, L_2, \dots,$ of length roughly $\frac{1}{2}\cdot\alpha\cdot(1/\epsilon)^i$ each, in the following way. The first segment L_1 starts at u, i.e., we write $u = u_1$. Given a left endpoint u_p, $p \ge 1$, of a segment L_p, we set the right endpoint v_p of L_p to be the farthest vertex of $\pi(u,v)$ from u_p which is closer to v than u_p, and such that $d_G(u_p,v_p) \le \frac{1}{2}\cdot\alpha\cdot(1/\epsilon)^i$ (if it exists).

If v_p does not exist, then the pth segment L_p is declared *void*, and we define $v_p = u_{p+1}$ to be the neighbor of u_p on $\pi(u,v)$ which is closer to v. If v_p does exist, then u_{p+1} is the "right" neighbor of v_p on $\pi(u,v)$, i.e., the neighbor of v_p which is closer to v than v_p. (It may not exist only if $v_p = v$.) Observe that in either case, if u_{p+1} exists, then $d_G(u_p,u_{p+1}) > \frac{1}{2}\cdot\alpha\cdot(1/\epsilon)^i$ (if it exists).

We also define *extended* segments $\hat{L}_p$ in the following way. If L_p is a void segment, then we define $\hat{L}_p = L_p$. Otherwise $\hat{L}_p$ is the segment of $\pi(u,v)$ connecting u_p with u_{p+1}, if u_{p+1} exists, and with v_p otherwise. (This may be the case only if $L_p = \hat{L}_p$ is the last, i.e., the rightmost, segment of the path $\pi(u,v)$.)

Observe that every nonvoid extended segment $\hat{L}_p$, except perhaps the last one, has length at least $\frac{1}{2}\cdot\alpha\cdot(1/\epsilon)^i$, and every segment L_p has length at most $\frac{1}{2}\cdot\alpha\cdot(1/\epsilon)^i$.

Next we construct a path $\pi'(u,v)$ in $G\cup H$, which has roughly the same length as $\pi(u,v)$, but consists of much fewer hops. Consider a segment L_p, with left endpoint u_p and right endpoint v_p, and its extended segment $\hat{L}_p$ with right endpoint u_{p+1}. We define a *substitute* segment L'_p in $G\cup H$, connecting u_p to u_{p+1} with a few hops, and of roughly the same length.

If L_p is a void segment, then L'_p is just the single edge (u_p,u_{p+1}), taken from $E = E(G)$. Observe that for a void segment,

$$\omega(\hat{L}_p) = \omega(L_p) = \omega(u_p,u_{p+1}) = d_G(u_p,u_{p+1}) \ .$$

Otherwise, if L_p is not a void segment, then $d_G(u_p,v_p) \le \frac{1}{2}\cdot\alpha\cdot(1/\epsilon)^i$. Observe also that since all vertices of $\pi(u,v)$ are $\hat{\mathcal{U}}^{(i)}$-clustered, this is also the case for the subpath $\pi(u_p,v_p)$. Hence the induction hypothesis is applicable to this subpath, and so there exists a path $\pi'(u_p,v_p)$ in $G\cup H$ with at most h_i hops, such that

$$\omega(\pi'(u_p,v_p)) \ \le \ d_G(u_p,v_p)\cdot(1+16\cdot c(i-1)\cdot\epsilon) + 8\cdot\alpha\cdot c\cdot(1/\epsilon)^{i-1} \ .$$

We define L'_p to be the concatenation of $\pi'(u_p, v_p)$ with the edge (v_p, u_{p+1}). (This edge is taken from G.) Since v_p lies on a shortest path between u_p and u_{p+1}, it follows that

$$\omega(L'_p) \;\leq\; (1 + 16c \cdot (i-1)\epsilon) \cdot d_G(u_p, u_{p+1}) + 8 \cdot \alpha \cdot c \cdot (1/\epsilon)^{i-1} \ ,$$

and L'_p contains up to $h_i + 1$ hops.

Finally, our ultimate path $\pi'(u,v)$ is the concatenation of all the substitute segments $L'_1 \circ L'_2 \circ \cdots \circ L'_q$, where $\pi(u,v) = \hat{L}_1 \circ \hat{L}_2 \circ \cdots \circ \hat{L}_q$. Since each extended segment has length at least $\frac{1}{2}\alpha \cdot (1/\epsilon)^{i-1}$, we conclude that

$$\begin{aligned} d_{G\cup H}^{((h_i+1)\cdot\lceil \frac{d_G(u,v)}{\frac{1}{2}\alpha(1/\epsilon)^i}\rceil)}(u,v) &\leq d_G(u,v)(1+16c(i-1)\epsilon) + \left\lceil \frac{d_G(u,v)}{\frac{1}{2}\alpha(1/\epsilon)^i} \right\rceil \cdot 8\alpha c \cdot (1/\epsilon)^{i-1} \\ &\leq d_G(u,v)\left(1 + 16c(i-1)\epsilon + \frac{8\cdot\alpha\cdot c(1/\epsilon)^{i-1}}{\frac{1}{2}\alpha\cdot(1/\epsilon)^i}\right) + 8\alpha c \cdot (1/\epsilon)^{i-1} \\ &= d_G(u,v)(1 + 16c\cdot i\cdot\epsilon) + 8\cdot\alpha c\cdot(1/\epsilon)^{i-1} \ . \end{aligned} \tag{8}$$

Now consider x, y such that $d_G(x,y) \leq \frac{1}{2}\alpha \cdot (1/\epsilon)^{i+1}$ and such that $\pi(x,y)$ is $\hat{\mathcal{U}}^{(i+1)}$-clustered. Let z_1 and z_2 denote the leftmost and rightmost $\hat{\mathcal{U}}_{i+1}$-clustered vertices on this path, and denote by C_1 and C_2 their respective clusters. Denote also $r_1 = r_{C_1}, r_2 = r_{C_2}$. (If z_1 and z_2 do not exist, then $\pi(x,y)$ is $\hat{\mathcal{U}}^{(i)}$-clustered, and this case was already taken care of.) Denote also by w_1 (resp., w_2) the neighbor of z_1 (resp., z_2) on the subpath $\pi(x, z_1)$ (resp., $\pi(z_2, y)$) of $\pi(x,y)$.

The path $\pi'(x,y)$ in $G\cup H$ between x and y is constructed in the following way. By (8), we can reach from x to w_1 while incurring a multiplicative stretch of $(1 + 16ci\cdot\epsilon)$ and an additive error of $8\cdot\alpha\cdot c\cdot(1/\epsilon)^{i-1}$, and using at most $b_1 = (h_i+1)\cdot\lceil\frac{d_G(x,w_1)}{\frac{1}{2}\alpha(1/\epsilon)^i}\rceil$ hops. The same is true for the pair w_2, y, except that the required number of hops is at most $b_2 = (h_i+1)\cdot\lceil\frac{d_G(w_2,y)}{\frac{1}{2}\alpha(1/\epsilon)^i}\rceil$. Finally, the path $\pi'(x,y)$ connects w_1 with w_2 via edges $(w_1,z_1), (z_2,w_2)$ that belong to $E(G)$, the edge (r_1,r_2) of the hopset H, and the paths $\pi(z_1,r_1), \pi(r_2,z_2)$ (each of $i+1$ hops) in H given by Claim 3.1. Hence,

$$\begin{aligned} d_{G\cup H}^{(h_{i+1})}(x,y) &\leq d_{G\cup H}^{(b_1)}(x,w_1) + d_G^{(1)}(w_1,z_1) + d_G^{(i+1)}(z_1,r_1) + d_H^{(1)}(r_1,r_2) \\ &\quad + d_G^{(i+1)}(r_2,z_2) + d_G^{(1)}(z_2,w_2) + d_{G\cup H}^{(b_2)}(w_2,y) \\ &\leq (1+16c\cdot i\cdot\epsilon)d_G(x,w_1) + d_G(w_1,z_1) + R_{i+1} + (d_G(z_1,z_2) + 2R_{i+1}) \\ &\quad + R_{i+1} + d_G(z_2,w_2) + (1+16ci\cdot\epsilon)d_G(w_2,y) + 2\cdot(8\alpha c\cdot(1/\epsilon)^{i-1}) \\ &\leq (1+16c\cdot i\cdot\epsilon)d_G(x,y) + 4\cdot\alpha c(1/\epsilon)^i + 16\alpha c\cdot(1/\epsilon)^{i-1} \\ &\leq (1+16c\cdot i\cdot\epsilon)d_G(x,y) + 8\cdot\alpha c\cdot(1/\epsilon)^i \ , \end{aligned}$$

where the required number of hops indeed satisfies

$$\begin{aligned} &(h_i+1)\left(\left\lceil\frac{d_G(x,w_1)}{\frac{1}{2}\alpha\cdot(1/\epsilon)^i}\right\rceil + \left\lceil\frac{d_G(w_2,y)}{\frac{1}{2}\alpha\cdot(1/\epsilon)^i}\right\rceil\right) + 2i + 5 \\ &\leq (h_i+1)\left(\frac{d_G(x,y)}{\frac{1}{2}\alpha(1/\epsilon)^i} + 2\right) + 2i + 5 \\ &\leq (h_i+1)(1/\epsilon+2) + 2i + 5 \\ &= h_{i+1}. \end{aligned} \tag{9}$$

□

The recursive equation $h_{i+1} = (h_i+1)(1/\epsilon+2)+2i+5$ solves to $h_i \leq 3\cdot(1/\epsilon+2)^i$ for $\epsilon < 1/10$, i.e., $h_\ell \leq 3\cdot(1/\epsilon+2)^\ell$. Write $\zeta = 16c(\ell+1)\cdot\epsilon$ and $\beta = 2h_\ell+1 \leq 6\cdot(1/\epsilon+2)^\ell+1$.

Corollary 3.6. *Let $x, y \in V$ be such that $d_G(x,y) \in (\hat{R}/2, \hat{R}]$. Then,*

$$d^{(\beta)}_{G\cup H}(x,y) \le (1+\zeta)\cdot d_G(x,y) \ .$$

Proof. Let $\pi(x,y)$ be the shortest path in G between x, y. By an argument similar to (and simpler than) the one in Lemma 3.5, we can see that there exists an edge $(u,v) \in E$ such that both u, v are on $\pi(x,y)$, and also that $d_G(x,u) \le \hat{R}/2$ and $d_G(y,v) \le \hat{R}/2$. Applying Lemma 3.5 on these pairs with $i = \ell$, recalling that $\frac{1}{2}\alpha\cdot(1/\epsilon)^{\ell} = \hat{R}/2$ and that every vertex is clustered in $\hat{\mathcal{U}}^{(\ell)}$, it follows that

$$d^{(h_\ell)}_{G\cup H}(x,u) \le d_G(x,u)(1+16c(\ell-1)\cdot\epsilon) + 8c\cdot\epsilon\cdot\hat{R} \ .$$

Similarly, it also follows that

$$d^{(h_\ell)}_{G\cup H}(y,v) \le d_G(x,u)(1+16c(\ell-1)\cdot\epsilon) + 8c\cdot\epsilon\cdot\hat{R} \ .$$

Since $\beta = 2h_\ell + 1$ and $(u,v)\in E$, we obtain

$$\begin{aligned} d^{(\beta)}_{G\cup H}(x,y) &\le d^{(h_\ell)}_{G\cup H}(x,u) + d^{(1)}_{G}(u,v) + d^{(h_\ell)}_{G\cup H}(y,v) \\ &\le (d_G(x,u) + d_G(u,v) + d_G(v,y))\cdot(1+16c(\ell-1)\cdot\epsilon) + 16c\cdot\epsilon\cdot\hat{R} \\ &\le d_G(x,y)\cdot(1+16c(\ell-1)\cdot\epsilon) + 32c\cdot\epsilon\cdot d_G(x,y) \\ &= d_G(x,y)\cdot(1+\zeta). \end{aligned}$$ ◻

Recall that $\ell = \lfloor\log(\kappa\rho)\rfloor + \lceil\frac{\kappa+1}{\rho\kappa}\rceil - 1 \le \log(\kappa\rho) + \lceil 1/\rho\rceil$ is the number of phases of the algorithm (for the sake of brevity, from now on we ignore the ceiling of $1/\rho$). When we rescale $\epsilon = \zeta$ as the stretch factor, then $\beta = O(\ell/\epsilon)^{\ell} = O(\frac{\log\kappa + 1/\rho}{\epsilon})^{\log\kappa+1/\rho}$.

Our ultimate hopset H is created by $H \leftarrow \bigcup_{k>\log\beta-1} H_k$; i.e., H is the union of up to $\lceil\log\Lambda\rceil$ hopsets, each of which takes care of its own distance range. As a result, the number of edges in H is $O(n^{1+1/\kappa}\cdot\log\kappa\cdot\log\Lambda)$, and its expected construction time is $O((|E| + n\log n)\cdot n^{\rho}/\rho\cdot\log\Lambda)$. (The factor $\log\kappa$ in the hopset's size will be eliminated in section 3.1.1.) By executing the algorithm a number of times until the first time that the number of edges in the hopset is at most some constant multiplied by $\mathbb{E}(|H|)$, one can ensure that the bound on $|H|$ holds with probability 1, while the expected running time grows only by a constant factor.

The following theorem summarizes this result.

Theorem 3.7. *For any graph $G = (V,E)$ with n vertices and diameter Λ, $2 \le \kappa \le (\log n)/4$, $1/2 \ge \rho \ge 1/\kappa$, and $0 < \epsilon \le 1$, our algorithm constructs a (β,ϵ)-hopset H with $O(n^{1+1/\kappa}\cdot\log\Lambda)$ edges, in expected time $O((|E| + n\log n)(n^{\rho}/\rho\cdot\log\Lambda))$, with $\beta = O(\frac{\log\kappa+1/\rho}{\epsilon})^{\log\kappa+1/\rho}$.*

Moreover, the hopset consists of up to $\lceil\log\Lambda\rceil$ single-scale hopsets. Each of these hopsets H_k has the same β, and its expected size is $|H_k| = O(n^{1+1/\kappa})$. It can be constructed in $O((|E| + n\cdot\log n)(n^{\rho}/\rho))$ time.

In section 4 we will show how to remove the dependence on the aspect ratio Λ and replace it with n, which yields the following.

Theorem 3.8. *For any graph $G = (V,E,\omega)$ with n vertices, $2 \le \kappa \le (\log n)/4$, $1/2 \ge \rho \ge 1/\kappa$, and $0 < \epsilon \le 1$, our algorithm constructs a (β,ϵ)-hopset H with $O(n^{1+1/\kappa}\cdot\log n)$ edges, in expected time $O((|E| + n\log n)(n^{\rho}/\rho\cdot\log n))$, with $\beta = O(\frac{\log\kappa+1/\rho}{\epsilon})^{\log\kappa+1/\rho}$.*

Finally, we note that our assumption that $\rho > \log\log n/(2\log n)$ is justified; otherwise we get $\beta \ge n$, in which case an empty hopset will do. Also, $\epsilon \le 1$ because we rescaled it by a factor of $16c(\ell+1) > 10$.

3.1.1. Improved hopset size. Here we show how to select a refined degree sequence that will eliminate the term of $\log \kappa$ from the size of the hopset constructed in section 3.1 by increasing the number of phases by 1 (and thus the exponent of β by an additive 1). Specifically, one can set $deg_i = n^{2^i/\kappa}/2^{2^i-1}$ for each $i = 0, 1, \ldots, i_0 = \lfloor \log(\kappa\rho) \rfloor$. As a result we get that $\mathbb{E}[|\hat{\mathcal{P}}_i|] = n \cdot \prod_{j=0}^{i-1} 1/deg_j = n^{1-\frac{2^i-1}{\kappa}} \cdot 2^{2^i-1-i}$, and thus the expected number of edges inserted at phase $i \leq i_0$ is at most

$$O(\mathbb{E}[|\hat{\mathcal{P}}_i|] \cdot deg_i) = O(n^{1+1/\kappa}/2^i) \ ,$$

and thus it is $O(n^{1+1/\kappa})$ over all phases $i = 0, 1, \ldots, i_0$. When the first stage concludes, we run the phase $i_0 + 1$ with $deg_{i_0+1} = n^{\rho/2}$ and run all subsequent phases with $deg_i = n^\rho$. To bound the expected number of edges added at phase $i_0 + 1$ we need to note that $2^{2^{i_0+1}} \leq 2^{2\kappa\rho} \leq n^{\rho/2}$ as long as $\kappa \leq (\log n)/4$. (The latter can be assumed without affecting any of the parameters by more than a constant factor.) It follows that $\mathbb{E}[|\hat{\mathcal{P}}_{i_0+1}|] \cdot deg_{i_0+1} = n^{1-\frac{2^{i_0+1}-1}{\kappa}} \cdot 2^{2^{i_0+1}-1-(i_0+1)} \cdot n^{\rho/2} \leq n^{1+1/\kappa}$. In the remaining phases, $\mathbb{E}[|\hat{\mathcal{P}}_{i_0+i}|] \leq n^{1+1/\kappa-(i-1)\rho}$ for $i \geq 2$, and the contribution of these phases is a converging sequence. In particular, at phase $i_0 + 2$ we have $\mathbb{E}[|\hat{\mathcal{P}}_{i_0+2}|] \leq n^{1+1/\kappa-\rho}$. We can take $i_1 = i_0 + \lceil \frac{\kappa+1}{\kappa\rho} \rceil - 1$, and finally $\mathbb{E}[|\hat{\mathcal{P}}_{i_1+1}|] \leq n^{1+1/\kappa-(i_1+1-i_0-1)\rho} \leq n^\rho$, as required to bound the running time of the final interconnection phase. The total number of phases is now $\ell = i_1 + 1 = \lfloor \log(\kappa\rho) \rfloor + \lceil \frac{\kappa+1}{\kappa\rho} \rceil$, which might be larger by an additive 1 than the bound claimed in Theorem 3.8. For the sake of brevity, we ignore this small overhead.

3.2. Distributed implementation in Congested Clique model. In this section we argue that our hopset construction admits an efficient implementation in the distributed Congested Clique model, albeit with somewhat worse parameters.

A direct implementation of the algorithm from section 3.1 in this model requires up to $O(n)$ distributed time, because a Dijkstra algorithm invoked during the hopset's construction may explore paths with up to n hops. To overcome this issue we use the following idea, which dates back to Cohen's work [Coh00]. Specifically, we use hopsets $\bigcup_{\log\beta-1<j\leq k-1} H_j$ to construct the hopset H_k. Indeed, the hopset H_k needs to take care of vertex pairs u, v with $d_G(u, v) \in (2^k, 2^{k+1}]$, $\hat{R}^{(k)} = 2^{k+1}$, while $E \cup H^{(k-1)}$ (with $H^{(k-1)} = \bigcup_{j\leq k-1} H_j$) provides $(1 + \epsilon)$-approximate shortest paths with up to β hops for pairs u, v with $d_G(u, v) \leq 2^k$. Denote $E'^{(k-1)} = E \cup H^{(k-1)}$.

Consider a vertex pair $u, v \in V$, with $d_G(u, v) \in (2^k, 2^{k+1}]$, and let $\pi(u, v)$ be a shortest path between them. Let $x \in V(\pi(u, v))$ be the farthest vertex of $\pi(u, v)$ from u which is at distance no greater than 2^k from u, and let y be its neighbor such that $d_G(u, y) > 2^k$. Observe that the hopset $H^{(k-1)}$ provides (together with the edge set E of the original graph) a $(1 + \epsilon)$-approximate shortest $u - v$ path with at most $2\beta + 1$ hops. This path $\pi'(u, v)$ is a concatenation of a $(1 + \epsilon)$-approximate $u - x$ path $\pi'(u, x)$ in $E'^{(k-1)}$ with the edge (x, y) and with a $(1 + \epsilon)$-approximate $y - v$ path $\pi'(y, v)$ in $E'^{(k-1)}$.

Next we generalize this observation.

LEMMA 3.9. *For any index p, $\log \beta - 1 \leq p \leq k - 1$, the hopset $H^{(p-1)}$ provides (together with edges of E) $(1+\epsilon)$-approximate shortest paths with at most $2^{k+2-p}(\beta+1)$ hops for pairs u,v, with $d_G(u, v) \in (2^k, 2^{k+1}]$.*

Proof. Partition $\pi(u, v)$ into segments (except void segments) of length at most 2^{p-1}, (See the proof of Lemma 3.5 for the definition of void, extended, and substitute segments.) For each nonvoid extended segment we use at most $\beta + 1$ hops of the substitute segment in $G \cup H^{(p-1)}$, and for each void extended segment we use just one hop. Every extended segment has length at least 2^{p-1}, and thus $\pi(u,v)$ is partitioned into at most 2^{k+2-p} such segments. Hence the assertion of the lemma follows. (In fact, in the last segment, β hops suffice, and thus the estimate can be refined to $2^{k+2-p}(\beta+1)-1$.) ∎

The first variant of our distributed algorithm builds hopsets $H_{k_0}, H_{k_0+1}, H_{k_0+2}, \ldots$ consecutively, where k_0 is the largest integer such that $\hat{R}^{(k_0)} = 2^{k_0+1} \le \beta$. Suppose that all hopsets H_j, $j < k$, were already built, and we are now building H_k. We only need to describe how the superclustering and the interconnection steps on a phase i, $0 \le i \le \ell$, are implemented. (On phase ℓ there is no superclustering step, and there we only need to implement the interconnection step.)

Denote by $\zeta = \zeta_{k-1}$ the value such that $E'^{(k-1)}$ guarantees stretch $1+\zeta$. We slightly modify the sequence of distance thresholds δ_i; specifically, we multiply all of them by a factor of $1+\zeta$.

We define $R'_i = (1+\zeta)R_i$ and $\delta'_i = (1+\zeta)\delta_i$ for every $i \in [0,\ell]$. Here $\alpha = \alpha^{(k)} = \frac{\hat{R}^{(k)}}{(1/\epsilon)^{\ell}}$, where $\hat{R}^{(k)} = 2^{k+1}$. The distributed variant of our algorithm uses distance thresholds $(\delta'_i \mid 0 \le i \le \ell)$, and as a result upper bounds on the radii of i-level clusters become $R'_i = R_i(1+\zeta)$.

The superclustering step of our *centralized* algorithm conducts a Dijkstra exploration from the set $Roots = \{r_c \mid C \in \mathcal{S}_i\}$ in G to distance $\delta_i = \alpha \cdot (1/\epsilon)^i + 4R_i \le \hat{R}^{(k)}$. In the distributed version we instead invoke the Bellman–Ford algorithm originated in *Roots* over the edge set $E'^{(k-1)} = E \cup H^{(k-1)}$ for $4\beta+2$ rounds.

Specifically, vertices $r_C \in Roots$ initialize their distance estimates $\hat{d}(r_C) = 0$, and other vertices initialize them as ∞. On every round, every vertex v sends its estimate $\hat{d}(v)$ to all other vertices in the network. Every vertex u sets locally

$$\hat{d}(u) \leftarrow \min\left\{\hat{d}(u), \min_{v\in V}\{\hat{d}(v) + \omega_{E'^{(k-1)}}(v,u)\}\right\} .$$

This computation proceeds for $4\beta+2$ rounds.

As a result every vertex v computes its $(4\beta+2)$-limited distance in $E'^{(k-1)}$ from *Roots*. For v such that $d_G(Roots, v) \le 2^{k+1} = \hat{R}^{(k)}$, we have

$$d_G(Roots,v) \;\le\; d^{(4\beta+2)}_{E'^{(k-1)}}(Roots,v) \;\le\; d^{(2\beta+1)}_{E'^{(k-1)}}(Roots,v) \;\le\; (1+\zeta)d_G(Roots,v) .$$

For every $i \in [0, \ell-1]$,

$$\begin{aligned}\delta_i \le \delta_{\ell-1} &= \alpha\cdot(1/\epsilon)^{\ell-1} + 4R_{\ell-1} \le \alpha\cdot(1/\epsilon)^{\ell-1} + 8\alpha(1/\epsilon)^{\ell-2}\\ &= \alpha(1+8\epsilon)(1/\epsilon)^{\ell-1} \le \alpha\cdot(1/\epsilon)^{\ell} = \hat{R}^{(k)}\end{aligned}$$

for $\epsilon \le 1/10$. (Recall that on phase ℓ there is no superclustering step, and thus the maximum exploration of any superclustering step is $\delta_{\ell-1}$.)

We conclude that for every $v \in Ball(Roots, \delta_i)$, its distance estimate $\hat{d}(v)$ satisfies

$$d_G(Roots,v) \le \hat{d}(v) \le (1+\zeta)\cdot d_G(Roots,v) . \tag{10}$$

Moreover, execution of the Bellman–Ford algorithm also constructs a forest F rooted at the centers $Roots = \{r_C \mid C \in \mathcal{S}_i\}$ of i-level selected clusters, and every vertex v with $\hat{d}(v) < \infty$ knows the root r_C of its tree in F.

For any cluster center $r_{C'}$, $C' \in \hat{\mathcal{P}}_i \setminus \mathcal{S}_i$, such that $\hat{d}(r_{C'}) \le \delta'_i = (1+\zeta)\delta_i$, the algorithm connects r_C with $r_{C'}$ via a hopset edge $e = (r_C, r_{C'})$ (i.e., $H \leftarrow H \cup \{e\}$)

of weight $\omega(e) = \hat{d}(r_{C'})$, where r_C is the root of the tree of F to which $r_{C'}$ belongs. We also create a supercluster rooted at C (more specifically, at r_C) which contains all vertices of C' as above. Observe that (by (10)), if $d_G(r_C, r_{C'}) \leq \delta_i$, then $\hat{d}(r_{C'}) \leq (1+\zeta)\delta_i = \delta'_i$. Then the edge $(r_C, r_{C'})$ will be added to the hopset, and the cluster C' will be superclustered in an i-level cluster created around C. (See also inequality (11) below.)

This completes the description of the superclustering step of phase i. The interconnection step is conducted in a similar way, but now the Bellman–Ford algorithm is conducted from each cluster center in $\mathit{URoots} = \{r_C \mid C \in \hat{\mathcal{U}}_i\}$ *separately.* (This is in contrast to the superclustering step, in which the Bellman–Ford algorithm is conducted from the set *Roots*.) Also, this version of Bellman–Ford is conducted to depth $2\beta+1$ rather than $4\beta+2$. As a result, every vertex v maintains estimates $\hat{d}(r_C, v)$ for cluster centers $r_C \in \mathit{URoots}$. In each step, v sends all of its estimates $\hat{d}(r_C, v)$ which satisfy $\hat{d}(r_C, v) \leq \frac{1}{2}\delta'_i$ to the entire network. Recall Lemma 3.3, which implies that whp, no vertex v has more than $O(\mathit{deg}_i \cdot \log n)$ cluster centers $\{r_C \mid C \in \hat{\mathcal{U}}_i\}$ at $(2\beta+1)$-limited distance at most $\frac{1}{2}\delta'_i$ from it in $E'^{(k-1)}$. (Note that if v has $c \cdot \mathit{deg}_i \cdot \log n$ cluster centers r_C as above at $(2\beta+1)$-limited distance at most $\frac{1}{2}\delta'_i$ from it for some sufficiently large c, then each of these cluster centers is at $(4\beta+2)$-limited distance at most δ'_i from all the others. Hence, with probability at least $n^{-(c-O(1))}$, i.e., whp, one of them is sampled, and none of the respective clusters C belongs to $\hat{\mathcal{U}}_i$.) Hence, in each step each vertex v needs to send $O(\mathit{deg}_i \cdot \log n)$ messages to the entire network, and this requires $O(\mathit{deg}_i \cdot \log n)$ rounds (whp). Since $\mathit{deg}_i = O(n^\rho)$ for all i, we conclude that the algorithm requires $O(n^\rho \cdot (1/\rho)\log n \cdot \beta)$ time whp. This is because in each phase we perform $O(\beta)$ steps, each of which lasts $O(\mathit{deg}_i \cdot \log n)$ rounds, whp. We also remark that if large messages of size $n^\rho \log n$ are allowed, then the running time becomes only $O(\beta/\rho)$.

As a result, for every pair of clusters $C, C' \in \hat{\mathcal{U}}_i$ such that $d_G(r_C, r_{C'}) \leq \frac{1}{2}\delta_i$, the edge $(r_C, r_{C'})$ is inserted into the hopset. Indeed, observe that

$$\frac{1}{2} \cdot \delta_i \leq \frac{1}{2} \cdot \delta_\ell \leq \alpha \cdot (1/\epsilon)^\ell (1+8\epsilon)/2 \leq \alpha \cdot (1/\epsilon)^\ell = \hat{R}^{(k)}$$

for $\epsilon < 1/10$. Hence $d_G(r_C, r_{C'}) \leq \hat{R}^{(k)}$. Denote by $\hat{d}(r_C, r_{C'})$ the estimate of $d_G(r_C, r_{C'})$ computed by $r_{C'}$. Then we have

$$(11)\quad \hat{d}(r_C, r_{C'}) \;=\; d^{(2\beta+1)}_{E'^{(k-1)}}(r_C, r_{C'}) \;\leq\; (1+\zeta) d_G(r_C, r_{C'}) \;\leq\; \frac{1}{2}(1+\zeta)\delta_i \;=\; \frac{1}{2}\delta'_i\ .$$

Next we analyze the properties of the resulting hopset $H = \bigcup_{k \geq k_0} H_k$. The size of the hopset is the same as with the centralized algorithm, but in the stretch analysis we incur additional overhead in comparison with the centralized algorithm. This is because in the centralized construction, every pair of sufficiently close $\hat{\mathcal{U}}_i$ cluster centers were interconnected via an edge of length *exactly equal* to the distance in G between them, while now the length of this edge is equal to the distance between them in $E'^{(k-1)}$, i.e., it is a $(1+\zeta)$-approximation of the distance in G between them.

The following lemma is a distributed analogue of Lemma 3.5.

LEMMA 3.10. *For x, y as in Lemma 3.5 it holds that*

$$(12)\quad d^{(h_i)}_{G \cup H_k}(x,y) \;\leq\; d_G(x,y)(1+\zeta)(1+16c(i-1)\epsilon) + (1+\zeta)\cdot 8 \cdot \alpha c \cdot (1/\epsilon)^{i-1}\ ,$$

with $h_{i+1} \leq (h_i+1)(1/\epsilon+2) + 2i + 5$.

Remark. Note that the hopset H_k alone suffices for approximating single-scale distances; i.e., one does not need hopsets H_j with $j < k$ for the inequality (12) to hold.

Proof. Recall that the radius of an i-level cluster in the distributed variant of our algorithm is at most $R'_i \leq (1+\zeta)R_i$, where $R_i \leq \alpha \cdot c \cdot (1/\epsilon)^{i-1}$. The proof is again by an induction on i. The base case is $i = 0$.

Base case. If $d_G(x,y) \leq \frac{1}{2}\alpha = \frac{1}{2}\delta_0$ and all vertices of $\pi(x,y)$ are clustered in $\hat{\mathcal{U}}_0$, then there is a hopset edge in H_k between x and y, and its weight is at most $(1+\zeta)d_G(x,y)$, as required.

Inductive step. We follow closely the proof of Lemma 3.5. In inequality (8) we get that for a pair u, v of vertices such that all vertices of $\pi(u,v)$ are clustered in $\hat{\mathcal{U}}^{(i)}$, it holds that

$$d_{G\cup H_k}^{(h_i+1)\cdot\lceil \frac{d_G(u,v)}{\frac{1}{2}\alpha\cdot(1/\epsilon)^i}\rceil}(u,v) \;\leq\; d_G(u,v)(1+\zeta)(1+16c\cdot i\cdot\epsilon) + 8\cdot\alpha c\cdot(1/\epsilon)^{i-1}\cdot(1+\zeta)\ .$$

Now we consider a pair of vertices x, y such that $d_G(x,y) \leq \frac{1}{2}\alpha \cdot (1/\epsilon)^{i+1}$ and such that $\pi(x,y)$ is $\hat{\mathcal{U}}^{(i+1)}$-clustered. Let $w, z_1, r_1, r_2, z_2, w_2, C_1$, and C_2 be as in the proof of Lemma 3.5. In an analogous way we conclude that

$$d_{G\cup H_k}^{(h_{i+1})}(x,y) \;\leq\; (1+\zeta)(1+16c\cdot i\cdot\epsilon)d_G(x,y) + (1+\zeta)8\cdot\alpha c\cdot(1/\epsilon)^i\ ,$$

with $h_{i+1} \leq (h_i+1)(1/\epsilon+2)+2i+5$. For $\epsilon < 1/4$, we have $16\alpha\cdot c\cdot(1/\epsilon)^{i-1} \leq 4\cdot\alpha\cdot c\cdot(1/\epsilon)^i$, and the assertion of the lemma follows. ◻

For a pair of vertices $u, v \in V$ with $\hat{R}^{(k)}/2 < d_G(u,v) \leq \hat{R}^{(k)}$, $\hat{R}^{(k)} = 2^{k+1}$, the additive term of $(1+\zeta)8\alpha c\cdot(1/\epsilon)^{\ell-1}$ in (12) can be incorporated into the multiplicative stretch, i.e., we get

$$d_{G\cup H_k}^{(h_\ell)}(x,y) \leq (1+\zeta)\Big((1+16c(\ell-1)\epsilon)d_G(x,y) + 8c\hat{R}^{(k)}\cdot\epsilon\Big) \leq (1+\zeta)(1+16c\ell\cdot\epsilon)d_G(x,y)\ .$$

Set now $\epsilon' = 16c\cdot\ell\cdot\epsilon \leq 16c(\log(\kappa\rho)+1/\rho)\cdot\epsilon$. We get $\beta = O(\frac{\log\kappa+1/\rho}{\epsilon})^{\log\kappa+1/\rho}$ and stretch $(1+\zeta)(1+\epsilon)$.

Recall that $\zeta = \zeta_{k-1}$ is the value such that E'_{k-1} provides stretch $1+\zeta$. For the largest k_0 such that $\hat{R}^{(k_0)} = 2^{k_0+1} \leq \beta$, we have $\zeta_{k_0} = 0$. On the next scale we have $\zeta_{k_0+1} = \epsilon$, and generally, $1+\zeta_k = (1+\zeta_{k-1})(1+\epsilon)$, i.e., $1+\zeta_k = (1+\epsilon)^k$.

Hence the overall stretch of our hopset is $(1+\epsilon)^{\log\Lambda}$. By rescaling $\epsilon'' = \frac{\epsilon}{2\log\Lambda}$, we get stretch $(1+\epsilon'')^{\log\Lambda} \leq 1+\epsilon$. The number of hops becomes

$$\beta = O\left(\frac{\log\Lambda}{\epsilon}\cdot(\log\kappa+1/\rho)\right)^{\log\kappa+1/\rho}\ . \tag{13}$$

The expected number of edges in the hopset is $O(n^{1+1/\kappa}\cdot\log\Lambda)$, and its construction time is, whp, $O(n^\rho/\rho\cdot\log n\cdot\beta\cdot\log\Lambda)$ rounds.

THEOREM 3.11. *For any graph $G = (V,E,\omega)$ with n vertices and diameter Λ, $2 \leq \kappa \leq (\log n)/4$, $1/2 \geq \rho \geq 1/\kappa$, and $0 < \epsilon < 1$, our distributed algorithm for the Congested Clique model computes a (β,ϵ)-hopset H with expected size $O(n^{1+1/\kappa}\cdot\log\Lambda)$ in $O(n^\rho/\rho\cdot\log n\cdot\beta\cdot\log\Lambda)$ rounds, whp, with β given by* (13).[11]

[11]When messages of size $n^\rho\log n$ are allowed, the number of rounds is $O(\beta\cdot\log\Lambda/\rho)$.

Moreover, a single-scale hopset H_k that provides stretch at most $1+\epsilon$ using at most β hops for pairs u, v with $d_G(u,v) \in (2^k, 2^{k+1}]$, and expected size $O(n^{1+1/\kappa})$, can be computed in the same number of rounds.

Next we show that whenever $\Lambda = \text{poly(n)}$, β can be made independent of Λ and of n. In section 4 we remove this assumption on Λ.

Fix a parameter $1 \le t \le \log \Lambda$. We partition the set of at most $\log \Lambda$ indices k for which we build hopsets H_k into at most $(\log \Lambda)/t + 1$ groups, each consisting of t consecutive indices. Consider a single group $\{k_1, k_1+1, \ldots, k_1+t-1\}$ of indices. (Except maybe one group which can contain less than t indices.) We will compute all hopsets in this group using the hopset $H^{(k_1)}$; i.e., when conducting a Bellman–Ford exploration to depth at most $\delta \le 2^{k+1}$ for some index k in the group, we will conduct this exploration in $G \cup H^{(k_1)}$ for $O(\beta \cdot 2^{k+2-k_1}) = O(\beta \cdot 2^t)$ rounds. (The exploration depth of the superclustering step will still be two times larger than that of the interconnection step, in terms of both distance and hop-distance.) As a result we spend more time constructing each individual hopset H_k for $k > k_1$ in the group, but the hopsets that we compute provide a better approximation (because they rely on $(1+\zeta_{k_1})$-approximate distances that $G \cup H^{(k_1)}$ provides rather than on $(1+\zeta_{k-1})$-approximate distances that $H^{(k-1)}$ provides).

As a result the ultimate stretch of our hopset becomes just $(1+\epsilon)^{(\log \Lambda)/t}$. For a sufficiently small ϵ, this stretch is at most $1 + O(\epsilon \cdot \log \Lambda)/t$. We now rescale $\epsilon' = (\epsilon \cdot \log \Lambda)/t$. Our β becomes

$$\beta = O\left(\frac{(\log \kappa + 1/\rho) \log \Lambda}{\epsilon \cdot t}\right)^{\log \kappa + 1/\rho} . \tag{14}$$

Finally, the number of rounds becomes greater than it was in Theorem 3.11 by a factor of 2^t, i.e, it is now $O(n^\rho/\rho \cdot \log n \cdot \beta \cdot \log \Lambda \cdot 2^t)$, and we have the following result.

THEOREM 3.12. *For any graph $G = (V, E, \omega)$ with n vertices and diameter Λ, $2 \le \kappa \le (\log n)/4$, $1/2 \ge \rho \ge 1/\kappa$, $1 \le t \le \log \Lambda$, and $0 < \epsilon < 1$, a variant of our distributed algorithm for the Congested Clique model computes a (β, ϵ)-hopset H with expected size $O(n^{1+1/\kappa} \cdot \log \Lambda)$ in $O(n^\rho/\rho \cdot \log n \cdot \beta \cdot \log \Lambda \cdot 2^t)$ rounds whp, with β given by* (14).

Moreover, a single-scale hopset with expected size $O(n^{1+1/\kappa})$ and with the same β can also be constructed within this running time.

We note that our algorithm computes a single-scale hopset of expected size of just $O(n^{1+1/\kappa})$, because hopsets of previous scales are only used to compute (approximate) distances, whereas the stretch analysis only uses the current scale edges and the graph edges.

When $t = 1$ this recaptures Theorem 3.11. A useful assignment is $t = \rho \log n$, which increases the number of rounds by a factor of n^ρ. Rescaling $\rho' = 2\rho$, we obtain the same size and running time as in Theorem 3.11, with

$$\beta = O\left(\frac{(\log \kappa + 1/\rho) \log \Lambda}{\epsilon \cdot \rho \log n}\right)^{\log \kappa + 2/\rho} . \tag{15}$$

The constant factor 2 in the exponent can be made arbitrarily close to 1 at the expense of increasing the constant hidden in the O-notation in the base of the exponent. See also Theorem 3.15 for a result about constructing *path-reporting* (to be defined) hopsets in the Congested Clique model.

3.3. Distributed implementation in CONGEST model. In this section we consider a scenario where we have an underlying "backbone" n-vertex network $G = (V, E)$ of hop-diameter D, and a"virtual" weighted m-vertex network $\tilde{G} = (\tilde{V}, \tilde{E}, \tilde{\omega})$, $\tilde{V} \subseteq V$. Our objective is to compute a hopset H for $\tilde{G}$. Observe that the hopset H needs only to approximate distances in $\tilde{G}$, defined by the weight function $\tilde{\omega}$. The latter may have nothing to do with the distance function d_G of G.

Constructing hopsets in this framework turns out to be particularly useful for shortest paths computation and routing in the CONGEST model [Nan14, HKN16, EN16b].

The algorithm itself is essentially the same as in the distributed Congested Clique model, except that most of the communication is conducted via a BFS (breadth first search) tree τ rooted at a vertex rt of the backbone network G. (The tree has hop-diameter D, and it can be constructed in $O(D)$ distributed time [Pel00].)

Similarly to the Congested Clique model, we construct hopsets $H_{k_0}, H_{k_0+1}, \ldots, H_{\lceil \log \Lambda \rceil}$ consecutively, where k_0 is the maximum integer k such that $2^{k+1} \le \beta$. (H_{k_0} is set as $\emptyset$.) We consider a fixed phase i and describe how the superclustering and interconnection steps of this phase are implemented. In the superclustering step, we conduct the Bellman–Ford algorithm from the set $\mathit{Roots} = \{r_C \mid C \in \mathcal{S}_i\}$ to depth $4\beta + 2$ in $\tilde{G} \cup H^{(k-1)}$. Each vertex $v \in \tilde{V}$ maintains an estimate $\hat{d}(v)$ initialized as 0 if $v \in \mathit{Roots}$, and ∞ otherwise. For each Bellman–Ford step, we collect all of the m distance estimates at the root rt of τ via pipelined convergecast, and we broadcast all estimates to the entire graph via pipelined broadcast over τ. This requires $O(m + D)$ time. Since we have $O(\beta)$ such steps, overall the superclustering step of phase i, for any i, requires $O((m + D) \cdot \beta)$ time.

Next, we implement the interconnection step. Here we need to conduct a Bellman–Ford to hop-depth at most $2\beta + 1$ and to weighted depth at most $\delta'_i/2 = (1 + \zeta)\delta_i/2$ in $G' \cup H^{(k-1)}$ from all vertices of *URoots* separately in parallel. As was argued in the previous sections, each vertex v has to maintain $O(\mathit{deg}_i \cdot \log m) = O(m^{\rho} \cdot \log m)$ distance estimates, whp. (In fact, in expectation the total number of these estimates is $O(m \cdot \mathit{deg}_i) = O(m^{1+\rho})$. Note that only estimates less than or equal to $\delta'_i/2$ are broadcasted.)

This broadcast is also implemented via pipelined convergecast and broadcast, and it requires $O(D + m^{1+\rho} \cdot \log m)$ time, whp. Since we implement it for $O(\beta)$ steps, we obtain an overall time of $O((D + m^{1+\rho} \cdot \log m) \cdot \beta)$ for implementing a single step, and an overall time of $O((D + m^{1+\rho} \cdot \log m) \cdot \beta/\rho)$ for all steps. (The steps of stage 2 require this much time, while the running time of interconnection steps of stage 1 is dominated by this expression.)

Similarly to the Congested Clique model, we can decrease the β here too by grouping the $\log \Lambda$ scales into groups of size t. (Note that we can compute all hopsets within the same group in parallel. This multiplies the number of messages broadcast in each step by t, but the number of steps becomes smaller by the same factor.) We summarize the result in the following theorem.

THEOREM 3.13. *For any graph $G = (V, E, \omega)$ with diameter Λ and hop-diameter D, any m-vertex weighted graph $\tilde{G} = (\tilde{V}, \tilde{E}, \tilde{\omega})$ embedded in G, and any $2 \le \kappa \le (\log m)/4$, $1/2 \ge \rho \ge 1/\kappa$, $1 \le t \le \log \Lambda$, $0 < \epsilon < 1$, our distributed algorithm in the CONGEST model computes a (β, ϵ)-hopset H for $\tilde{G}$ with expected size $O(m^{1+1/\kappa} \cdot \log \Lambda)$ in $O((D + m^{1+\rho} \cdot \log m \cdot t)\beta/\rho \cdot \log \Lambda \cdot 2^t/t)$ rounds whp,*[12] *with β given by* (14).

[12]When messages of size $m^{\rho} \cdot \log m \cdot t$ are allowed, the number of rounds is $O((D+m)\beta/\rho \cdot \log \Lambda \cdot 2^t/t)$.

3.4. Path-reporting hopsets in distributed models. Next we introduce a property of distributed hopset construction which we call *awareness*, and we argue that our distributed construction satisfies this property. This property is useful for certain distributed applications of hopsets, such as in constructions of routing tables and sketches; cf. [EN16b].

This section focuses on the distributed CONGEST model; cf. section 3.3. At the end of this section we also indicate how these results apply to the Congested Clique model.

The awareness property stipulates that for every hopset edge $(u, v) \in H$, there exists a path $\pi(u, v)$ in G between u and v of weight $\omega_G(\pi(u, v)) = \omega_H(u, v)$, and moreover, every vertex x on this path is *aware* that it lies on $\pi(u, v)$ and knows its distances $d_{\pi(u,v)}(u, x)$ and $d_{\pi(u,v)}(x, v)$ to u and to v on this path, respectively, and its two G-neighbors u' and v' that lie on $\pi(u, v)$ along with the orientation. (That is, x knows that u' (resp., v') is its $\pi(u, v)$-neighbor that leads to u (resp., v).) Note that x may belong to multiple such paths. We also call a hopset with this property a *path-reporting hopset*, because it can be used to report approximate shortest paths.

Next we adapt our algorithm for constructing hopsets in the CONGEST model so that the awareness property will hold. Assume inductively that the property holds for hopsets $\bigcup_{j \le k-1} H_j$, and we will now show how to make it hold for H_k. (The induction base case holds vacuously, because for the maximum value k_0 such that $\hat{R}^{(k_0)} = 2^{k_0+1} \le \beta$, the hopset $H_{k_0} = \emptyset$.)

The first modification to the algorithm that we introduce is that Bellman–Ford executions will propagate not only distance estimates but also the actual paths that implement these estimates. First, consider the variant of our algorithm that provides (relatively) large β, i.e., the β given by (13). Since all of our Bellman-Ford invocations in this variant are $O(\beta)$-limited, these paths are of length up to $O(\beta)$, and the algorithm incurs a slowdown by only a factor of $O(\beta)$ as a result of this modification. Specifically, the Bellman–Ford invocations now require $O((D + \beta m^{1+\rho} \cdot \log m) \cdot \beta/\rho)$ overall time (whp) rather than $O((D + m^{1+\rho} \cdot \log m) \cdot \beta/\rho)$ time, which we had in section 3.3. Also, if messages of size $b' = O(\beta \cdot m^\rho \cdot \log m)$ are allowed, then these invocations can be implemented in $O((D + m)\beta/\rho)$ time.

With this modification, when a vertex $r_{C'}$ decides to add an edge $(r_C, r_{C'})$ to the hopset, it knows the entire path $\pi_{k-1}(r_C, r_{C'})$ in $E'^{(k-1)}$ which implements this edge. (Note that $|\pi_{k-1}(r_C, r_{C'})| = O(\beta)$.) After the construction of the hopset H_k is over, all vertices $v \in \tilde{V}$ broadcast all hopset edges (of H_k) along with their respective paths to the entire graph. We will refer to this broadcast as the *paths' broadcast*. Since in expectation $|H_k| = O(m^{1+1/\kappa})$, it follows that this broadcast requires expected $O(m^{1+1/\kappa} \cdot \beta + D)$ time. Using messages of size b', this expected time can be reduced to $O(D + m)$.

For an edge $e = (r_C, r_{C'}) \in H_k$, every vertex $v \in \tilde{V}(\pi_{k-1}(r_C, r_{C'}))$ hears this broadcast of e and of $\pi_{k-1}(r_C, r_{C'})$ and records to itself that it (i.e., the vertex v) belongs to $\tilde{V}(\pi_{k-1}(r_C, r_{C'}))$, calculates its distances to the endpoints r_C and $r_{C'}$, and computes its neighbors u and u' on $\pi_{k-1}(r_C, r_{C'})$ in the direction of r_C and $r_{C'}$, respectively.

Now vertex x involved in a path $\hat{\pi}(v, u) \subseteq \tilde{E}$ that implements an edge (v, u) of $\pi_{k-1}(r_C, r_{C'})$ needs also to record that it belongs to the path $\hat{\pi}(r_C, r_{C'})$ which implements the H_k-edge $e = (r_C, r_{C'})$ via edges of $\tilde{E}$. (This is in contrast to $\pi_{k-1}(r_C, r_{C'})$, which implements the same H_k-edge $(r_C, r_{C'})$ via edges of $E'^{(k-1)}$.) Since x hears of the edge $(v, u) \in \hat{\pi}(r_C, r_{C'})$, and since v stores the distances to v and u, it can infer from $\hat{\pi}(r_C, r_{C'})$ the distances to $r_C, r_{C'}$ (e.g., if v is the endpoint closer to r_C, then the distance to r_C that x stores is $d_{\pi(r_C,r_{C'})}(r_C,v)+d_{\pi(v,u)}(v,x))$. The appropriate neighbors of x are the same as those it stores for the edge (v,u).

To summarize, this modification of the algorithm ensures that our hopset constructing algorithm satisfies the awareness property. It does so by having its running time increased to $O((m^{1+1/\rho}\cdot\log m\cdot\beta+D)\beta/\rho\log\Lambda)$ rounds, whp. (The time to broadcast $O(m^{1+1/\kappa}\beta)$ messages is dominated by $m^{1+\rho}\cdot\log m\cdot\beta$ here.) For the variant in which we group t scales together, each path may consist of $O(\beta\cdot 2^t)$ hops, so we pay this factor in the number of messages sent.

Theorem 3.14. *For any graph $G=(V,E,\omega)$ with diameter Λ and hop-diameter D, $2\le\kappa\le(\log n)/4$, $1/2\ge\rho\ge 1/\kappa$, $1\le t\le\log\Lambda$, $0<\epsilon\le 1$, and any m-vertex weighted graph $\tilde{G}=(\tilde{V},\tilde{E},\tilde{\omega})$ embedded in G, our distributed algorithm in the CONGEST model computes a* path-reporting *(β,ϵ)-hopset H for $\tilde{G}$ with expected size $O(m^{1+1/\kappa}\cdot\log\Lambda)$ in $O((D+m^{1+\rho}\cdot\log m\cdot t\cdot\beta\cdot 2^t)\beta/\rho\cdot\log\Lambda\cdot 2^t/t)$ rounds whp, with β given by* (14). *Using messages of size $O(\beta\cdot 2^t\cdot t\cdot m^\rho\cdot\log m)$, the running time can be reduced to $O((D+m)\beta/\rho\cdot\log\Lambda\cdot 2^t/t)$.*

Moreover, a single-scale hopset with expected $O(m^{1+1/\kappa})$ edges and with the same β can also be constructed within this running time.

A similar adaptation enables us to construct path-reporting hopsets in the Congested Clique model, again by transmitting entire paths rather than distance estimates. As paths are of length at most $\beta\cdot 2^t$, this incurs such a factor in the number of rounds or in the message size. Specifically, we obtain the following result.

Theorem 3.15. *For any graph $G=(V,E,\omega)$ with n vertices and diameter Λ, $2\le\kappa\le(\log n)/4$, $1/2\ge\rho\ge 1/\kappa$, $1\le t\le\log\Lambda$, and $0<\epsilon\le 1$, a variant of our distributed algorithm for the Congested Clique model computes a* path-reporting *(β,ϵ)-hopset H with expected size $O(n^{1+1/\kappa}\cdot\log\Lambda)$ in $O(n^\rho/\rho\cdot\log n\cdot\beta^2\cdot\log\Lambda\cdot 2^{2t})$ rounds whp, with β given by* (14). *If messages of size $O(n^\rho\cdot\rho\cdot\log n\cdot\beta\cdot 2^t)$ are allowed, then the running time reduces to $O(\beta\cdot 2^t\cdot\log\Lambda)$.*

Moreover, a single-scale hopset with expected size $O(n^{1+1/\kappa})$ and with the same β can also be constructed within this running time.

Particularly useful settings of the parameter t are $t=\log(m^\rho)$ in the CONGEST model and $t=\log(n^{\rho/2})$ in the Congested Clique model. Then in Theorem 3.14 we obtain running time $O((D+m^{1+2\rho}\cdot\log^2 m\cdot\beta\rho)\beta/\rho^2\cdot m^\rho)$, and

$$\beta = O\left(\frac{\log\kappa+1/\rho}{\epsilon\cdot\rho}\right)^{\log\kappa+1/\rho}, \tag{16}$$

assuming $\Lambda\le\mathrm{poly}(m)$.

In Theorem 3.15 we obtain running time $O(n^{2\rho}/\rho\cdot\log^2 n\cdot\beta^2)$ with the same value of β as in (16), assuming $\Lambda\le\mathrm{poly}(n)$. One can also rescale $\rho'=2\rho$ and get running time $O(n^\rho/\rho\cdot\log^2 n\cdot\beta^2)$, with

$$\beta=O\left(\frac{\log\kappa+1/\rho}{\epsilon\cdot\rho}\right)^{\log\kappa+2/\rho}.$$

The constant factor 2 in the exponent can be made arbitrarily close to 1 at the expense of increasing a constant factor hidden in the O-notation in the base of the exponent. Also the assumption that $\Lambda\le\mathrm{poly}(n)$ will be removed in section 4. We also remark that the distributed algorithms for constructing the hopsets described so far can be seamlessly implemented in the Broadcast Congested Clique and Broadcast CONGEST models.

3.5. Streaming model. Our implementation of the hopset constructing algorithm in the streaming model follows closely our implementation from section 3.2 of the algorithm in the Congested Clique model.

Here too we construct the hopsets $H_{k_0}, H_{k_0+1}, \ldots, H_{\lceil \log \Lambda \rceil}$ consecutively. The hopset $H_{k_0} = \emptyset$. Next we describe how to construct a hopset H_k (for distances in the range $(2^k, 2^{k+1}]$, $2^{k+1} = \hat{R}^{(k)}$), assuming that the hopsets H_j, $j \leq k-1$, were already constructed.

We describe the streaming implementation in two regimes. In the first regime the space will be $\tilde{O}(n^{1+\rho})$, and the number of passes will be $\text{polylog}(n) \cdot \beta$. (In fact, in this section it will be $\text{polylog}(\Lambda) \cdot \beta$, but in section 4 we will replace the dependence on Λ by a similar dependence on n.) In the second regime we use $O(n^{1+1/\kappa} \cdot \log \Lambda)$ space, but the number of passes is much larger. (Specifically, it is $O(n^\rho \cdot \beta \cdot \log n \cdot \log \Lambda)$.)

To conduct the superclustering step of a phase i, in every pass over the stream E of edges of G we update distance estimates $\hat{d}(v)$ of the distances $d_G(\mathit{Roots}, v)$. After each pass we also read again the hopset $H^{(k-1)}$ and adjust distance estimates according to hopset edges. As a result of $4\beta + 2$ such passes we implement a $(4\beta + 2)$-limited Bellman–Ford algorithm in $E'^{(k-1)} = E \cup H^{(k-1)}$ which originated at *Roots*. As was argued above, this provides us with $(1 + \zeta_{k-1})$-approximate distances $d_G(\mathit{Roots}, v)$ for all v such that $d_G(\mathit{Roots}, v) \leq \delta_i$. This completes the description of the superclustering step. The space required for it is the space needed to keep the hopset, i.e., expected $O(n^{1+1/\kappa} \cdot \log \Lambda)$, and $O(n)$ space for distance estimates.

For the interconnection step, we conduct a similar $(2\beta + 1)$-limited Bellman–Ford exploration, but to depth at most $\delta'_i/2 = \frac{\delta_i(1+\zeta_{k-1})}{2}$, and from all cluster centers *URoots* separately in parallel. This necessitates that each vertex maintains in expectation $O(\mathit{deg}_i) = O(n^\rho)$ estimates, so we will use $O(n^{1+\rho}/\rho)$ space to guarantee (with constant probability) that none of the $1/\rho$ phases of stage 2 overflows (note that in stage 1, deg_i is much smaller than n^ρ). This completes the description of the streaming algorithm in the first regime. The space required is $O(n^{1+\rho}/\rho + n^{1+1/\kappa} \cdot \log \Lambda)$, and the number of passes is $O(\beta \cdot \log \Lambda)$.

For applications in which we use hopsets to provide approximate paths rather than distances, we can store actual paths in $E'^{(k-1)}$ for every edge of H_k that we create. Recall that a hopset appended with this information is a *path-reporting hopset.* Since every edge of H_k is implemented using at most $2\beta{+}1$ edges of $E'^{(k-1)}$, we can construct the path-reporting variant of the above hopset using space $O(n^{1+\rho}/\rho + n^{1+1/\kappa} \cdot \beta \cdot \log \Lambda)$, and the same number of passes as above.

Next, consider the regime where we allow (a much more limited) space $O(n^{1+1/\kappa} \cdot \log \Lambda)$. The superclustering steps can be implemented in the same way as described above. The interconnection steps, however, require a certain adaptation. Specifically, partition the interconnection step of phase i to $c \cdot \mathit{deg}_i \cdot \log n$ subphases for a sufficiently large constant c. On each subphase each exploration source, which was not sampled on previous subphases, samples itself iar with probability $1/\mathit{deg}_i$. Then the sampled exploration sources conduct the $\delta'_i/2$-distance-bounded $(2\beta{+}1)$-limited Bellman–Ford explorations. Recall Lemma 3.3, which asserts whp that every vertex is visited by at most $O(\mathit{deg}_i \cdot \log n)$ explorations in each phase. Since in each subphase every exploration happens with probability $1/\mathit{deg}_i$, the Chernoff bound implies that whp no vertex is visited by more than $O(\log n)$ explorations. We conclude that it suffices to use $O(n \cdot \log n)$ memory for all phases to maintain distance estimates. Here we also take a union-bound on all of the $\log \Lambda$ different scales, assuming that $\log \Lambda \leq \text{poly}(n)$. After $c \cdot \mathit{deg}_i \cdot \log n$ subphases, whp, each exploration source is sampled on at least one of the subphases, and thus the algorithm performs all the required explorations.

So we have space $O(n^{1+1/\kappa}\cdot\log\Lambda)$, and the number of passes is $O(n^{\rho}\cdot\log n\cdot\beta\cdot\log\Lambda)$. The size, stretch, and hopbound analysis of the resulting hopset is identical to that for the distributed Congested Clique model.

We can reduce the value of β by employing the idea used for the proof of Theorem 3.12. (See the discussion right before Theorem 3.12.) Specifically, fix some integer $t \geq 1$. The set of at most $\log\Lambda$ indices k for which we build hopsets H_k is partitioned into at most $(\log\Lambda)/t+1$ groups, each consisting of t consecutive indices (except maybe one which can consist of less than t indices). In a single group $\{k_1, k_1+1, \ldots, k_1+t-1\}$ of indices, each hopset H_k from the group is computed using hopset $H^{(k_1)}$ rather than $H^{(k-1)}$.

As a result, we now conduct Bellman–Ford explorations to depth $O(\beta\cdot 2^t)$, rather than just $O(\beta)$, and hence the number of passes increases by a factor of 2^t. Hopsets in the same group can be computed"in parallel," so the space increases by a factor of t. (Recall that in the path-reporting setting, we need to store paths of length $\beta\cdot 2^t$, which increases the space needed to store the hopset by this factor.) The hopbound β improves to the value in (14). We summarize this discussion in the next theorem.

THEOREM 3.16. *For any n-vertex graph $G=(V,E,\omega)$ of diameter Λ, any $2 \leq \kappa \leq (\log n)/4$, $1/2 \geq \rho \geq 1/\kappa$, $0 < \epsilon \leq 1$, and any $1 \leq t \leq \log\Lambda$, our streaming algorithm computes a (β,ϵ)-hopset with β given by* (14) *and with expected size $O(n^{1+1/\kappa}\log\Lambda)$. The resource usage is either*

1. *space $O(t\cdot n^{1+\rho}/\rho+n^{1+1/\kappa}\cdot\log\Lambda)$ (resp., space $O(t\cdot n^{1+\rho}/\rho+n^{1+1/\kappa}\cdot\beta\cdot 2^t\cdot\log\Lambda)$ for path-reporting), in expectation, and $O(\beta\log\Lambda\cdot 2^t)$ passes; or*
2. *space $O(n^{1+1/\kappa}\cdot\log\Lambda)$ (resp., $O(n^{1+1/\kappa}\cdot\beta\cdot 2^t\cdot\log\Lambda)$ for path-reporting) in expectation, and $O(n^{\rho}\cdot\beta\cdot\log n\cdot\log\Lambda\cdot 2^t)$ passes.*[13]

Moreover, a single-scale hopset H_k with expected size $O(n^{1+1/\kappa})$ can be computed within the space and pass complexities stated above.

In item 2 of Theorem 3.16 in the path-reporting case, it makes sense to set $t=\log(n^{1/\kappa})$. As a result we obtain space $O(n^{1+2/\kappa}\cdot\beta\cdot\log\Lambda)$, in expectation, and $O(n^{\rho+1/\kappa}\cdot\log\Lambda)$ passes, and β is given by (16). One can also rescale $\kappa'=\kappa/2$ and get expected space $O(n^{1+1/\kappa}\cdot\beta\cdot\log\Lambda)$, $O(n^{\rho+\frac{1}{2\kappa}})$ passes, and $\beta = O(\frac{(\log\kappa+1/\rho)\kappa}{\epsilon})^{\log\kappa+1/\rho+1}$.

3.6. PRAM model.

We construct hopsets $H_{k_0}, H_{k_0+1}, \ldots, H_{\lambda}$, $\lambda=\lceil\log\Lambda\rceil$, consecutively. Suppose that the hopset $H^{(k-1)}=\bigcup_{j=k_0}^{k-1} H_j$ has already been constructed. We now construct the hopset H_k for distances in the range $(2^k, 2^{k+1}]$, $2^{k+1}=\hat{R}^{(k)}$.

We designate a set $P_v=\{p_{v,1},\ldots,p_{v,\Delta}\}$ of $\Delta=c\cdot n^{\rho}\cdot\log n$ processors for every vertex $v\in V$, and a set $P_e=\{p_{e,1},\ldots,p_{e,\Delta}\}$ of Δ processors for every edge $e\in E'^{(k-1)}=E\cup H^{(k-1)}$.

Next, we describe how to implement the superclustering step of a phase i, and later we will explain how the interconnection step is implemented.

For the superclustering step, we use just one processor $p_v\in P_v$ for each vertex $v\in V$, and one processor $p_e\in P_e$ for each edge $e\in E'^{(k-1)}$. In the superclustering step we run a $(2\beta+1)$-limited Bellman–Ford algorithm in $E'^{(k-1)}$ which originated at *Roots*. At the beginning of an iteration of the Bellman–Ford algorithm, for every vertex $v\in V$, its processor p_v maintains an estimate $\tilde{d}(v)$ of its distance from *Roots*, and if $\tilde{d}(v)<\infty$, then p_v also stores the identity of a root $r_C\in$ *Roots*, such that $\tilde{d}(v)$

[13]Here we assume $\Lambda \leq 2^{\text{poly}(n)}$.

reflects the length of a path from r_C to v.[14] In the path-reporting case, p_v also stores an edge (u, v) through which v acquired this estimate.

To implement the iteration, for each edge $e = (u, v) \in E'^{(k-1)}$ incident on v, the processor p_e computes $\tilde{d}(u) + \omega(e)$. (Toward this end, all processors $\{p_e \mid u \in e\}$ need to read $\tilde{d}(u)$ concurrently. This, however, can be implemented in the EREW model in $O(\log n)$ time; cf. [JaJ92, Theorem 10.1].) The minimum $\min\{\tilde{d}(u) + \omega(u, v) \mid u \in \Gamma(v)\}$ can now be computed by the processors $\{p_e \mid v \in e\}$ in $O(\log n)$ parallel time. If this minimum is smaller than the current value of $\tilde{d}(v)$, then the estimate $\tilde{d}(v)$ is updated to be equal to this minimum. Hence the total EREW parallel time for one iteration of Bellman–Ford is $O(\log n)$, and the overall time for the superclustering step is, therefore, $O(\beta \cdot \log n)$.

Now, we turn to implementing the interconnection step. Here we implement a $(2\beta + 1)$-limited Bellman–Ford exploration, to depth at most $\delta'_i/2$, from all cluster centers *URoots* separately, in parallel.

Consider a single iteration of the Bellman–Ford algorithm. At the beginning of the iteration, every vertex v maintains estimates $\{\tilde{d}(v, x) \mid x \in \mathit{URoots}\}$ for all $x \in \mathit{URoots}$ that it heard from. Other estimates are (implicitly) set to ∞. For every edge $e = (u, v) \in E'^{(k-1)}$, incident on v, whp, there are at most $\Delta = c \cdot n^\rho \cdot \log n$ Bellman–Ford explorations that traverse this edge. Recall that we we have Δ processors $\{p_{e,1}, \ldots, p_{e,\Delta}\} = P_e$ designated to this edge. We designate a separate processor from P_e to each exploration that traverses e. With some notational ambiguity, we will denote by $p_{e,x}$ the processor from P_e designated to the exploration originated at a vertex x, traversing the edge e.

All processors $\{p_{e,x} \mid e = (v, u), \text{ for some } u\}$ read the value $\tilde{d}(u, x)$. (This concurrent read can be implemented in $O(\log n)$ time in EREW PRAM.) They also calculate the value $\tilde{d}(u, x) + \omega(u, v)$, and the minimum of these values over all neighbors u of v (separately for each x) is computed within additional $O(\log n)$ EREW PRAM time. If this minimum is smaller than the current $\tilde{d}(v, x)$, and if it is no greater than $\delta'_i/2$, then the value $\tilde{d}(v, x)$ is updated (by the processor $p_{v,x}$, designated to handle at v the exploration originated at x) to the new value.

To summarize, one iteration of Bellman–Ford explorations in an interconnection step can also be implemented in $O(\log n)$ EREW PRAM time. Hence, overall, the superclustering and the interconnection steps of a given phase require $O(\beta \cdot \log n)$ EREW PRAM time. Therefore, the total parallel time for computing a single-scale hopset H_k is $O(\beta \cdot \ell \cdot \log n) = O(\beta \cdot (\log \kappa + 1/\rho) \cdot \log n)$. Computing hopsets for all of the $\lceil \log \Lambda \rceil$ scales requires $O(\beta \cdot (\log \kappa + 1/\rho) \cdot \log n \cdot \log \Lambda)$ parallel time. The number of processors is $O(|E'^{(\lambda)}| \cdot n^\rho \cdot \log n) = O((|E| + |H^{(\lambda)}|) \cdot n^\rho \cdot \log n)$, and $\mathbb{E}(|H^{(\lambda)}|) = O(n^{1+1/\kappa} \cdot \lambda)$. Each single-scale hopset has size $O(n^{1+1/\kappa} \cdot \log n)$, whp, i.e., $H^{(\lambda)}$ has size $O(n^{1+1/\kappa} \cdot \log n \cdot \log \Lambda)$, whp.

THEOREM 3.17. *For any n-vertex graph $G = (V, E, \omega)$ of diameter Λ, and any $2 \leq \kappa \leq (\log n)/4$, $1/2 \geq \rho \geq 1/\kappa$, $0 < \epsilon \leq 1$, our parallel algorithm computes a (β, ϵ)-hopset with β given by* (13)*, and with expected size $O(n^{1+1/\kappa} \cdot \log \Lambda)$, in $O(\beta \cdot (\log \kappa + 1/\rho) \cdot \log n \cdot \log \Lambda)$ EREW PRAM time, using $O((|E| + n^{1+1/\kappa} \cdot \log n \cdot \log \Lambda) \cdot n^\rho \log n)$ processors, whp.*

More generally, we can also group scales into $\lceil \frac{\lambda}{t} \rceil$ groups, of size t each (except maybe one of them, which can be smaller) for a parameter t. We then compute all

[14]When we say that a processor p stores a value of a variable x, we mean that there is a memory location x, designated to the processor p, from which p can read the value of x. Other processors can also read from and write to this location.

hopsets in a group via explorations in the lowest-scale hopset of that group. As a result, the explorations become $O(2^t \cdot \beta)$-limited, instead of $(2\beta+1)$-limited, but β decreases. (β is now given by (14).) The number of processors grows by a factor of t, because all hopsets in the same group are now computed in parallel.

Theorem 3.18. *For any n-vertex graph $G = (V, E, \omega)$ of diameter Λ, any $2 \le \kappa \le (\log n)/4$, $1/2 \ge \rho \ge 1/\kappa$, $0 < \epsilon \le 1$, and any $1 \le t \le \log \Lambda$, our parallel algorithm computes a (β, ϵ)-hopset with β given by (14) and with expected size $O(n^{1+1/\kappa} \cdot \log \Lambda)$, in $O(\beta \cdot (\log \kappa + 1/\rho) \cdot \log n \cdot 2^t \cdot \frac{\log \Lambda}{t})$ EREW PRAM time, using $O((|E| + n^{1+1/\kappa} \cdot \log n \cdot \log \Lambda) \cdot n^{\rho} \log n \cdot t)$ processors, whp.*

Moreover, a single-scale hopset of expected size $O(n^{1+1/\kappa})$ can be computed using the same resources.

Naturally, Theorem 3.18 can be made path-reporting by keeping for every hopset edge a path (of length $O(2^t \cdot \beta)$) of lower-scale hopset edges that implement the hopset edge.

4. Eliminating dependence on the aspect ratio.

In this section, we show a general reduction that removes the dependence on the aspect ratio of the graph from both the running time and the hopset size. This reduction is based on a classical reduction of [KS97]. The latter reduction, however, applies just to shortest paths computation, not to computation of hopsets. Also, the reduction of [KS97] was devised for the PRAM setting (see also [MPVX15]), while our reduction is applicable in the Congested Clique, CONGEST, and streaming models as well.

Assume, without loss of generality, that the minimal distance in the graph $G = (V, E)$ is 1. Fix a parameter $0 < \epsilon < 1/2$. For any scale index $k \ge 1$, we define a graph G_k that contains the edges of weight at most 2^{k+2}, and in which every edge of weight less than $(\epsilon/n) \cdot 2^k$ is contracted. By contraction we mean identifying the edge endpoints while keeping the shortest edge among parallel edges. We refer to the vertices of G_k as *nodes*, where each node is a subset of V. The weight of an edge $(X, Y) \in E(G_k)$ is set to be

$$\mathcal{W}(X,Y) = \omega(x,y) + (\epsilon/n) \cdot 2^k \cdot (|X| + |Y|) \ , \tag{17}$$

where $x \in X$, $y \in Y$, and the edge $(x, y) \in E$ is the shortest edge between a vertex of X to a vertex of Y. (The purpose of the additional term $(\epsilon/n) \cdot 2^k \cdot (|X| + |Y|)$ is to guarantee that distances in G_k are no shorter than those in G, while ensuring that for pairs of distance $\ge 2^k$, the distance in G_k does not increase by too much.)

In order to guarantee a small number of hops even for contracted vertices, we add an additional set of edges S to the hopset. We refer to the edges of S as *star edges.* Every node U in G_k has a designated center $u \in U$, and we add edges from u to every vertex in U to the hopset. Consider a contraction of an edge (x', y'), $x' \in X$, $y' \in Y$, connecting nodes X, Y, with centers $x \in X$ and $y \in Y$. Assuming $|X| \ge |Y|$, x is declared the center of $U = X \cup Y$, and we add to S edges from x to every vertex of Y. The weight of the edge (x, z) for each $z \in Y$ is set as

$$\mathcal{W}(x,z) = (\epsilon/n) \cdot 2^k \cdot |U| \ . \tag{18}$$

This value dominates $d_G(x, z)$, as there exists a path in G from x to z consisting of at most $|U| - 1$ edges, each of weight at most $(\epsilon/n) \cdot 2^k$.

Claim 4.1. $|S| \le n \log n$.

Proof. Assume inductively that every node U of size $s = |U|$ has at most $s \log s$ internal edges added to the hopset by the above process. This holds for singletons $|U| = 1$, which have 0 internal edges. When we combine X and Y of sizes s_1, s_2 (respectively), we add at most $s_2 = \min\{s_1, s_2\}$ edges. By induction there were already at most $s_1 \log s_1 + s_2 \log s_2$ edges, so the total number of edges in S between vertices of U is at most

$$\begin{aligned} &s_1 \log s_1 + s_2 \log s_2 + s_2 = s_1 \log s_1 + s_2 \log(2s_2) \\ &\le s_1 \log(s_1 + s_2) + s_2 \log(s_1 + s_2) = (s_1 + s_2) \log(s_1 + s_2) \ . \end{aligned}$$

When the scale index k is sufficiently large we have at a certain point a graph with a single node V, and at this point we added at most $n \log n$ edges throughout the process. ◻

CLAIM 4.2. *Let $x, y \in V$ such that $d_G(x, y) \in (2^k, 2^{k+1}]$, and let $X, Y \in V(G_k)$ be the two nodes containing x, y (respectively) in G_k; then*

$$d_G(x, y) \le d_{G_k}(X, Y) \le (1 + 2\epsilon) d_G(x, y) \ .$$

Proof. We start with the right-hand-side inequality. Let $x = x_0, \ldots, x_q = y$ be the shortest path P in G between x and y. Let $X = X_0, X_1, \ldots, X_q = Y$ be the corresponding nodes in G_k (that is, $x_j \in X_j$), and let $X = X_0, \ldots, X_p = Y$ be that path with all repetitions and loops removed. For $0 \le j \le p$, denote by $s(j)$ (resp., $t(j)$) the index of the first (resp., last) vertex of P that X_j contains. Since by (17) the weight of each edge (X_{j-1}, X_j) for $1 \le j \le p$ in G_k is defined using the shortest weight edge, we have $\mathcal{W}(X_{j-1}, X_j) \le \omega(x_{t(j-1)}, x_{s(j)}) + (\epsilon/n) \cdot 2^k \cdot (|X_{j-1}| + |X_j|)$. As each term $|X_j|$ appears at most twice, and $\sum_{j=0}^{p} |X_j| \le n$, we obtain that

$$\begin{aligned} d_{G_k}(X, Y) &\le \sum_{j=1}^{p} \mathcal{W}(X_{j-1}, X_j) \\ &\le \sum_{j=1}^{p} (\omega(x_{t(j-1)}, x_{s(j)}) + (\epsilon/n) \cdot 2^k \cdot (|X_{j-1}| + |X_j|)) \\ &\le d_G(x, y) + 2\epsilon \cdot 2^k \\ &\le (1 + 2\epsilon) \cdot d_G(x, y) \ . \end{aligned}$$

We now turn to prove the left-hand-side inequality. Let $X = Y_0, \ldots, Y_r = Y$ be the shortest path in G_k from X to Y. For each $1 \le j \le r$, denote by $(y_{j-1}, z_j) \in E$ the edge of minimal weight connecting Y_{j-1} and Y_j, with $y_{j-1} \in Y_{j-1}$ and $z_j \in Y_j$. Since each Y_j consists of $|Y_j| - 1$ edges that were contracted, each of weight at most $(\epsilon/n) \cdot 2^k$, we have that $d_G(z_j, y_j) \le \mathrm{diam}(Y_j) \le (\epsilon/n) \cdot 2^k \cdot |Y_j|$. Moreover, this inequality holds for every pair z'_j, y'_j of vertices in Y_j. Hence,

$$\begin{aligned} &d_{G_k}(X, Y) = \sum_{j=1}^{r} \mathcal{W}(Y_{j-1}, Y_j) \\ &\overset{(17)}{=} \sum_{j=1}^{r} \left[\omega(y_{j-1}, z_j) + (\epsilon/n) \cdot 2^k \cdot (|Y_{j-1}| + |Y_j|)\right] \\ &\ge \sum_{j=1}^{r} [d_G(y_{j-1}, z_j) + d_G(z_j, y_j)] + d_G(x, y_0) + d_G(z_r, y) \\ &\ge d_G(x, y). \end{aligned}$$

◻

Some of the scales k are redundant—define K to be the set of scales k so that there exists an edge of weight in the range $[2^k/n, 2^{k+1}]$. We will refer to the scales in K as *relevant* scales. Observe that if there is no edge in this range, then there is no pair of vertices whose distance in G is in the range $(2^k, 2^{k+1}]$, so we do not need a hopset for this scale. We can see that $|K| \le \tilde{O}(|E|)$, as every edge can induce a logarithmic number of scales to K.

For every $k \in K$ and every connected component of G_k, we will execute the algorithm for constructing a single-scale (β, ϵ)-hopset H_k as in Theorem 3.7. Whenever we add a hopset edge between two nodes X, Y, we put the same hopset edge between their centers.

LEMMA 4.3. *The set $H = S \cup \bigcup_{k \in K} H_k$ is a $(6\beta + 5, 6\epsilon)$-hopset for G.*

Proof. Fix $x, y \in V$, and let $k \in K$ be the scale such that $d_G(x, y) \in (2^k, 2^{k+1}]$. Let $X, Y \in V(G_k)$ so that $x \in X$ and $y \in Y$. Since H_k is a (β, ϵ)-hopset for the range $(2^k, 2^{k+1}]$ in G_k, it is also a $(2\beta + 1, \epsilon)$-hopset for the range $(2^k, 2^{k+2}]$, and by Claim 4.2 we have that, indeed, $2^k < d_{G_k}(X, Y) \le (1 + 2\epsilon)2^{k+1} < 2^{k+2}$. It follows that there exists a path $(X = X_0, X_1, \ldots, X_p = Y)$ in $G_k \cup H_k$ containing $p \le 2\beta + 1$ edges of length at most $(1+\epsilon)d_{G_k}(X, Y)$. For every X_j, denote by u_j its center. Note that each edge (X_{j-1}, X_j) could be either a hopset edge or an edge of G_k. In the latter case there are vertices $y_{j-1} \in X_{j-1}$ and $x_j \in X_j$ so that $(y_{j-1}, x_j) \in E$, and the weight of the edge by (17) is $\hat{\omega}(X_{j-1}, X_j) = \omega(y_{j-1}, x_j) + (\epsilon/n) \cdot 2^k \cdot (|X_{j-1}| + |X_j|)$. For ease of notation, in the former case (a hopset edge) we write $y_{j-1} = u_{j-1}$ and $x_j = u_j$, and we let $\hat{\omega}(y_{j-1}, x_j)$ denote the weight of the hopset edge (recall that this edge indeed connects nodes' centers u_{j-1}, u_j). Then the following is a path from x to y in $G \cup H$:

$$P = (x = x_0, u_0, y_0, x_1, u_1, y_1, x_2, u_2, y_2, \ldots, x_p, u_p, y_p = y) \ .$$

First, note that the path contains at most $2(p + 1)$ star edges (i.e., edges of S) and p edges between nodes. Since $p \le 2\beta + 1$, this path has at most $6\beta + 5$ edges. Next, we bound the stretch. We have that the total weight of the edges in $P \cap S$ is

$$\sum_{j=0}^{p} (\hat{\omega}(x_j, u_j) + \hat{\omega}(y_j, u_j)) \overset{(18)}{=} 2 \sum_{j=0}^{p} (\epsilon/n) \cdot 2^k \cdot |X_j| \le (\epsilon/n) \cdot 2^{k+1} \cdot n \le 2\epsilon \cdot d_G(x, y) \ .$$

We note that the length of the path $X_0, X_1, \ldots, X_p$ is

$$\sum_{j=1}^{p} \hat{\omega}(y_{j-1}, x_j) \le (1 + \epsilon) d_{G_k}(X, Y) \le (1 + \epsilon)(1 + 2\epsilon) d_G(x, y) \ ,$$

where the last inequality is by Claim 4.2. Combining these inequalities implies that the length of P is at most

$$2\epsilon \cdot d_G(x, y) + (1 + 4\epsilon) d_G(x, y) = (1 + 6\epsilon) d_G(x, y) \ . \qquad \square$$

We say that a node U in the graph G_k is *active* if it has degree at least 1, and we denote by n_k the number of active nodes in G_k. The nodes of the graphs $\{G_k\}_{k \in K}$ induce a laminar family $\mathcal{L}$ on V, which contains at most $2n - 1$ distinct sets. We will refer to $\mathcal{L}$ as the *laminar family* of the algorithm.

CLAIM 4.4. $\sum_{k \in K} n_k = O(n \log n)$.

Proof. In order to bound the total number of active nodes in all of these graphs, it suffices to show that each node can be active in at most $\log(n/\epsilon) + 2$ scales. To this end, consider a node U that is active for the first time in G_k, so U has an edge containing U of weight at most 2^{k+2}. (That is, k is the smallest scale such that U is active in G_k.) After $q = \log(n/\epsilon) + 2$ scales, in G_{k+q}, this edge will be of weight at most $(\epsilon/n) \cdot 2^{k+q}$. Thus it will be contracted, and the node U will merge with some other node and never appear again in $G_{k'}$ for $k' \geq k + q$. ☐

Corollary 4.5. *$|H| = O(n^{1+1/\kappa} \cdot \log n)$, in expectation.*

Proof. By Claim 4.1, we have that $|S| \leq n \log n$. For each $k \in K$, by Theorem 3.7, the expected size of the single-scale hopset H_k is at most $O(n_k^{1+1/\kappa})$ (observe that isolated nodes do not participate in the hopset). We conclude by Claim 4.4 that

$$\begin{aligned}|H| &\leq |S| + \sum_{k \in K} |H_k| \leq n \log n + \sum_{k \in K} O(n_k^{1+1/\kappa}) \\ &\leq n \log n + n^{1/\kappa} \sum_{k \in K} O(n_k) = O(n^{1+1/\kappa} \cdot \log n).\end{aligned}$$ ☐

4.1. Implementation in the centralized model. Computing the graphs G_k can be done in $\tilde{O}(|E|)$ time in a straightforward manner. First, sort the edges by weight, add all edges of weight at most 2 to obtain G_0, and create G_k from G_{k-1} by adding edges of weight in the range of $(2^k, 2^{k+1}]$ and contracting edges of weight in range of $(\epsilon/n) \cdot (2^{k-1}, 2^k]$. Adding the star edges of S is also straightforward.

Note that for any $k \in K$, the aspect ratio of G_k is $O(n/\epsilon)$. By Theorem 3.7, the expected running time for computing the hopset H_k is $O(|E(G_k)| + n_k \log n) \cdot n^\rho/\rho$. Observe that each edge participates in at most $\log(n/\epsilon) + 2$ scales. We have $\sum_{k \in K} |E(G_k)| \leq O(|E| \cdot \log n)$. Also, we spend time only on relevant scales, and the number of relevant scales is at most $K = O(|E| \cdot \log n)$. By Claim 4.4, we conclude that the total expected running time is

$$\sum_{k \in K} O(|E(G_k)| + n_k \cdot \log n) \cdot n^\rho/\rho \;=\; O(|E| + n \log n) \cdot n^\rho/\rho \cdot \log n \;.$$

We thus have the following theorem.

Theorem 4.6. *For any graph $G = (V, E, \omega)$ with n vertices, $2 \leq \kappa \leq (\log n)/4$, $1/2 \geq \rho \geq 1/\kappa$, and $0 < \epsilon < 1/2$, our algorithm constructs a (β, ϵ)-hopset H with $O(n^{1+1/\kappa} \cdot \log n)$ edges in expectation, in expected time $O((|E| + n \log n)(n^\rho/\rho \cdot \log n))$, with $\beta = O\left(\frac{\log \kappa + 1/\rho}{\epsilon}\right)^{\log \kappa + 1/\rho}$.*

4.2. Implementation in the Congested Clique model. In this model, the computation of the nodes of the graphs $\{G_k\}_{k \in K}$ can be done in $O(\log n)$ rounds. Recall our assumption that any polynomial combination of edge weights can be sent in a single message. To compute the vertices of $\{G_k\}$, we use a variant of computing in a distributed manner a minimum spanning tree (MST). Consider a single iteration of Boruvka's algorithm for computing the MST [CLRS09]. Boruvka's algorithm starts from a collection of MST fragments. We assume that at the beginning of the iteration every vertex v knows the entire current partition of G into MST fragments (subtrees of the MST). This assumption holds at the beginning of the first iteration, where the fragments are singleton vertices. (Every vertex v sends a message with its identity to every other vertex in G, and then the assumption holds at the beginning of the first iteration.) Generally, in every Boruvka iteration every vertex v sends to all other vertices of G a message containing its identity, the identity of its fragment F_v, and the lightest cross-fragment edge e, along with its weight, incident to v.

Every vertex $x \in V$ hears all of these messages and computes locally the Boruvka forest of fragments. In particular, as a result of this computation x knows which edges incident on it cross between different fragments on the next iteration. Let F'_x be the fragment containing x in the next iteration. F'_x can be viewed as a connected component of the Boruvka forest. Vertices of the Boruvka forest are fragments of the current iteration, and its edges are minimum weight outgoing (or crossing) edges of these fragments. We will refer to them as MWOEs.

To build node sets of graphs $\{G_k\}$, the vertex x processes these MWOEs in order of increasing scale index. Let k be the smallest scale index of MWOEs in F'_x, and let $F'_x(k)$ be the subgraph of F'_x obtained by leaving only the scale-k MWOEs and removing all other MWOEs. We write ${F'}_x^{(0)} = F'_x(k)$, $k = k^{(0)}$. The connected components of $F'_x(k)$ are now locally merged by x into nodes of G_k. The vertex x records these nodes and updates the (initially empty) lists $L(x)$. Each of these components/nodes is now contracted, and we obtain ${F'}_x^{(1)}$. The same procedure is now repeated locally with the second smallest scale $k^{(1)}$, such that an MWOE of scale $k^{(1)}$ appears in F'_x (in other words, the smallest scale of an edge of ${F'}_x^{(1)}$). Then it is performed with the third smallest scale index, the fourth smallest, etc.

It is easy to verify that as a result of this procedure, within $O(\log n)$ time (i.e., the time required for computing MST by this algorithm), the node sets of all graphs G_k are computed and are known to all vertices.

Finally, the root will appoint a coordinator vertex $c(U) \in U$ for every node U, who will be in charge of communications for that node. We require that every vertex participates at most once as a coordinator for a nontrivial node (a node of size > 1). Note that when we create a new node U by combining two others X, Y, we can maintain the property that there will be a vertex $u \in U$ that never was a coordinator—this holds by induction for $x \in X$ and $y \in Y$, so we can simply set x as the coordinator for U, and $u = y$ will be the "free" vertex. (We do not use the center as coordinator, since the same vertex can be a center in numerous scales.) This computation can be performed in the Broadcast Congested Clique model.

Similarly to the case of the streaming model, we run the hopset algorithm of section 3.2 for all graphs G_k in parallel. Let us briefly review how to implement each step in the algorithm. Recall that the main ingredients are Bellman–Ford explorations, in every iteration of which every vertex sends its current estimate to all of its neighbors. In the graph G_k, for every node U the coordinator $c(U)$ will send the appropriate estimate $\hat{d}$ (along with the scale k of the node U) to all the vertices in the graph. Every vertex $u \in V$ that receives this message, sends to its coordinator at level k the updated estimate $\hat{d} + \hat{\omega}$, where $\hat{\omega}$ is the shortest length of an edge (given by (17)) connecting u to a vertex in U. (Recall that each vertex knows the entire laminar family $\mathcal{L}$ of sets, and hence can compute locally the nodes of G_k.) The coordinator of each node will keep the shortest of these as the estimate for its node. Thus, for each step of Bellman–Ford, we need two rounds in which all communication is over edges containing a coordinator.

We now analyze the required number of rounds. We charge the cost of each exploration step to the coordinators of the nodes. The point is that every vertex can be a coordinator in at most $\log(n/\epsilon) + 2$ different scales $k \in K$, because once a node is active, after so many scales it must be merged with another node, which will necessarily have a different coordinator. We conclude that the load on any edge, arising from its participation in many different graphs, is only $O(\log n)$. Hence the number of rounds as a result of this simulation grows only by a factor of $O(\log n)$ in the Congested Clique model. In the Broadcast Congested Clique model, however, a vertex v cannot within the same round send distinct messages to all coordinators of nodes that contain it. Instead, $O(\log \Lambda)$ rounds will be required for this. Also, recall that the aspect ratio of each G_k is $O(n/\epsilon)$. By applying the single-scale versions of Theorems 3.12 and 3.15 on each G_k, we conclude with the following.

THEOREM 4.7. *For any graph $G = (V, E, \omega)$ with n vertices, $2 \le \kappa \le (\log n)/4$, $1/2 \ge \rho \ge 1/\kappa$, $1 \le t \le \log n$, and $0 < \epsilon < 1/2$, our distributed algorithm for the Congested Clique model computes a (β, ϵ)-hopset H with expected size $O(n^{1+1/\kappa} \cdot \log n)$ in $O(n^\rho/\rho \cdot \log^3 n \cdot \beta \cdot 2^t)$ rounds whp,*[15] *with β given by* (22)*. For a path-reporting hopset, the number of rounds becomes larger by a factor of $\beta \cdot 2^t$. In the Broadcast Congested Clique model, the respective running times are larger by a factor of* $\log_n \Lambda$.

To get a hopset with β independent of n, we set $t = \log n^{\mu\rho}$ for an arbitrarily small constant $\mu > 0$. We then rescale $\rho' = (1 + \mu)\rho$. As a result, we obtain

$$\beta \;=\; O\left(\frac{1}{\epsilon \cdot \rho}(\log \kappa + 1/\rho)\right)^{\log \kappa + \frac{1+\mu}{\rho}} . \tag{19}$$

The O-notation in (19) hides a (constant) factor of $1/\mu$ in the base of the exponent. The number of rounds becomes, whp, $O(n^\rho/\rho \cdot \log^3 n \cdot \beta)$. In the path-reporting case, we set $t = \log n^{(\mu/2)\rho}$ and rescale in the same way as above. As a result, the running time becomes $O(n^\rho/\rho \cdot \log^3 n \cdot \beta^2)$, whp. The hopbound and the hopset size are the same as in the non–path-reporting case.

COROLLARY 4.8. *For any graph $G = (V, E, \omega)$ with n vertices, $2 \le \kappa \le (\log n)/4$, $1/2 \ge \rho \ge 1/\kappa$, and $0 < \epsilon < 1/2$, and any constant $\mu > 0$, our distributed algorithm for the Congested Clique model computes a (β, ϵ)-hopset H with expected size $O(n^{1+1/\kappa} \cdot \log n)$ in $O(n^\rho/\rho \cdot \log^3 n \cdot \beta)$ rounds whp, with β given by* (19)*. For a path-reporting hopset, the number of rounds is $O(n^\rho/\rho \cdot \log^3 n \cdot \beta^2)$, whp. In the Broadcast Congested Clique model, the respective running times are larger by a factor of* $\log_n \Lambda$.

4.3. Implementation in the CONGEST model. In this model we are given a "virtual" graph $\tilde{G} = (\tilde{V}, \tilde{E}, \tilde{\omega})$, where $\tilde{V} \subseteq V$, $|\tilde{V}| = m$, on which we wish to compute a hopset. For each scale k, we construct a virtual graph $\tilde{G}_k = (\tilde{V}_k, \tilde{E}_k)$, whose nodes are subsets of $\tilde{V}$. To construct the node sets of the graphs $\tilde{G}_k$, we do essentially the same as we did in the Congested Clique model. Specifically, in the Congested Clique model a vertex v sent a message to other vertices. In the CONGEST model we do the same via a broadcast over a BFS tree T of the entire graph. For every scale k, in each iteration of the Bellman–Ford algorithm in $\tilde{G}_k$, we send $O(|\tilde{V}_k|)$ messages. So the total number of messages sent is $M = O(\sum_{k \in K} |\tilde{V}_k|) = O(|\tilde{V}| \log m) = O(m \cdot \log m)$. These M messages can be convergecasted and broadcasted over the BFS tree T of G in $O(M + D) = O(D + m \cdot \log m)$ rounds, where D is the hop-diameter of G.

The hopset algorithm from section 3.3 will require a subtle modification: When a coordinator broadcasts a distance estimate $\hat{d}$, we cannot afford to have every vertex in every node notify its coordinator of its own estimate. Instead, we conduct a pipelined

[15]When messages of size $n^\rho \log^2 n$ are allowed, the number of rounds is $O(\log n \cdot \beta \cdot 2^t/\rho)$. For the path-reporting hopset, we can have the same number of rounds if messages of size $n^\rho \cdot \log^2 n \cdot \beta \cdot 2^t$ are allowed.

convergecast on the global BFS tree T while computing the minimum estimate among estimates within each node U. Then the root broadcasts the updated distance estimates to all of the coordinators. Since the total number of computed minimums is M, this computation requires only $O(M + D)$ rounds. See, e.g., [Pel00, Lemma 3.4.6] for a detailed analysis of pipelined convergecast.

By Theorems 3.13 and 3.14, the number of rounds required for computing all t hopsets of a given group of scales is $O((D + m^{1+\rho} \cdot \log m \cdot t)\beta/\rho \cdot 2^t)$ in the non–path-reporting case and is $O((D + m^{1+\rho} \cdot \log m \cdot t \cdot \beta \cdot 2^t)\beta/\rho \cdot 2^t)$ in the path-reporting case. (Both bounds are whp.) To get a hopset for all scales, this expression was multiplied by the number of groups, i.e., $\lceil \frac{\log \Lambda}{t} \rceil$. When computing the hopsets H_k of G_k, the number of groups is $\lceil \frac{\log O(m/\epsilon)}{t} \rceil = O(\frac{\log m}{t})$, because the aspect ratio of each G_k is $O(m/\epsilon)$. The number of messages convergecasted and broadcasted over the BFS tree τ of the entire network in each iteration of the Bellman–Ford algorithm (which is executed now in parallel in all graphs $\tilde{G}_k$) is $\sum_{k \in K} O(m_k^{1+\rho} \log m_k \cdot t) = O(m^{1+\rho} \cdot \log^2 m \cdot t)$, whp, in the non–path-reporting case, and is $\sum_{k \in K} O(m_k^{1+\rho} \cdot \log m_k \cdot t \cdot \beta \cdot 2^t) = O(m^{1+\rho} \cdot \log^2 m \cdot t \cdot \beta \cdot 2^t)$, whp, in the path-reporting case. When messages of size b are allowed, for a parameter b, pipelined convergecast and broadcast of M messages can be performed in $O(D + M/b)$ time. We also note that this algorithm can be seamlessly implemented in the Broadcast CONGEST model.

We summarize this discussion with the following theorem.

THEOREM 4.9. *For any graph $G = (V, E)$ with hop-diameter D, any m-vertex weighted graph $\tilde{G} = (\tilde{V}, \tilde{E}, \tilde{\omega})$ embedded in G, and any $2 \le \kappa \le (\log m)/4$, $1/2 \ge \rho \ge 1/\kappa$, $1 \le t \le \log m$, $0 < \epsilon < 1/2$, our distributed algorithm in the Broadcast CONGEST model computes a (β, ϵ)-hopset H for $\tilde{G}$ with expected size $O(m^{1+1/\kappa} \cdot \log m)$, in $O((D + m^{1+\rho} \cdot \log^2 m \cdot t) \cdot \beta/\rho \cdot \log m \cdot 2^t/t)$ rounds whp,*[16] *with β given by (22), with n replaced by m. For a path-reporting hopset, the number of rounds becomes $O((D + m^{1+\rho} \cdot \log^2 m \cdot t \cdot \beta \cdot 2^t) \cdot \beta/\rho \cdot \log m \cdot 2^t/t)$.*

To get β independent of m, we set $t = \log m^{\mu\rho}$. As a result we get

$$\beta = O\left(\frac{1}{\epsilon\rho}(\log \kappa + 1/\rho)\right)^{\log \kappa + \frac{1}{\rho}} . \tag{20}$$

The running time is $O((D + m^{1+\rho} \cdot \log^3 m \cdot \rho) \cdot \beta/\rho^2 \cdot m^{\mu\rho})$, whp, in the non–path-reporting case.

In the path-reporting case we get time $O((D + m^{\rho(1+\mu)} \cdot \log^3 m \cdot \rho \cdot \beta) \cdot \frac{\beta}{\rho^2} \cdot m^{\mu\rho})$, whp. By rescaling $\rho' = \rho(1 + \mu)$, we get

$$\beta = O\left(\frac{1}{\epsilon\rho}(\log \kappa + 1/\rho)\right)^{\log \kappa + \frac{1+\mu}{\rho}} , \tag{21}$$

and the running time is $O((D + m^{\rho} \cdot \log^3 m \cdot \rho \cdot \beta) \cdot \frac{\beta}{\rho^2} \cdot m^{\mu\rho})$, whp.

COROLLARY 4.10. *For any graph $G = (V, E)$ with hop-diameter D, any m-vertex weighted graph $\tilde{G} = (\tilde{V}, \tilde{E}, \tilde{\omega})$ embedded in G, any $2 \le \kappa \le (\log m)/4$, $1/2 \ge \rho \ge 1/\kappa$, and $0 < \epsilon < 1/2$, and any constant $\mu > 0$, our distributed algorithm for the Broadcast CONGEST model computes a (β, ϵ)-hopset H for $\tilde{G}$ with expected size $O(m^{1+1/\kappa} \cdot$*

[16]When messages of size $m^{\rho} \log^2 m \cdot t$ are allowed, the number of rounds is $O((D + m) \cdot \beta/\rho \cdot \log m \cdot 2^t/t)$. For the path-reporting case, we can have the same number of rounds with messages of size $m^{\rho} \cdot \log^2 m \cdot t \cdot \beta \cdot 2^t$.

$\log m$), in $O((D + m^{1+\rho} \cdot \log^3 m \cdot \rho) \cdot \beta/\rho^2 \cdot m^{\mu\rho})$ rounds whp, with β given by (20)*. For a path-reporting hopset, β is given by* (21)*, and the number of rounds becomes $O((D + m^{1+\rho} \cdot \log^3 m \cdot \beta \cdot \rho) \cdot m^{\rho\mu} \cdot \beta/\rho^2)$.*

4.4. Implementation in the streaming model. We assign $O(n \log n)$ words of memory for storing a data structure for nodes of the graphs $\{G_k\}_{k \in K}$. The main observation is that whenever we contract an edge between nodes X, Y with $|X| \geq |Y|$, only the vertices of Y get a new center, but the size of the node containing them is at least doubled. This implies that each vertex changes the center of the node containing it at most $\log n$ times. Every $x \in V$ stores a list $L(x)$ of pairs, where a pair $(i, v) \in L(x)$ indicates that at scale $i \in K$ the node containing x was merged with a larger node centered at v. Initially, $(0, x) \in L(x)$. The lists $\text{Lists} = \{L(v) \mid v \in V\}$ that our algorithm maintains enable us to maintain a part $\mathcal{L}'$ of the laminar family $\mathcal{L}$ that was constructed so far. Observe that $\mathcal{L}'$ can be viewed as a forest of sets, and this forest is partial to the tree $\mathcal{L}$. The nodes of each G_k can be reproduced from the lists $\{L(x)\}_{x \in V}$. (Specifically, to compute the nodes of G_k, for each vertex $x \in V$ find the maximum index $i \leq k$, for which there is an entry $(i, v) \in L(x)$, and x will be a part of a G_k-node centered at v.)

Therefore, the algorithm does not really need to store the list of the edges that it has seen so far. Rather, the information stored in Lists, along with the new edge $e = (x, y)$, which the algorithm processes in the current stage, is sufficient for updating the set Lists, and the latter is sufficient for deducing the node sets of graphs $\{G_k\}_{k \in K}$. More concretely, given the set Lists_h, which was constructed from a sequence $(e_1, e_2, \ldots, e_h)$ of edges, for some positive integer h and a newly arriving edge $e = e_{h+1}$ of weight $\omega = \omega(e)$, the algorithm constructs the set Lists_{h+1}, which reflects the appended edge sequence $(e_1, e_2, \ldots, e_{h+1})$, in the following way. It processes the scales $k = 0, 1, 2, \ldots$ consecutively from the bottom up. (Recall that we are not limited in processing time but rather only in memory.) Initially, Lists_{h+1} is empty. In each scale k, the algorithm processes all scale-k merges recorded in Lists_h, and, if the newly arrived edge e causes a k-scale merge, then the algorithm processes this new edge as well. We also recompute the set S_{h+1} of hopset edges in each iteration. For every node Z, these edges connect the node center z with every other vertex $v \in Z$. That is, the previous set $S = S_h$ is discarded, and the new set $S = S_{h+1}$ is computed from scratch.

This completes the description of the first pass of our algorithm, i.e., of the pass that computes the set Lists, and as a result, the node sets of graphs G_k, for all relevant scales k. The space required for this computation is proportional to the maximum size of the data structure Lists, which is, by Claim 4.4, at most $O(n \cdot \log n)$.

The correctness of this procedure hinges on the observation that if two nodes X, Y merge on scale k, it is immaterial which of the edges from $(X \times Y) \cap E$ caused this merge. Moreover, the weight $\omega(e)$ of this edge is also immaterial. (By the very fact that the merge occurred, we know that $\omega(e) < \frac{\epsilon}{n} \cdot 2^k$.)

Let us now review the execution of the hopset algorithm in the consequent passes over the stream. We compute single-scale hopsets H_k in parallel for all $k \in K$. For each $k \in K$ we run the hopset construction given by Theorem 3.16. Initially, the vertices of each G_k can be derived from the data structure we store. Whenever an edge $(x, y) \in E$ of weight $\omega(x, y)$ is read from the stream, we know it is active in at most $\log(n/\epsilon) + 2$ different scales. For each such scale $k \in K$ with $(\epsilon/n) \cdot 2^k \leq \omega(x, y) < 2^{k+2}$, we use the data structure to find the centers of nodes containing x, y in G_k, and we execute the hopset algorithm as if an edge connecting these centers (of weight given by (17)) were just read from the stream.

TABLE 3

Summary of results for (β, ϵ)-hopsets in the streaming model; all are with expected size $O(n^{1+1/\kappa} \log n)$ and stretch $1 + \epsilon$. The space bounds are in expectation, the bounds on the number of passes hold whp, and the bounds on β hold deterministically. The last column indicates whether the hopset is path-reporting.

Space	# of passes	The hopbound β	Paths
$O(n^{1+\rho} \log n/\rho + n^{1/\kappa} \log^2 n)$	$O(\beta \log n)$	$O\left(\frac{\log n}{\epsilon}(\log(\rho\kappa) + \frac{1}{\rho})\right)^{\log(\rho\kappa)+\frac{1}{\rho}}$	No
$O(n^{1+\rho} \log n/\rho + n^{1+\frac{1}{\kappa}} \beta \log^2 n)$	$O(\beta \log n)$	$O\left(\frac{\log n}{\epsilon}(\log(\rho\kappa) + \frac{1}{\rho})\right)^{\log(\rho\kappa)+\frac{1}{\rho}}$	Yes
$O(n^{1+\frac{1}{\kappa}} \cdot \log^2 n)$	$O(n^\rho \cdot \beta \cdot \log^2 n)$	$O\left(\frac{\log n}{\epsilon}(\log(\rho\kappa) + \frac{1}{\rho})\right)^{\log(\rho\kappa)+\frac{1}{\rho}}$	No
$O(n^{1+\frac{1}{\kappa}} \cdot \log^2 n \cdot \beta)$	$O(n^\rho \cdot \beta \cdot \log^2 n)$	$O\left(\frac{\log n}{\epsilon}(\log(\rho\kappa) + \frac{1}{\rho})\right)^{\log(\rho\kappa)+\frac{1}{\rho}}$	Yes
$O(n^{1+\frac{1}{\kappa}} \cdot \log^2 n)$	$O(n^\rho \cdot \beta \cdot \log^2 n)$	$O\left(\frac{1}{\epsilon\cdot\rho}(\log(\rho\kappa) + \frac{1}{\rho})\right)^{\log(\rho\kappa)+\frac{1+\mu}{\rho}}$	No
$O(n^{1+\frac{1}{\kappa}} \cdot \log^2 n \cdot \beta)$	$O(n^{\frac{\mu}{\kappa}+\rho} \beta \log^2 n)$	$O\left(\frac{\kappa}{\epsilon}(\log(\rho\kappa) + \frac{1}{\rho})\right)^{\log(\rho\kappa)+\frac{1}{\rho}}$	Yes

We have two possible trade-offs between space and number of passes for given parameters κ, ϵ, ρ, and t. Since we run in parallel in all scales, the fact that there are many graphs does not affect the number of passes. The size of each H_k is only $O(n_k^{1+1/\kappa})$. Using the fact that each G_k has aspect ratio at most $\Lambda_k = O(n/\epsilon)$, we can essentially replace $\log \Lambda$ by $\log(n/\epsilon) = O(\log n)$ in Theorem 3.16. The total space used by the algorithm in the second regime of Theorem 3.16 is $\sum_{k\in K} n_k^{1+1/\kappa} \cdot \log(n/\epsilon) \leq O(n^{1+1/\kappa} \log^2 n)$ rather than $O(n^{1+1/\kappa} \log \Lambda)$. The calculation for the first regime is analogous. Formally, we derive the following theorem.

THEOREM 4.11. *For any graph $G = (V, E, \omega)$ with n vertices, any $2 \leq \kappa \leq (\log n)/4$, $1/2 \geq \rho \geq 1/\kappa$, $0 < \epsilon < 1/2$, and any $1 \leq t \leq O(\log n)$, our streaming algorithm computes a (β, ϵ)-hopset with expected size $O(n^{1+1/\kappa} \cdot \log n)$ and with β given by*

$$\beta = O\left(\frac{(\log \kappa + 1/\rho) \log n}{\epsilon \cdot t}\right)^{\log \kappa + 1/\rho} . \tag{22}$$

The resource usage is either

1. *expected space $O(t \cdot n^{1+\rho} \cdot \log n/\rho + n^{1+1/\kappa} \cdot \log^2 n)$ (resp., space $O(t \cdot n^{1+\rho} \cdot \log n/\rho + n^{1+1/\kappa} \cdot \beta \cdot 2^t \cdot \log^2 n)$ for path-reporting) and $O(\beta \cdot \log n \cdot 2^t)$ passes, whp; or*
2. *expected space $O(n^{1+1/\kappa} \cdot \log^2 n)$ (resp., $O(n^{1+1/\kappa} \cdot \beta \cdot 2^t \cdot \log^2 n)$ for path-reporting) and $O(n^\rho \cdot \beta \cdot \log^2 n \cdot 2^t)$ passes, whp.*

A few possible trade-offs are summarized in Table 3. The first four results in the table are obtained by choosing $t = 1$ in Theorem 4.11. The first two follow from the first item, and the following two from the second item. The fifth and sixth results also follow from the second item of Theorem 4.11 but have an improved β (i.e., a constant independent of n). In the fifth one we set $t = \mu \cdot \rho \cdot \log n$ for an arbitrarily small constant $\mu > 0$. Then we rescale $\rho' = (1 + \mu) \cdot \rho$. This induces the term of $1 + \mu$ in the exponent of β. To get the path-reporting version of this bound, in the sixth result we set a smaller $t = \mu \cdot (\log n)/k$ and rescale $\kappa' = \frac{\kappa}{1+\mu}$. The O-notation of the β-column in the fifth and sixth rows of Table 3 hides a (constant) factor of $1/\mu$ in the base of the exponent.

The following corollary summarizes the fifth and sixth rows of Table 3, which provide efficient (requiring roughly $\tilde{O}(n^\rho)$ time) streaming algorithms for constructing (β,ϵ)-hopsets with $\tilde{O}(n^{1+1/\kappa})$ edges, and with $\beta = \beta(\epsilon,\kappa,\rho)$ independent of n. All four parameters ϵ, $1/\kappa$, ρ, and μ can simultaneously be made arbitrarily close to 0 constants while still having constant β.

COROLLARY 4.12. *For any graph $G = (V,E,\omega)$ with n vertices, any $2 \le \kappa \le (\log n)/4$, $1/2 \ge \rho \ge 1/\kappa$, $0 < \epsilon < 1/2$, and any arbitrarily small constant $\mu > 0$, our streaming algorithm computes a (β,ϵ)-hopset (resp., path-reporting hopset) with expected size $O(n^{1+1/\kappa}\cdot \log n)$, and with $\beta = O(\frac{1}{\epsilon\cdot\rho}(\log(\rho\kappa)+\frac{1}{\rho}))^{\log(\rho\kappa)+\frac{1+\mu}{\rho}}$ (resp., $\beta = O(\frac{\kappa}{\epsilon}(\log(\rho\kappa)+\frac{1}{\rho}))^{\log(\rho\kappa)+\frac{1}{\rho}}$) and expected space $O(n^{1+\frac{1}{\kappa}}\cdot\log^2 n)$ (resp., $O(n^{1+\frac{1}{\kappa}}\cdot\log^2 n\cdot\beta)$) in $O(n^\rho\cdot\beta\cdot\log^2 n)$ passes (resp., $O(n^{\rho+\frac{\mu}{\kappa}}\cdot\beta\cdot\log^2 n)$), whp.*

4.5. PRAM model. Klein and Subramanian [KS93] (see also [Coh97]) showed that the graphs $G_1, G_2, \ldots, G_\lambda$, $\lambda = \lceil \log\Lambda\rceil$, can be computed in the EREW PRAM model in $O(\log^2 n)$ time, using $O(|E|)$ processors. We next compute hopsets H_k, for all $k \in K$, in parallel. The overall expected size of the resulting hopset is, by Theorem 3.18, $\mathbb{E}(|H|) = \sum_{k\in K} n_k^{1+1/\kappa} = O(n^{1+1/\kappa}\cdot\log n)$. The aspect ratio of each graph G_k is $O(n/\epsilon)$, and thus the number of processors used is

$$O\left(\sum_{k\in K}(|E(G_k)|+n_k^{1+1/\kappa}\cdot\log^2 n)\cdot n^\rho\cdot\log n\cdot t\right)=O(n^\rho\cdot\log n\cdot t\cdot(|E|\cdot\log n+n^{1+1/\kappa}\cdot\log^3 n))\,.$$

We summarize with the following.

THEOREM 4.13. *For any n-vertex graph $G = (V,E,\omega)$ of diameter Λ, any $2 \le \kappa \le (\log n)/4$, $1/2 \ge \rho \ge 1/\kappa$, $0 < \epsilon < 1/2$, and any $1 \le t \le \log\Lambda$, our parallel algorithm computes a (β,ϵ)-hopset with $O(n^{1+1/\kappa}\cdot\log n)$ edges in expectation, and with β given by*

$$\beta \;=\; O\left(\frac{(\log\kappa+1/\rho)\cdot\log n}{\epsilon\cdot t}\right)^{\log\kappa+1/\rho}$$

in $O(\beta\cdot(\log\kappa+1/\rho)\cdot\log^2 n\cdot\frac{2^t}{t})$ EREW PRAM time, using $O((|E|+n^{1+1/\kappa}\cdot\log^2 n)\cdot n^\rho\cdot\log^2 n\cdot t)$ processors, whp.

In particular, by setting $t = 1$ we get

$$\beta \;=\; O\left(\frac{(\log\kappa+1/\rho)\cdot\log n}{\epsilon}\right)^{\log\kappa+1/\rho} \tag{23}$$

in $O(\beta\cdot(\log\kappa+1/\rho)\cdot\log^2 n)$ EREW PRAM time, using $O((|E|+n^{1+1/\kappa}\cdot\log^2 n)\cdot n^\rho\cdot\log^2 n)$ processors. One can also set $t = \log n^\zeta$, for a parameter $\zeta > 0$, and get $O(\beta\cdot(\log\kappa+1/\rho)\cdot\log n\cdot n^\zeta/\zeta)$ time, with

$$\beta = O\left(\frac{\log\kappa+1/\rho}{\epsilon\cdot\zeta}\right)^{\log\kappa+1/\rho}\,.$$

Note that β becomes independent of n, i.e., it is constant whenever ρ, ϵ, ζ, and $1/\kappa$ are. So one can compute a (β,ϵ)-hopset, with arbitrarily small constant $\epsilon > 0$, in EREW PRAM time $O(n^\zeta)$, for an arbitrarily small constant $\zeta > 0$, using $O((|E| + n^{1+1/\kappa}\cdot\log^2 n)\cdot n^\rho\cdot\log^3 n)$ processors, for arbitrarily small constants $\rho, 1/\kappa > 0$, and still have a constant hopbound β.

5. Applications.

In this section we describe applications of our improved constructions of hopsets to compute approximate shortest paths for a set $S \times V$ of vertex pairs for a subset $S \subseteq V$ of designated sources. Let $G = (V, E, \omega)$ be a weighted graph with n vertices, $0 < \epsilon < 1/2$, and let $S \subseteq V$ be a set of $s = |S|$ sources.

5.1. Congested Clique model.

In order to compute shortest paths from every vertex in S to every vertex in V, we first apply Corollary 4.8 to the graph with parameters $\kappa = \log_s n$ and $\rho = 1/\kappa$ (we assume $s \geq 16$ for the bound on κ to hold). We use $\mu = 0.01$. This yields a (β, ϵ)-hopset H with $\beta = O((\log_s n)/\epsilon)^{2.02 \cdot \log_s n}$. Now each of the S sources (in parallel) conducts β iterations of Bellman–Ford exploration in $G \cup H$, and as a result obtains $1 + \epsilon$ approximate distance estimations to all other vertices.

The number of rounds to compute the hopset is

$$O(n^\rho \cdot \log^4 n \cdot \beta) = s \cdot O((\log_s n)/\epsilon)^{2.02 \cdot \log_s n} \cdot \log^4 n$$

whp, and the number of rounds to conduct s Bellman–Ford explorations to range β is at most $O(s \cdot \beta)$ (see, e.g., section 3.2). We conclude that the total number of rounds is $s \cdot O((\log_s n)/\epsilon)^{2.02 \cdot \log_s n} \cdot \log^4 n$. In the case when $s = n^{\Omega(1)}$, we can in fact set $\kappa = \log_{s/\lg^4 n} n$, which yields $\beta = (1/\epsilon)^{O(1)}$, and the number of rounds will be essentially linear in s, specifically, $s \cdot (1/\epsilon)^{O(1)}$. In the case when $s \leq 2^{\sqrt{\log n \log\log n}}$, it is more beneficial to choose $\kappa = \sqrt{\frac{\log n}{\log\log n}}$. This yields

$$\beta = (1/\epsilon)^{O(\sqrt{\frac{\log n}{\log\log n}})} \cdot 2^{O(\sqrt{\log n \cdot \log\log n})} \tag{24}$$

in $\tilde{O}(n^\rho \cdot \beta) = O(1/\epsilon)^{\sqrt{\frac{\log n}{\log\log n}}} \cdot 2^{O(\sqrt{\log n \log\log n})}$ rounds.

If we are interested in the actual paths, rather than just distances, then we employ our path-reporting variant of hopsets. This increases the number of rounds by an additional factor of β.

THEOREM 5.1. *For any graph $G = (V, E, \omega)$ with n vertices, a parameter $0 < \epsilon < 1/2$, and $S \subseteq V$ of size s, there is an algorithm in the Congested Clique model that whp computes $(1+\epsilon)$-approximate $S \times V$ shortest paths in $s \cdot O((\log_s n)/\epsilon)^{2.02 \cdot \log_s n} \cdot \log^4 n$ rounds. In the case $s = n^{\Omega(1)}$ we can achieve $s \cdot (1/\epsilon)^{O(1)}$ rounds, and in the case $s \leq 2^{\sqrt{\log n \log\log n}}$ the number of rounds can be set as $s \cdot (1/\epsilon)^{O(\sqrt{\frac{\log n}{\log\log n}})} \cdot 2^{O(\sqrt{\log n \log\log n})}$.*

We also have the following result for approximate single-source shortest paths from some $u \in V$, assuming messages of size n^ρ are allowed, for some $\rho > 0$. By applying Theorem 4.7 with $\kappa = 1/\rho$ and $t = 1$, we get a hopset H with $\beta = O(\frac{\log n}{\epsilon \cdot \rho^2})^{2/\rho}$ in $O(\log^3 n \cdot \beta/\rho)$ rounds. Conducting β iterations of Bellman–Ford exploration from u in $G \cup H$ yields $(1+\epsilon)$-approximate distance estimations to all other vertices.

COROLLARY 5.2. *For any graph $G = (V, E, \omega)$ with n vertices, parameters $0 < \rho, \epsilon < 1/2$, and a vertex $u \in V$, there is an algorithm in the Congested Clique model with message size n^ρ that whp computes $(1+\epsilon)$-approximate shortest paths from u in $(\frac{\log n}{\epsilon})^{O(1/\rho)}$ rounds.*

The results of Theorem 5.1 and Corollary 5.2 apply also to the Broadcast Congested Clique model, except that the respective running times become larger by a factor of $\log_n \Lambda$.

5.2. CONGEST model. Computing approximate shortest paths in the CONGEST model using hopsets is somewhat more involved. We follow the method of [HKN16] and give full details for completeness. First, we use a lemma of [Nan14], which efficiently computes hop-limited distances from a given set of sources.

LEMMA 5.3 ([Nan14]). *Given a weighted graph $G = (V, E, \omega)$ of hop-diameter D and diameter Λ, a set $\tilde{V} \subseteq V$, and parameters $t \geq 1$ and $0 < \epsilon < 1/2$, there is a distributed algorithm in the Broadcast CONGEST model that whp runs in $\tilde{O}((|\tilde{V}| + t + D) \cdot \log \Lambda)/\epsilon$ rounds, so that every $u \in V$ will know values $\{\tilde{d}(u,v)\}_{v \in \tilde{V}}$ satisfying*[17]

$$d_G(u,v) \leq \tilde{d}(u,v) \leq (1+\epsilon) d_G^{(t)}(u,v) \ . \tag{25}$$

Remark 5.4. While not explicitly stated in [Nan14], the proof also yields that each $v \in V$ knows, for every $u \in \tilde{V}$, a parent $p = p_u(v)$ which is a neighbor of v satisfying

$$\tilde{d}(v,u) \leq \omega(v,p) + \tilde{d}(p,u) \ . \tag{26}$$

Let $\tilde{V} \subseteq V$ be a random set of vertices, such that each $v \in V$ is included in $\tilde{V}$ independently with probability $1/\sqrt{sn}$. Note that whp $|\tilde{V}| \leq \sqrt{n/s} \cdot \ln n$, so that $s \cdot |\tilde{V}| = \tilde{O}(\sqrt{ns})$. The following claim argues that the random sample $\tilde{V}$ hits every shortest path somewhere in its first $\tilde{O}(\sqrt{sn})$ vertices.

CLAIM 5.5. *The following holds whp: For every $x, y \in V$, there exists $u \in \tilde{V} \cup \{y\}$ on the shortest path from x to y in G, such that $d_G^{(4\sqrt{sn} \cdot \ln n)}(x,u) = d_G(x,u)$.*

Proof. Fix some $x, y \in V$. If it is the case that the shortest path $\pi(x,y)$ between them in G is comprised of at most $4\sqrt{sn} \cdot \ln n$ vertices, then we can take $u = y$. Otherwise, the probability that none of the first $4\sqrt{sn} \cdot \ln n$ vertices on $\pi(x,y)$ is sampled to $\tilde{V}$ is bounded by $(1 - 1/\sqrt{sn})^{4\sqrt{sn} \cdot \ln n} \leq n^{-4}$. Taking a union-bound on the $O(n^2)$ pairs concludes the proof. ∎

Let $\tilde{G} = (\tilde{V}, \tilde{E})$ be the graph on the vertex set $\tilde{V}$ of size $m = |\tilde{V}|$, with edge weights $\tilde{d}(u,v)$ given by applying Lemma 5.3 on G with parameters $t = 4\sqrt{sn} \cdot \ln n$ and ϵ. This will take $\tilde{O}((D + \sqrt{sn}) \cdot \log \Lambda)/\epsilon$ rounds. Next, construct a (β, ϵ)-hopset H for $\tilde{G}$ (embedded in G) as in Corollary 4.10, with $\kappa = \sqrt{\log m / \log \log m}$, $\rho = 1/\kappa$ (and, say $\mu = 0.01$). This results in

$$\beta = (1/\epsilon)^{O(\sqrt{\frac{\log m}{\log \log m}})} \cdot 2^{O(\sqrt{\log m \cdot \log \log m})} \ , \tag{27}$$

and the number of rounds required is $(D+m) \cdot (1/\epsilon)^{O(\sqrt{\log m / \log \log m})} \cdot 2^{O(\sqrt{\log m \cdot \log \log m})}$. Now, for each $s \in S$, each $u \in \tilde{V}$ holds an initial estimate $\tilde{d}(u,s)$ given by Lemma 5.3. We conduct β iterations of Bellman–Ford explorations in $\tilde{G} \cup H$ for each of the vertices of S. That is, in every iteration, every $u \in \tilde{V}$ sends s messages containing its current distance estimate for each $s \in S$ and updates its estimates according to the messages of other vertices. This requires additional $O(D + ms) \cdot \beta$ rounds. As a result, for every pair $s \in S$ and $u \in \tilde{V}$, the vertex u holds an estimate $\hat{d}(s,u)$. We broadcast all of these values to the entire graph in $O(D + sm)$ rounds.

Finally, for each $v \in V$ and $s \in S$, the vertex v computes the value $\hat{d}(v,s) = \min_{u \in \tilde{V}} \{\tilde{d}(v,u) + \hat{d}(u,s)\}$ as its approximate distance to s. The total number of

[17]The computed values are symmetric, that is, $\tilde{d}(u,v) = \tilde{d}(v,u)$ whenever $u, v \in \tilde{V}$.

rounds required is $(D+\sqrt{ns})\cdot[(1/\epsilon)^{O(\sqrt{\frac{\log m}{\log\log m}})}\cdot 2^{O(\sqrt{\log m\cdot\log\log m})}+O(\log\Lambda)/\epsilon]$. As above, whenever $s=n^{\Omega(1)}$ it is better to set $\kappa=\log_{s^{1-\mu}/\log^4 n} n$ and $\rho=1/\kappa$. Then $\beta=(1/\epsilon)^{O(1)}$, and the number of rounds will be $(D+\sqrt{ns})\cdot(1/\epsilon)^{O(1)}\cdot\log\Lambda$.

If we are interested in the actual paths, then we use Remark 5.4 to trace down the parents, and the actual approximate path from any $v\in V$ to any $u\in\tilde{V}$ can be derived. Also, we use the path-reporting version of our hopset. This introduces an additional factor of β to the number of rounds and enables every vertex in the graph to find out the actual paths that implement every hopset edge (for every hopset edge, we broadcast also the path of length at most β that implements it). In particular, v will be able to infer the paths for both $\tilde{d}(v,u)$ and $\hat{d}(u,s)$.

It remains to prove the correctness of the algorithm. First, consider any $y\in\tilde{V}$ and $s\in S$. Let $u\in\tilde{V}$ be the vertex on $\pi(s,y)$ guaranteed by Claim 5.5 (it could be that $u=y$). By Lemma 5.3,

$$\tilde{d}(s,u)\le(1+\epsilon)d_G^{(t)}(s,u)=(1+\epsilon)d_G(s,u)\ . \tag{28}$$

We also have that

$$d_{\tilde{G}}(y,u)\le(1+\epsilon)d_G(y,u)\ , \tag{29}$$

where (29) holds because every edge along the shortest path from y to u was stretched in $\tilde{G}$ by at most $1+\epsilon$. Finally, the property of hopsets suggests that

$$d^{(\beta)}_{\tilde{G}\cup H}(y,u)\le(1+\epsilon)d_{\tilde{G}}(y,u)\overset{(29)}{\le}(1+\epsilon)^2 d_G(y,u)\ . \tag{30}$$

Note that in the Bellman–Ford iterations, the vertex y could have heard the estimate from u using a path of length β in $\tilde{G}\cup H$. Combining (28) and (30) yields that

$$\hat{d}(y,s)\le d^{(\beta)}_{\tilde{G}\cup H}(y,u)+\tilde{d}(s,u)\le(1+3\epsilon)d_G(y,u)+(1+\epsilon)d_G(s,u)\le(1+3\epsilon)d_G(y,s)\ . \tag{31}$$

Consider now some arbitrary $v\in V$ and $s\in S$. By Claim 5.5, there exists $u\in\tilde{V}\cup\{s\}$ on the shortest path from v to s in G with $d_G^{(t)}(v,u)=d_G(v,u)$. By Lemma 5.3,

$$\hat{d}(v,s)\le\tilde{d}(v,u)+\hat{d}(u,s)\overset{(31)}{\le}(1+\epsilon)d_G(v,u)+(1+3\epsilon)d_G(u,s)\le(1+3\epsilon)d_G(v,s)\ .$$

We summarize with the following theorem.

THEOREM 5.6. *For any graph $G=(V,E,\omega)$ with n vertices and hop-diameter D, a parameter $0<\epsilon<1/2$, and $S\subseteq V$ of size s, there is an algorithm in the Broadcast CONGEST model that whp computes $(1+\epsilon)$-approximate $S\times V$ shortest paths in*

$$(D+\sqrt{ns})\cdot\left[(1/\epsilon)^{O(\sqrt{\frac{\log n}{\log\log n}})}\cdot 2^{O(\sqrt{\log n\cdot\log\log n})}+O(\log\Lambda)/\epsilon\right]$$

rounds. Whenever $s=n^{\Omega(1)}$, we have only $\tilde{O}(D+\sqrt{ns})\cdot(1/\epsilon)^{O(1)}\cdot\log\Lambda$ rounds.

We also have the following result for approximate single-source shortest paths from some $u\in V$, assuming messages of size n^ρ are allowed, for some $\rho>0$. By computing $\tilde{G}$ as above with $\tilde{V}$ being a random sample of $m\approx\sqrt{n}$ vertices, and applying Theorem 4.9 with $\kappa=1/\rho$ and $t=1$, we get a hopset H for $\tilde{G}$ with $\beta=O(\frac{\log n}{\epsilon\cdot\rho^2})^{2/\rho}$ in $O((D+m\cdot\log^2 m)\cdot\beta/\rho\cdot\log m)$ rounds. Conducting the same computation as above for $\tilde{G}$ with $S=\{u\}$ and extending to G, we obtain $1+\epsilon$ approximate distance estimations from u to all other vertices.

COROLLARY 5.7. *For any graph $G = (V, E, \omega)$ with n vertices, parameters $0 < \rho, \epsilon < 1/2$, and a vertex $u \in V$, there is an algorithm in the Broadcast CONGEST model with message size n^ρ that whp computes $(1+\epsilon)$-approximate shortest paths from u in $(D + \sqrt{n}) \cdot (\frac{\log n}{\epsilon})^{O(1/\rho)}$ rounds.*

5.3. Streaming model. Similarly to the Congested Clique model, we first compute a (β, ϵ)-hopset H for G, and then using additional $O(s \cdot \beta)$ passes over the stream, conduct β iterations of Bellman–Ford exploration in $G \cup H$ separately for each of the s sources. Note that each exploration requires linear in n space, plus the space required to store the hopset. The latter space will be sublinear in the output size, so we need to assume that after computing distances (or paths) from a certain source, we may output the result and erase it to free up memory.

In the case of the path-reporting hopset, for each edge e of the hopset, the hopset stores a path with at most β edges $e_1, \ldots, e_\beta$ of a subhopset of lower scale that implements it. The same is true for each of the edges $e_1, \ldots, e_\beta$, recursively. So the paths can be retrieved given our path-reporting hopset.

We use the hopset given by Corollary 4.12 with parameter $\kappa = (\log n)/4$. Thus $\beta = ((\log\log n + 1/\rho)/\epsilon)^{O(\log\log n + 1/\rho)}$, while for the path-reporting case, $\beta_{\text{path}} = ((\log n)/\epsilon)^{O(\log\log n + 1/\rho)}$. The space requirement is $O(n \log^2 n)$ (for path-reporting it is larger by a factor of β_{path}), and the number of passes required is $O(s \cdot \beta + n^\rho \cdot \beta \cdot \log^2 n)$.

We consider several possible regimes. Whenever $s > n^{1/\log\log n}$ we set $\rho = 1/\log\log n$, when $2^{\sqrt{\log n \log\log n}} < s \leq n^{1/\log\log n}$ we take $\rho = \frac{\log s}{\log n}$, and for $s \leq 2^{\sqrt{\log n \log\log n}}$ we choose $\rho = \sqrt{\log\log n / \log n}$. The resulting algorithms are described in the following theorem.

THEOREM 5.8. *For any graph $G = (V, E, \omega)$ with n vertices, a parameter $0 < \epsilon < 1/2$, and $S \subseteq V$ of size s, there is an algorithm in the streaming model that whp computes $(1 + \epsilon)$-approximate $S \times V$ shortest paths, with the following resources:*

- *Whenever $s > n^{1/\log\log n}$, the algorithm performs $s \cdot (\log n)^{O(\log^{(3)} n + \log 1/\epsilon)}$ passes over the stream and uses $O(n \log^2 n)$ space. For path-reporting the number of passes is $s \cdot (\log n)^{O(\log\log n + \log 1/\epsilon)}$, while the space increases to $n \cdot (\log n)^{O(\log\log n + \log 1/\epsilon)}$.*
- *Whenever $2^{\sqrt{\log n \log\log n}} < s \leq n^{1/\log\log n}$, the algorithm makes $s \cdot n^{O(\log\log n + \log 1/\epsilon)/\log s}$ passes, and the space requirement is $O(n \log^2 n)$ (or $n^{1+O(\log\log n + \log 1/\epsilon)/\log s}$ space for path-reporting).*
- *Whenever $s \leq 2^{\sqrt{\log n \log\log n}}$, the algorithm uses $2^{O(\sqrt{\log n \log\log n})}$ passes, and the space is $O(n \log^2 n)$ (or $n \cdot 2^{O(\sqrt{\log n \log\log n})}$ space for path-reporting).*

We also remark that whenever $s = n^{\Omega(1)}$, one can also choose a smaller $\kappa = O(1)$, increasing the space to $O(n^{1+1/\kappa} \cdot \log^2 n)$, while setting $\rho = 1/\kappa$ so that β is a constant. This will yield near-optimal $s \cdot (1/\epsilon)^{O(1)}$ passes over the stream (also for path-reporting).

5.4. PRAM model. We use Theorem 4.13 with $t = 1$ to construct a (β, ϵ)-hopset H of expected size $O(n^{1+1/\kappa} \cdot \log n)$, with β given by (23), in $O(\beta \cdot (\log\kappa + 1/\rho) \cdot \log^2 n)$ parallel time, using $O((|E| + n^{1+1/\kappa} \cdot \log^2 n) \cdot n^\rho \cdot \log^2 n)$ processors.

To compute $(1+\epsilon)$-approximate shortest distances (or paths) for $S \times V$, for some subset $S \subseteq V$ of vertices, we now conduct β-limited Bellman–Ford explorations, in parallel, separately from each of the $|S|$ origins, in $G \cup H$. We use $s = |S|$ processors $p_{v,1}, \ldots, p_{v,s}$ for every vertex $v \in V$, and s processors $p_{e,1}, \ldots, p_{e,s}$ for every edge $e \in G \cup H$. As argued in section 3.6, these explorations can now be completed in $O(\beta \cdot \log n)$ EREW PRAM time.

If we are interested in paths (rather than distance estimates), then we use a path-reporting hopset. As a result, for every pair $(s,v) \in S \times V$, we obtain a path $\pi(s,v)$ with at most β edges, some of which may belong to G, and others belong to the hopset H. For each edge $e \in \pi(s,v) \cap H$, we store a path with at most β other edges e' of $G \cup H$, and the same is true for each hopset edge e', recursively. The depth of the induced tree of edges is bounded by $\log O(n/\epsilon)$, that is, by the aspect ratio of each of the graphs G_k for which single-scale hopsets are constructed.

Hence, the entire path can be retrieved in $O(\log n)$ parallel time, using processors that were used for conducting the Bellman–Ford explorations from vertices of S. We cannot, however, retrieve all the paths simultaneously within these resource bounds. So our algorithm provides an implicit solution for the $(1+\epsilon)$-approximate shortest paths problem; i.e., it returns a data structure from which each of the $S \times V$ approximate shortest paths can be efficiently extracted. Though not stated explicitly, to the best of our knowledge, this is also the case with Cohen's parallel $(1+\epsilon)$-approximate shortest paths algorithm [Coh00].

THEOREM 5.9. *For any n-vertex graph $G = (V, E, \omega)$ of diameter Λ, a set $S \subseteq V$, and any $2 \leq \kappa \leq (\log n)/4$, $1/2 \geq \rho \geq 1/\kappa$, $0 < \epsilon \leq 1$, our parallel algorithm computes $(1+\epsilon)$-approximate shortest distances (and implicit $(1+\epsilon)$-approximate shortest paths; see above) in $O(\frac{(\log \kappa + 1/\rho)\cdot \log n}{\epsilon})^{\log \kappa + 1/\rho + 1} \cdot \log^2 n$ EREW PRAM time, using $O((|E| + n^{1+1/\kappa} \cdot \log^2 n) \cdot (n^\rho \cdot \log^2 n + |S|))$ processors, whp.*

Since $\rho \geq 1/\kappa$, the number of processors in Theorem 5.9 can be written as $O(|E| \cdot n^\rho \cdot \log^4 n + |E| \cdot |S| \cdot \log^2 n)$, while incurring a term $2/\rho$ instead of $1/\rho$ in the exponent of the running time. (This increase occurs as a result of rescaling $\rho' = 2\rho$.)

When κ, ρ, and ϵ are constant, this running time is polylogarithmic in n. In Cohen's result [Coh00, Theorem 1.1], the running time is polylogarithmic as well, but the exponent is roughly $O(\frac{1}{\rho} \cdot \log \kappa)$, rather than $\frac{1}{\rho} + \log \kappa + O(1)$ as in our Theorem 5.9.

Acknowledgments. We are grateful to the anonymous reviewers of FOCS'16 and SICOMP for extensive comments that improved the presentation of this paper.

REFERENCES

[ABCP93] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg, *Near-linear cost sequential and distributed constructions of sparse neighborhood covers*, in Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1993, pp. 638–647.

[ABP16] A. Abboud, G. Bodwin, and S. Pettie, *A hierarchy of lower bounds for sublinear additive spanners*, in Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, Barcelona, Spain, 2017, pp. 568–576, https://doi.org/10.1137/1.9781611974782.36.

[AP90] B. Awerbuch and D. Peleg, *Sparse Partitions (extended abstract),* in Proceedings of the 31st Annual Symposium on Foundations of Computer Science, Vol. II, St. Louis, MO, 1990, pp. 503–513.

[AS87] N. Alon and B. Schieber, *Optimal Preprocessing for Answering On-Line Product Queries*, Technical report, School of Mathematical Sciences, Tel Aviv University, 1987.

[Ber09] A. Bernstein, *Fully dynamic $(2+\epsilon)$ approximate all-pairs shortest paths with fast query and close to linear update time*, in Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2009), Atlanta, GA, 2009, pp. 693–702.

[BKKL16] R. Becker, A. Karrenbauer, S. Krinninger, and C. Lenzen, *Approximate Undirected Transshipment and Shortest Paths via Gradient Descent*, preprint, https://arxiv.org/abs/1607.05127v1, 2016.

[CG06] T-H. H. Chan and A. Gupta, *Small hop-diameter sparse spanners for doubling metrics*, Discrete Comput. Geom., 41 (2009), pp. 28–44.

[CKK+15] K. Censor-Hillel, P. Kaski, J. H. Korhonen, C. Lenzen, A. Paz, and J. Suomela, *Algebraic methods in the congested clique*, in Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing (PODC 2015), Donostia-San Sebastián, Spain, 2015, pp. 143–152.

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed., The MIT Press, 2009.

[Coh93] E. Cohen, *Fast algorithms for constructing t-spanners and paths with stretch t*, in Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1993, pp. 648–658.

[Coh97] E. Cohen, *Using selective path-doubling for parallel shortest-path computations*, J. Algorithms, 22 (1997), pp. 30–56.

[Coh00] E. Cohen, *Polylog-time and near-linear work approximation scheme for undirected shortest paths*, J. ACM, 47 (2000), pp. 132–166.

[Elk01] M. Elkin, *Computing almost shortest paths*, ACM Trans. Algorithms, 1 (2005), pp. 283–323.

[EN16a] M. Elkin and O. Neiman, *Hopsets with constant hopbound, and applications to approximate shortest paths*, in Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2016), New Brunswick, NJ, 2016, pp. 128–137.

[EN16b] M. Elkin and O. Neiman, *On efficient distributed construction of near optimal routing schemes: Extended abstract*, in Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing (PODC '16), New York, NY, 2016, pp. 235–244.

[EN17] M. Elkin and O. Neiman, *Linear-Size Hopsets with Small Hopbound, and Distributed Routing with Low Memory*, preprint, https://arxiv.org/abs/1704.08468, 2017.

[EP01] M. Elkin and D. Peleg, *(1+epsilon, beta)-spanner constructions for general graphs*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, Hersonissos, Greece, 2001, pp. 173–182.

[EP04] M. Elkin and D. Peleg, *$(1+\epsilon,\beta)$-spanner constructions for general graphs*, SIAM J. Comput., 33 (2004), pp. 608–631, https://doi.org/10.1137/S0097539701393384.

[EZ06] M. Elkin and J. Zhang, *Efficient algorithms for constructing $(1+\epsilon,\beta)$-spanners in the distributed and streaming models*, Distrib. Comput., 18 (2006), pp. 375–385.

[HKN14] M. Henzinger, S. Krinninger, and D. Nanongkai, *Decremental single-source shortest paths on undirected graphs in near-linear total update time*, in Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2014), Philadelphia, PA, 2014, pp. 146–155.

[HKN16] M. Henzinger, S. Krinninger, and D. Nanongkai, *A deterministic almost-tight distributed algorithm for approximating single-source shortest paths*, in Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing (STOC '16), Cambridge, MA, 2016, ACM, pp. 489–498.

[HP19] S.-En Huang and S. Pettie, *Thorup-Zwick emulators are universally optimal hopsets*, Inform. Process. Lett., 142 (2019), pp. 9–13.

[JaJ92] J. JaJa, *An Introduction to Parallel Algorithms*, Addison-Wesley Publishing Company, 1992.

[KS93] P. N. Klein and S. Subramanian, *A linear-processor polylog-time algorithm for shortest paths in planar graphs*, in Proceedings of the 34th Annual Symposium on Foundations of Computer Science, Palo Alto, CA, 1993, pp. 259–270.

[KS97] P. N. Klein and S. Subramanian, *A randomized parallel algorithm for single-source shortest paths*, J. Algorithms, 25 (1997), pp. 205–220.

[LP15] C. Lenzen and B. Patt-Shamir, *Fast partial distance estimation and applications*, in Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing (PODC 2015), Donostia-San Sebastián, Spain, 2015, pp. 153–162.

[MPVX15] G. L. Miller, R. Peng, A. Vladu, and S. C. Xu, *Improved parallel algorithms for spanners and hopsets*, in Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '15), Portland, OR, 2015, pp. 192–201.

[Nan14] D. Nanongkai, *Distributed approximation algorithms for weighted shortest paths*, in Proceedings of the Symposium on Theory of Computing, STOC 2014, New York, NY, 2014, pp. 565–573.

[Pel00] D. Peleg, *Distributed Computing: A Locality-Sensitive Approach*, Discrete Math. Appl. 5, SIAM, Philadelphia, 2000, https://doi.org/10.1137/1.9780898719772.

[SS99] H. Shi and T. H. Spencer, *Time–work tradeoffs of the single-source shortest paths problem*, J. Algorithms, 30 (1999), pp. 19–32.

[TZ01] M. Thorup and U. Zwick, *Approximate distance oracles*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, Hersonissos, Greece, 2001, pp. 183–192.

[TZ06] M. Thorup and U. Zwick, *Spanners and emulators with sublinear distance errors*, in Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, Miami, FL, 2006, pp. 802–809.

[UY91] J. D. Ullman and M. Yannakakis, *High-probability parallel transitive-closure algorithms*, SIAM J. Comput., 20 (1991), pp. 100–125, https://doi.org/10.1137/0220006.

[Yao82] A. C.-C. Yao, *Space-time tradeoff for answering range queries (extended abstract),* in Proceedings of the 14th Annual ACM Symposium on Theory of Computing, San Francisco, CA, 1982, pp. 128–136.