Linear-Size Hopsets with Small Hopbound, and Constant-Hopbound Hopsets in RNC

Michael Elkin*[1] and Ofer Neiman†[1]

[1]Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel.
Email: {elkinm,neimano}@cs.bgu.ac.il

Abstract

For a positive parameter β, the *β-bounded distance* between a pair of vertices u, v in a weighted undirected graph $G = (V, E, \omega)$ is the length of the shortest $u - v$ path in G with at most β edges, aka *hops*. For β as above and $\epsilon > 0$, a (β, ϵ)-*hopset* of $G = (V, E, \omega)$ is a graph $G_H = (V, H, \omega_H)$ on the same vertex set, such that all distances in G are $(1 + \epsilon)$-approximated by β-bounded distances in $G \cup G_H$.

Hopsets are a fundamental graph-theoretic and graph-algorithmic construct, and they are widely used for distance-related problems in a variety of computational settings. Currently existing constructions of hopsets produce hopsets either with $\Omega(n \log n)$ edges, or with a hopbound $n^{\Omega(1)}$. In this paper we devise a construction of *linear-size* hopsets with hopbound (ignoring the dependence on ϵ) $(\log \log n)^{\log \log n + O(1)}$. This improves the previous hopbound for linear-size hopsets almost *exponentially*.

We also devise efficient implementations of our construction in PRAM and distributed settings. The only existing PRAM algorithm [EN16a] for computing hopsets with a constant (i.e., independent of n) hopbound requires $n^{\Omega(1)}$ time. We devise a PRAM algorithm with polylogarithmic running time for computing hopsets with a constant hopbound, i.e., our running time is *exponentially* better than the previous one. Moreover, these hopsets are also significantly sparser than their counterparts from [EN16a]. We apply these hopsets to achieve the following online variant of shortest paths in the PRAM model: preprocess a given weighted graph within polylogarithmic time, and then given any query vertex v, report all approximate shortest paths from v in *constant time*. All previous constructions of hopsets require either polylogarithmic time per query or polynomial preprocessing time.

1 Introduction

1.1 Hopsets

Consider a weighted undirected graph $G = (V, E, \omega)$. Consider another graph $G_H = (V, H, \omega_H)$ on the same vertex set, that satisfies that for every $(u, v) \in H$, $\omega_H(u, v) \geq d_G(u, v)$, where $d_G(u, v)$ stands for the *distance* between u and v in G. For a positive integer parameter β, and a positive parameter $\epsilon > 0$, the graph G_H is called a (β, ϵ)-*hopset* of G, if for every pair of vertices $u, v \in V$, the *β-bounded distance* $d^{(\beta)}_{G \cup G_H}(u, v)$ between u and v in $G \cup G_H$ is within a factor $1+\epsilon$ from the distance $d_G(u, v)$ between these vertices in G, i.e., $d_G(u, v) \leq d^{(\beta)}_{G \cup G_H}(u, w) \leq (1+\epsilon) d_G(u, v)$. The *$\beta$-bounded distance* between u and v in G is the length of the shortest *β-bounded u-v-path* $\Pi_{u,v}$ in G, i.e., of a shortest path $\Pi_{u,v}$ with at most β hops/edges.

Hopsets constitute an important algorithmic and combinatorial object, and they are extremely useful for exact and approximate distance computations in a large variety of computational settings, including the distributed model [Nan14, HKN16, EN16a, EN16b, Elk17], parallel (PRAM) model [KS97, SS99, Coh00, MPVX15, EN16a, FL16], streaming model [Nan14, HKN16, EN16a, Elk17], dynamic setting [Ber09, HKN14], for routing [LP15, EN16a,

*This research was supported by the ISF grant No. (724/15).
†Supported in part by ISF grant No. (1817/17) and by BSF grant No. 2015813.

EN18], and also for approximation algorithms in PRAM [FL16]. The notion of hopsets was coined in a seminal paper of Cohen [Coh00]. (Though some first implicit constructions appeared a little bit earlier [UY91, KS97, SS99, Coh97].)

In [Coh00], Cohen also devised landmark constructions of hopsets. Specifically, she showed that for any parameters $\kappa = 1, 2, \ldots$, and $\epsilon > 0$, and any n-vertex graph G, there exists a (β, ϵ)-hopset with $O(n^{1+1/\kappa} \cdot \log n)$ edges, with $\beta = O\left(\frac{\log n}{\epsilon}\right)^{O(\log \kappa)}$. Moreover, she showed that hopsets with comparable attributes can be efficiently constructed in the centralized and parallel settings. Specifically, in the centralized setting her algorithm takes an additional parameter $\rho > 0$, and constructs a hopset of size $O(n^{1+1/\kappa} \cdot \log n)$ edges in $O(|E|n^{\rho})$ time, with hopbound $\beta = O\left(\frac{\log n}{\epsilon}\right)^{\frac{O(\log \kappa)}{\rho}}$. In the PRAM model, her hopset has size $O(n^{1+1/\kappa}) \cdot O(\log n)^{O(\frac{\log \kappa}{\rho})}$, and β is as above, and it is constructed in roughly $O(\beta)$ (i.e., polylogarithmic time), and with $O(|E| \cdot n^{\rho})$ work.

Cohen [Coh00] also raised the open question of existence and efficient constructability of hopsets with better attributes; she called it an "intriguing research problem". In the two decades that passed since Cohen's work [Coh00], numerous algorithms for constructing hopsets in various settings were devised [Ber09, HKN14, Nan14, MPVX15, EN16a]. The hopsets of [Ber09, HKN14, Nan14, HKN16] are no better than those of [Coh00] in terms of their attributes. (But they are constructed in settings to which Cohen's algorithm is not known to apply.) The algorithm of [MPVX15] builds hopsets of size $O(n)$, but with a large hopbound $\beta = n^{\Omega(1)}$. In [EN16a], the current authors showed that there always exist (β, ϵ)-hopsets of size $O(n^{1+1/\kappa} \cdot \log n)$, with *constant* (i.e., independent of n) hopbound $\beta = O(\frac{\log \kappa}{\epsilon})^{\log \kappa + O(1)}$. In the PRAM model, [EN16a] showed two results. First, that for parameters $\epsilon, \rho, \zeta > 0$ and $\kappa = 1, 2, \ldots$, hopsets with constant hopbound $\beta = O\left(\frac{\log \kappa + 1/\rho}{\epsilon\zeta}\right)^{\log \kappa + 2/\rho + O(1)}$ can be constructed in time $O(n^{\zeta} \cdot \beta)$, using $O(|E| \cdot n^{\rho+\zeta})$ work. Second, [EN16a] devised a PRAM algorithm with polylogarithmic time $O\left(\frac{\log n}{\epsilon}\right)^{\log \kappa + 1/\rho + O(1)}$, albeit with a polylogarithmic β roughly equal to the running time. The hopset's size is $O(n^{1+1/\kappa} \cdot \log n)$ in the second result as well.

These results of [EN16a] strictly outperformed the longstanding tradeoff of [Coh00] in all regimes, and proved the existence of hopsets with constant hopbound. However, they left a significant room for improvement. First, the hopsets of [EN16a] have $\Omega(n \log n)$ edges in all regimes. As a result, the only currently existing sparser hopsets [UY91, KS97, SS99, MPVX15] have hopbound of $n^{\Omega(1)}$. Hence it was left open if hopsets with size $o(n \log n)$ and hopbound $n^{o(1)}$ exist. Second, the question whether hopsets with constant hopbound can be constructed in *polylogarithmic* PRAM time was left open (recall that the previous best algorithm requires $n^{\Omega(1)}$ PRAM time).

In this paper we answer both these questions in the affirmative. Specifically, we show that for any $\kappa = 1, 2, \ldots$, there exists a $(\beta(\kappa, \epsilon), \epsilon)$-hopset, for all $\epsilon > 0$ *simultaneously*, with size $O(n^{1+1/\kappa})$ and $\beta = \beta(\kappa, \epsilon) = O\left(\frac{\log \kappa}{\epsilon}\right)^{\log \kappa + O(1)}$. In particular, by setting $\kappa = \log n$, we obtain a *linear-size* hopset, and its hopbound is $\beta = O\left(\frac{\log \log n}{\epsilon}\right)^{\log \log n + O(1)}$. This is an almost exponential improvement of the previously best known upper bound (due to [MPVX15]) on the hopbound of linear-size hopsets.

Second, in the PRAM setting, for any $\kappa = 1, 2, \ldots$, $\epsilon > 0$, and $\rho > 0$, our algorithm constructs hopsets of size $O(n^{1+1/\kappa} \cdot \log^* n)$, *constant hopbound* $\beta = O\left(\frac{(\log \kappa + 1/\rho)^2}{\epsilon}\right)^{\log \kappa + 1/\rho + O(1)}$, in *polylogarithmic* time $O((\log n)/\epsilon)^{\log \kappa + 1/\rho + O(1)}$, using work $O(|E| \cdot n^{\rho})$. This is an *exponential* improvement of the parallel running time of the previously best-known algorithm for constructing hopsets with constant hopbound due to [EN16a]. The result also implies that the problem of constructing sparse hopsets with constant hopbound is in RNC.

We can also shave the $\log^* n$ factor in the size, that allows for a linear-size hopset, but then β grows to be roughly the running time. See Table 1 for a concise comparison between existing and new results concerning hopsets in the PRAM model.

On the lower bound frontier, Abboud et al. [ABP17] showed that the dependency on ϵ in the hopbound β from [EN16a] is nearly tight. Our hopbound in the current construction is the same as in [EN16a]. Specifically, the lower bound of [ABP17] asserts that a (β, ϵ)-hopset with $O(n^{1+1/\kappa})$ edges must have $\beta = \Omega\left(\frac{1}{\epsilon \log \kappa}\right)^{\log \kappa - 1}$. Note that this lower bound does not preclude the existence of (β, ϵ)-hopsets with arbitrarily small constant $\epsilon > 0$, linear size and hopbound $\beta = \log^{O(\log 1/\epsilon)} n = \log^{O(1)} n$. Hopsets that we devise in the current paper have $\beta = (\log n)^{O(\log \log \log n)}$

in this regime (i.e., linear size and arbitrarily small constant $\epsilon > 0$).

An additional property of our hopset is that it comes with a *compact path-recovery mechanism*. Specifically, in applications in distributed computing, such as approximate shortest paths computation and routing, hopset edges e have to be replaced by actual paths P_e in the original graph G, that implement these edges. To do it, previous distributed hopsets' constructions [Nan14, HKN14, HKN16, EN16a] required every vertex v to store all hopset edges e such that P_e traverses v. This resulted in a prohibitively large blow-up in the memory used by graph vertices. The new path-recovery mechanism that we develop in this paper enables us to avoid this blow-up, and to keep the individual memory requirements of vertices in check. We exploit this mechanism in our low-memory approximate shortest paths algorithm, and in our new low-memory distributed routing scheme, developed in the companion paper [EN18].

Our algorithm also provides improved results for constructing hopsets in distributed CONGEST and Congested Clique models (see Section 3 for definition of these models). In all these models, our algorithm constructs linear-size hopsets. Also, the running time of our algorithms in all these models does not depend on the aspect ratio Λ of the graph. [1] In contrast, previous algorithms [Nan14, HKN16, EN16a] for constructing hopsets in the CONGEST model all have running time proportional to $\log \Lambda$.

Reference	Size	β = Hopbound	Time	Work
[KS97, SS99]	$O(n)$	$O(\sqrt{n})$	$O(\sqrt{n}\log n)$	$O(\lvert E\rvert \cdot \sqrt{n})$
[MPVX15]	$O(n)$	$O(n^{\frac{4+\alpha}{4+2\alpha}})$	$O(n^{\frac{4+\alpha}{4+2\alpha}})$	$O(\lvert E\rvert \cdot \log^{3+\alpha} n)$
	$O(n)$	$n^{\Omega(1)}$	$n^{\Omega(1)}$	$O(\lvert E\rvert \cdot \log^{O(1)} n)$
[Coh00]	$n^{1+1/\kappa} \cdot (\log n)^{O(\frac{\log\kappa}{\rho})}$	$(\log n)^{O(\frac{\log\kappa}{\rho})}$	$(\log n)^{O(\frac{\log\kappa}{\rho})}$	$O(\lvert E\rvert \cdot n^{\rho})$
[EN16a]	$O(n^{1+\frac{1}{\kappa}} \cdot \log n)$	$(\log n)^{\log\kappa+\frac{1}{\rho}+O(1)}$	$(\log n)^{\log\kappa+\frac{1}{\rho}+O(1)}$	$O(\lvert E\rvert \cdot n^{\rho})$
	$O(n^{1+\frac{1}{\kappa}} \cdot \log n)$	$\left(\frac{\log\kappa+\frac{1}{\rho}}{\zeta}\right)^{\log\kappa+O(\frac{1}{\rho})}$	$O(n^{\zeta} \cdot \beta)$	$O(\lvert E\rvert \cdot n^{\rho+\zeta})$
New	$O(n^{1+\frac{1}{\kappa}})$	$(\log n)^{\log\kappa+\frac{1}{\rho}+O(1)}$	$(\log n)^{\log\kappa+\frac{1}{\rho}+O(1)}$	$O(\lvert E\rvert \cdot n^{\rho})$
	$O(n^{1+\frac{1}{\kappa}} \cdot \log^* n)$	$\left(\log\kappa+\frac{1}{\rho}\right)^{O(\log\kappa+\frac{1}{\rho})}$	$(\log n)^{\log\kappa+\frac{1}{\rho}+O(1)}$	$O(\lvert E\rvert \cdot n^{\rho})$

Table 1: Comparison between (β, ϵ)-hopsets in the PRAM model (neglecting the dependency on ϵ). The hopsets of [KS97, SS99] provide exact distances. The one-before-last line provides the *first* construction of *linear-size* hopsets with hopbound $n^{o(1)}$. The last line provides the *first* RNC algorithm for constructing sparse hopsets with constant hopbound.

In this context we are thinking about the variant of the CONGEST model in which in every round, every vertex v is allowed to send $O(1)$ edge weights or identity numbers over each edge $e = (v, u)$ incident on v, *regardless* of the length of bit representations of these edge weights. We call this variant of the CONGEST model the *CONGEST RAM model*, because this assumption is analogous to the common assumption of the (centralized) RAM model, in which it is assumed that every edge weight fits in one single memory cell, and can be processed in $O(1)$ time. Even in the standard CONGEST model, in numerous settings, the assumption that edge weights can be sent in a single message is *not* equivalent to having message size $\Theta(\log \Lambda)$. Whenever a $1 + \epsilon$ multiplicative error in edge weights is allowed (as is the case in our setting), one can use only $O(\log\log\Lambda + \log(1/\epsilon))$ bits to specify an edge weight. In particular, our bounds stay the same even when edge weights are exponential in $\mathrm{poly}(n)$, rather than just polynomial in n. See Appendix 6 for further discussion on the CONGEST RAM model and bit complexity of messages.

1.2 Applications

We use our new constructions of hopsets to derive improved distributed approximate single-source shortest paths (SSSP) and s-source shortest paths (s-SSP) algorithms. These are extremely well-studied central problems in distributed computing [Elk01, EZ06, LPP16, Nan14, HKN16, EN16a, BKKL17]. All known $(1 + \epsilon)$-approximate al-

[1] The aspect ratio Λ of a graph G is given by $\Lambda = \frac{\max_{u,v\in V} d_G(u,v)}{\min_{u,v\in V, u\neq v} d_G(u,v)}$.

gorithms for the single-source problem in the CONGEST model have running time that depends at least linearly on $\log \Lambda$. Specifically, the state-of-the-art bound [BKKL17] is $O(\epsilon^{-O(1)}(D+\sqrt{n}) \cdot \log^{O(1)} n \cdot \log \Lambda)$. The best-known algorithm for this problem with running time independent of Λ is an exact algorithm of [Elk17], with running time $\tilde{O}(D^{2/3}n^{1/3} + n^{5/6})$.

Our new constructions of hopsets give rise to an algorithm for $(1+\epsilon)$-approximate SSSP with running time $O(D+\sqrt{n})(2+1/\epsilon)^{O(\sqrt{\log n \log\log n})}$, i.e., the same running time as in [HKN16], but without dependence on $\log \Lambda$. Moreover, our algorithm is the first one in which vertices only use a small *memory* throughout the computation. Specifically, the maximum memory per vertex required in our algorithm is $2^{O(\sqrt{\log n \log\log n})}$, while in all previous algorithms for this problem the memory requirement is at least $\Omega(n^{1/3})$. Furthermore, we can even achieve *polylogarithmic memory* requirement of $\log^{O(1/\mu)} n$, for a positive constant parameter $\mu > 0$, at the expense of increasing the time to $O((D+\sqrt{n}) \cdot n^{\mu})$. See Table 2 for a concise comparison of existing and new results about SSSP problem.

For the s-SSP problem, for sufficiently large s (i.e., $s = n^{\zeta}$, for an arbitrarily small constant $\zeta > 0$), the state-of-the-art bound is $\tilde{O}(D+\sqrt{ns}) \cdot \log \Lambda$, due to [EN16a]. Using our new construction of hopsets we eliminate the dependence on $\log \Lambda$ here too, and achieve running time of $\tilde{O}(D+\sqrt{ns})$ for this problem.[2]

Reference	Time	Memory
[HKN16]	$O(D+\sqrt{n}) \cdot 2^{O(\sqrt{\log n \log\log n})} \cdot \log \Lambda$	$\tilde{O}(\sqrt{n})$
[BKKL17]	$\tilde{O}(D+\sqrt{n}) \cdot \log \Lambda$	$\tilde{O}(\sqrt{n})$
[Elk17]	$\tilde{O}(D^{2/3}n^{1/3} + n^{5/6}) \cdot \log_n \Lambda$, exact	$\tilde{O}(n^{1/3})$
New	$O(D+\sqrt{n}) \cdot 2^{O(\sqrt{\log n \log\log n})} \cdot \log_n \log \Lambda$	$2^{O(\sqrt{\log n \log\log n})}$
	$O(D+\sqrt{n}) \cdot n^{\mu} \cdot \log_n \log \Lambda$	$\log^{O(1/\mu)} n$

Table 2: Summary of existing and new results for the approximate SSSP problem in the distributed CONGEST model. The memory requirement in [Elk17] is $\tilde{O}(n^{1/3})$, for $D = O(\sqrt{n \log n})$, and it improves to $\tilde{O}((n/D)^{1/3})$ for larger D. In the CONGEST RAM model there is no dependence on Λ in the last three lines.

1.2.1 PRAM Approximate Shortest Paths with Preprocessing

In the PRAM setting, our improved construction of hopsets gives rise to improved running time for the following variant of the s-SSP problem, which we call *PRAM Approximate Shortest Paths with Preprocessing*, or PRAM ASPP. We are allowed to first preprocess the graph, and then the sources S arrive in an online manner. Our objective is then to answer each query for a source $u \in S$, by returning all approximate shortest paths from u, in a very small parallel time.

We can use our hopsets to preprocess the input graph in polylogarithmic in n time and using $\tilde{O}(|E| \cdot n^{1/k})$ work, for a constant parameter $k \geq 1$. Then, given a source u, our algorithm returns $(1+\epsilon)$-approximate shortest paths from u to all other vertices in *constant* $(k/\epsilon)^{O(k)}$ time, using $\approx O(|E| \cdot n^{1/k})$ work. See Theorem 12 in Section 5.2. This should be compared with a solution based on our previous hopsets' construction [EN16a], where polylogarithmic time preprocessing did not really help. Even after constructing the hopset of [EN16a], one still needed a *polylogarithmic* (with a high exponent), as opposed to a constant, time to process each source. We note that the algorithm of Becker et al. [BKKL17] does not apply to the PRAM model, as pointed out in [BKKL17].

1.3 Technical Overview

Cohen's algorithm [Coh00] is based on a subroutine for constructing pairwise covers [Coh93, ABCP93], i.e., collections of small-radii clusters with small maximum overlap (no vertex belongs to too many clusters). The algorithm is a top-down recursive procedure: it interconnects large clusters of the cover via hopset edges, and recurses in small clusters. To keep the overall overlap of all recursion levels in check, Cohen used the radius parameter $O(\log n)$ for the

[2] To the best of our understanding, the algorithm of [BKKL17] scales linearly with s for the s-SSP problem, and so, for large s, the state-of-the-art bounds are based on hopsets.

covers. This resulted in a hopbound, which is at least polylogarithmic in n. Cohen's hopset is also built separately for each distance scale $[2^i, 2^{i+1})$, $i = 0, 1, \ldots, \log \Lambda$, and the ultimate hopset is the union of all these single-scale hopsets.

The hopsets construction of [EN16a] (due to the current authors) also builds a single-scale hopset for each distance scale, and then takes their union as an ultimate hopset. The construction of single-scale hopsets in [EN16a] is based upon ideas from the construction of $(1+\epsilon, \beta)$-spanners of [EP04] for unweighted graphs. It starts with a partition of the vertex set V into singleton clusters $\mathcal{P}_0 = \{v \mid v \in V\}$, and alternates superclustering and interconnection steps. In a superclustering step some of the clusters of the current partition are merged into larger clusters (of $\mathcal{P}_{i+1}$), while the other clusters are interconnected with one another via hopset edges.

This approach (of [EN16a]) enabled us to prove existence of hopsets with constant hopbound, but it appears to be incapable of producing hopsets of size $o(n \log n)$. Indeed, even if each single-scale hopset is of linear size (which is indeed the case in [EN16a]), their union is doomed to be of size $\Omega(n \log n)$. Moreover, in parallel and distributed settings, one produces a hopset $i+1$ based upon a hopset of scale i. This results in *accumulation of stretch* from $1+\epsilon$ to $(1+\epsilon)^{\log n}$. To alleviate this issue, one needs to rescale ϵ. However, then the hopbound grows from constant to polylogarithmic. To get around this, [EN16a] used a smaller number of scales, and this indeed enables [EN16a] to construct hopsets with constant hopbound in these settings, albeit the running time becomes proportional to the ratio between consequent scales, i.e., it becomes $n^{\Omega(1)}$.

The closest to our current construction of hopsets is the line of research of [Ber09, HKN14, HKN16, Nan14], which is based on a construction of distance oracles due to Thorup and Zwick [TZ01]. To construct their oracles, [TZ01] used a hierarchy of sets $V = A_0 \supseteq A_1 \supseteq \ldots A_{\kappa-1} \supseteq A_\kappa = \emptyset$, where each vertex of A_i, for all $i = 0, 1, \ldots, \kappa-2$, is sampled with probability $n^{-1/\kappa}$ for inclusion into A_{i+1}. For each vertex $u \in V$, [TZ01] defined for every $i = 0, 1, \ldots, \kappa-1$, the ith *pivot* $p_i(u)$ to be its closest vertex in A_i, and the ith *bunch* $B_i(u) = \{v \mid d_G(u,v) < d_G(u, A_{i+1})\} \cup \{p_{i+1}(u)\}$, (for $i = \kappa$, let $\{p_\kappa(u)\} = \emptyset$), and the entire bunch $B(u) = \bigcup_{i=0}^{\kappa-1} B_i(u)$. They also defined the dual sets, *clusters*, $C(v) = \{u \mid v \in B(u)\}$. Bernstein and others [Ber09, HKN14, Nan14, HKN16] used this construction with $\kappa = \tilde{\Theta}(\sqrt{\log n})$, and built Thorup-Zwick clusters with respect to $2^{\tilde{O}(\sqrt{\log n})}$-bounded distances. As a result, they obtained a so-called $2^{\tilde{O}(\sqrt{\log n})}$-*bounded hopset*, i.e., a hopset which takes care only of pairs $u, v \in V$ of vertices that admit a $2^{\tilde{O}(\sqrt{\log n})}$-bounded shortest path. They then used this bounded hopset in a certain recursive fashion (see the so-called *hop reduction* of Nanongkai [Nan14]) to obtain their ultimate hopset.

Thorup-Zwick's construction with $\kappa = \tilde{\Theta}(\sqrt{\log n})$ alone introduces into the hopset $n \cdot 2^{\tilde{\Theta}(\sqrt{\log n})}$ edges, and thus, such a hopset cannot be very sparse. In addition, the recursive application of the hop reduction technique results in a hopbound of $2^{\tilde{\Omega}(\sqrt{\log n})}$. Finally, these constructions [HKN14, Nan14, HKN16] and their analyses are rather complicated.

Our construction of hopsets is based upon a construction of Thorup-Zwick's sublinear-error emulators[3] [TZ06], as opposed to Thorup-Zwick distance oracles and spanners that served as a starting point of the algorithms [Ber09, HKN14, Nan14, HKN16]. Specifically, to obtain the hierarchy $V = A_0 \supseteq A_1 \supseteq \ldots A_{\log\kappa-1} \supseteq A_{\log\kappa} = \emptyset$, one samples each vertex of A_i, for $i = 0, 1, \ldots, \log\kappa - 2$, with probability roughly $n^{-2^i/\kappa}$ for inclusion in A_{i+1}. Then one defines the bunch of a vertex $u \in A_i$ as $B(u) = \{v \in A_i \mid d_G(v,u) < d_G(v, A_{i+1})\} \cup \{p_{i+1}(u)\}$, and sets

$$H = \bigcup_{u \in V} \{(u,v) \mid v \in B(u)\} . \tag{1}$$

For *unweighted* graphs G, [TZ06] showed that H given by (1) is a sublinear-error emulator with stretch $\alpha(d) = O(\log\kappa \cdot d^{1-1/(\log\kappa-1)})$ and $O(\log\kappa \cdot n^{1+1/\kappa})$ edges. By a different proof argument, we show that the *very same* construction for weighted graphs provides a (β, ϵ)-hopset of the same size and with $\beta = O\left(\frac{\log\kappa}{\epsilon}\right)^{\log\kappa-1}$, for all $\epsilon > 0$ *simultaneously*. Moreover, by adjusting the sampling probabilities, we also shave the $\log\kappa$ factor from both the hopset's and the emulator's size, while increasing the exponent of β by 1. (This also gives rise to the first linear-size emulator with sub-linear additive stretch for unweighted graphs.) Our analysis is different than the analysis of [TZ06]: we need to handle weighted graphs (as opposed to the sublinear-error emulator, which works only for unweighted graphs), and also we analyze the hopbound, as opposed to the stretch, which is analyzed in [TZ06].

[3] A graph $G' = (V, E', \omega')$ is called a *sublinear-error emulator* of an unweighted graph $G = (V, E)$, if for every pair of vertices $u, v \in V$, we have $d_G(u,v) \leq d_{G'}(u,v) \leq d_G(u,v) + \alpha(d_G(u,v))$ for some sub-linear stretch function α. If G' is a subgraph of G, it is called a sublinear-error *spanner* of G.

As a result, we obtain a construction of hopsets, which is (arguably) by far simpler than the previous Thorup-Zwick-based constructions of hopsets [Ber09, HKN14, Nan14, HKN16].[4] As was discussed above, it also provides hopsets with much better parameters, and it is more adaptable to efficient implementation in various computational settings. Our construction is also much simpler than the constructions of [Coh00, EN16a], which are not based on Thorup-Zwick's hierarchy.

Parallel and distributed implementations of our hopset's construction proceed in scales $\ell = 0, 1, \ldots, \log n$, where on scale ℓ the algorithm constructs a 2^ℓ-bounded hopset. This is different from the situation in [Coh00, EN16a, Nan14, HKN16], where scale ℓ takes care of distances in the range $[2^\ell, 2^{\ell+1})$. An important advantage of this is that we no longer need to take the union of all single-scale hopsets into our hopset; rather we just take the largest-scale hopset as our ultimate hopset. This saves a factor of $\log n$ in the size, and enables us to construct *linear-size* hopsets in parallel and distributed settings. The fact that we do not work with distance scales, but rather with hop-distance-scales, makes it possible to avoid the dependence on $\log \Lambda$ in the distributed construction time. All previous distributed algorithms for constructing approximate hopsets [Nan14, HKN16, EN16a] have running time proportional to $\log \Lambda$. (A distributed construction of an *exact* hopset [Elk17] by the first-named author, however, avoids this dependence too. Alas, it has a much higher running time.)

The fact that the construction's scales are with respect to hop-distances, as opposed to actual distances, enables us to essentially avoid accumulation of error. This is done by the following recursive procedure. First, we build 2^ℓ-bounded hopsets $H^{(\ell)}$, $\ell = 0, 1, \ldots, \log n$, and set $H(1) = H^{(\log n)}$ to be the highest-scale hopset. This process involves accumulation of stretch, and after rescaling the stretch parameter ϵ, the hopset $H(1)$ ends up having polylogarithmic hopbound β_1. Its construction time is roughly β_1, i.e., polylogarithmic as well. Now we add the hopset into the original graph, and recurse on $G \cup H(1)$. Note that now we only need to process $\log \beta_1 \approx \log \log n$ scales, rather than $\log n$ ones. Hence the accumulation of stretch in the resulting hopset $H(2)$ is much more mild than in $H(1)$. As a result, after rescaling the stretch parameter ϵ, the hopbound β_2 of $H(2)$ is roughly $poly(\log \log n)$. By repeating this recursive process for $\log^* n$ iterations, we eventually achieve a hopset with constant hopbound in parallel polylogarithmic time. As was mentioned above, this dramatically improves the $n^{\Omega(1)}$ parallel time required in [EN16a] to construct a hopset with constant hopbound.

In the context of distributed computation, like in [LP15, EN16b], our hopset H is constructed on top of a *virtual* (aka skeleton) graph $G' = (V', E')$, where V' is a collection of $\approx \sqrt{n}$ vertices, sampled from the original vertex set V independently with probability $\approx n^{-1/2}$. There is an edge $(u', v') \in E'$ iff there is a $\tilde{O}(\sqrt{n})$-bounded $u' - v'$ path in G. Surprisingly, we show that the hopset H for G' can be constructed without ever constructing G' itself! We only compute those edges of G', which are required for constructing the hopset H. (A similar phenomenon shows up in the construction of [Elk17] of *exact* hopsets.) This is one of the keys to drastically decreasing the memory requirement of our approximate SSSP algorithm, and of our low-memory routing schemes, devised in our companion paper [EN18]. Another crucial idea that we employ to guarantee a small individual memory requirement is ensuring that our hopset H has small *arboricity*, i.e., that its edges can be oriented in such a way that every vertex has only a small out-degree. This out-degree is proportional to the ultimate individual memory requirement of our algorithm.

Another key to decreasing the memory requirements is our *compact* path-recovery mechanism. Unlike previous algorithms that use hopsets for shortest paths and routing [HKN14, Nan14, HKN16, LP15, EN16a], in our algorithm a vertex v cannot afford storing all hopset edges e whose respective paths $P(e)$ traverse v. Instead, we show that even with limited memory one can execute a protocol that will recover for each hopset edge the path that implements it in G.

1.4 A Bibliographical Note

A preliminary version of this paper [EN17] was published in the arXiv in the end of April, 2017. A few days later, independently of us Huang and Pettie posted [HP17] a proof that Thorup-Zwick's emulators give rise to linear-size hopsets, and that the same construction applies for all ϵ simultaneously. The choice of sampling probabilities in [HP17] is slightly better than ours, improving the exponent of our β by an additive 1. (That is, in our existential construction, if

[4] Our construction is simpler than that of [Ber09] in two aspects: First, [Ber09] use *truncated* TZ-bunches, while we use the standard TZ bunches. Second, [Ber09] applies the construction for all possible distances scales, while we apply it just once. The distributed implementations of [HKN14, Nan14, HKN16] are even more complicated, since the hop-reduction technique requires a recursive application of the construction.

the size is $O(n^{1+1/\kappa})$ then $\beta = O\left(\frac{\log \kappa}{\epsilon}\right)^{\log \kappa}$, while in theirs it is $O\left(\frac{\log \kappa}{\epsilon}\right)^{\log \kappa - 1}$, and the latter is tight as a function of ϵ [ABP17].) The manuscript of [HP17] does not show, however, that these hopsets can be efficiently constructed in parallel or distributed models of computation.

1.5 Organization

Our linear-size hopsets appear in Section 2. The distributed construction of hopsets is in Section 3.1 for Congested Clique and Section 3.2 for the CONGEST model. The hopsets in the PRAM model are presented in Section 4. The applications to shortest paths in the CONGEST and PRAM models appear in Section 5.

2 Linear Size Hopsets

Let $G = (V, E)$ be a weighted graph, and fix a parameter $k \geq 1$. Let $\nu = 1/(2^k - 1)$ (one should think of $\kappa = 2^k - 1 = 1/\nu$). Let $V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_k = \emptyset$ be a sequence of sets, such that for all $0 \leq i < k-1$, A_{i+1} is created by sampling every element from A_i independently with probability $p_i = n^{-2^i \cdot \nu}$.[5] It follows that for $0 \leq i \leq k-1$ we have

$$N_i := \mathbb{E}[|A_i|] = n \cdot \prod_{j=0}^{i-1} p_j = n^{1-(2^i-1)\nu} ,$$

and in particular $N_{k-1} = n^{(1+\nu)/2}$.

For every $0 \leq i \leq k-1$ and every vertex $u \in A_i \setminus A_{i+1}$, define the pivot $p(u) \in A_{i+1}$ as a vertex satisfying $d_G(u, A_{i+1}) = d_G(u, p(u))$ (note $p(u)$ does not necessarily exist, e.g. for $u \in A_{k-1}$), and define the bunch

$$B(u) = \{v \in A_i \ : \ d_G(u,v) < d_G(u, A_{i+1})\} \cup \{p(u)\} .$$

The hopset is created by taking $H = \{(u,v) \ : \ u \in V, v \in B(u)\}$, where the length of the edge (u,v) is set as $d_G(u,v)$. As argued in [TZ01], for any $0 \leq i \leq k-2$ and $u \in A_i \setminus A_{i+1}$, the size of $B(u)$ is bounded by a random variable sampled from a geometric distribution with parameter p_i (this corresponds to the first vertex of A_i, when ordered by distance to u, that is included in A_{i+1}). So $\mathbb{E}[|B(u)|] \leq 1/p_i = n^{2^i \cdot \nu}$. For $u \in A_{k-1}$ we have $\mathbb{E}[|B(u)|] = N_{k-1} = n^{(1+\nu)/2}$. The expected size of the hopset H is at most

$$\sum_{i=0}^{k-2} (N_i \cdot n^{2^i \cdot \nu}) + N_{k-1} \cdot N_{k-1} = k \cdot n^{1+\nu} .$$

The following lemma bounds the number of hops and stretch of H. Recall that $d_G^{(t)}(u,v)$ is the length of the shortest-path between u, v in G that consists of at most t edges.

Lemma 1. *Fix any $0 < \delta < 1/(8k)$ and any $x, y \in V$. Then for every $0 \leq i \leq k-1$, at least one of the following holds:*

1. $d_{G \cup H}^{((3/\delta)^i)}(x,y) \leq (1 + 8\delta i) \cdot d_G(x,y)$.
2. *There exists $z \in A_{i+1}$ such that* $d_{G \cup H}^{((3/\delta)^i)}(x,z) \leq 2 d_G(x,y)$.

Proof. The proof is by induction on i. We start with the basis $i = 0$. If it is the case that $y \in B(x)$, then we added the edge (x,y) to the hopset, i.e. $d_H^{(1)}(x,y) = d_G(x,y)$, and so the first item holds. Otherwise, consider the case that $x \in A_1$: then we can take $z = x$, so the second item holds trivially. The remaining case is that $x \in A_0 \setminus A_1$ and $y \notin B(x)$, so by definition of $B(x)$ we get that $d_G(x,y) \geq d_G(x, A_1)$. By taking $z = p(x)$, there is a single edge between x, z in H of length $d_G(x,z) = d_G(x, A_1) \leq d_G(x,y)$, which satisfies the second item.

[5] Our definition is slightly different than that of [TZ06], which used $p_i = |A_i|/n^{1+\nu}$, but it gives rise to the same expected size of A_i. We use our version since it allows efficient implementation in various models of computation.

Assume the claim holds for i, and we prove for $i+1$ (note we may assume $i \leq k-2$, since the maximal i we need is $k-1$). Consider the path $\pi(x,y)$ between x,y in G, and partition it into $J \leq 1/\delta$ segments $\{L_j = [u_j, v_j]\}_{j\in[J]}$, each of length at most $\delta \cdot d_G(x,y)$, and at most $1/\delta$ edges $\{(v_j, u_{j+1})\}_{j\in[J]}$ of G connecting these segments. This can be done as follows: define $u_1 = x$, and for $j \geq 1$, walk from u_j on $\pi(x,y)$ (towards y) until the point v_j, which is the vertex so that the next edge will take us to distance greater than $\delta \cdot d_G(x,y)$ from u_j (or until we reached y). By definition, $d_G(u_j, v_j) \leq \delta \cdot d_G(x,y)$. Define u_{j+1} to be the neighbor of v_j on $\pi(x,y)$ that is closer to y (if exists). If u_{j+1} does not exist (which can happen only when $v_j = y$) then define $u_{j+1} = y$ and $J = j$. Observe that for all $1 \leq j \leq J-1$, $d_G(u_j, u_{j+1}) > \delta \cdot d_G(x,y)$, so indeed $J \leq 1/\delta$.

We use the induction hypothesis for all pairs (u_j, v_j) with parameter i. Consider first the case that the first item holds for all of them, that is, $d^{((3/\delta)^i)}_{G\cup H}(u_j, v_j) \leq (1+8\delta i)\cdot d_G(u_j, v_j)$. Then we take the path in $G \cup H$ that consists of the $(3/\delta)^i$-hops between each pair u_j, v_j, and the edges (v_j, u_{j+1}) of G. Since $(3/\delta)^{i+1} \geq (1/\delta)\cdot(3/\delta)^i + 1/\delta$, we have

$$\begin{aligned} d^{((3/\delta)^{i+1})}_{G\cup H}(x,y) &\leq \sum_{j\in[J]} d^{((3/\delta)^i)}_{G\cup H}(u_j, v_j) + d^{(1)}_G(v_j, u_{j+1}) \\ &\leq (1+8\delta i)\cdot d_G(x,y)\ . \end{aligned}$$

The second case is that there are pairs (u_j, v_j) for which only the second item holds. Let $l \in [J]$ (resp., $r \in [J]$) be the first (resp., last) index for which the first item does not hold for the pair (u_l, v_l) (resp., (u_r, v_r)). Then there are $z_l, z_r \in A_{i+1}$ such that

$$\begin{aligned} d^{((3/\delta)^i)}_{G\cup H}(u_l, z_l) \leq 2d_G(u_l, v_l) \text{ and} \\ d^{((3/\delta)^i)}_{G\cup H}(v_r, z_r) \leq 2d_G(u_r, v_r). \end{aligned} \tag{2}$$

(Note that we used v_r and not u_r in the second inequality. This can be done since the lemma's assertion holds for the pair (v_r, u_r) as well, and as the first item is symmetric with respect to u_r, v_r, it does not hold for the pair (v_r, u_r) as well.) Consider now the case that $z_r \in B(z_l)$. In this case we added the edge (z_l, z_r) to the hopset, and by the triangle inequality,

$$\begin{aligned} d^{(1)}_H(z_l, z_r) &= d_G(z_l, z_r) \\ &\leq d^{((3/\delta)^i)}_{G\cup H}(u_l, z_l) + d_G(u_l, v_r) + d^{((3/\delta)^i)}_{G\cup H}(z_r, v_r)\ . \end{aligned} \tag{3}$$

Next, apply the inductive hypothesis on segments $\{L_j\}$ for $j < l$ and $j > r$, and in between use the detour via u_l, z_l, z_r, v_r. Since $l \leq r$ there is at least one segment we skipped, so the total number of hops is bounded by $(1/\delta - 1)\cdot(3/\delta)^i + 1/\delta + 2(3/\delta)^i + 1$. (The additive term of $1/\delta$ accounts for the edges (v_j, v_{j+1}), $0 \leq j \leq J-1$.) This expression is at most $(3/\delta)^{i+1}$ whenever $\delta < 1/2$. It follows that

$$\begin{aligned} & d^{((3/\delta)^{i+1})}_{G\cup H}(x,y) \\ &\leq \sum_{j=1}^{l-1}\left[d^{((3/\delta)^i)}_{G\cup H}(u_j, v_j) + d^{(1)}_G(v_j, u_{j+1})\right] + d^{((3/\delta)^i)}_{G\cup H}(u_l, z_l) \\ &\quad + d^{(1)}_H(z_l, z_r) + d^{((3/\delta)^i)}_{G\cup H}(z_r, v_r) + d^{(1)}_G(v_r, u_{r+1}) \\ &\quad + \sum_{j=r+1}^{J}\left[d^{((3/\delta)^i)}_{G\cup H}(u_j, v_j) + d^{(1)}_G(v_j, u_{j+1})\right] \\ &\overset{(3)}{\leq} (1+8\delta i)d_G(x, u_l) + d_G(u_l, v_r) + (1+8\delta i)d_G(v_r, y) \\ &\quad + 2d^{((3/\delta)^i)}_{G\cup H}(u_l, z_l) + 2d^{((3/\delta)^i)}_{G\cup H}(z_r, v_r) \\ &\overset{(2)}{\leq} 8\delta\cdot d_G(x,y) + (1+8\delta i)d_G(x,y) \\ &= (1+8\delta(i+1))\cdot d_G(x,y)\ . \end{aligned} \tag{4}$$

This demonstrates item 1 holds in this case. The final case to consider is that $z_r \notin B(z_l)$. Assume first that $z_l \notin A_{i+2}$. Then taking $z = p(z_l) \in A_{i+2}$, the definition of $B(z_l)$ implies that $d_G(z_l, z) \leq d_G(z_l, z_r)$. We now claim that item 2 holds for such a choice of z. Indeed, since $(z_l, z) \in H$, we have

$$\begin{aligned}
& d_{G\cup H}^{((3/\delta)^{i+1})}(x,z) \\
&\leq \sum_{j=1}^{l-1}\left[d_{G\cup H}^{((3/\delta)^{i})}(u_j,v_j) + d_G^{(1)}(v_j,u_{j+1})\right] + d_{G\cup H}^{((3/\delta)^{i})}(u_l,z_l) \\
&\quad + d_H^{(1)}(z_l,z) \\
&\leq (1+8\delta i)d_G(x,u_l) + d_{G\cup H}^{((3/\delta)^{i})}(u_l,z_l) + d_G(z_l,z_r) \\
&\overset{(3)}{\leq} (1+8\delta i)d_G(x,u_l) + 2d_{G\cup H}^{((3/\delta)^{i})}(u_l,z_l) + d_G(u_l,v_r) \\
&\quad + d_{G\cup H}^{((3/\delta)^{i})}(z_r,v_r) \\
&\overset{(2)}{\leq} (1+8\delta i)d_G(x,v_r) + 6\delta\cdot d_G(x,y) \\
&\leq 2d_G(x,y),
\end{aligned}$$

where the last inequality used that $\delta < 1/(8k)$. The case that $z_l \in A_{i+2}$ is simpler, since we may take $z = z_l$. □

Fix any $0 < \epsilon < 1$, and apply the lemma on any pair x, y with $\delta = \epsilon/(8k)$ and $i = k-1$. It must be that the first item holds (since $A_k = \emptyset$), so we have that

$$d_{G\cup H}^{(\beta)}(x,y) \leq (1+\epsilon)d_G(x,y)\ ,$$

where the number of hops is given by $\beta = (24k/\epsilon)^{k-1}$. We derive the following theorem.

Theorem 1. *For any weighted graph $G = (V,E)$ on n vertices, and any $k \geq 1$, there exists H of size at most $O(k\cdot n^{1+1/(2^k-1)})$, which is a (β,ϵ)-hopset for any $0<\epsilon<1$ with $\beta = O(k/\epsilon)^{k-1}$.*

2.1 Improved Hopset Size

Here we show how to remove the k factor from the hopset size, at the cost of increasing the exponent of β by an additive 1. Note that we may assume w.l.o.g. that $k \leq \log\log n - 1$, as for larger values of k, both β and the size of the hopset (which becomes $O(kn)$), grow with k. We will increase the number of sets by 1, and sample $V = A'_0 \supseteq A'_1 \supseteq \cdots \supseteq A'_{k+1} = \emptyset$ using the following probabilities: $p'_i = n^{-2^i\cdot\nu}\cdot 2^{2^i-1}$ (the restriction on k ensures $p'_i < 1$). Now for $0 \leq i \leq k$,

$$N'_i := \mathbb{E}[|A'_i|] = n\cdot\prod_{j=0}^{i-1} p'_j = n^{1-(2^i-1)\nu}\cdot 2^{2^i-i-1}\ ,$$

and in particular $N'_k \leq 2^{2^k-k} \leq n^{1/2}$. The expected size of H becomes at most

$$\sum_{i=0}^{k-1}(N'_i/p'_i) + N'_k\cdot N'_k \leq \sum_{i=0}^{k-1}(n^{1+\nu}/2^i) + n \leq 3n^{1+\nu}\ .$$

The hopset construction and the stretch analysis in Lemma 1 remains essentially the same. There is an additional set now which increases the exponent of β by 1.

Theorem 2. *For any weighted graph $G = (V,E)$ on n vertices and any $k \geq 1$, there exists H of size at most $O(n^{1+1/(2^k-1)})$, which is a (β,ϵ)-hopset for any $0<\epsilon<1$ with $\beta = O(k/\epsilon)^k$.*

Since our construction is based on the [TZ06] emulator construction, following their analysis we obtain an emulator with additive stretch that can have *linear* size.

Corollary 2. *For any un-weighted graph $G = (V, E)$ on n vertices and any $k \geq 1$, there exists an emulator H of size at most $O(n^{1+1/(2^k-1)})$, with additive stretch $O(k \cdot d^{1-1/k})$ for pairs at distance d.*

2.2 Efficient Implementation in the Standard Model

We consider the construction and notation of Section 2.1, with the slightly stronger assumption $k \leq \log\log n - 2$.

It was (implicitly) shown in [TZ01] that for any $0 \leq i < k$, the sets $B(u)$ (and the corresponding distances) for all $u \in A'_i \setminus A'_{i+1}$ can be computed in expected time $O(|E| + n\log n)/p'_i$. (In fact, in [TZ01], the set $B(u)$ contains more vertices, not only those in A'_i. However, we can remove the extra vertices easily.) The running time becomes larger as i grows, and in order to keep it under control, we use the method of [EN16a]: introduce a parameter $2\nu < \rho < 1$, and redefine the probabilities as follows. Set $i_0 = \lfloor \log(\rho/\nu) \rfloor$ and $i_1 = i_0 + 1 + \left\lceil \frac{1}{\rho} \right\rceil$. For $0 \leq i \leq i_0$, let $p'_i = n^{-2^i \cdot \nu} \cdot 2^{2^i - 1}$ as in Section 2.1. Set also $p'_{i_0+1} = n^{-\rho/2}$, and for the remaining levels $i_0 + 2 \leq i \leq i_1$, set $p'_i = n^{-\rho}$. Finally, let $A'_{i_1+1} = \emptyset$. Note that for $0 \leq i \leq i_0 + 1$ we have that

$$N'_i = n \prod_{j=0}^{i-1} p'_j = n^{1-(2^i-1)\nu} \cdot 2^{2^i - i - 1} \,. \tag{5}$$

In particular, using that $\rho/\nu \leq 2^{i_0+1} \leq 2\rho/\nu$, we get

$$N'_{i_0+1} \leq n^{1-(\rho/\nu-1)\nu} \cdot 2^{2\rho/\nu} \leq n^{1+\nu-\rho/2} \,, \tag{6}$$

where the last inequality used that $k \leq \log\log n - 2$, thus $1/\nu = (2^k - 1) \leq \log n/4$, so that $2^{2\rho/\nu} \leq n^{\rho/2}$. Thus for any $i \geq i_0 + 2$ we see that

$$\begin{aligned} N'_i &= N'_{i_0+1} \prod_{j=i_0+1}^{i-1} p'_j \overset{(6)}{\leq} n^{1+\nu-\rho/2} \cdot n^{-\rho/2} \cdot n^{-(i-1-(i_0+1))\cdot\rho} \\ &= n^{1+\nu} \cdot n^{-(i-(i_0+1))\cdot\rho} \,. \end{aligned} \tag{7}$$

The expected number of edges inserted until phase $i_0 + 1$ is at most

$$\begin{aligned} \sum_{i=0}^{i_0} N'_i/p'_i &\overset{(5)}{\leq} \sum_{i=0}^{i_0} n^{1-(2^i-1)\nu} \cdot 2^{2^i-i-1} \cdot n^{2^i\cdot\nu}/2^{2^i-1} \\ &= \sum_{i=0}^{i_0} n^{1+\nu}/2^i \leq 2n^{1+\nu} \,. \end{aligned}$$

The expected number of edges at phase $i_0 + 1$ is bounded by

$$N'_{i_0+1}/p'_{i_0+1} \overset{(6)}{\leq} n^{1+\nu-\rho/2} \cdot n^{\rho/2} = n^{1+\nu} \,.$$

The remaining phases until i_1 introduce at most

$$\begin{aligned} \sum_{i=i_0+2}^{i_1-1} N'_i/p'_i &\overset{(7)}{\leq} \sum_{i=i_0+2}^{i_1-1} n^{1+\nu} \cdot n^{-(i-(i_0+1))\cdot\rho} \cdot n^{\rho} \\ &\leq n^{1+\nu} \cdot \sum_{i=0}^{\infty} n^{-i\rho} \leq 2n^{1+\nu} \,, \end{aligned}$$

as this summation converges. Finally, since

$$N'_{i_1} \leq n^{1+\nu} \cdot n^{-(i_1-(i_0+1))\cdot\rho} \leq n^{\nu} \ , \tag{8}$$

the last phase i_1 contributes at most $N'_{i_1} \cdot N'_{i_1} \leq n^{2\nu} \leq n^{1+\nu}$ edges to the hopset. We conclude that the total number of edges is $O(n^{1+\nu})$.

Recall that the expected running time of the Dijkstra explorations at level $i < i_1$ is $O(|E| + n\log n)/p'_i$. Thus the expected running time of the first i_0 levels converges to $O(|E| + n\log n) \cdot n^{\rho}$, while each of the at most $\lceil 1/\rho \rceil + 1$ remaining levels will take also $O(|E| + n\log n) \cdot n^{\rho}$ time. The final level is expected to take $O(|E| + n\log n) \cdot n^{\rho}$ as well, since there are expected $O(n^{\nu})$ vertices at A_{i_1} from which Dijkstra is performed, and $\nu < \rho$. The price we pay is in a higher number of sets, which increases the exponent of β by at most an additive $1/\rho$. We have thus proven the following theorem.

Theorem 3. *For any weighted graph $G = (V, E)$ on n vertices, and any $1 < k \leq \log\log n - 2$, $\frac{2}{2^k-1} < \rho < 1$, there is a randomized algorithm that runs in expected time $O(|E| + n\log n) \cdot n^{\rho}/\rho$, and computes H of size at most $O(n^{1+1/(2^k-1)})$, which is a (β, ϵ)-hopset for any $0 < \epsilon < 1$, where*

$$\beta = O\left(\frac{k + 1/\rho}{\epsilon}\right)^{k+1/\rho+1} .$$

By substituting $\kappa = 2^k - 1$, we obtain an improved version of the hopsets of [EN16a], where both the size of the hopset and the running time are smaller by a factor of $\log n$, while the other parameters remain the same. Another notable advantage is that it yields a single hopset which works for all $0 < \epsilon < 1$ simultaneously.

Corollary 3. *For any weighted graph $G = (V, E)$ on n vertices, and any $\kappa \geq 1$, $\frac{2}{\kappa} < \rho < 1$, there is a randomized algorithm that runs in expected time $O(|E| + n\log n) \cdot n^{\rho}/\rho$, and computes H of size at most $O(n^{1+1/\kappa})$, which is a (β, ϵ)-hopset for any $0 < \epsilon < 1$, where*

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

3 Distributed Models

We will consider two models in distributed computing: the Congested Clique model, and the CONGEST RAM 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 an identity of a vertex, an edge weight, a distance in the graph, or anything else of no larger (up to a fixed constant factor) size.[6] The local computation is assumed to require zero time, and we are interested in algorithms that run for as few rounds as possible. In the Congested Clique model, we assume that all vertices are interconnected via direct edges, while in the CONGEST RAM model, every vertex can send messages only to its G-neighbors (the weight of edges is irrelevant to the communication time).

3.1 Congested Clique Model

We first show how to construct the hopset in the Congested Clique model. In order to avoid a high number of rounds when computing distances for determining the bunches $\{B(u)\}$, we build the hopset in $\log n$ levels, where each level ℓ hopset will only "take care" of pairs that have a shortest path with *at most 2^{ℓ} hops*. This is somewhat different from previous works [Coh00, Nan14, HKN16, EN16a], in which the level ℓ hopset handled pairs with *distance in the range* $[2^{\ell}, 2^{\ell+1}]$. A few advantages of our current approach: it easily avoids the dependency on the ratio between largest and

[6] 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 to capture 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.

smallest distance, and also the final hopset is just the level $\log n$ one, so we can obtain a linear size hopset (unlike previous works which took the union of all levels).

There are a few technical difficulties in implementing the algorithm of Section 2 in a distributed setting. The first is that the [TZ01] method for computing the bunches $B(u)$ was to compute their "inverses" – called clusters.[7] Alas, it is not known how to efficiently compute these clusters in a distributed manner. Rather, we compute the bunches directly, and to avoid the potential large congestion (a vertex may be a part of many bunches, and needs to send messages for all of them), we replace the bunches with *half-bunches* (i.e., taking only points closer than half the distance to the pivot). See just below for the formal definition. The second issue (of congestion) is more subtle, and arises since hop-bounded distances do not obey the triangle inequality. For the stretch analysis to go through, we need that the weight of the hopset edges will be bounded by a certain path between the end-points of the edge (see (14)). In order to ensure that this happens, we build each hopset for level ℓ gradually, i.e. the bunches are created first for $A_0 \setminus A_1$, then for $A_1 \setminus A_2$ and so on, where each time the partial hopset is added to the graph on which we compute distances.

We say that H is a (β, ϵ, t)-hopset if $G \cup H$ provides $(1+\epsilon)$ approximation with at most β hops for all pairs $x, y \in V$ such that $d_G(x,y) = d_G^{(t)}(x,y)$ (i.e., the pairs that have a shortest path consisting of at most t edges between them). Note that the empty set is a $(1,0,1)$-hopset (and thus also a $(\beta,\epsilon,1)$-hopset for all $\beta \geq 1$ and $\epsilon > 0$). Given a $(\beta, \epsilon_{\ell-1}, 2^{\ell-1})$-hopset $H^{(\ell-1)}$, we build a $(\beta, \epsilon_\ell, 2^\ell)$-hopset $H^{(\ell)}$, where $1+\epsilon_\ell = (1+\epsilon)^\ell$ for some $0 < \epsilon < 1/(5\log n)$. The final hopset will be $H^{(\log n)}$. (We stress that the previous hopsets $H^{(1)}, \ldots, H^{(\ell-1)}$ are only used to compute $H^{(\ell)}$, and are not contained in it.)

Observe that $H^{(\ell-1)}$ is a $(2\beta, \epsilon_{\ell-1}, 2^\ell)$-hopset, since every path with at most 2^ℓ hops can be partitioned into two paths of at most $2^{\ell-1}$ hops each, and $H^{(\ell-1)}$ provides a $(1+\epsilon_{\ell-1})$-approximation with β hops for each of these. It follows that for any $x, y \in V$,

$$d^{(2\beta)}_{G \cup H^{(\ell-1)}}(x,y) \leq (1+\epsilon_{\ell-1}) \cdot d_G^{(2^\ell)}(x,y) \; . \tag{9}$$

We sample sets $V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_{k'+1} = \emptyset$ as in Section 2.2, where $k' = i_1 \leq k + 1/\rho + 1$ is the number of sets. We introduce a subtle change to the construction – in the previous section we defined for each $u \in A_i \setminus A_{i+1}$ a set $B(u)$, and added the edges (u,v) for all $v \in B(u)$, simultaneously for all $0 \leq i \leq k'$. Here we shall build the hopset $H = H^{(\ell)}$ gradually: For each $i = 0, 1, \ldots, k'$ we define a set of edges $H_i = H_i^{(\ell)}$ corresponding to the bunches of vertices in $V \setminus A_{i+1}$, and finally take $H = H_{k'}$.

Fix some $0 \leq i \leq k'$, and assume we built the set H_{i-1} (when $i = 0$ define $H_{-1} = \emptyset$). We shall work in the graph G_i, defined by

$$G_i = G \cup H^{(\ell-1)} \cup H_{i-1} \; .$$

The algorithm consists of two stages. On the first stage, for 8β rounds, run a Bellman-Ford exploration in G_i rooted at A_{i+1}, to obtain for each $u \in V$ the value $\hat{d}(u, A_{i+1}) = d^{(8\beta)}_{G_i}(u, A_{i+1})$. (If some vertex $v \in V$ was not found in the exploration, then we set $\hat{d}(v, A_{i+1}) = \infty$.) Also for each vertex $u \in A_i \setminus A_{i+1}$ with $\hat{d}(u, A_{i+1}) < \infty$, store $p(u)$ as a vertex $p \in A_{i+1}$ satisfying $\hat{d}(u, A_{i+1}) = d^{(8\beta)}_{G_i}(u,p)$.

To determine the bunches (this is the second stage), each vertex $u \in A_i \setminus A_{i+1}$ conducts another Bellman-Ford exploration in the graph G_i rooted at u, this time for only 4β rounds, i.e., half the number of hops 8β of the first exploration, to distance less than $\hat{d}(u, A_{i+1})/2$ (i.e., the messages whose origin is u contain the value $\hat{d}(u, A_{i+1})$, and only vertices within this distance from u forward the message in the next round). Define the half-bunch $B(u) = \{v \in A_i \; : \; d^{(4\beta)}_{G_i}(u,v) < \hat{d}(u, A_{i+1})/2\} \cup \{p(u)\}$. Let

$$H_i = H_{i-1} \cup \bigcup_{u \in A_i \setminus A_{i+1}} \{(u,v) \; : \; v \in B(u)\} \; , \tag{10}$$

and set the weight of the edge (u,v) as the distance discovered in the exploration (i.e. $d^{(4\beta)}_{G_i}(u,v)$ for $v \in B(u)$ and $d^{(8\beta)}_{G_i}(u,p(u))$ for the pivot).

Claim 4. $\mathbb{E}[|H|] \leq O(n^{1+\nu})$.

[7] The cluster $C(v)$ is defined as follows: each point $u \in C(v)$ iff $v \in B(u)$.

Proof. The argument is essentially the same as the one in Section 2.2. The only difference is that when analyzing in step $0 \leq i < k'$ the expected size of a bunch, i.e. $\mathbb{E}[|B(u)|]$ for some $u \in A_i \setminus A_{i+1}$, we consider the ordering on V given by the 4β-bounded distance from u in the graph G_i, i.e,. according to $d^{(4\beta)}_{G_i}(u,\cdot)$. Then the size of $B(u)$ is bounded by the index of the first vertex in this ordering that is included in A_{i+1}. Since every $v \in A_i$ is included in A_{i+1} independently with probability p'_i, we have that $\mathbb{E}[|B(u)|] \leq 1/p'_i$. In fact, $B(u)$ may have a smaller size than the first index included in A_{i+1}, since we use more hops for computing distances to pivots (which reduces the distance threshold for being in $B(u)$), and since we only take into the bunch points that are less than half the distance to the pivot. Finally, for the last level k' we have that for $u \in A_{k'}$, $\mathbb{E}[|B(u)|] \leq \mathbb{E}[|A_{k'}|] \overset{(8)}{\leq} n^{\nu}$. Combining this with bounds on $N'_i = \mathbb{E}[|A_i|]$ in Section 2.2 we can bound the size of H in the same manner. □

In fact, since $|B(u)|$ is stochastically bounded by a geometric distribution with parameter $p'_i \geq n^{-\rho}$, it follows that with high probability for all $v \in V$,

$$|B(v)| \leq 4n^{\rho} \cdot \ln n \; . \tag{11}$$

Claim 5. *The number of rounds required is whp $O(n^{\rho} \cdot k' \cdot \log^2 n \cdot \beta)$.*

Proof. The sampling of the sets A_i is done independently for each vertex, therefore it requires no communication. For each $1 \leq \ell \leq \log n$ and $0 \leq i < k'$ we conduct a single Bellman-Ford exploration in G_i rooted at A_{i+1} for 8β rounds. Since in the Congested Clique model all edges are present, this requires $O(\beta)$ rounds per exploration (every vertex sends just a single message to all its neighbors every round). The more expensive step is the explorations to range 4β rooted at each $u \in A_i \setminus A_{i+1}$. The number of rounds in these explorations is affected by the number of messages a vertex $v \in V$ needs to forward to its neighbors at each round. In what follows we prove that with high probability this number is at most $O(n^{\rho} \log n)$ for every $v \in V$.

Fix $0 \leq i < k'$. Consider $v \in V$, and order the vertices of A_i according to their 4β-bounded distance to v in G_i, that is, according to $d^{(4\beta)}_{G_i}(v,\cdot)$. Since $p'_i \geq n^{-\rho}$, the probability that none of the first $2n^{\rho} \ln n$ vertices in that ordering is sampled to A_{i+1} is at most

$$\left(1 - n^{-\rho}\right)^{2n^{\rho} \ln n} \leq 1/n^2 \; .$$

So by the union bound on the n vertices, with high probability, for each $v \in V$ at least one of the first $2n^{\rho} \ln n$ vertices in its ordering of A_i is sampled to A_{i+1}. Denote by $z \in A_{i+1}$ the first vertex in the ordering of v that was chosen to A_{i+1}. We claim that no vertex $u \in A_i$, that appears after z in the ordering of v, will cause v to forward messages concerning $B(u)$. This is because in the first stage we performed the Bellman-Ford exploration rooted at A_{i+1} for 8β rounds. Thus

$$\hat{d}(u, A_{i+1}) \leq d^{(8\beta)}_{G_i}(u,z) \leq d^{(4\beta)}_{G_i}(u,v) + d^{(4\beta)}_{G_i}(v,z) \leq 2d^{(4\beta)}_{G_i}(u,v) \; ,$$

where the last inequality uses the assumption that u appeared after z in v's ordering. We obtained $d^{(4\beta)}_{G_i}(u,v) \geq \hat{d}(u, A_{i+1})/2$. So by the definition of half bunch, $v \notin B(u)$ and thus, v will not forward u's messages.

We still have to argue about the last level $i = k'$ (since no vertex is chosen to $A_{k'+1}$). Recall that the expected size of $A_{k'}$ is bounded by n^{ν}, as shown in (8). It can be easily checked that whp $|A_{k'}| \leq O(n^{\nu} \cdot \log n) \leq O(n^{\rho} \cdot \log n)$. (Recall that $\rho \geq 2\nu$.) We conclude that whp every vertex needs to send at most $O(n^{\rho} \log n)$ messages to implement a single step of Bellman-Ford. There are $O(\beta)$ rounds for each $\ell = 1, \ldots, \log n$ and each $0 \leq i \leq k'$, so the total number of rounds required is $O(n^{\rho} \cdot k' \cdot \log^2 n \cdot \beta)$. □

Next, we prove the analogue of Lemma 1 for the distributed setting. There are several subtle differences described in the beginning of the section, so we provide the complete proof.

Lemma 6. *Fix any $0 < \delta < 1/(15k')$, set $\beta = (3/\delta)^{k'}$, and let $x, y \in V$ be such that $d_G(x,y) = d^{(2^{\ell})}_G(x,y)$. Then for every $0 \leq i \leq k'$, at least one of the following two assertions holds:*

1. $d^{((3/\delta)^i)}_{G \cup H_i}(x,y) \leq (1 + \epsilon_{\ell-1}) \cdot (1 + 12\delta i) \cdot d_G(x,y)$.
2. *There exists $z \in A_{i+1}$ such that $d^{((3/\delta)^i)}_{G \cup H_i}(x,z) \leq 3d_G(x,y)$.*

Proof. The proof is by induction on i. We start with the base case $i = 0$. If it is the case that $x \in A_1$ then we can take $z = x$ and the second item holds trivially. Otherwise, consider the case that $x \in A_0 \setminus A_1$ and $y \in B(x)$. Then we added an edge (x, y) to H_0 of weight $d^{(4\beta)}_{G_0}(x, y)$ (the case that $y = p(x)$ is similar, replacing 4β by 8β). Recall that $G_0 = G \cup H^{(\ell-1)}$. Hence

$$d^{(1)}_{H_0}(x, y) = d^{(4\beta)}_{G\cup H^{(\ell-1)}}(x, y) \overset{(9)}{\leq} (1 + \epsilon_{\ell-1}) \cdot d^{(2^\ell)}_G(x, y) = (1 + \epsilon_{\ell-1}) \cdot d_G(x, y) \ ,$$

so the first item holds. The last case is that $x \in A_0 \setminus A_1$ and $y \notin B(x)$. By the definition of H_0 it must be that

$$\hat{d}(x, A_1) \leq 2d^{(4\beta)}_{G_0}(x, y) \ . \tag{12}$$

Since $p(x) \in B(x)$, the edge $(x, p(x))$ is in the hopset, and its weight is

$$d^{(8\beta)}_{G_0}(x, p(x)) = \hat{d}(x, A_1) \overset{(12)}{\leq} 2d^{(4\beta)}_{G_0}(x, y) \leq 2(1 + \epsilon_{\ell-1}) \cdot d_G(x, y) < 3d_G(x, y) \ ,$$

where for the last two inequalities we again used (9) and the fact that $1+\epsilon_{\ell-1} < 3/2$. (The latter holds since we assume $\epsilon < 1/(5 \log n)$ and $\ell \leq \log n$, so $(1 + \epsilon_\ell) = (1 + \epsilon)^\ell < e^{1/5}$). This proves the second item with $z = p(x) \in A_1$.

Assume the claim holds for i, and we prove for $i + 1$. Consider the shortest path $\pi(x, y)$ between x, y in G that contains at most 2^ℓ edges, and partition it into $J \leq 1/\delta$ segments $\{L_j = [u_j, v_j]\}_{j\in[J]}$ as in the proof of Lemma 1. We use the induction hypothesis for all pairs (u_j, v_j) with parameter i. (By the virtue of lying on a shortest path that has at most 2^ℓ edges, all these pairs satisfy $d^{(2^\ell)}_G(u_j, v_j) = d_G(u_j, v_j)$). Consider first the case that the first item holds for all of them, that is, $d^{((3/\delta)^i)}_{G\cup H_i}(u_j, v_j) \leq (1 + \epsilon_{\ell-1}) \cdot (1 + 12\delta i) \cdot d_G(u_j, v_j)$. Then we take the path in $G \cup H_i$ that consists of the $(3/\delta)^i$-hops between each pair u_j, v_j, and the edges (v_j, u_{j+1}) of G. Since by (10), $H_i \subseteq H_{i+1}$, we have

$$d^{((3/\delta)^{i+1})}_{G\cup H_{i+1}}(x, y) \leq \sum_{j\in[J]} (d^{((3/\delta)^i)}_{G\cup H_i}(u_j, v_j) + d^{(1)}_G)(v_j, u_{j+1}) \leq (1 + \epsilon_{\ell-1}) \cdot (1 + 12\delta i) \cdot d_G(x, y) \ ,$$

which concludes the proof for the first case. The second case is that there are pairs (u_j, v_j) for which only the second item holds. Let $l \in [J]$ (resp., $r \in [J]$) be the first (resp., last) index for which the first item does not hold for the pair (u_l, v_l) (resp., (u_r, v_r)). Then there are $z_l, z_r \in A_{i+1}$ such that

$$d^{((3/\delta)^i)}_{G\cup H_i}(u_l, z_l) \leq 3d_G(u_l, v_l) \ \text{ and } \ d^{((3/\delta)^i)}_{G\cup H_i}(v_r, z_r) \leq 3d_G(u_r, v_r) \ . \tag{13}$$

Consider first the case that $z_l \in A_{i+2}$. Then we take $z = z_l$, and derive

$$\begin{aligned} d^{((3/\delta)^{i+1})}_{G\cup H_{i+1}}(x, z) &\leq \sum_{j=1}^{l-1} \left(d^{((3/\delta)^i)}_{G\cup H_i}(u_j, v_j) + d^{(1)}_G(v_j, u_{j+1}) \right) + d^{((3/\delta)^i)}_{G\cup H_i}(u_l, z_l) \\ &\overset{(13)}{\leq} (1 + \epsilon_{\ell-1}) \cdot (1 + 12\delta i) \cdot d_G(x, u_l) + 3d_G(u_l, v_l) \\ &\leq 3d_G(x, y) \ , \end{aligned}$$

where in the second inequality we used that the first item holds for all intervals until the l-th one, and in the final one that $1 + \epsilon_{\ell-1} < 3/2$ and $1 + 12\delta i < 2$.

From now on assume $z_l \in A_{i+1} \setminus A_{i+2}$. Recall that the Bellman-Ford explorations that constructed H_{i+1} were conducted in the graph $G_{i+1} = G \cup H^{(\ell-1)} \cup H_i$. These explorations were conducted to hop-depth 8β on the first stage, and 4β on the second. This allows us to provide the following bound

$$\begin{aligned} d^{(4\beta)}_{G_{i+1}}(z_l, z_r) &\leq d^{(\beta)}_{G\cup H_i}(z_l, u_l) + d^{(2\beta)}_{G\cup H^{(\ell-1)}}(u_l, v_r) + d^{(\beta)}_{G\cup H_i}(v_r, z_r) \\ &\overset{(9)}{\leq} d^{((3/\delta)^i)}_{G\cup H_i}(z_l, u_l) + (1 + \epsilon_{\ell-1}) \cdot d_G(u_l, v_r) + d^{((3/\delta)^i)}_{G\cup H_i}(v_r, z_r) \ . \end{aligned} \tag{14}$$

Here the first inequality follows by the triangle inequality, the second using that $(3/\delta)^i \le \beta$, that u_l, v_r lie on a shortest path with at most 2^ℓ hops, and that $H^{(\ell-1)}$ is a $(2\beta, \epsilon_{\ell-1}, 2^\ell)$ hopset.

Consider the case that $z_r \in B(z_l)$, then we have a hopset edge (z_l, z_r) that was introduced in H_{i+1}. In particular, since we used 4β steps in the exploration from z_l, we have that

$$d^{(1)}_{H_{i+1}}(z_l, z_r) = d^{(4\beta)}_{G_{i+1}}(z_l, z_r) \overset{(14)}{\le} d^{((3/\delta)^i)}_{G\cup H_i}(z_l, u_l) + (1+\epsilon_{\ell-1}) \cdot d_G(u_l, v_r) + d^{((3/\delta)^i)}_{G\cup H_i}(v_r, z_r) \,. \tag{15}$$

Next, apply the inductive hypothesis on segments $\{L_j\}$ for $j < l$ and $j > r$, and in between use the detour via u_l, z_l, z_r, v_r. Since there are at most $1/\delta - 1$ intervals for which we use the first item in the inductive hypothesis, the total number of hops we will need is at most $(1/\delta - 1)\cdot(3/\delta)^i + 1/\delta + 2(3/\delta)^i + 1$. This is at most $(3/\delta)^{i+1}$ whenever $\delta < 1/2$. It follows that

$$\begin{aligned}
d^{((3/\delta)^{i+1})}_{G\cup H_{i+1}}(x,y) \quad &\le \quad \sum_{j=1}^{l-1}\left[d^{((3/\delta)^i)}_{G\cup H_i}(u_j,v_j) + d^{(1)}_G(v_j,u_{j+1})\right] + d^{((3/\delta)^i)}_{G\cup H_i}(u_l,z_l) + d^{(1)}_{H_{i+1}}(z_l,z_r) \\
&\qquad + d^{((3/\delta)^i)}_{G\cup H_i}(z_r,v_r) + d^{(1)}_G(v_r,u_{r+1}) + \sum_{j=r+1}^{J}\left[d^{((3/\delta)^i)}_{G\cup H_i}(u_j,v_j) + d^{(1)}_G(v_j,u_{j+1})\right] \\
&\overset{(15)}{\le} \quad (1+\epsilon_{\ell-1})\cdot\left[(1+12\delta i)\cdot d_G(x,u_l) + d_G(u_l,v_r) + (1+12\delta i)\cdot d_G(v_r,y)\right] \\
&\qquad + 2d^{((3/\delta)^i)}_{G\cup H_i}(u_l,z_l) + 2d^{((3/\delta)^i)}_{G\cup H_i}(z_r,v_r) \\
&\overset{(13)}{\le} \quad (1+\epsilon_{\ell-1})\cdot(1+12\delta i)\cdot d_G(x,y) + 12\delta\cdot d_G(x,y) \\
&\le \quad (1+\epsilon_{\ell-1})\cdot(1+12\delta(i+1))\cdot d_G(x,y) \,.
\end{aligned}$$

In the penultimate inequality we used that both $d_G(u_l, v_l), d_G(u_r, v_r) \le \delta \cdot d_G(x, y)$. This demonstrates that item 1 holds in this case.

The final case to consider is that $z_r \notin B(z_l)$ (and $z_l \in A_{i+1} \setminus A_{i+2}$). Let $z = p(z_l) \in A_{i+2}$. Since $z_r \in A_{i+1}$, the definition of $B(z_l)$ implies that

$$d^{(1)}_{H_{i+1}}(z_l, z) = \hat{d}(z_l, A_{i+2}) \le 2d^{(4\beta)}_{G_{i+1}}(z_l, z_r) \,. \tag{16}$$

(Recall that $G_{i+1} = G \cup H^{(\ell-1)} \cup H_i$.)

We now claim that item 2 holds for such a choice of z. Indeed, by (13), we have

$$3\cdot d^{(3/\delta)^i}_{G\cup H_i}(u_l,z_l) + 2\cdot d^{(3/\delta)^i}_{G\cup H_i}(v_r,z_r) \;\le\; 15\cdot d_G(x,y) \,. \tag{17}$$

Hence,

$$\begin{aligned}
d^{((3/\delta)^{i+1})}_{G\cup H_{i+1}}(x,z) \quad &\le \quad \sum_{j=1}^{l-1}\left[d^{((3/\delta)^i)}_{G\cup H_i}(u_j,v_j) + d^{(1)}_G(v_j,u_{j+1})\right] + d^{((3/\delta)^i)}_{G\cup H_i}(u_l,z_l) + d^{(1)}_{H_{i+1}}(z_l,z) \\
&\overset{(16)\wedge(14)}{\le} \quad (1+\epsilon_{\ell-1})\cdot(1+12\delta i)\cdot d_G(x,u_l) + d^{((3/\delta)^i)}_{G\cup H_i}(u_l,z_l) \\
&\qquad + 2\cdot\left[d^{((3/\delta)^i)}_{G\cup H_i}(z_l,u_l) + (1+\epsilon_{\ell-1})\cdot d_G(u_l,v_r) + d^{((3/\delta)^i)}_{G\cup H_i}(v_r,z_r)\right] \\
&\overset{(17)}{\le} \quad (1+\epsilon_{\ell-1})\cdot(1+12\delta i)\cdot d_G(x,u_l) + 15\delta\cdot d_G(x,y) + 2(1+\epsilon_{\ell-1})\cdot d_G(u_l,v_r) \\
&\le \quad 3d_G(x,y) \,,
\end{aligned}$$

where the last inequality we used that $\delta < 1/(15k)$, $k \ge 2$ and $1 + \epsilon_{\ell-1} < e^{1/5} < 5/4$, so that both $(1+\epsilon_{\ell-1})\cdot(1 + 12\delta i) + 15\delta \le 3$ and $2(1+\epsilon_{\ell-1}) + 15\delta \le 3$. □

Taking $\delta = \epsilon/(15k')$ and picking $i = k'$, the second item of Lemma 6 cannot hold for any $x, y \in V$ (because $A_{k'+1} = \emptyset$), so we have for every $x, y \in V$ such that $d_G^{(2^\ell)}(x,y) = d_G(x,y)$ that

$$d^{(\beta)}_{G \cup H^{(\ell)}}(x,y) \leq (1+\epsilon_{\ell-1}) \cdot (1+\epsilon) \cdot d_G^{(2^\ell)}(x,y) = (1+\epsilon_\ell) \cdot d_G(x,y) \; .$$

Recall that $\beta = (3/\delta)^{k'} = (45k'/\epsilon)^{k'}$. Rescaling $\epsilon' = \epsilon/\log n$ and taking $\ell = \log n$, we derive the following theorem.

Theorem 4. *For any weighted graph $G = (V, E)$ on n vertices, an integer $k > 1$, and parameters $0 < \rho < 1$, $0 < \epsilon < 1/5$, there is a distributed algorithm in the Congested Clique model running in $\tilde{O}(n^\rho \cdot \beta)$ rounds, that computes a (β, ϵ)-hopset H of size at most $O(n^{1+1/(2^k-1)})$, where*

$$\beta = O\left(\frac{(k+1/\rho) \cdot \log n}{\epsilon}\right)^{k+1/\rho+1} \; . \tag{18}$$

Remark 1. *We note that by (11) and Claim 5, the memory requirement from every vertex is $\tilde{O}(n^\rho)$. This is because the latter shows that this is a bound on the number of messages every vertex needs to send in each round, and the former indicates that whp storing $B(v)$ for any $v \in V$ requires only so much space.*

We remark that one can achieve β independent of n by either applying the construction recursively, as we do in Section 4 for the parallel implementation, or by using an idea from [EN16a]. We next describe the latter: fix a parameter t, and use the hopset $H^{(\ell)}$ to compute the hopset $H^{(\ell+t)}$; Since $H^{(\ell)}$ is also a $(2^t \cdot \beta, \epsilon, 2^{\ell+t})$-hopset, we need explorations to range $2^t \cdot \beta$ in order for an appropriate variant of (9) to hold. There will be only $(\log n)/t$ levels until $H^{(\log n)}$ is built, so we gain a factor of t in β. We derive the following result.

Theorem 5. *For any weighted graph $G = (V, E)$ on n vertices, integers $k > 1$, $t \geq 1$, and parameters $0 < \rho < 1$, $0 < \epsilon < 1/5$, there is a distributed algorithm in the Congested Clique model that runs in $\tilde{O}(n^\rho \cdot \beta \cdot 2^t/t)$ rounds, and computes H of size at most $O(n^{1+1/(2^k-1)})$, which is a (β, ϵ)-hopset, where*

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

In particular, taking $t = \rho \log n$ and rescaling $\rho' = 2\rho$, gives

Corollary 7. *For any weighted graph $G = (V, E)$ on n vertices, an integer $k > 1$, and parameters $0 < \rho < 1/2$, $0 < \epsilon < 1/5$, there is a distributed algorithm in the Congested Clique model that runs in $\tilde{O}(n^\rho \cdot \beta)$ rounds, and computes H of size at most $O(n^{1+1/(2^k-1)})$, which is a (β, ϵ)-hopset, where*

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

3.2 CONGEST Model

Given a weighted graph $G = (V, E, w)$ representing the network, in the CONGEST model we will be interested in a setting where there is a "virtual" graph $G' = (V', E', w')$ embedded in G, i.e. $V' \subseteq V$. We would like to construct a hopset for G'. It is motivated by distributed applications of hopsets for approximate shortest paths computation, distance estimation and routing [HKN14, Nan14, HKN16, LP15, EN16b, EN16a], which require a hopset for a virtual graph embedded in the underlying network in the above way. For simplicity we only consider the relevant case when $m \approx \sqrt{n}$, and the edges of G' correspond to $\tilde{O}(\sqrt{n})$-bounded distances in G.

The following lemma provides a way to perform Bellman-Ford explorations in a small-arboricity hopset with small memory.

Lemma 8. *Let $G'' = (V', E' \cup H)$ be a virtual graph on m vertices embedded in a graph $G = (V, E)$ of hop-diameter D, such that edges in E' correspond to B-bounded distances in G, and H has arboricity α (i.e., one can orient the edges of H to have out-degree at most α). Then one can compute distances from a given root vertex (or set), that are at most those given by β iterations of Bellman-Ford in G'' (and no less than those in G) in the CONGEST model, within $O((m \cdot \alpha + B + D) \cdot \beta \cdot \log n)$ rounds, so that every vertex requires only $O(\alpha + \log n)$ memory.*

Proof. To implement a single iteration of the Bellman-Ford exploration, every vertex $v \in V'$ which holds a current distance estimate will need to communicate it to its neighbors in G''. First it will initiate an exploration in G for B rounds. In each round, every vertex $u \in V$ will forward the smallest value it received so far. This guarantees that if $\{v, w\} \in E'$, then w will receive v's message (or a smaller value).

We now have to handle the edges of H, and may assume every $v \in V'$ knows of at most α such edges touching it. Let T be a spanning tree of G with hop-depth D. Every $v \in V'$ will broadcast via T its value to the entire graph, and will also send all the existing edges of H touching it that v knows about. All vertices $w \in V'$ that know of a hopset edge $\{v, w\}$ (or that learn about it from v's message) will update their value accordingly. Since there are $O(m \cdot \alpha)$ messages, this can be done in $O(m \cdot \alpha + D)$ rounds. In order to guarantee small internal memory, each v selects at random a number from $\{1, 2, \ldots, m \cdot \alpha\}$ for each message it sends, as a round to start its broadcast (clearly this increases the number of rounds by at most $m \cdot \alpha$); Since each message of v will reach every vertex of T at most once, the probability that some $u \in V$ receives t messages in a single round is at most $\binom{m \cdot \alpha}{t} \cdot 1/(m \cdot \alpha)^t \leq (e/t)^t$, thus with high probability no vertex will receive more than $O(\log n)$ messages each round. By increasing the number of rounds by a factor of $O(\log n)$, whp there will be no congestion. The total number of rounds required is thus $O(m \cdot \alpha + B + D) \cdot \beta \cdot \log n$. □

We now show how to use Lemma 8 to construct a hopset for G' (without computing G' explicitly). Recall that in the i-th step of constructing $H = H^{(\ell)}$ we already built the previous hopset $H^{(\ell-1)}$ and the partial hopset H_{i-1}. Since we aim at using limited memory, every vertex v stores only the "outgoing" hopset edges, those to vertices in its bunch $B(v)$. Recall that by (11) whp $|B(v)| \leq O(m^\rho \cdot \log n)$, for all $v \in V'$.

We work in the graph $G_i = G' \cup H^{(\ell-1)} \cup H_{i-1}$. In order to implement the $O(\beta)$-bounded exploration rooted at A_{i+1} we simply apply Lemma 8 on G_i with $\alpha = O(m^\rho \cdot \log n)$ (recall that $m \approx \sqrt{n}$ and $B = \tilde{O}(m)$). The explorations from vertices of $A_i \setminus A_{i+1}$ are done in a similar manner. However, there is larger congestion due to the multiple sources of limited explorations. Recall that in the limited exploration whose origin is $v \in A_i \setminus A_{i+1}$, each intermediate node $x \in V'$ forwards the message iff its current estimation is strictly less than $\hat{d}(v, A_{i+1})/2$ (this value is part of the message v sends). We enforce the exact same rule for vertices $u \in V$ as well. If a message concerning v should pass in G' from x to its neighbor y, then all vertices on the B-bounded path in G that implements the edge $(x, y) \in E'$ will have estimates smaller than that of y, therefore will forward the message on. In the proof of Claim 5 we saw that each $x \in V'$ participates whp in at most $O(m^\rho \cdot \log n)$ explorations for each i. The argument is identical for $u \in V$ as well, so the congestion induced in the first stage of Lemma 8 (the exploration in G for B rounds) by multiple sources is only $O(m^\rho \cdot \log n)$. Note that the second phase (broadcasting the edges of H) the number of messages increases to $O(m \cdot \alpha + m \cdot m^\rho \cdot \log n)$. Thus, the total number of rounds required is still $\tilde{O}(m^{1+\rho} + D) \cdot \beta$. We summarize the discussion with the following result.

Theorem 6. *For any weighted graph $G = (V, E)$ on n vertices with hop-diameter D, an integer $k > 1$, and parameters $0 < \rho < 1$, $0 < \epsilon < 1/5$, and a virtual graph $G' = (V', E')$ embedded in G on $|V'| = \Theta(\sqrt{n})$ vertices, where E' corresponds to $B = \tilde{O}(\sqrt{n})$-bounded distances in G. Then there is a distributed algorithm in the CONGEST model that runs in $\tilde{O}(n^{(1+\rho)/2} + D) \cdot \beta$ rounds,[8] that computes H, which is a (β, ϵ)-hopset for G', of size at most $O(n^{(1+1/(2^k-1))/2})$, where*

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

and both the arboricity of H and the internal memory per vertex are bounded by $\tilde{O}(n^{\rho/2})$.

3.2.1 Compact Path-Recovery Mechanism

Every hopset edge is implemented via some path in G. In order for our hopset to be useful in settings when one wishes to report shortest paths, or for compact routing schemes, we need to be able to efficiently recover those paths in G that implement the hopset edges. It was shown in [EN16a] how to adapt the Bellman-Ford exploration so that paths information can be stored as well, so every vertex knows about all the hopset edges whose paths contain it. This feature

[8] The stated number of rounds is in fact for the CONGEST RAM model; in the CONGEST model one needs to multiply the bound by an $O(\log_n \log \Lambda)$ factor.

increased the size of messages by a factor of $O(\beta)$. However, this approach provides no guarantee on the number of hopset edges a vertex $u \in V$ can be a part of, which can be devastating when one desires small memory per vertex. We describe now an approach that eliminates the need for the message's size increase, and also ensures that each vertex needs to store a small amount of information. The drawback is that the vertices no longer know about al the hopset edges they implement, but we require a protocol for efficient reconstruction of the paths. Bearing in mind the current applications of hopsets to shortest paths and routing schemes, our path recovery mechanism enables the vertices on paths implementing hopset edges to compute distances to a certain set of roots (which correspond to the sources of shortest path computation, or to the roots of trees in which routing in conducted).

Definition 1. *Given a graph $G = (V, E)$ with hop diameter D, a hopset H has a* path-recovery mechanism*, if every edge $e \in H$ corresponds to a path $P(e)$ with $\omega(P(e)) = \omega_H(e)$ in G, and also the following property holds. Suppose we are given a collection of root vertices $R \subseteq V$, so that each hopset edge $e = (x, y)$ is associated with a subset of them $R(e) \subseteq R$, and has approximate distance $\hat{d}(x, z)$ from x to each $z \in R(e)$ (and thus also from y). Then there exists a* protocol *that enables one to inform each $v \in V$ on $R(v) = \bigcup_{e\,:\,v\in P(e)} R(e)$, and also to compute the approximate distances to all roots in $R(v)$. The approximate distance $\hat{d}(v, z)$ to $z \in R(v)$ is bounded by $d_{P(e)}(v, x) + \hat{d}(x, z)$. In addition, v will know of a* parent*, a neighbor in some path $P(e)$, so that $v \in P(e)$, implementing $\hat{d}(v, z)$. The running time of this protocol is $\tilde{O}(|H| \cdot C + D) \cdot \beta$, where $C = \max_{v\in V} |R(v)|$, and the required memory per vertex is proportional to that required by the hopset construction.*

During the construction of our hopset, for each vertex $u \in V'$ and for every $O(\beta)$-depth Bellman-Ford exploration originated at $v \in V'$ that reached u (and that u forwarded it to its neighbors), u stores the parent $p_v(u)$ (the last vertex which updated the distance u has from v), and the distance to v along the path the message traveled. The vertices of $V \setminus V'$ also store parent and distance information for every virtual edge in E' that was used in the exploration rooted at v. Consider level $1 \leq \ell \leq \log n$ and step $0 \leq i \leq k'$. At the construction of H_i, u forwards at most 1 message originated in A_{i+1}, and by Claim 5 at most $O(m^\rho \cdot \log n)$ messages from $v \in A_i \setminus A_{i+1}$. Hence the memory required from any vertex to store this information is only $\tilde{O}(m^\rho)$.

Path-Recovery Protocol. In order to recover every vertex on the path $P(e)$, for all hopset edges $e \in H$, and compute their distances to the appropriate roots, we run the following protocol, which essentially reconstructs the hopset edges from the top down. Recall that every hopset edge is created as an $O(\beta)$ bounded path in a graph consisting of virtual edges E', and hopset edges of previous levels, and every vertex stored its parent on this path.

Fix some $1 \leq \ell \leq \log n$ and $0 \leq i \leq k'$, and assume we already handled hopset edges in levels ℓ' and i' with $\ell' > \ell$, or $\ell' = \ell$ and $i' > i$. For each hopset edge $e = (x, y) \in H_i^{(\ell)}$, let $Q(e)$ be the set of $O(\beta)$ edges of $E' \cup H^{(\ell-1)} \cup H_{i-1}$ that comprise it. Since every vertex u of these $O(\beta)$ edges stored its parent and $d_{P(e)}(u, x)$ (the distance to x on this path), we can in $O(\beta)$ broadcast phases inform all of them on $R(e)$ and the distances $\{\hat{d}(x, z)\}_{z\in R(e)}$. Each vertex u can infer $\hat{d}(u, z) = d_{P(e)}(u, x) + \hat{d}(x, z)$ (assuming x is the closer vertex to z, otherwise, given $\omega_H(e)$, u can compute the distance to z via y as well). Each hopset edge e' of $Q(e)$ will append $R(e)$ to its set of roots $R(e')$. So in each broadcast phase we have at most $|H|$ hopset edges generating messages of size at most $O(C)$ (since every vertex, and thus every hopset edge e, has $|R(e)| \leq C$), so this can be executed in $O(|H| \cdot C + D)$ rounds. The total number of rounds is thus $O((|H| \cdot C + D) \cdot \beta \cdot \log n \cdot k')$, since there are $O(\beta)$ such phases, and we do this for all $1 \leq \ell \leq \log n$ and $0 \leq i \leq k'$.

When these phases conclude, every vertex $u \in V'$ knows about $R(u)$ and the corresponding distances. In order to propagate this information to vertices in V, and to compute parents, we execute $O(B \cdot C) = \tilde{O}(\sqrt{n} \cdot C) \leq \tilde{O}(|H| \cdot C)$ additional rounds in which every virtual edge $e' = (x', y') \in E'$ informs $R(e')$ and the approximate distance $\hat{d}(x', z)$ for $z \in R(e')$, to the vertices on the B-bounded path in G that implements e'. Since each of those vertices stored its parent, once again this could be done in B phases, each phase we send $O(C)$ messages per vertex ($O(1)$ messages for each root). In addition, every vertex $u \in V$ can infer for every $z \in R(u)$ the approximate distance to z as above, and also its parent on the path implementing this distance. If a vertex happens to receive multiple messages regarding the same root z, it stores only the one with shortest distance.

We summarize with the following theorem.

Theorem 7. *For any weighted graph $G = (V, E)$ with hop-diameter D, an integer $k > 1$, and parameters $0 < \rho < 1$, $0 < \epsilon < 1/5$, and a virtual graph $G' = (V', E')$ embedded in G on $|V'| = \Theta(\sqrt{n})$ vertices, where E' corresponds to $B = \tilde{O}(\sqrt{n})$-bounded distances in G. Then there is a distributed algorithm in the CONGEST model that runs in $\tilde{O}(n^{(1+\rho)/2} + D) \cdot \beta$ rounds, that computes a (β, ϵ)-hopset for G' with a path-recovery mechanism, of size at most $O(n^{(1+1/(2^k-1))/2})$, where*

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

and both the arboricity of H and the internal memory per vertex are bounded by $\tilde{O}(n^{\rho/2})$.

4 PRAM Model

The algorithm described in Section 3.1 can be easily adapted to the PRAM model. For each $\ell = 1, 2, \ldots, \log n$ we build the hopset $H^{(\ell)}$ based on the previous hopset $H^{(\ell-1)}$. Each of the $O(\beta)$-bounded Bellman-Ford explorations for constructing H_i can be implemented in parallel time $O(\beta)$, where the congestion of $\tilde{O}(n^{\rho})$ per vertex translates to extra work (rather than multiply the number of rounds, as was the case in the distributed models). Since there are $\log n$ values of ℓ, and $k' \leq k + 1/\rho + 1$ steps in each level, the total time is only $O((k + 1/\rho) \cdot \log n \cdot \beta)$. We have the following result.

Theorem 8. *For any weighted graph $G = (V, E)$ on n vertices, an integer $k > 1$, and parameters $0 < \rho < 1$, $0 < \epsilon < 1/5$, there is a parallel algorithm running in time $O((k + 1/\rho) \cdot \log n \cdot \beta)$ and $\tilde{O}(|E| \cdot n^{\rho})$ work, that computes H of size at most $O(n^{1+1/(2^k-1)})$, which is a (β, ϵ)-hopset, where*

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

We can also apply the construction recursively: If $H(1)$ is the hopset given by Theorem 8 with β_1 as in (19), then apply the construction on the graph $G \cup H(1)$, but only for levels ℓ up to $\ell_2 = \log \beta_1$, to obtain a hopset $H(2)$. Since for any $x, y \in V$ we have $d^{(\beta_1)}_{G \cup H(1)}(x, y) \leq (1 + \epsilon) d_G(x, y)$, then adding both $H(1)$ and $H(2)$ guarantees $d^{(\beta_2)}_{G \cup H(1) \cup H(2)}(x, y) \leq (1 + \epsilon)^2 d_G(x, y)$, where $\beta_2 = \left(\frac{3c \cdot (k+1/\rho) \cdot \log \beta_1}{\epsilon}\right)^{k+1/\rho+1}$, where c is the constant hidden by the $O(\cdot)$ notation in (19). This bound follows because ϵ needs to be rescaled by $3\ell_2 = 3 \log \beta_1$; the rescaling by $\log \beta_1$ is to compensate for the number of levels, and by 3 to reduce the error from $(1 + \epsilon)^2$ back to $1 + \epsilon$. Continuing in this manner for the next level with $\ell_3 = \log \beta_2$ levels, we obtain in general a recursion for $\beta_{i+1} = \left(\frac{3c \cdot (k+1/\rho) \cdot \log \beta_i}{\epsilon}\right)^{k+1/\rho+1}$, and it can be shown by induction that as long as $\log^{(i)} n \geq 3c \log(k + 1/\rho)$ we have

$$\beta_i \leq \left(\frac{8c \cdot (k + 1/\rho)^2 \cdot \left\lceil \log(3c(k + 1/\rho)/\epsilon) + \log^{(i)} n \right\rceil}{\epsilon}\right)^{k+1/\rho+1}.$$

After at most $t = \log^* n$ iterations, we get that $\beta_t = O\left(\frac{(k+1/\rho)^2}{\epsilon}\right)^{(1+o(1)) \cdot (k+1/\rho)}$. To summarize, this yields a hopset with constant parameter β that is computed in time $\mathrm{polylog}(n)$.

Theorem 9. *For any weighted graph $G = (V, E)$ on n vertices, an integer $k > 1$, and parameters $0 < \rho < 1$, $0 < \epsilon < 1/5$, there is a parallel algorithm running in time $O(\left(\frac{(k+1/\rho) \cdot \log n}{\epsilon}\right)^{k+1/\rho+2})$ and $\tilde{O}(|E| \cdot n^{\rho})$ work, that computes H of size at most $O(n^{1+1/(2^k-1)} \cdot \log^* n)$, which is a (β, ϵ)-hopset, where*

$$\beta = O\left(\frac{(k + 1/\rho)^2}{\epsilon}\right)^{(1+o(1)) \cdot (k+1/\rho)}. \tag{20}$$

5 Approximate Shortest Paths

5.1 CONGSET Model

In several papers [Nan14, HKN16, EN16a], it was shown how to use hopsets to distributively compute $(1+\epsilon)$-approximate shortest paths from a single source $z \in V$, or from multiple sources. However, all these results have running time $(\sqrt{n}+D)\cdot(1/\epsilon)^{\tilde{O}(\sqrt{\log n})}\cdot\log\Lambda$, where Λ is the aspect ratio of the input graph. Here we eliminate the dependence on the aspect ratio. In addition, every vertex needs only a small memory during the computation. Since the derivation of shortest paths from hopsets already appeared in several places, we only give a brief sketch.

Theorem 10. *For any weighted graph $G=(V,E)$ with hop-diameter D, a source vertex $z\in V$, and a parameter $0<\epsilon<1/5$, there is a distributed algorithm in the CONGEST model that runs in $(\sqrt{n}+D)\cdot(1/\epsilon)^{\tilde{O}(\sqrt{\log n})}$ rounds, and computes $(1+\epsilon)$-approximate shortest paths from z to every $v\in V$. Furthermore, every vertex requires $2^{\tilde{O}(\sqrt{\log n})}$ memory.*

Proof. Sample each vertex to V' independently with probability $1/\sqrt{n}$, and apply Theorem 6 to compute a (β,ϵ)-hopset H for $G'=(V',E',w')$, where the weights w' corresponds to $B=4\sqrt{n}\ln n$ bounded distances in G. The parameters we choose are $k=1/\rho=\sqrt{\log n}$, so $\beta=(1/\epsilon)^{\tilde{O}(\sqrt{\log n})}$. Next, run β iterations of Bellman-Ford in G' rooted in the source z. As described in the broadcast algorithm of Lemma 8, each message of $v\in V'$ will be sent for B rounds in G (to implement the edges of G'), and also broadcast to the entire graph via a BFS tree of G (to implement the hopset edges H). Since whp there are $O(\sqrt{n})$ vertices in V', it will take $O(\sqrt{n}+D)\cdot\beta$ rounds to implement this (intermediate vertices in G forward the smallest estimate, if they receive more than one, so there is no congestion). Now every vertex $v\in V'$ has a value $b_z(v)$ which is a $1+\epsilon$ approximation to $d_G(z,v)$. Initiate another B rounds of Bellman-Ford in G, each $v\in V'$ starts off with $b_z(v)$ and $u\in V\setminus V'$ starts off with ∞. It can be checked that whp each shortest path in G has a vertex of V' in its first B hops. Thus for any $u\in V$, there is a $v\in V'$ at most B hops away on the shortest path from u to z. After the B iterations conclude, u will receive from this v its $1+\epsilon$ approximation to $d_G(z,u)$. □

By selecting a smaller $\rho=\mu\log\log n/\log n$ for some small enough constant $\mu>0$, we obtain the following corollary.

Corollary 9. *For any weighted graph $G=(V,E)$ with hop-diameter D, a source vertex $z\in V$, and parameters $1/\log^{O(1)} n<\epsilon<1/5$ and $\mu>0$, there is a distributed algorithm in the CONGEST model that runs in $(\sqrt{n}+D)\cdot n^{O(\mu)}$ rounds, and computes $(1+\epsilon)$-approximate shortest paths from z to every $v\in V$. Furthermore, every vertex requires $\log^{O(1/\mu)} n$ memory.*

Multi-source Approximate Shortest Paths. In this setting we are given a set $S\subseteq V$ of size $|S|=s$, and would like to approximately compute shortest path for all pairs in $S\times V$. Using a similar approach (but picking each vertex into V' with probability $1/\sqrt{sn}$), we obtain the following where s is large (see [EN16a] for details, and other regimes of s).

Theorem 11. *For any weighted graph $G=(V,E)$ with hop-diameter D, a set $S\subseteq V$ of size $|S|=n^{\Omega(1)}$, and a parameter $0<\epsilon<1/5$, there is a distributed algorithm in the CONGEST model that runs in $(\sqrt{sn}+D)\cdot(1/\epsilon)^{O(1)}$ rounds, and computes $(1+\epsilon)$-approximate shortest paths for all pairs in $S\times V$.*

5.2 PRAM Approximate Shortest Paths with Preprocessing

We consider the following variant of s-SSP in the PRAM model, where the sources S arrive in an online manner. We are allowed to first preprocess the graph, and then, for each query $u\in S$ that arrives, we would like to compute $1+\epsilon$ approximation to all shortest paths rooted at u in *constant* parallel time.

Fix a constant $k\geq 1$, that will determine the tradeoff between time and work. The algorithm starts with computing the hopset H from Theorem 9 with parameter $\rho=1/k$. The preprocessing time (i.e., the time required to construct

the hopset) is $\left(\frac{\log n}{\epsilon}\right)^{O(k)}$, and the preprocessing work is $\tilde{O}(|E| \cdot n^{1/k})$. This hopset has hopbound $\beta = \left(\frac{k}{\epsilon}\right)^{O(k)}$, i.e., constant as long as k and ϵ are.

Given a source u, we compute a $(1+\epsilon)$-approximate $u \times V$ shortest paths in G, by running β iterations of Bellman-Ford in $G \cup H$, starting from u. We allocate $O(deg_{G \cup H}(v) \cdot n^{1/k})$ processors to every vertex v. Kucera [Kuc81] and Shiloach and Vishkin [SV81] (see also Chaudhuri's book [Cha92], Ch. 5.2.1) showed that using $O(N^{1+1/k})$ processors, one can find minimum of N values in $O(k)$ CRCW PRAM time.

One iteration of B-F amounts to computing the minimum of $deg_{G \cup H}(v)$ values, in each vertex $v \in V$. Hence, we implement β iterations of B-F in $O(k \cdot \beta) = O(1)$ parallel time. The overall number of processors used for this computation is

$$O(n^{1/k}) \sum_{v \in V} deg_{G \cup H}(v) \;=\; O(n^{1/k} \cdot (|E| + n^{1+1/(2^k-1)} \log^* n)) \; .$$

We remark that when constructing the hopset, each vertex can keep track of all hopset edges $e \in H$ whose path $P(e)$ in G contain it. This means that the actual approximate paths are reported.

Theorem 12. ***(PRAM ASPP)*** *For constant parameters $k \geq 1$ and $\epsilon > 0$, using preprocessing that can be implemented in $\left(\frac{\log n}{\epsilon}\right)^{O(k)}$ time and $\tilde{O}(|E|n^{1/k})$ work, one can then compute $(1+\epsilon)$-approximate shortest paths $u \times V$ from any source $u \in V$ in constant query time (specifically, $\left(\frac{k}{\epsilon}\right)^{O(k)}$), and $O((|E| + n^{1+1/(2^k-1)} \log^* n) \cdot n^{1/k})$ work, in the CRCW PRAM model. (In EREW PRAM model, the query time of this algorithm is $O(\beta \log n) = O(\log n)$.)*

6 CONGEST RAM Model

Here we briefly discuss the assumption that edge weights can be sent in a single message in distributed models of computation. Recall that in the CONGEST RAM model, every vertex v is allowed to send $O(1)$ edge weights or Identity numbers over each edge $e = (v, u)$ incident on v, *regardless* of the length of bit representations of these edge weights.

In the sorting literature, for example, one distinguishes between algorithms that work in the RAM model, and those that manipulate with bits of input elements (see [HT02, Han04], and the references therein). This is also the case in the context of algorithms for maximum flow, where the state-of-the-art algorithm with running time that does not depend on edge capacities requires $O(mn)$ time [Orl13], while the state-of-the-art algorithm whose running time depends on edge capacities requires $O(m^{10/7}U^{1/7})$ time [Mad16], where U is the ratio between maximum and minimum edge capacities. See also the recent work of Cohen et al. [CMSV17] about centralized shortest paths computations with negative weights; in this context too the literature separates between algorithms with running times that do and do not depend on edge weights.

An argument that can be made against our CONGEST RAM model is that in "real" networks, the bandwidth is bounded by a fixed threshold value T, and therefore the running time has to be a function of Λ. Note, however, that precisely the same argument can be made against the RAM model, as "real" computer words are of size at most some T, and so the centralized running time also has to be a function of Λ.

We also show that in many settings where a small error $1 + \delta$ is allowed, one can indicate an edge weight using only $O(\log \log \Lambda + \log(1/\delta))$ bits. To see this, partition the range of possible weights, w.l.o.g. $[1, \Lambda]$, into intervals $I_i = [(1+\delta)^{i-1}, (1+\delta)^i]$, and note that there are $\left\lceil \frac{\log \Lambda}{\delta} \right\rceil$ possible values of i. To indicate an edge weight w up to a $1 + \delta$ error, it suffices to indicate for which i we have $w \in I_i$, which requires $O(\log(\log \Lambda/\delta))$ bits. We observe that for computing hopsets with error $1 + O(\epsilon)$, we can suffer $1 + \epsilon$ error in edge weights, so the bit complexity of messages is only $O(\log \log \Lambda + \log(1/\epsilon))$. Assuming Λ is exponential in $\mathrm{poly}(n)$, the messages can be encoded with $O(\log n)$ bits. Generally, when messages are of size $O(\log n)$, the overhead is $O(\log_n \log \Lambda)$.

References

[ABCP93] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Near-linear cost sequential and distribured constructions of sparse neighborhood covers. In *34th Annual Symposium on Foundations of Computer Science, Palo Alto, California, USA, 3-5 November 1993*, pages 638–647, 1993.

[ABP17] Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 568–576, 2017.

[Ber09] Aaron Bernstein. Fully dynamic (2 + epsilon) approximate all-pairs shortest paths with fast query and close to linear update time. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 693–702, 2009.

[BKKL17] Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 7:1–7:16, 2017.

[Cha92] Pranay Chaudhuri. *Parallel algorithms - designs and analysis*. Advances in computer science series. Prentice Hall, 1992.

[CMSV17] Michael B. Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in $\tilde{o}\,(m^{10/7} \log W)$ time (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 752–771, 2017.

[Coh93] Edith Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. In *34th Annual Symposium on Foundations of Computer Science, Palo Alto, California, USA, 3-5 November 1993*, pages 648–658, 1993.

[Coh97] Edith Cohen. Using selective path-doubling for parallel shortest-path computations. *J. Algorithms*, 22(1):30–56, 1997.

[Coh00] Edith Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *J. ACM*, 47(1):132–166, 2000.

[Elk01] M. Elkin. Computing almost shortest paths. In *Proc. 20th ACM Symp. on Principles of Distributed Computing*, pages 53–62, 2001.

[Elk17] Michael Elkin. Distributed exact shortest paths in sublinear time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 757–770, New York, NY, USA, 2017. ACM.

[EN16a] Michael Elkin and Ofer Neiman. Hopsets with constant hopbound, and applications to approximate shortest paths. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 128–137, 2016.

[EN16b] Michael Elkin and Ofer 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, pages 235–244, New York, NY, USA, 2016. ACM.

[EN17] Michael Elkin and Ofer Neiman. Linear-size hopsets with small hopbound, and distributed routing with low memory. *CoRR*, abs/1704.08468, 2017.

[EN18] Michael Elkin and Ofer Neiman. Near-optimal distributed routing with low memory. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 207–216, 2018.

[EP04] Michael Elkin and David Peleg. (1+epsilon, beta)-spanner constructions for general graphs. *SIAM J. Comput.*, 33(3):608–631, 2004.

[EZ06] Michael Elkin and Jian Zhang. Efficient algorithms for constructing (1+epsilon, beta)-spanners in the distributed and streaming models. *Distributed Computing*, 18(5):375–385, 2006.

[FL16] Stephan Friedrichs and Christoph Lenzen. Parallel metric tree embedding based on an algebraic view on moore-bellman-ford. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '16, pages 455–466, New York, NY, USA, 2016. ACM.

[Han04] Yijie Han. Deterministic sorting in o($n$loglog$n$) time and linear space. *J. Algorithms*, 50(1):96–105, 2004.

[HKN14] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 146–155, 2014.

[HKN16] Monika Henzinger, Sebastian Krinninger, and Danupon 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, pages 489–498, New York, NY, USA, 2016. ACM.

[HP17] Shang-En Huang and Seth Pettie. Thorup-zwick emulators are universally optimal hopsets. *CoRR*, abs/1705.00327, 2017.

[HT02] Yijie Han and Mikkel Thorup. Integer sorting in 0(n sqrt (log log n)) expected time and linear space. In *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*, pages 135–144, 2002.

[KS97] Philip N. Klein and Sairam Subramanian. A randomized parallel algorithm for single-source shortest paths. *J. Algorithms*, 25(2):205–220, 1997.

[Kuc81] L. Kucera. Parallel computation and conflicts in memory access. *Inform. Process. Lett.*, 14:93–96, 1981.

[LP15] Christoph Lenzen and Boaz 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, July 21 - 23, 2015*, pages 153–162, 2015.

[LPP16] Christoph Lenzen, Boaz Patt-Shamir, and David Peleg. Distributed distance computation and routing with small messages, 2016. manuscript.

[Mad16] Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 593–602, 2016.

[MPVX15] Gary L. Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. Improved parallel algorithms for spanners and hopsets. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '15, pages 192–201, New York, NY, USA, 2015. ACM.

[Nan14] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 565–573, 2014.

[Orl13] James B. Orlin. Max flows in o(nm) time, or better. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 765–774, 2013.

[SS99] Hanmao Shi and Thomas H. Spencer. Time-work tradeoffs of the single-source shortest paths problem. *J. Algorithms*, 30(1):19–32, 1999.

[SV81] Yossi Shiloach and Uzi Vishkin. Finding the maximum, merging, and sorting in a parallel computation model. *J. Algorithms*, 2(1):88–102, 1981.

[TZ01] M. Thorup and U. Zwick. Approximate distance oracles. In *Proc. of the 33rd ACM Symp. on Theory of Computing*, pages 183–192, 2001.

[TZ06] M. Thorup and U. Zwick. Spanners and emulators with sublinear distance errors. In *Proc. of Symp. on Discr. Algorithms*, pages 802–809, 2006.

[UY91] Jeffrey D. Ullman and Mihalis Yannakakis. High-probability parallel transitive-closure algorithms. *SIAM J. Comput.*, 20(1):100–125, 1991.