Distributed Exact Shortest Paths in Sublinear Time

MICHAEL ELKIN, Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

The distributed single-source shortest paths problem is one of the most fundamental and central problems in the message-passing distributed computing. Classical Bellman-Ford algorithm solves it in $O(n)$ time, where n is the number of vertices in the input graph G. Peleg and Rubinovich [49] showed a lower bound of $\tilde{\Omega}(D + \sqrt{n})$ for this problem, where D is the hop-diameter of G.

Whether or not this problem can be solved in $o(n)$ time when D is relatively small is a major open question. Despite intensive research [10, 17, 33, 41, 45] that yielded near-optimal algorithms for the *approximate* variant of this problem, no progress was reported for the original problem.

In this article, we answer this question in the affirmative. We devise an algorithm that requires $O((n \log n)^{5/6})$ time, for $D = O(\sqrt{n \log n})$, and $O(D^{1/3} \cdot (n \log n)^{2/3})$ time, for larger D. This running time is sublinear in n in almost the entire range of parameters, specifically, for $D = o(n/\log^2 n)$.

We also generalize our result in two directions. One is when edges have bandwidth $b \geq 1$, and the other is the s-sources shortest paths problem. For both problems, our algorithm provides bounds that improve upon the previous state-of-the-art in almost the entire range of parameters. In particular, we provide an all-pairs shortest paths algorithm that requires $O(n^{5/3} \cdot \log^{2/3} n)$ time, even for $b = 1$, for all values of D.

We also devise the first algorithm with non-trivial complexity guarantees for computing exact shortest paths in the *multipass semi-streaming* model of computation.

From the technical viewpoint, our distributed algorithm computes a hopset G'' of a skeleton graph G' of G *without first computing* G' itself. We then conduct a Bellman-Ford exploration in $G' \cup G''$, while computing the required edges of G' *on the fly*. As a result, our distributed algorithm computes *exactly* those edges of G' that it really needs, rather than computing approximately the entire G'.

CCS Concepts: • **Theory of computation → Distributed algorithms**;

Additional Key Words and Phrases: Distributed graph algorithms, shortest paths tree

ACM Reference format:
Michael Elkin. 2020. Distributed Exact Shortest Paths in Sublinear Time. *J. ACM* 67, 3, Article 15 (May 2020), 36 pages.
https://doi.org/10.1145/3387161

1 INTRODUCTION

1.1 Single-Source Shortest Paths

We study the *distributed single-source shortest paths* (henceforth, SSSP) problem in the *CONGEST* model of distributed computing. In this model, a communication network is modeled by a weighted

This research was supported by the ISF Grants No. (724/15) and (2344/19).
Authors' address: M. Elkin, Department of Computer Science, Ben-Gurion University of the Negev, P.O.B. 653, Beer-Sheva, Israel, 84105; email: elkinm@cs.bgu.ac.il.

0004-5411/2020/05-ART15 $15.00
https://doi.org/10.1145/3387161

undirected n-vertex graph $G = (V, E, \omega)$, $\omega(e) \geq 0$ for every edge $e \in E$, whose vertices host autonomous processors. The processors have distinct identity numbers (shortly, *Id*s), typically[1] from the range $\{1, \ldots, n\}$. The processors communicate via edges of the graph in synchronous rounds. In every round every vertex v is allowed to send *short* messages to its neighbors. A message sent at the beginning of a round, arrives by the end of the same round. By "short" one typically means $O(\log n)$ bits; alternatively, and somewhat more generally, one can also allow a message to contain up to $O(1)$ edge weights or/and vertex *Id*s. In a yet more general *CONGEST*$(b \log n)$ model, for an integer parameter $b \geq 1$, one can deliver $O(b \log n)$ bits in each message, or more generally, $O(b)$ edges weights and/or vertex *Id*s. When $b = 1$, we write *CONGEST* for *CONGEST*$(\log n)$.

The running time of an algorithm in this model is the (worst-case) number of rounds of distributed computation that it requires. At the beginning of an algorithm every vertex v knows its *Id* number and the weights of edges incident on it. By the end of the algorithm, v needs to know its *exact* distance to a designated source vertex r, which is given as a part of the problem's input.[2] In the closely related *shortest path tree* (SPT) problem, v also needs to know the identity of its parent $p(v)$ in the tree, and which edges (v, u) among those incident on it belong to the SPT.

The distributed SSSP problem is among the most central, extensively studied, and fundamentally important problems in this area. The classical Bellman-Ford algorithm [8, 25, 44] has running time $O(n)$. There are instances, such as a weighted n-cycle C_n, in which the problem requires $\Omega(n)$ time, and thus the problem is said to be a *global* one, i.e., a problem for which one may need to traverse the entire network to solve it. (The notion of "global" problem is due to Garay et al. [30].) Moreover, Peleg and Rubinovich [49] showed that the problem (as well as the MST problem) requires $\tilde{\Omega}(D + \sqrt{n}/b)$ time; this was later improved to $\tilde{\Omega}(D + \sqrt{n/b})$ in Reference [14], where D is the *hop-diameter* of G, i.e., the maximum unweighted distance between a pair of vertices u and v in G.

The results of References [8, 14, 25, 44, 49] left open the gap between the upper bound of $O(n)$ and the lower bound of $\tilde{\Omega}(D + \sqrt{n/b})$. Remarkably, the gap persists (and is actually even wider) if one restricts their attention to general graphs with small diameter $D = O(1)$. In this case the lower bound is $\tilde{\Omega}(D + n^{1/2-O(1/D)})$ [14, 42]. To the best of our knowledge, only for the case $D = 1$ (the so-called Congested Clique model) sublinear in n bounds are known [12].

Addressing this gap is one of the most central open problems in this area. The author heard this question for the first time from Rubinovich some fifteen years ago; consequently, it was raised in the author's survey [15]. (In the quotation below $B = b \log n$.)

What is the complexity of the shortest path tree problem in the message-passing model with small bandwidth parameter B (i.e., $B = \log n$)?

This question was recently raised again in the Open Problems section of Nanongkai's paper [45].

Another question that should be very interesting is understanding the exact case:
Problem I.2 Can we solve SSSP exactly *in sublinear time?*

(The emphasis on "exactly" is in Reference [45].) He then motivates the problem by saying:

In some settings, an exact algorithm for computing shortest paths is crucial; e.g., some Internet protocols such as OSPF and IS-IS use edge weights to control the traffic and using approximate shortest paths is unacceptable.

[1] They can be as well from a larger range. In fact, our algorithm does not require any assumption about the range of IDs, because distinct IDs from the range $\{1, \ldots, n\}$ can be easily computed from scratch in $O(D)$ time in our model.
[2] Vertices do not have to know r at the beginning of the computation. It is sufficient that every vertex knows if it is the source r or not.

The same question was raised again in the Conclusions and Open Problems section of Reference [33]. They wrote (the emphasis on "approximately" and "exactly" is in Reference [33]):
Finally, while our paper essentially settles the running time for computing single-source shortest paths approximately, the best running time for solving this problem exactly is still linear (by the Bellman-Ford algorithm). Whether there is a sublinear algorithm is a major open problem. In fact, in the past few years we have much better understood how to approximately *solve basic graph problems, such as minimum cut, single-source shortest paths, all-pairs shortest paths, and maximum flows, on distributed networks (e.g. [NS14, GK13, GKK+15]). However, when it comes to solving these problems* exactly*, almost nothing is known. Understanding the complexity of exact algorithms is an important open problem.*

Lack of progress on this problem led researchers to consider its relaxed version, in which one is interested in *approximate* distances from a designated source r, rather than in the *exact* ones. In STOC'13, Lenzen and Patt-Shamir [41] devised an $O(k \log k)$-approximate algorithm for SSSP with running time $\tilde{O}(D + n^{1/2+1/k})$. In STOC'14, Nanongkai came up with a $(1 + o(1))$-approximate algorithm that requires $\tilde{O}(n^{1/2}D^{1/4} + D)$ time. In STOC'16, Henzinger et al. [33] improved this bound further to $\tilde{O}(n^{1/2+o(1)} + D)$ time. Finally, very recently, Becker et al. [10] devised a $(1 + \epsilon)$-approximate algorithm with running time $\tilde{O}(\epsilon^{-O(1)}(D + \sqrt{n}))$. We note that a lower bound of $\tilde{\Omega}(D + \sqrt{n/b})$ [14, 50] applies to approximate variants of the problem as well.

Nevertheless, despite this intensive research [10, 14, 33, 41, 45, 50], the fundamental question of whether *exact* SSSP can be solved in $o(n)$ time when the diameter D is small (i.e., sublinear in n) remained wide open. In this article, we answer this question in the affirmative. Specifically, we devise a randomized algorithm that solves the problem in $O((n \log n)^{5/6})$ time when $D = O(\sqrt{n \log n})$, and more generally, in $O(D^{1/3} \cdot (n \log n)^{2/3})$ time, for larger D. (The result applies to the *CONGEST* model, i.e., $B = \log n$.) Observe that this running time is sublinear in n in *almost the entire range* of parameters, that is, as long as $D = o(n/\log^2 n)$. Moreover, our algorithm can compute an SPT rooted at a designated source r within the same time.

We note also that all previous sublinear-time $(1 + \epsilon)$-approximate SSSP algorithms [10, 17, 33, 45] require time proportional to $\log \Lambda$, where $\Lambda = \frac{\max\{\omega(e)|e \in E\}}{\min\{\omega(e)|e \in E\}}$ is the aspect ratio of the graph. This is, of course, unavoidable if the bandwidth is $O(\log n)$, because just delivering a single edge weight over an edge requires $O(\log_n \Lambda)$ time. However, in the natural model in which the bandwidth allows us to deliver a single edge weight or/and *Id* number through an edge in a round, the running time can be independent of Λ. This is the situation for a much-better-understood MST problem; the near-optimal algorithm of Kutten and Peleg [39] requires $O(D + \sqrt{n} \log^* n)$ time in this model, even if the aspect ratio is huge. However, the state-of-the-art $(1 + \epsilon)$-approximate SSSP algorithm [10] requires $\tilde{O}(\epsilon^{-O(1)}(\sqrt{n} + D)) \cdot \log \Lambda$ time, i.e., it is sublinear in n only if $\Lambda = 2^{o(\sqrt{n})}$. This is also the case with the approximate algorithms from [17, 33, 45].[3] However, the running time of our exact SSSP algorithm, like the running time of the MST algorithm of Kutten and Peleg [39], is *truly sublinear*, i.e., in particular, independent of the aspect ratio Λ.

1.2 Extensions and Applications

1.2.1 s Sources. We also extend our result in two directions. First, we consider the *exact* s-sources shortest paths (henceforth, s-SSP) problem, i.e., given a set S of $|S| = s$ sources, we want to compute shortest paths for all pairs of vertices in $S \times V$.

[3]The authors of Reference [17] eliminate the dependence on Λ from their SSSP algorithm for the Congested Clique and streaming models. However, similarly to References [33, 45], the running time of their algorithm for the *CONGEST* model is proportional to $\log \Lambda$.

To the best of our knowledge, the only existing solution to this problem is to run the Bellman-Ford (henceforth, B-F) algorithm in parallel from all s sources; due to congestion it requires $O(n \cdot s)$ time. Our algorithm solves the s-SSP problem (1) in time $O((n \log n)^{5/6} \cdot s^{2/3})$ for $D = O(s\sqrt{n \log n})$ and $s = O(\sqrt{n \log n})$; and (2) in time $O((n \log n)^{2/3} \cdot s)$, for $s = \Omega(\sqrt{n \log n})$ (applicable for all values of D). Together, these bounds improve the trivial $O(n \cdot s)$ bound of B-F in the entire range of parameters. In particular, our algorithm solves the *all-pairs shortest paths* (APSP) problem in time $O(n^{5/3} \cdot \log^{2/3} n)$, for all values of D.

We remark that faster *approximate* s-SSP algorithms are known [17, 33]; their running time is $\tilde{O}(D + \sqrt{ns}) \cdot \log \Lambda$, for a sufficiently large s, and $\tilde{O}(D + \sqrt{ns}) \cdot n^{o(1)} \cdot \log \Lambda$ for all s. Also, Holzer and Wattenhofer [37] devised an algorithm that solves *unweighted* APSP in $O(n)$ time. Their algorithm also provides meaningful bounds for the *unweighted* s-SSP problem.

1.2.2 Large Bandwidth. We also extend our algorithm to work more efficiently when larger bandwidth $O(b \cdot \log n)$ is available, for $1 \leq b \leq n$, i.e., in the $CONGEST(b \cdot \log n)$ model. (In fact, we assume that up to b edge weights can be delivered through an edge in a single round.)

We are aware of only two (both of which are trivial) existing solutions for the *single-source* problem (SSSP) in this model. One is the B-F algorithm, which makes no use of larger bandwidth, and as a result requires $O(n)$ time. The other one builds an auxiliary spanning BFS tree τ for G rooted at an arbitrary vertex rt. Then the trivial algorithm collects the entire topology of G into rt; solves the problem locally in rt, and disseminates the solution. This algorithm requires $O(D + |E|/b)$ time. Observe, however, that for dense graphs this expression is also not sublinear in n.

The results that we described in Section 1.1 are sublinear even when the bandwidth is small. However, using larger bandwidth our algorithm solves the SSSP problem in the $CONGEST(b \cdot \log n)$ model (1) in time $O(\frac{(n \log n)^{5/6}}{b^{1/2}})$, if $b \leq (n \log n)^{1/3}$, and $D = O(\sqrt{\frac{n \log n}{b}})$; and (2) in time $O(\frac{(n \log n)^{3/4}}{b^{1/4}})$, if $b \geq (n \log n)^{1/3}$, $D = O(\sqrt{\frac{n \log n}{b}})$; and (3) in time $O((D/b)^{1/3}(n \log n)^{2/3})$, for larger values of D, under certain mild restrictions on D and b (see Theorem C.1 for details).

Note that bound (2) above gives running time $\tilde{O}(\sqrt{n})$, when the bandwidth b is really large (i.e., $b \approx n/polylog(n)$), and the diameter D is rather small ($D \leq polylog(n)$). We remark, however, that the lower bound in $CONGEST(b \cdot \log n)$ model in this regime behaves like $\tilde{\Omega}(\sqrt{\frac{n}{b}})$ [14]; i.e., polylogarithmic-time SSSP might be possible for such a large bandwidth and small diameter.

Finally, we also extend our algorithm to the s-SSP problem in the $CONGEST(b \cdot \log n)$ model. In this case the B-F algorithm can be sped up, and it requires $O(n\lceil s/b \rceil)$ time. The topology-collecting algorithm, described above, requires $O(D + \frac{|E|+ns}{b})$ time. Our results for this setting generalize our single-source and/or unit-bandwidth results, and improve the existing (trivial) bounds in almost the entire range of parameters. (See Theorem C.3, and the discussion that follows it, for details.)

1.2.3 Streaming Model. A variant of our algorithm provides also the first non-trivial upper bound for the SSSP problems in the multipass *semi-streaming* model. In this model, the algorithm is allowed to read the stream of edges of the input n-vertex graph $G = (V, E)$ multiple times (aka passes), while storing only a limited amount of information at all times.

The problem of computing SSSP (with respect to a designated root vertex r) using a small number of passes and small memory is one of the most central open questions in the area of

semi-streaming[4] graph algorithms. Feigenbaum et al. [22, 23] pioneered the study of graph problems in this model. They observed that the greedy algorithm for constructing graph spanners [2] provides a single-pass streaming approximate algorithm for computing graph distances, and devised a more efficient (in terms of processing time-per-edge) algorithm for this problem. Their results were consequently improved in References [7, 16]. Following this research direction, [18, 20] devised efficient streaming multipass algorithms for computing sparse $(1+\epsilon,\beta)$-spanners for unweighted graphs, and as a result, derived improved multipass streaming algorithms for computing approximate distances and paths. Ahn et al. [3] devised efficient algorithms for these problems in the so-called *turnstile* (aka *dynamic*) multipass semi-streaming model. See Reference [3] for the definition of this model.

On the lower bound frontier, Feigenbaum et al. [22, 23] showed that even in *unweighted* undirected graphs, computing a p-neighborhood of a given vertex , for a positive integer parameter $p = O(\frac{\log n}{\log\log n})$, using $p-1$ passes or less, requires $n^{1+\Omega(1/p)}$ space. Guruswami and Onak [32] showed that nearly the same lower bound (up to factors polynomial in $\log n$ and in p) applies to an easier problem of computing an exact distance between a given pair of vertices, which are at distance $\Theta(p)$ from one another.

Note, however, that no non-trivial *upper* bounds are known for the fundamental problem of *exact* SSSP computation, even in *unweighted* graphs. This problem appears explicitly in the collection of open problems from Kanpur's Workshop on Algorithms for Data Streams, 2006, compiled by McGregor [43]. Specifically, problem 14 in this collection says:

"Clearly, $d_G(u,v)$ can be computed exactly in $d_G(u,v)$ passes, but for large $d_G(u,v)$ this is infeasible. Can we do better?"

In other words, there are two trivial upper bounds for exact SSSP in this model. The first one (outlined in McGregor's question) uses $O(n)$ passes and $O(n)$ memory, and the second one (which stores the entire graph) uses just one single pass, but $O(|E|)$ memory.

In the current article, we devise a deterministic algorithm that smoothly interpolates between these two trivial bounds. This algorithm, for a parameter k, $1 \le k \le n$, computes SSSP in weighted undirected graphs with non-negative edge weights, in $O(n/k)$ passes over the stream and $O(nk)$ memory. We also generalize this result to the s-SSP problem, and show that for any parameter $k \ge s$, our algorithm solves s-SSP using the same number of passes and memory as in the single-source case. (See Section 7 for further details.)

Moreover, using randomization, we extend this result to directed graphs with possibly negative edge weights. However, the number of passes in this generalized result becomes larger by a logarithmic factor, i.e., it becomes $O((n/k)\log n)$.

1.2.4 Approximating Diameter. Once single-source distances from a designated root vertex r are computed, it is easy to compute the radius with respect to r (i.e., the maximum distance between some vertex $v \in V$ and r) within additional $O(D)$ distributed time. Observe that the (weighted) *diameter* (i.e., the maximum distance between some pair $u, v \in V$ of vertices) is at least the radius, and is at most twice the radius. Hence our algorithm also provides a 2-approximation of the (weighted) diameter for the input graph, within the same time bounds. Previous sublinear-time algorithms for approximating weighted diameter [33, 45] provide $(2+o(1))$-approximation. On the lower bound frontier, Henzinger et al. [33], citing [37], state that a $(2-\epsilon)$-approximation of

[4]When the allowed memory is $\Omega(n)$, the model is typically called "semi-streaming," as opposed to the "strict" streaming model, in which the allowed space is $o(n)$. Since most graph problems require $\Omega(n)$ space, they are typically studied in the semi-streaming model [22, 23].

weighted diameter, for any constant arbitrarily small $\epsilon > 0$, requires $\tilde{\Omega}(n)$ distributed time. The problem of estimating diameter in distributed setting was extensively studied; cf. References [21, 35, 36, 37], and the references therein.

1.3 Technical Overview

The basic approach in many recent distributed[5] approximate shortest paths algorithms [17, 33, 41, 45] is the following one: One samples[6] roughly $\sqrt{n}$ "virtual" vertices. Denote the set of virtual vertices by V'. Then the algorithm builds a virtual or "skeleton" graph $G' = (V', E', \omega')$ on the set V' of virtual vertices. A pair (u', v') of virtual vertices forms an edge in E' if there exists a path $\pi(u', v')$ between them in G with at most $c \cdot \sqrt{n} \cdot \ln n$ hops, for an appropriate constant c. If it is the case, then the weight $\omega'(u', v')$ of the edge $(u', v') \in E'$ is the weight of the shortest such a $(c \cdot \sqrt{n} \cdot \ln n)$-limited $u' - v'$ path in G. (A path is said to be *h-limited*, for a parameter h, if it contains at most h hops. The smallest weight of an h-limited $u' - v'$ path in G is called the *h-limited distance* between u' and v', and is denoted $d_G^{(h)}(u', v')$.)

Once G' is constructed, the algorithm builds a *hopset* G'' for G'. A graph $G'' = (V', H', \omega'')$ is said to be a (β, ϵ)-*hopset* for G', for an integer parameter $\beta > 0$ and a real parameter $\epsilon > 0$, if for every pair $u', v' \in V'$ of vertices in G', we have

$$d_{G'}(u', v') \leq d^{(\beta)}_{G' \cup G''}(u', v') \leq (1 + \epsilon) d_{G'}(u', v').$$

Here $G' \cup G'' = (V', E' \cup H', \hat{\omega})$, where the weight function $\hat{\omega}$ gives preference to hopset edges, i.e., for $e \in H'$, we have $\hat{\omega}(e) = \omega''(e)$, and for $e \in E' \setminus H'$, we have $\hat{\omega}(e) = \omega'(e)$. Efficient distributed constructions of sparse hopsets with $\beta = n^{o(1)}$ can be found in References [17, 33]. The main precursor of these constructions is the PRAM construction of hopsets from Reference [13].

After the graph G' on the vertex set V' and the hopset G'' for G' are constructed, the algorithm conducts a B-F exploration in $G' \cup G''$ originated at the designated source vertex r. (It can be assumed that $r \in V'$.) In each iteration of this B-F exploration, every vertex v' from V' broadcasts its current distance estimate (which is an upper bound on $d_{G'}(r, v')$) to the entire graph. The number of iterations of this B-F exploration is at most the hopbound β of the hopset G''. Since $\beta = n^{o(1)}$, the entire algorithm is efficient.

When one tries to use this approach for computing *exact* shortest paths, the main hurdle is that it is not even clear how to compute the virtual graph G'. All existing algorithms [17, 33, 45] rely on *approximate* computation of paths with limited number of hops. Specifically, Nanongkai [45] developed an elegant and sophisticated routine that computes h-limited $(1 + \epsilon)$-approximate shortest paths from s designated sources in $\tilde{O}(D + h + s) \cdot \log \Lambda$ time. In the context of the single-source problem, it holds that $h \approx s \approx \sqrt{n}$, and therefore an *approximate* version of the virtual graph G' can be computed efficiently, i.e., in $\tilde{O}(D + \sqrt{n}) \cdot \log \Lambda$ time. However, obviously, once G' is computed approximately, the entire scheme is doomed to compute approximate distances, even if one were using an exact hopset G'' of G'.

Our main idea is to *bypass* the computation of G'. We show that, perhaps surprisingly, one can compute a hopset G'' for G' *without* computing the virtual graph G' first! Once this is done, we still need to conduct a B-F exploration in $G' \cup G''$, and a-priori this also seems to require the vertices to know G'. We observe, however, that the entire graph G' is not needed. Rather one can compute only those edges of G' that the B-F exploration in $G' \cup G''$ traverses. Our algorithm does

[5] Similar ideas appeared also in some centralized algorithms. See, e.g., Reference [1].

[6] The authors of Reference [33] replaced this sampling by a deterministic selection. Also, the exact size of V' may vary between different algorithms, and depend on problem's parameters, such as the diameter, the bandwidth and the number of sources.

this *on the fly*, i.e., during the exploration. Albeit, these edges are computed *exactly*, rather than approximately.

Next, we sketch how our algorithm constructs a hopset G'' without computing the virtual graph G' first. Naturally, the hopset G'' needs to be *exact*, i.e., with $\epsilon = 0$, as we aim at exact distances. Exact hopsets were built in References [40, 51, 52]. We use the hopset of Shi and Spencer [51], called the *k-shortcut hopset*, for a parameter $k \geq 1$. The hopset $G'^{(k)}$ for $G' = (V', E', \omega')$ contains, for every vertex $v' \in V'$, edges $(v', v'_1), \ldots, (v', v'_k)$ connecting v' to the k closest vertices $v'_1, \ldots, v'_k \in V'$ to v'.[7] It is easy to see that $G'^{(k)}$ is an $(O(n'/k), 0)$-hopset for G', where $n' = |V'|$ is the number of vertices in G'. We show that an exact k-shortcut hopset $G'^{(k)}$ can be constructed without knowing G' in $\tilde{O}(\sqrt{n} \cdot k^2)$ time.

The algorithm that builds this hopset is actually very simple. We run a B-F exploration in the original graph G from virtual vertices $v' \in V'$ in parallel to depth $\tilde{O}(\sqrt{n} \cdot k)$. In each iteration of this exploration, every vertex $v \in V$ records and forwards only the k closest virtual vertices that it knows. As a result, due to congestion, every iteration lasts for $O(k)$ rounds. At the end of the exploration, every vertex $v \in V$ knows the k closest virtual vertices to it (with respect to distance in G) that are reachable from v via $\tilde{O}(\sqrt{n} \cdot k)$-limited paths from the original graph G. It is not hard to see that, whp, these are exactly the k closest to v virtual vertices *with respect to distance in* G', i.e., this exploration actually computes the desired k-shortcut hopset.

Once the hopset G'' is computed, we run a B-F exploration in $G' \cup G''$ originated at r for $\beta = O(|V'|/k)$ iterations. Each iteration involves computing the required edges of G', and updating distance estimates via the edges of G' that were just computed and via the hopset edges of G''. The former is done by an inner B-F exploration in G, and the latter is done via a broadcast in the auxiliary BFS tree τ of the entire graph. For concreteness, consider the special (but a very important) case of single source and small diameter. Then each edge of G' corresponds to a path of length $\approx \sqrt{n}$ in G, and thus the B-F exploration in G goes to depth $\approx \sqrt{n}$. The broadcast in the BFS tree τ requires $O(D + |V'|) \approx \sqrt{n}$ time. Thus, a single iteration of this B-F exploration in $G' \cup G''$ is implemented in $\approx \sqrt{n}$ time. Since there are $\beta = O(|V'|/k) \approx \sqrt{n}/k$ iterations, this entire exploration requires $\approx n/k$ time. This expression is balanced with the time required to construct the hopset G'' (roughly $\sqrt{n} \cdot k^2$); this yields time $\approx n^{5/6}$.

We note that our entire algorithm is pretty simple, and it involves no heavy local computations. The algorithm amounts to a number of carefully combined B-F explorations. In particular, it is much simpler than the existing approximate SSSP solutions [10, 17, 33, 45]. Indeed, References [17, 33, 45] all involve sophisticated "light-weight" approximate shortest path computations to compute G', and then employ intricate constructions of approximate hopsets. The recent algorithm in Reference [10] bypasses hopsets altogether. It, however, employs a sophisticated and technically very involved constrained gradient descent method.

1.4 Related Work

To the best of our knowledge, the research thread that aims at solving global distributed problems in sublinear in n time when the diameter D is relatively small was initiated by Peleg [47] in the context of Leader Election problem. Awerbuch [6], citing an unpublished manuscript by Peleg, says, "*Peleg points out that the difference between $O(V)$ and $O(D)$ time can be very significant in many existing networks, e.g. the ARPANET, where $D \ll V$.*"

[7] The observation that the structure that Reference [51] computes is a hopset with hopbound $O(|V'|/k)$ appears explicitly in Reference [13]. Nanongkai [45] provided an explicit analysis and a distributed construction of this hopset, given an approximate virtual graph G'.

Bellman-Ford algorithm [8, 25] was developed a few decades before before the distributed message-passing model was formalized. Explicit descriptions of distributed Bellman-Ford algorithm were given, e.g., by Gallager in Reference [9, 27, 28].

In addition to the SSSP and MST, other global problems for which sublinear in n time algorithms were devised include the asynchronous BFS [6], and the minimum cut problem [29, 46]. Distributed exact shortest paths algorithms were also devised in References [6, 26, 38].

The idea of computing a spanner on a virtual graph without first computing the virtual graph itself appeared in Reference [41]. Note, however, that once a spanner is constructed, one no longer needs the virtual graph, but rather can conduct distance computations in the spanner itself. This is not the case with a hopset. Even when the hopset G'' of the virtual graph G' is already constructed, one still needs the graph G' to estimate distances. (Recall that distance computations are then conducted in $G' \cup G''$.) The observation that one can conduct all these computations without ever computing the virtual graph G' itself is crucial for our result.

Our streaming algorithm is closely related to Nanongkai's $O(\sqrt{n})$-time SSSP algorithm for the broadcast congested clique model [45].

2 SUBSEQUENT WORK

Our result triggered a long line of recent advances. Huang et al. [34], Agarwal and Ramachandran [4], Agarwal et al. [5], Bernstein and Nanongkai [11] devised exact APSP algorithms. Forster and Nanongkai [24] and Ghaffari and Li [31] devised improved algorithms for exact single-source shortest paths problem, albeit for graphs with polynomially bounded integer weights.

The idea of building a hopset for virtual graph G' without ever fully computing G' itself, introduced in this article in the context of exact hopsets, was subsequently employed by Neiman and the author in Reference [19] in the context of approximate hopsets.

2.1 Structure of the Paper

In Section 3 we describe our algorithm that constructs a k-shortcut hopset for the skeleton graph G'. In Section 4 we show how the hopset can be used to compute single-source distances in sublinear time. For simplicity of presentation, the bounds derived in Sections 3 and 4 are not the best ones that we can get. We derive sharper bounds in Sections 5 and 6. Our streaming algorithm and its analysis are provided in Section 7. In Appendix C we adapt our algorithms to the scenario that the bandwidth parameter b is larger than one. Some proofs are deferred to Appendix A. Finally, In Appendix B we generalize a classical upcast algorithm to $CONGEST(b \log n)$ model.

3 COMPUTING THE K-SHORTCUT HOPSET

Consider a weighted undirected graph $G = (V, E, \omega)$, and let k be a positive integer parameter. For a vertex $v \in V$, let $S_G[k](v)$ (or, shortly, $S[k](v)$, when G can be understood from the context) denote the set of k closest (reachable) vertices to G, not including v. Ties are broken arbitrarily. We define the following graph $G^{(k)} = (V, H, \omega^{(k)})$ on top of G.

$$H = \{(u, v) \mid u \in S_G[k](v) \text{ or } v \in S_G[k](u)\}.$$

The weight function $\omega^{(k)}$ is defined by $\omega^{(k)}(u, v) = d_G(u, v)$. The graph $G^{(k)}$ will be referred to as the *k-shortcut hopset* of G. The following theorem (from References [13, 45, 51]) justifies the name of $G^{(k)}$. We provide the proof of this theorem, which follows closely the proof from Reference [45], in Appendix A for the sake of completeness.

THEOREM 3.1 [45, 51]. *$G^{(k)}$ is an exact hopset of G with hopbound $h = O(n/k)$.*

We sample $N = \Theta(\sqrt{n}\log n)$ vertices $V' = \{v_1, v_2, \ldots, v_N\}$, i.e., every vertex is sampled independently at random with probability $q = \frac{c\ln n}{\sqrt{n}-1}$, for a sufficiently large constant c. By Chernoff's bound, $N = \Theta(\sqrt{n}\log n)$. Consider the virtual graph $G' = (V', E')$, where

$$E' = \{(v', u') \mid v', u' \in V' \; \exists v' - u' \text{ path in } G \text{ with } \leq \sqrt{n} \text{ hops}\}.$$

The weight function ω' is defined by $\omega'(v', u') = d_G^{(\sqrt{n})}(v', u')$.

3.1 The Algorithm

Our first objective is to compute a k-shortcut hopset $G'^{(k)}$ of G', for a parameter k, *without* first computing the virtual graph G'. This section is devoted to the following variant of the B-F algorithm that we employ for this task.

We divide the computation into *super-rounds*, each lasting for $O(k)$ rounds. At the beginning of each super-round i, $i = 0, 1, \ldots$, every vertex v maintains the set $S'^{(i)}_G[k](v)$, defined as the set of k closest selected (aka, virtual) vertices $v' \in V'$ to v, closest with respect to i-limited distance in G. In other words, if one orders virtual vertices $v'_1, \ldots, v'_N$ in the order of non-decreasing i-limited distance from v in G (i.e., $d_G^{(i)}(v, v'_1) \leq d_G^{(i)}(v, v'_2) \leq \cdots \leq d_G^{(i)}(v, v'_N)$), then $S'^{(i)}_G[k](v)$ contains the first k elements in this ordering. (It is also required that $d_G^{(i)}(v, v'_j) < \infty$, for any $v'_j \in S'^{(i)}_G[k](v)$.)

Here ties are broken in the following way. For v'_h, v'_j, $1 \leq h < j \leq N$, suppose that $d_G^{(i)}(v, v'_h) = d_G^{(i)}(v, v'_j)$. Let h' (respectively, j') be the number of hops in the shortest i-limited $v - v'_h$ (respectively, $v - v'_j$) path $\pi(v, v'_h)$ (respectively, $\pi(v, v'_j)$) in G. Then if $h' < j'$, then we write $v'_h <_v v'_j$, i.e., we prefer v'_h to v'_j in the ordering $<_v$. Symmetrically, if $h' > j'$, then $v'_h >_v v'_j$. Finally, if $h' = j'$, then the paths $\pi(v, v'_h), \pi(v, v'_j)$ are compared in the lexicographical order, starting with v. (We assume that all vertices have distinct *Id* numbers.) We write $\pi(v, v'_h) <_v \pi(v, v'_j)$, if $\pi(v, v'_h)$ is lexicographically smaller than $\pi(v, v'_j)$, and then also $v'_h <_v v'_j$.

This completes the definition of $S'^{(i)}_G[k](v)$. Recall that we assume inductively that at the beginning of a super-round i, $i = 0, 1, \ldots$, every vertex v knows $S'^{(i)}_G[k](v)$. For every $v' \in S'^{(i)}_G[k](v)$, the vertex v also keeps the i-limited distance $d_G^{(i)}(v', v)$, the number of hops $h^{(i)}(v', v)$ in the shortest i-limited $v' - v$ path, and the *parent* or *predecessor* $u = p(v)$ from which v received this tuple. Observe that the induction base case $i = 0$ holds.

In super-round i, every vertex v sends to all its neighbors its entire set $\mathcal{S}^{(i)}(v) = S'^{(i)}_G[k](v)$. Then v selects k smallest estimates among those that it received, using appropriate (see below) tie-breaking rules. As a result, the vertex v computes its set $\mathcal{S}^{(i+1)}(v)$. We will show that it is equal to $S'^{(i+1)}_G[k](v)$.

Specifically, v receives from a neighbor u a tuple $(x, d^{(i)}(x, u), h^{(i)}(x, u), p(u))$. Then v computes its own tuple $(x, \delta_u^{(i+1)}(x, v) = d^{(i)}(x, u) + \omega(u, v), h_u^{(i+1)}(x, v) = h^{(i)}(x, u) + 1, p(v) = u)$. (Here $\delta_u^{(i+1)}(x, v)$ is the estimate of $(i+1)$-limited $x - v$ distance that v learns from u, and $h_u^{(i+1)}(x, v)$ is the number of hops in this esimate.)

Then for each origin x that v hears from, it computes the estimate $d^{(i+1)}(x, v) = \min_u \delta_u^{(i+1)}(x, v)$. In case of equality, v keeps the tuple with smaller $h_u^{(i+1)}(x, v)$. If these are equal, too, then it prefers the tuple with lexicographically smaller parent u. When comparing tuples from different origins, v uses the same rules again, and selects k smallest tuples. (For a pair of such distinct tuples σ, σ' that v compares, we write $\sigma <_v \sigma'$ iff v prefers σ over σ' using the above rules of comparison.)

Next, we argue that $\mathcal{S}^{(i+1)}(v) = S'^{(i+1)}_{G}[k](v)$. For convenience, super-rounds are numbered below starting with 0. The proof of this lemma is in Appendix A.

Lemma 3.2. *The set $\mathcal{S}^{(i+1)}(v)$ that each vertex v computes at the end of super-round i is equal to $S'^{(i+1)}_{G}[k](v)$.*

To summarize, after $\sqrt{n} \cdot k$ super-rounds of this Bellman-Ford algorithm (which last for $O(\sqrt{n} \cdot k^2)$ time), every vertex v computes $S'^{(\sqrt{n}\cdot k)}_{G}[k](v)$.[8]

3.2 The Analysis

In this section, we analyze the algorithm for constructing the k-shortcut hopset that was described in Section 3.1.

Next, we define the sets $S'^{(i)}_{G'}[k](v)$, for vertices $v' \in V'$, for some fixed integer i. For a vertex $v' \in V'$, the set $S'^{(i)}_{G'}[k](v')$ is defined as the set of k closest sampled vertices (i.e., vertices of V') to v' with respect to i-limited distances in G'. (The set will only contain vertices reachable within i hops in G' from v'. An analogous restriction applies also to the set $S'^{(i)}_{G}[k](v')$.) In other words, let

$$d^{(i)}_{G'}(v', v'_1) \leq d^{(i)}_{G'}(v', v'_2) \leq \cdots d^{(i)}_{G'}(v', v'_N).$$

Then $S'^{(i)}_{G'}[k](v') = \{v'_1, v'_2, \ldots, v'_k\}$, where the ties are broken according to the paths in G (and not in G').

Specifically, consider a pair of vertices $u', w' \in V'$ such that $d^{(i)}_{G'}(v', u') = d^{(i)}_{G'}(v', w')$. Let $\pi(v', u')$ (respectively, $\pi(v', w')$) be the shortest $(i \cdot \sqrt{n})$-limited $v' - u'$ (respectively, $v' - w'$) path in the *original* graph G with the smallest number of hops. Then $u' <_{v'} w'$ iff $\pi(v', u') <_{v'} \pi(v', w')$.

Our next objective is to show that, with high probability (henceforth, we will write "whp"), $S'^{(k)}_{G'}[k](v') = S'^{(\sqrt{n}\cdot k)}_{G}[k](v')$. Before proving it, we define a collection of i-limited paths, for all $i \in [n-1]$, between all pairs of vertices, and argue that this collection admits useful nesting properties.

Let $\mathcal{P} = \{P^{(i)}(u, v) \mid 1 \leq i \leq n-1, u \neq v, u, v \in V\}$ be the collection of i-limited shortest paths computed if we were to run B-F on G, with the above rules to break ties, from all vertices of the graph, one after another. (Note that our tie-breaking rules prefer paths that are lexicographically smaller when viewing them from the *tail to head*.) We remark that the algorithm does not actually do this computation. We just imagine that it is done for definitional purposes.

The proofs of the following two standard lemmas can be found in Appendix A.

Lemma 3.3. *Let $P^{(i)}(u, v), P'^{(i')}(u', v') \in \mathcal{P}$ be an i-limited and an i'-limited paths, respectively, that traverse some pair of vertices x and y in the same order, and the number of hops ℓ in $P^{(i)}(u, v)$ between x and y is equal to the number of hops in $P'^{(i')}(u', v')$ between x and y. Then the entire subpaths $P^{(\ell)}(x, y)$ and $P'^{(\ell)}(x, y)$ of the two respective paths are equal. In particular, it follows that $P^{(\ell)}(x, y) = P'^{(\ell)}(x, y) \in \mathcal{P}$.*

Lemma 3.4. *Whp, (specifically, with probability at least $1 - n^{-(c-3)}$), any path $P^{(i)}(u, v) \in \mathcal{P}$ with $|P^{(i)}(u, v)| \geq \sqrt{n}$ (i.e., the number of hops in the path is at least $i \geq \sqrt{n}$), contains at least one internal vertex from V'.*

Denote by $\mathcal{A}$ the event of Lemma 3.4. We showed that $\mathbb{P}(\mathcal{A}) \geq 1 - n^{-(c-3)}$.

[8] As pointed out by an anonymous reviewer of STOC'17 conference, this fact can also be derived using the "short-range scheme" of Reference [41].

Lemma 3.5. *Conditioned on $\mathcal{A}$, for every virtual vertex $v' \in V'$, and every $u' \in S'^{(\sqrt{n}\cdot k)}_{G}[k](v')$, there is a k-limited path in G' of weight $d^{(\sqrt{n}k)}_{G}(u', v')$.*

Proof. Consider $P^{(\sqrt{n}k)}(v', u')$, i.e., the shortest $v' - u'$ $(\sqrt{n}k)$-limited path in G with the fewest number of hops, smallest with respect to $<_{v'}$ among such paths. (If there is no such a path, then $d^{(\sqrt{n}k)}_{G}(v', u') = \infty$, and there always exists a weight-∞ (or less) k-limited path in G' between v' and u'.)

Let $v' = v'_0, v'_1, \ldots, v'_{h-1}, v'_h = u'$ be the distinct selected vertices in $P^{(\sqrt{n}k)}(v', u')$, in the order of their appearence on $P^{(\sqrt{n}k)}(v', u')$. If $h > k$ (i.e., $h - 1 \geq k$), then $v'_1, v'_2, \ldots, v'_{h-1}$ are all closer to v' than u' with respect to $(\sqrt{n}k)$-bounded distance in G. (Or, if zero weights are allowed, then some of these vertices may be at the same distance from v' as u', but the number of hops between v' and them is smaller than in the $v' - u'$ path.) This is a contradiction to the assumption that $u' \in S'^{(\sqrt{n}\cdot k)}_{G}[k](v')$. (As there are at least k closer than u' to v' selected vertices.)

Hence $h \leq k$. For every $i \in [0, h-1]$, the number of hops on $P^{(\sqrt{n}k)}(v', u')$ between v'_i and v'_{i+1} is, conditioned on $\mathcal{A}$, at most $\sqrt{n}$. (Otherwise there were another selected vertex between them on the path.) Hence there are edges (v'_i, v'_{i+1}), for every $i \in [0, h-1]$, in G' of weight $\omega'(v'_i, v'_{i+1})$ equal to the weight $\omega(P(v'_i, v'_{i+1}))$ of the subpath $P(v'_i, v'_{i+1})$ of $P^{(\sqrt{n}k)}(v', u')$ between v'_i and v'_{i+1}. Hence there is a $v' - u'$ path $(v' = v'_0, v'_1, \ldots, v'_{h-1}, v'_h = u')$, $h \leq k$, in G' of weight $d^{(\sqrt{n}k)}_{G}(v', u')$. □

Lemma 3.6. *Conditioned on $\mathcal{A}$, for every $v' \in V'$, and for every $u' \in S'^{(\sqrt{n}\cdot k)}_{G}[k](v')$, we have $d^{(k)}_{G'}(v', u') = d^{(\sqrt{n}k)}_{G}(v', u')$.*

Proof. By Lemma 3.5, $d^{(k)}_{G'}(v', u') \leq d^{(\sqrt{n}k)}_{G}(v', u')$. Suppose for contradiction that there is a strict inequality. But then we can translate the k-limited $v' - u'$ path in G' of weight $d^{(k)}_{G'}(v', u')$ into a $(\sqrt{n}k)$-limited $v' - u'$ path in G of the same weight. This however results in a $(\sqrt{n}k)$-limited path between them of length strictly smaller than $d^{(\sqrt{n}k)}_{G}(v', u')$, contradiction. □

Lemma 3.7. *Conditioned on $\mathcal{A}$, for every $v' \in V'$,*

$$S'^{(k)}_{G'}[k](v') \subseteq S'^{(\sqrt{n}k)}_{G}[k](v').$$

Proof. Consider $u' \in S'^{(k)}_{G'}[k](v')$. There exists a $(\sqrt{n}k)$-limited path $\pi(v', u')$ in G of length at most $d^{(k)}_{G'}(v', u')$. (This path is obtained by concatenating $\sqrt{n}$-limited paths from G that correspond to edges of the k-limited $v' - u'$ path in G'.)

If $u' \notin S'^{(\sqrt{n}k)}_{G}[k](v')$, then it follows that there are some other k vertices $u'_1, u'_2, \ldots, u'_k \in S'^{(\sqrt{n}k)}_{G}[k](v')$, such that for all $i \in [k]$, $u'_i \neq u'$, and *either* $d^{(\sqrt{n}k)}_{G}(v', u'_i) < d^{(\sqrt{n}k)}_{G}(v', u')$ *or* $d^{(\sqrt{n}k)}_{G}(v', u'_i) = d^{(\sqrt{n}k)}_{G}(v', u')$ and the number of hops in the shortest $(\sqrt{n}k)$-limited $v' - u'_i$ path $\pi(v', u'_i)$ in G is smaller than in the shortest $(\sqrt{n}k)$-limited $v' - u'$ path $\pi(v', u')$ in G, *or* also $|\pi(v', u'_i)| = |\pi(v', u')|$, but $\pi(v', u'_i) <_{v'} \pi(v', u')$.

Also, by Lemmas 3.5 and 3.6, $d^{(\sqrt{n}k)}_{G}(v', u') = d^{(k)}_{G'}(v', u')$, and $d^{(\sqrt{n}k)}_{G}(v', u'_i) = d^{(k)}_{G'}(v', u'_i)$, for all $i \in [k]$. Hence, for all $i \in [k]$, either $d^{(k)}_{G'}(v', u'_i) < d^{(k)}_{G'}(v', u')$, or $d^{(k)}_{G'}(v', u'_i) = d^{(k)}_{G'}(v', u')$, but $|\pi(v', u'_i)| < |\pi(v', u')|$, or also $|\pi(v', u'_i)| = |\pi(v', u')|$, but $\pi(v', u'_i) <_{v'} \pi(v', u')$. In either case, this is a contradiction to the assumption that $u' \in S'^{(k)}_{G'}[k](v')$. (Recall that the ties in $S'^{(k)}_{G'}[k](v')$ are broken using respective paths in G, and not in G'.) □

Lemma 3.8. *Conditioned on $\mathcal{A}$, for every $v' \in V'$,*

$$S'^{(\sqrt{n}k)}_{G}[k](v') \subseteq S'^{(k)}_{G'}[k](v').$$

Proof. If $|S'^{(k)}_{G'}[k](v')| = k$, then since by Lemma 3.7, $S'^{(k)}_{G'}[k](v') \subseteq S'^{(\sqrt{n}k)}_{G}[k](v')$, and $|S'^{(\sqrt{n}k)}_{G}[k](v')| \leq k$, it follows that the two sets are equal.

Otherwise, $|S'^{(k)}_{G'}[k](v')| < k$. Suppose for contradiction that there exists $u' \in S'^{(\sqrt{n}k)}_{G}[k](v') \setminus S'^{(k)}_{G'}[k](v')$. By Lemma 3.6, $d^{(k)}_{G'}(v', u') = d^{(\sqrt{n}k)}_{G}(v', u')$. Since $u' \in S'^{(\sqrt{n}k)}_{G}[k](v')$, by definition of this set, $d^{(k)}_{G'}(v', u') = d^{(\sqrt{n}k)}_{G}(v', u') < \infty$. But then u' must have been included in $S'^{(k)}_{G'}[k](v')$, because $|S'^{(k)}_{G'}[k](v')| < k$ implies that for all vertices $u' \in V' \setminus S'^{(k)}_{G'}[k](v')$, we have $d^{(k)}_{G'}(v', u') = \infty$. This is a contradiction. □

Corollary 3.9. *Conditioned on $\mathcal{A}$, for every vertex $v' \in V'$,*

$$S'^{(\sqrt{n}k)}_{G}[k](v') = S'^{(k)}_{G'}[k](v').$$

Finally, we argue that for any vertex $v' \in V'$, $S'^{(k)}_{G'}[k](v') = S'_{G'}[k](v')$. (The latter set is the set of k closest vertices in G' to v', where the paths are no longer required to be k-limited. The ties are broken in the same way as before, i.e., using the respective paths in G.) The proof of the following lemma is in Appendix A.

Lemma 3.10. *Conditioned on $\mathcal{A}$, for every $v' \in V'$,*

$$S'^{(k)}_{G'}[k](v') = S'_{G'}[k](v').$$

Hence, conditioned on $\mathcal{A}$, $S'^{(\sqrt{n}k)}_{G}[k](v') = S'_{G'}[k](v')$, i.e., as a result of our computation, after $O(\sqrt{n} \cdot k^2)$ rounds every virtual vertex $v' \in V'$ knows its $S'_{G'}[k](v')$. We also want to ensure that every vertex $u' \in V'$ that belongs to $S'_{G'}[k](v')$, for some $v' \in V'$, knows about this hopset edge (v', u'). (We just showed that v' does know about it.)

To ensure this, we conduct an upcast and pipelined broadcast of all the computed edges $\{(v', u') \mid u' \in S'_{G'}[k](v'), v' \in V'\} = G'^{(k)}$ over an auxiliary BFS tree τ of the entire graph G. Since there are $O(|V'| \cdot k) = O(\sqrt{n} \cdot \log n \cdot k)$ edges in $G'^{(k)}$, it follows that this step requires $O(D + \sqrt{n} \cdot k \cdot \log n)$ time.

Corollary 3.11. *Whp, in overall time $O(D + \sqrt{n} \cdot k^2 + \sqrt{n}k \cdot \log n)$, the k-shortcut hopset $G'^{(k)}$ of the virtual graph G' can be computed from scratch.*

We remark that the virtual graph G' itself is *not* known to the vertices of V', even after the computation of the hopset $G'^{(k)}$ for G' has been completed.

4 COMPUTING DISTANCES

In this section, we employ the k-shortcut hopset computed in Section 3 to compute distances from a designated root vertex r to all other vertices of the graph. We will also extend this algorithm to compute a shortest paths tree (henceforth, SPT) rooted at r.

We can assume that the root r belongs to V'. (It can be just added to V' after sampling all other vertices of V'. This has no effect on the analysis, except that the expected size of V' grows by an additive 1.) Our first objective at this stage is to compute all distances $\{d_G(r, v') \mid v' \in V'\}$. Specifically, we want every $v' \in V'$ to know its respective distance $d_G(r, v')$. The algorithm will utilize an auxiliary BFS tree τ of G rooted at a vertex rt. It can be constructed in $O(D)$ time. (See, e.g., Reference [48], Chapter 5.)

Recall that (see Theorem 3.1) the hopbound of $G'^{(k)}$ is $h' = O(N/k) = O(\frac{\sqrt{n}\log n}{k})$, whp. The stage that computes distances runs for h' iterations, i.e., it is a B-F to depth h' rooted at r, in $G' \cup G'^{(k)}$. (By "B-F to depth h" we mean that the B-F explores vertices that are at *hop*-distance at most h from the origin.) At the beginning of 0th iteration, every vertex $v' \in V' \setminus \{r\}$ initializes its estimate $\delta(v') = \infty$, and r initializes its estimate $\delta(r) = 0$. In iteration i, $0 \le i \le h' - 1$, every virtual vertex $v' \in V'$ that has a finite estimate $\delta(v')$ wishes to deliver its estimate to all its neighbors in $G' \cup G'^{(k)}$. Before iteration i starts, every vertex v initializes two auxiliary estimates $\delta^I(v) = \delta^{II}(v) = \delta(v)$. The iteration lasts for $O(D + |V'|) + O(\sqrt{n}) = O(D + \sqrt{n}\log n)$ rounds, and it consists of two parts. In the first part, all vertices v' upcast their estimates to the root rt of the auxiliary tree τ, and the root broadcasts them to the entire graph. Every vertex $u' \in V'$ that hears an estimate $\delta(v')$ of its neighbor v' in the hopset $G'^{(k)}$, computes its own estimate $\delta_{v'}(u') = \delta(v') + \omega^{(k)}(v', u')$, compares it with the minimum estimate $\delta^I(v')$ it knows, and updates the minimum estimate if needed. (Note, however, that on the second part of the iteration v' still disseminates its original estimate $\delta(v')$ and not the updated estimate $\delta^I(v')$.) This part of the computation requires $O(|D| + |V'|) = O(D + \sqrt{n}\log n)$ time.

In the second part of the iteration, every vertex v' initiates a B-F in G to depth $\sqrt{n}$. The message v' broadcasts is $\delta(v')$. Every intermediate vertex $v \in V$ that hears from each of its neighbors $\{u_1, \ldots, u_d\}$ estimates $\{\delta(u_1), \ldots, \delta(u_d)\}$ of their respective distances from r, computes the smallest value $\delta(u_i) + \omega((u_i, v))$, compares it with its current estimate $\delta(v)$, sets the minimum as its new estimate, and sends the latter to its neighbors. (We also want every vertex to record its *parent*, i.e., the vertex from which it learned its current estimate.) This process continues for $\sqrt{n}$ rounds. In each round, every vertex sends one message to all its neighbors.

Since each edge $e = (v', u')$ of G' corresponds to a $\sqrt{n}$-limited $v' - u'$ path in G, this $\sqrt{n}$-limited B-F exploration serves to imitate one phase of a B-F exploration in G'. Specifically, as a result of the second part of iteration i, every vertex $v' \in V'$ learns the value

$$\delta^{II}(v') = \min\{\delta(u') + d_G(v', u') \mid (v', u') \in E'\} = \min\{\delta(u') + \omega'(v', u') \mid (v', u') \in E'\}.$$

Now v' computes the minimum between $\delta^I(v')$ and $\delta^{II}(v')$, and sets it as its new estimate $\delta(v')$. The latter estimate will be used in the next iteration of the B-F in $G \cup G'^{(k)}$.

Observe that in the second part of each iteration, we essentially run the B-F in G that we used to compute the hopset $G'^{(k)}$, but with $k = 1$. (But we deliver just one single smallest value, and not the k smallest ones. Also, now every vertex v is interested in its distance from a designated root r, rather than in the distances from k closest vertices of V'.)

Hence a special case of the analysis that we used in Section 3 shows that after i, $0 \le i \le \sqrt{n}$, rounds of this B-F, every vertex $v \in V$ knows $\delta^{(i)}(v) = \min\{\delta(v') + d_G^{(i)}(v', v)\}$, where the minimum is taken over all $v' \in V'$ reachable from v via at most i hops in G. In particular, after $\sqrt{n}$ rounds of this process, every $u' \in V'$ knows its $\delta^{(\sqrt{n})}(u')$.

Hence, overall, iteration i requires $O(D + \sqrt{n}\log n)$ time, and it imitates one iteration of B-F in $G' \cup G'^{(k)}$, rooted at r. Since the hopbound of $G'^{(k)}$ is $h' = O(N/k) = O(\frac{\sqrt{n}\log n}{k})$, it follows that within h' such iterations, every vertex $v' \in V'$ knows its correct distance estimate $d^{(h')}_{G' \cup G'^{(k)}}(r, v') = d_{G'}(r, v') = d_G(r, v')$. (The last equality holds whp. It is true because, by Lemma 3.4, the shortest $r - v'$ path in G' contains vertices of V' every $\sqrt{n}$ hops or less, i.e., it can be replaced by an $r - v'$ path in G' of the same length.)

Corollary 4.1. *After the hopset $G'^{(k)}$ has already been constructed, within additional $O(D + \sqrt{n}\log n) \cdot O(\frac{\sqrt{n}\log n}{k})$ time, all distances $\{d_G(r, v') \mid v' \in V'\}$ can be computed.*

Hence the overall time spent by the algorithm so far (including the time required to construct the k-shortcut hopset $G'^{(k)}$) is $O(D+\sqrt{n}\log n)\cdot O(\frac{\sqrt{n}\log n}{k})+O(\sqrt{n}\cdot k^2+\sqrt{n}\cdot k\log n)$.

Now, when all virtual vertices $v'\in V'$ know their exact disatnces $d_G(r,v')$ from r, we conduct a $\sqrt{n}$-limited B-F exploration in G from all vertices of V'. (Every vertex $v\in V$, in every round, selects one single smallest estimate of its distance from r, and forwards it.) This step requires $\sqrt{n}$ additional time.

Lemma 4.2. *Conditioned on $\mathcal{A}$ (i.e., whp), for every vertex $v\in V$, after the last step of the algorithm we have $\delta(v)=d_G(r,v)$.*

Proof. Let $\pi(r,v)$ be a shortest $r-v$ path in G, with the smallest number of hops, smallest with respect to $<_v$ among such paths.

First, consider the case that $|\pi(r,v)|\le\sqrt{n}$. Recall that $r\in V'$. Then the distance $d_G(r,v)$ propagates from the root r to v along $\pi(r,v)$ during the last stage of the algorithm (i.e., during the $\sqrt{n}$-limited B-F), and we are done.

Now we turn to the case $|\pi(r,v)|>\sqrt{n}$. Let $(r=v'_0,v'_1,\ldots,v'_h,v)$, $v'_0,v'_1,\ldots,v'_h\in V'$ be the virtual vertices (i.e., vertices of V') appearing on $\pi(r,v)$, in the order of their appearance. Under $\mathcal{A}$, for every index $i\in[0,h-1]$, the number of hops between v'_i and v'_{i+1} in $\pi(r,v)$ is at most $\sqrt{n}$, and so is the number of hops between v'_h and v. At the last stage of the algorithm, the vertex v'_h holds $\delta(v'_h)=d_G(r,v'_h)$.

Denote by $\pi(v'_h,v)$ the subpath of $\pi(r,v)$ connecting v'_h and v. The estimate $\delta(v'_h)$ propagates along $\pi(v'_h,v)$ during the $\sqrt{n}$-limited B-F on the last stage of the algorithm, and at the end of the algorithm it holds that

$$\delta(v)\le\delta(v'_h)+\omega(\pi(v'_h,v))=d_G(r,v'_h)+d_G(v'_h,v)=d_G(r,v).$$

The last inequality is because v'_h lies on the shortest $r-v$ path in G.

Suppose for contradiction that $\delta(v)<d_G(r,v)$. But then there exists some virtual vertex $v'\in V'$, such that $\delta(v)=\delta(v')+d_G(v',v)$. (This is the virtual vertex through which v has acquired its estimate $\delta(v)$.) But $\delta(v')=d_G(r,v')$, and so there exists an $r-v$ path in G that passes through v' and has length $\delta(v)=d_G(r,v')+d_G(v',v)<d_G(r,v)$, contradiction. □

Next we show that the algorithm can also construct an SPT for G rooted at r. In the beginning we assume, for convenience of presentation, that all edge weights are positive.

The modification to the algorithm is in the last stage, where a $\sqrt{n}$-limited B-F in G from vertices of V' is conducted. There are two modifications. First, every vertex $v\in V$ records the neighbor $p(v)$ of v in G from which it received its final distance estimate. (For this end it always keeps the neighbor that supplied v its current estimate; at the end this neighbor is $p(v)$.) Second, every vertex $v'\in V'$ will now also receive updates on this last stage of the algorithm. (This is not necessary if one is only in interested in distances.) The vertex v' will record the neighbor $p(v')$ of v' in G, through which v' could receive a correct estimate $\delta(v')=d_G(r,v)$. That is, every neighbor u of v' sends it some value $\delta(u)$, and v' computes $\min\{\delta(u)+\omega((u,v'))\mid u\in\Gamma_G(v')\}$. This value is equal to the value $\delta(v')=d_G(r,v')$, which v' knows before the last phase begins. Nevertheless, the vertex v' records the neighbor $p(v')=u$ through which it can attain this value. Ties are broken in the same way as above, i.e., according to the number of hops between r and $p(v)$, and finally, in case of equality, using the *Id* numbers of neighbors $p(v)$. The argument that shows that this is indeed the same value is given in Lemma 4.2.

We now argue that the resulting edge set is an SPT of G with respect to r.

Lemma 4.3. *The edge set $T=\{(v,p(v))\mid v\in V\setminus\{r\}\}$ is an SPT of G with respect to r.*

Proof. The edge set T spans V. Moreover, it is acyclic, as $\delta(p(v)) < \delta(v)$, for all $v \in V \setminus \{r\}$. (Because edge weights are positive.) This also guarantees that if $u = p(v)$, then it cannot happen that $p(u) = v$, and so the number of edges in $T = \{(v, p(v)) \mid v \neq r\}$ is $n - 1$. Hence T is a spanning tree of G.

The proof that T is an SPT is by induction on the depth (the hop-distance from r) in T of a vertex v. The base case ($v = r$) clearly holds.

For the induction step, observe that by the induction hypothesis, $d_T(r, p(v)) = d_G(r, p(v))$. Also, $d_T(r, v) = d_T(r, p(v)) + \omega((p(v), v)) = d_G(r, p(v)) + \omega((p(v), v))$. But the latter is equal to the distance estimate $\delta(v) = \delta(p(v)) + \omega((p(v), v))$ of v (from r), and we have already established that $\delta(v) = d_G(r, v)$. Hence $d_T(r, v) = d_G(r, v)$. □

This argument can be extended to the case that zero weights are allowed, in the following way. Let T' be an SPT of $G' \cup G'^{(k)}$ with respect to r, that our algorithm implicitly constructs when it computes the distances $\{d_G(r, v') \mid v' \in V'\}$. Every vertex $v' \in V'$ will record its hop-distance $h'(v') \leq h'$ from r in T', i.e., the number of phase of the B-F over $G' \cup G'^{(k)}$ in which it acquired its final distance estimate. Also, in the last stage of the algorithm, a vertex v will prefer as its parent $p(v)$ the vertex u with smallest $\delta(u) + \omega((u, v))$, and among such vertices v will prefer u, which acquired its estimate through a virtual vertex v' with smallest $h'(v')$. Also, breaking ties among two such v'_1, v'_2 with equal $h'(v'_1) = h'(v'_2)$, the vertex v will prefer the one with smaller number of hops in G between v and this virtual vertex. (This corresponds to the iteration number of the last stage of the algorithm.) Finally, if those are equal, one breaks ties by *Id*s of v'_1, v'_2, and for two messages that come from the same v' via different neighbors u_1, u_2, the identities of u_1, u_2 will be used to break ties.

With this choice of parents, it is easy to see that the resulting T is acyclic, and contains $n - 1$ edges. The rest of the proof of Lemma 4.3 extends seamlessly.

The overall running time of the algorithm is, by Corollaries 3.11 and 4.1,

$$O(\sqrt{n} \cdot k^2 + D + \sqrt{n} \cdot k \cdot \log n) + (O(D + \sqrt{n} \cdot \log n) + O(\sqrt{n})) \cdot O\left(\frac{\sqrt{n} \log n}{k}\right) =$$

$$O\left(D \cdot \frac{\sqrt{n} \cdot \log n}{k} + \sqrt{n} \cdot k^2 + \frac{n \cdot \log^2 n}{k} + \sqrt{n} \cdot k \cdot \log n\right).$$

Set $k = n^{1/6} \cdot \log^{2/3} n$. We obtain time $O((D + n^{1/2} \cdot \log n) \cdot (n \cdot \log n)^{1/3})$. This is $O(n^{5/6} \cdot \log^{4/3} n)$, for $D = O(n^{1/2} \cdot \log n)$. This bound is sublinear in n for $D = o(n^{2/3} / \log^{1/3} n)$.

However, when D is large, i.e., $D = \Omega(\sqrt{n} \cdot \log n)$, it makes sense to set $k = (D \cdot \log n)^{1/3}$. Observe that $k = (D \cdot \log n)^{1/3} \leq \sqrt{n} \cdot \log n = N$, and so this is a valid choice of the parameter k. With this choice of k, the running time becomes

$$O(\sqrt{n} \cdot D^{2/3} \cdot \log^{2/3} n) + O\left(\frac{n \cdot \log^{5/3} n}{D^{1/3}}\right) = O(D^{2/3} \sqrt{n} \cdot \log^{2/3} n).$$

(The last inequality is because $D = \Omega(\sqrt{n} \cdot \log n)$.) This estimate is sublinear in n as long as $D = o(\frac{n^{3/4}}{\log n})$. It is also no worse than the estimate $O((D + \sqrt{n} \log n) \cdot (n \log n)^{1/3})$ in the entire range $D = \Omega(\sqrt{n} \log n)$.

To summarize:

Theorem 4.4. *Whp, the algorithm described above computes an exact single source shortest paths tree in the CONGEST model in $O(n^{5/6} \cdot \log^{4/3} n)$ time, whenever $D = O(\sqrt{n} \cdot \log n)$, and in $O(D^{2/3} \sqrt{n} \cdot \log^{2/3} n)$ time, for larger D.*

See Theorem 5.3 for yet sharper bounds.

5 IMPROVED BOUNDS

In this section, we show that by setting the sampling probability q more carefully, one obtains yet sharper bounds. Most notably, in this way we derive a variant of our algorithm that runs in sublinear in n time, for a much wider range of D. Specifically, the running time will be sublinear in n for $D = o(n/\log^2 n)$.

Recall that every vertex is selected independently at random with probability q. The expected number of selected vertices $\mathbf{E}(|V'|) = n \cdot q$, i.e., whp, $|V'| = O(n \cdot q)$. (We assume that $q = \Omega(\frac{\log n}{n})$.) Consider a path of $\frac{n}{n \cdot q} \cdot (c \cdot \ln n) = \frac{c \ln n}{q}$ hops, for a sufficiently large constant c. It contains no selected vertex with probability $(1-q)^{\frac{c \ln n}{q}} \leq \frac{1}{n^c}$. So, whp, (i.e., with probability at least $1 - \frac{1}{n^{c-3}}$), for every pair $u, v \in V$ of vertices and an index i such that $|P^{(i)}(u,v)| \geq \frac{c \ln n}{q}$, the path $P^{(i)}(u,v)$ contains a selected vertex.

We conduct a B-F from all vertices of V' to depth $\frac{c \cdot \ln n}{q} \cdot k$, while storing and forwarding k smallest values at every super-round. The total time required for this step is $O(\frac{\ln n}{q} \cdot k^2)$.

The upcast and pipelined broadcast (over the BFS tree τ of G) of the hopset edges requires whp $O(D + n \cdot q \cdot k)$ time.

Corollary 5.1. *Constructing the k-shortcut hopset $G'^{(k)}$ for G' requires $O(\frac{\log n}{q} \cdot k^2 + D + nqk)$ time, whp.*

We then conduct B-F rooted at the vertex r in $G' \cup G'^{(k)}$ for $h' = O(N/k) = O(\frac{n \cdot q}{k})$ phases.

Lemma 5.2. *This step requires $O(((D + nq) + \frac{\ln n}{q}) \cdot \frac{nq}{k})$ time, whp.*

Proof. Recall that every phase of this B-F consists of two parts. In the first part, $O(nq)$ estimates (whp) of distances of selected vertices from r are disseminated over the BFS tree τ. This requires $O(D + nq)$ time, whp. In the second part, a B-F exploration over G to depth $\frac{c \ln n}{q}$ is conducted from vertices of V', where each vertex forwards just the smallest estimate of distance from r that it knows. This requires $O(\frac{\ln n}{q})$ time. Since there are $O(\frac{n \cdot q}{k})$ phases of the B-F in $G' \cup G'^{(k)}$, it follows that the overall time of this step is $O(((D + nq) + \frac{\ln n}{q}) \cdot \frac{nq}{k})$, whp. □

Finally, when all virtual vertices (of V') already know their respective distances from r, another B-F in G to depth $\frac{c \ln n}{q}$ is conducted, to update all vertices of $V \setminus V'$. This step requires additional $O(\frac{\ln n}{q})$ time.

Hence, whp, the total running time of the algorithm is given by

$$T = O\left(\frac{\ln n}{q} \cdot k^2 + (D + nq \cdot k) + \left(D + nq + \frac{\ln n}{q}\right) \cdot \frac{nq}{k}\right). \tag{1}$$

To optimize the running time, we consider two regimes. The first regime is when D is relatively small, i.e., $D = O(\max\{nq, \frac{\ln n}{q}\})$. We then substitute $q = \sqrt{\frac{\ln n}{n}}$, $k = (n \ln n)^{1/6}$. (Observe that $k \leq nq$. This is required, because $N = \Theta(nq)$, and k in the k-shortcut hopset needs to be at most $N - 1$.) The bound becomes $T = O((n \ln n)^{5/6})$ (for $D = O(\sqrt{n \ln n})$). This slightly improves the bound $T = O(n^{5/6} \cdot \log^{4/3} n)$ that we had in Theorem 4.4.

Next we consider the regime of large D, i.e., $D \geq \max\{\frac{\ln n}{q}, nq\}$. We set $q = \frac{\ln n}{D}$, $k = (\frac{n \ln n}{D})^{1/3}$. (Note that $nq \geq k$, as required.) We also have $D = \frac{\ln n}{q} \geq nq$ in this case, as $D = \Omega(\sqrt{n \log n})$. Then

the running time is bounded by

$$T = O\left(\frac{\ln n}{q} \cdot k^2 + (D + nq \cdot k) + D \cdot \frac{nq}{k}\right) = O\left(D^{1/3}(n \ln n)^{2/3} + \left(\frac{n \ln n}{D}\right)^{4/3}\right)$$
$$= O(D^{1/3}(n \ln n)^{2/3}).$$

(The last inequality is because $D = \Omega(\sqrt{n \log n})$.)

Note that for $D = \Theta(\sqrt{n \log n})$, this estimate gives $T = O((n \log n)^{5/6})$, i.e., it agrees with the bound that we have in the small diameter regime. Also, this bound is sublinear in n as long as $D = o(n/\log^2 n)$.

We summarize this analysis with the next theorem.

Theorem 5.3. *Whp, our algorithm computes an exact shortest paths tree in the CONGEST model in time $O((n \log n)^{5/6})$, when $D = O(\sqrt{n \log n})$, and in time $O(D^{1/3}(n \log n)^{2/3})$, for larger D.*

6 MULTIPLE SOURCES

In this section, we extend our algorithm so that it will compute shortest paths between s sources $r_1, \ldots, r_s$ and all other vertices. The algorithm still builds the k-shortcut hopset $G'^{(k)}$ in the same way as in the single-source case. The hopbound is still $h' = O(N/k) = O(nq/k)$. The time required to construct it is, by Corollary 5.1, whp, $O(\frac{\log n}{q} \cdot k^2) + O(D + nqk)$.

We will assume that $s \leq nq$, and so the sources $\{r_1, \ldots, r_s\}$ can be added to V' without affecting its size by more than a constant factor.

Next, we conduct a B-F in $G' \cup G'^{(k)}$ from the s sources $r_1, \ldots, r_s$. As before, this B-F continues for $h' = O(nq/k)$ iterations, and each iteration consists of two parts. In the first part all vertices $v' \in V'$ upcast their s distance estimates via the BFS tree τ of G to the root of τ, and then these estimates are disseminated in the graph via pipelined broadcast in τ. This part requires $O(D + N \cdot s) = O(D + n \cdot q \cdot s)$ time.

In the second part of each iteration, a B-F in G to depth $O(\frac{\log n}{q})$ is conducted. Unlike the single-source variant of the algorithm, here every step of this B-F is a super-round that consists of s rounds. Every vertex v uses these s rounds to update its neighbors with the at most s estimates of its distances from the s sources. Hence this B-F requires $O(\frac{\log n}{q} \cdot s)$ time.

Thus, a single iteration of the B-F in $G' \cup G'^{(k)}$ requires $O(D + (nq + \frac{\log n}{q}) \cdot s)$ time, and overall this B-F requires

$$O(D + \left(nq + \frac{\log n}{q}\right) \cdot s) \cdot h' = O\left(\left(D + \left(nq + \frac{\log n}{q}\right) \cdot s\right) \cdot \frac{nq}{k}\right)$$

time.

Finally, on the last stage of the algorithm, vertices of $V \setminus V'$ learn their distances to the s sources via a B-F in G to depth $O(\frac{\log n}{q})$. Each step of this B-F requires now $O(s)$ rounds, as s estimates need to be proliferated. Hence the running time of this step is $O(\frac{\log n}{q} \cdot s)$.

To summarize, the running time of the entire algorithm becomes

$$T = O\left(\frac{\log n}{q} \cdot k^2 + D + nqk + \left(D + \left(nq + \frac{\log n}{q}\right) \cdot s\right) \cdot \frac{nq}{k} + \frac{\log n}{q} \cdot s\right). \tag{2}$$

To analyze this expression, we again consider two regimes. The first regime is when D is relatively small, i.e., $D \leq s(nq + \frac{\log n}{q})$. Here we set $q = \sqrt{\frac{\log n}{n}}$, $k = (n \log n)^{1/6} \cdot s^{1/3}$. The condition $k \leq nq$

holds. The running time becomes

$$T = O\left((n\log n)^{5/6}\cdot s^{2/3} + (n\log n)^{2/3}\cdot s^{1/3} + s\cdot\sqrt{n\log n}\cdot\frac{(n\log n)^{1/3}}{s^{1/3}} + \sqrt{n\log n}\cdot s\right)$$
$$= O((n\log n)^{5/6}\cdot s^{2/3}).$$

This bound is applicable as long as $D = O(s\sqrt{n\log n})$, $s \le nq = O(\sqrt{n\log n})$. (We will handle the case of larger s below.) Note that this bound is better than the only previously existing trivial bound $O(n\cdot s)$ in the entire range where it applies.

The second regime is when $D = \Omega(s(nq + \frac{\log n}{q}))$, i.e., in particular, $D = \Omega(s\sqrt{n\log n})$. By (2), the running time in this regime is given by $T = O(\frac{\log n}{q}\cdot k^2 + nqk + D\cdot\frac{nq}{k})$. We set $q = \frac{s\log n}{D} \le 1$ (as $D \ge s\log n$), and $k = (\frac{n\log n}{D})^{1/3}\cdot s^{2/3}$. Note that $k \le nq$ holds. We conclude that

$$T = O\left(\frac{\log n}{s\log n}\cdot D\cdot\left(\frac{n\log n}{D}\right)^{2/3}\cdot s^{4/3} + \frac{ns\log n}{D}\cdot\left(\frac{n\log n}{D}\right)^{1/3}\cdot s^{2/3} + D\cdot\frac{ns\log n\cdot D^{1/3}}{D\cdot(n\log n)^{1/3}\cdot s^{2/3}}\right)$$
$$= O((Ds)^{1/3}(n\log n)^{2/3} + \left(\frac{n\log n}{D}\right)^{4/3}\cdot s^{5/3}).$$

The second term is dominated by the first one when

$$D = \Omega(s^{4/5}(n\log n)^{2/5}). \tag{3}$$

Since we are in the regime that $D = \Omega(s\sqrt{n\log n})$, the condition (3) holds. Hence $T = O((Ds)^{1/3}\cdot(n\log n)^{2/3})$.

We summarize this result in the next theorem. (This is the case of a relatively small s, i.e., $s = O(\sqrt{n\log n})$.)

Theorem 6.1. *Whp, our algorithm computes exact shortest paths for pairs in $S\times V$, $|S| = s$, in the CONGEST model in time $O((n\log n)^{5/6}\cdot s^{2/3})$, whenever $D = O(s\sqrt{n\log n})$, $s = O(\sqrt{n\log n})$, and in time $O((Ds)^{1/3}(n\log n)^{2/3})$, for $D = \Omega(s\sqrt{n\log n})$. (In the latter case, in particular, it holds that $s = O(\sqrt{n/\log n})$.)*

Note that the two bounds agree when $D = \Theta(s\sqrt{n\log n})$. Also, this theorem generalizes Theorem 5.3. The second bound of Theorem 6.1 will be improved below (see Theorem 6.2).

In the case $s \ge \sqrt{n\log n}$, we set $q = s/n$, $k = \frac{s^{4/3}}{(n\log n)^{1/3}}$. Observe that $s \le nq = s$, and $k = \frac{s^{4/3}}{(n\log n)^{1/3}} \le nq = s$, for all $1 \le s \le n$. It also holds that $nq = s \ge \frac{\log n}{q} = \frac{\log n}{s}\cdot n$, i.e., $s \ge \sqrt{n\log n}$. Note also that $D = o(s(nq + \frac{\log n}{q}))$ in this case, as the right-hand side is $\omega(n)$.

Hence, by Equation (2), the running time is

$$T = O\left(\frac{\log n}{q}\cdot k^2 + nqk + s\cdot\frac{n^2q^2}{k}\right) = O((n\log n)^{1/3}\cdot s^{5/3}). \tag{4}$$

(Note that $nqk = \frac{s^{7/3}}{(n\log n)^{1/3}} \le s^{5/3}(n\log n)^{1/3}$, for all $1 \le s \le n$.)

This bound is not trivial (i.e., $o(ns)$) almost in the entire range of parameters, specifically, for $s = o(n/\sqrt{\log n})$. Observe that for $s = \Theta(\sqrt{n\log n})$, this bound agrees with the bound of Theorem 6.1, and gives running time $O((n\log n)^{7/6})$. A better bound for large s (i.e., $s = \Omega(\sqrt{n\log n})$) can be derived by partitioning the set S into $\alpha = \lceil s/\sqrt{n\log n}\rceil$ subsets $S_1, S_2, \ldots, S_\alpha$ of sizes $O(\sqrt{n\log n})$ each, and running this algorithm first for $S_1\times V$, then for $S_2\times V$, ..., and finally, for $S_\alpha\times V$. The overall running time of the algorithm becomes $O((n\log n)^{7/6}\cdot\frac{s}{\sqrt{n\log n}}) = O((n\log n)^{2/3}\cdot s)$. This

bound is no worse than the bound (4) in the entire range $s = \Omega(\sqrt{n \log n})$, and it is better than the bound in Equation (4) for $s = \omega(\sqrt{n \log n})$. We summarize this discussion below.

Theorem 6.2. *Whp, our algorithm computes exact shortest paths for $S \times V$, $|S| = s$, in the CONGEST model in time $O((n \log n)^{2/3} \cdot s)$, whenever $s = \Omega(\sqrt{n \log n})$, for all values of D.*

In particular, our algorithm computes *all-pairs shortest paths* in time $O(n^{5/3} \cdot \log^{2/3} n)$, for all values of D.

Note also that the bound of Theorem 6.2 outperforms the second bound of Theorem 6.1. Indeed, $(Ds)^{1/3} \cdot (n \log n)^{2/3} \leq s \cdot (n \log n)^{2/3}$ only if $D \leq s^2$. But the second bound of Theorem 6.1 applies only if $D = \Omega(s\sqrt{n \log n})$, i.e., $D = \Omega(\sqrt{Dn \log n})$. For the latter to hold, D needs to be $\Omega(n \log n)$, i.e., this never happens.

7 A STREAMING ALGORITHM

In this section, we present a variant of our algorithm for computing exact shortest paths in the multi-pass streaming setting. We start with undirected graphs, and then proceed to directed ones.

7.1 Undirected Graphs

Our algorithm for undirected graphs is closely related to Nanongkai's [45] $O(\sqrt{n})$-time SSSP algorithm for the broadcast congested clique model.

Fix an integer parameter k, $1 \leq k \leq n-1$. Every vertex v learns in one pass the k closest neighbors $u_1, u_2, \ldots, u_k$ of v in G, with ties broken by identity numbers. (If v happens to have degree smaller than k, then it learns all its neighbors.) This requires $O(n \cdot k)$ memory. As a result we obtain the *k-neighborhood graph* $\mathcal{N} = (V, F)$, $F = \{(v, u_i) \mid v \in V, i \in [k]\}$.

Lemma 7.1. *For a vertex $v \in V$, and $u \in S_G[k](v)$, any shortest path $P(v, u)$ between v and u in G with minimum number of hops is contained in the graph $\mathcal{N}$.*

Proof. Denote $P(v, u) = (v = v_0, v_1, \ldots, v_h = u)$, for some $h \leq k$. (Note that if $h > k$, then $u \notin S_G[k](v)$, as the vertices $v_1, v_2, \ldots, v_k$ would have been preferred over u. This is a contradiction to the assumption that $u \in S_G[k](v)$.) For every index $i \in [0, h-1]$, we argue that the vertex v_{i+1} is among k closest neighbors of v_i, and thus the edge (v_i, v_{i+1}) belongs to the edge set F of the k-neighborhood graph $\mathcal{N}$.

Indeed, otherwise there are neighbors $x_1, x_2, \ldots, x_k$ of v_i, which are all closer to v_i than v_{i+1}. (Alternatively, they may be at the same distance from v_i, but have smaller *Id*s than that of v_{i+1}.) But then all these vertices are closer to v than v_{i+1} as well, and consequently, they are also closer to v than $u = v_h$. (Note that $h \geq i + 1$.) This is, however, a contradiction to the assumption that $u \in S_G[k](v)$. Hence $(v_i, v_{i+1}) \in F$. □

Note that for the sake of the definition of $S_{\mathcal{N}}[k](v)$, the ties are broken in $\mathcal{N}$ the same way as in G.

Lemma 7.2. *For every $v \in V$, we have $S_{\mathcal{N}}[k](v) = S_G[k](v)$.*

Proof. First, we show that $S_G[k](v) \subseteq S_{\mathcal{N}}[k](v)$. Let $u \in S_G[k](v)$. By Lemma 7.1, any shortest $v - u$ path in G with minimum number of hops is contained in $\mathcal{N}$, and so, $d_{\mathcal{N}}(v, u) = d_G(v, u) < \infty$. Also, recall that $S_G[k](v)$ contains only vertices u reachable from v, and thus, such a path exists.

If $u \notin S_{\mathcal{N}}[k](v)$, then there are k vertices $\{y_1, \ldots, y_k\} = S_{\mathcal{N}}[k](v)$, $u \notin \{y_1, \ldots, y_k\}$, with $d_{\mathcal{N}}(v, y_i) \leq d_{\mathcal{N}}(v, u)$, and such that in case of equality, y_i is preferred over u in $\mathcal{N}$. But as $d_G(v, y_i) \leq d_{\mathcal{N}}(v, y_i)$, for every $i \in [k]$, we have

$$d_G(v, y_i) \leq d_{\mathcal{N}}(v, y_i) \leq d_{\mathcal{N}}(v, u) = d_G(v, u),$$

and in case of equality ($d_G(v, y_i) = d_G(v, u)$), the vertex y_i is preferred over u in G, too. But then $u \notin S_G[k](v)$, contradiction.

Next, we argue that $S_{\mathcal{N}}[k](v) \subseteq S_G[k](v)$.

If $|S_G[k](v)| = k$, then since $|S_{\mathcal{N}}[k](v)| \leq k$ and $S_G[k](v) \subseteq S_{\mathcal{N}}[k](v)$, it follows that the two sets are equal.

Otherwise $|S_G[k](v)| < k$. Let $u \in S_{\mathcal{N}}[k](v)$, and suppose for contradiction that $u \notin S_G[k](v)$. But u is reachable from v in $\mathcal{N}$, and so it is reachable from v in G, too, i.e., $d_G(v, u) < \infty$. However, when $|S_G[k](v)| < k$, it means that for every vertex $w \in V \setminus S_G[k](v)$, we have $d_G(v, w) = \infty$. This is a contradiction. Hence $S_{\mathcal{N}}[k](v) \subseteq S_G[k](v)$, proving the lemma. □

Corollary 7.3. *In one pass over the stream, using $O(nk)$ memory, one can compute the k-shortcut hopset $G^{(k)}$ via a deterministic algorithm.*

Proof. We saw that in one pass, using $O(nk)$ memory, one computes the k-neighborhood graph $\mathcal{N}$, and that for every $v \in V$, we have $S_G[k](v) = S_{\mathcal{N}}[k](v)$. Given the graph $\mathcal{N}$, we are now computing $\{S_{\mathcal{N}}[k](v) \mid v \in V\}$ offline (i.e., without any additional passes over the stream), and obtain as a result the sets $S_G[k](v)$, for every vertex $v \in V$. Recall that the edge set of the hopset $G^{(k)}$ is $\{(v, u) \mid u \in S_G[k](v), v \in V\}$. The algorithm has computed it. □

Recall that $G^{(k)}$ is an exact hopset with hopbound $h = O(n/k)$. So now we conduct h additional passes over G, and after each pass we relax also the edges of $G^{(k)}$. (In other words, we conduct Bellman-Ford in $G \cup G^{(k)}$ for $h = O(n/k)$ iterations.) Hence, after i passes, $0 \leq i \leq h$, we have computed i-limited distances in $G \cup G^{(k)}$ from a designated root vertex r to all other vertices. (For every vertex v, we store only its current distance estimate, and a parent through which this estimate was attained.) Hence, after $h = O(n/k)$ passes, we have computed $O(n/k)$-limited distances from the root r in $G \cup G^{(k)}$, which are equal (since $G^{(k)}$ is an exact hopset with hopbound $O(n/k)$) to exact $\{r\} \times V$ distances in G.

Observe that when computing hopset edges $\{(v, u) \mid v \in V, u \in S_{\mathcal{N}}[k](v) = S_G[k](v)\}$, the algorithm has also computed the shortest paths $P(v, u)$ in G implementing these hopset edges. We store them implicitly, i.e., for every vertex v, we store a shortest paths tree (SPT) of $S_{\mathcal{N}}[k](v) = S_G[k](v)$. (Note that its size is at most k.) Once we have computed the SPT in $G \cup G^{(k)}$ rooted at r, containing at most $n - 1$ edges, we can now replace every hopset edge of this tree with a path of length at most k, consisting of edges from G. In this way, we recover the shortest paths in G, while still employing $O(nk)$ memory.

Theorem 7.4. *For any n-vertex weighted undirected graph $G = (V, E)$, and any integer parameter k, $1 \leq k \leq n - 1$, our deterministic streaming $O(n/k)$-pass algorithm computes* exact *single-source shortest paths from a designated root r to all other vertices, using $O(nk)$ memory.*

Consider now the scenario that we want to compute s-SSP, i.e., shortest paths from s sources, for some $1 \leq s \leq n$. We denote the sources $r_1, \ldots, r_s$.

For a parameter k, we compute the hopset $G^{(k)}$ as described above, in $O(n/k)$ passes, using $O(nk)$ memory. We then conduct Bellman-Ford from designated sources in $G \cup G^{(k)}$, for $O(n/k)$ iterations, i.e., using $O(n/k)$ passes over the stream. (After each pass over the stream, we also scan again the hopset, stored in the local memory.) These passes use $O(n \cdot s)$ memory, i.e., $O(s)$ memory for every vertex, which it uses to store its s distance estimates. This is in addition to the memory used to store the hopset itself.

As a result, we obtain exact $S \times V$ distances in G, using $O(n(s + k))$ memory, in $O(n/k)$ passes. To obtain actual distances, we consider the s SPTs $\tau_1, \ldots, \tau_s$ in $G \cup G^{(k)}$, rooted at the s designated sources $r_1, \ldots, r_s$, respectively, which our algorithm has computed. We process these trees (offline)

one after another. We start by replacing every hopset edge in τ_1 by an actual path (containing at most k edges) in G. As a result we obtain a subgraph T_1' with $O(nk)$ edges, such that for every vertex $v \in V$, we have $d_{T_1'}(r_1, v) = d_G(r_1, v)$. We then compute an SPT T_1 of T_1' with respect to r_1, and erase T_1' from the local memory. Then we do the same with τ_2, then with τ_3, and so on.

We summarize this argument below.

Theorem 7.5. *For any n-vertex weighted undirected graph $G = (V, E)$, and any parameter k, $1 \leq k \leq n-1$, our deterministic streaming $O(n/k)$-pass algorithm computes exact s-source shortest paths, using $O(n(k+s))$ memory.*

A particularly useful setting of parameters here is $k = s$. Then we get $O(n/s)$ passes, $O(n \cdot s)$ memory, for the exact s-sources shortest paths problem. Generally, for a fixed s, it makes sense only to use this theorem with $k \geq s$.

7.2 Directed Graphs

In this section, we show that the results of Theorems 7.4 and 7.5 can be extended to directed graphs with positive and negative edge weights, at the expense of losing a logarithmic factor in the number of passes, and by using randomization. Note that the algorithm of Section 7.1 is deterministic.

We start with assuming that there are no cycles with negative weight in the graph. We will later show how to get rid of this assumption.

For a parameter k, $1 \leq k \leq n$, we sample every vertex $v \in V$ into the set V' of virtual vertices u.a.r. with probability k/n. We add the designated root vertex r into V'. (We consider the single-source case first.)

We conduct B-F explorations from all the vertices of V' in parallel, to depth $c \cdot \frac{n}{k} \cdot \ln n$, for a sufficiently large constant c. Denote $\Gamma = c \cdot \frac{n}{k} \cdot \ln n$. This involves Γ passes over the stream of edges, and for every vertex $v \in V$, on each iteration $i \leq \Gamma$, we store up to $|V'|$ current distance estimates $\{d_G^{(i)}(v', v) \mid v' \in V'\}$. Hence, we use expected $O(n \cdot k)$ memory for this step.

We obtain a virtual digraph $G' = (V', E')$, defined by

$$E' = \{\langle u', v' \rangle \mid v' \text{ is reachable from } u' \text{ via a } \Gamma - \text{limited path in } G\},$$

and with weight function $\omega'(\langle u', v' \rangle) = d_G^{(\Gamma)}(u', v')$.

Lemma 7.6. *Whp, for every $u', v' \in V'$, we have $d_{G'}(u', v') = d_G(u', v')$.*

Proof. Since any $u' - v'$ path in G' can be implemented by $u' - v'$ in G of the same length, it follows that $d_{G'}(u', v') \geq d_G(u', v')$. In the opposite direction, let $P = P(u', v')$ be a shortest $u' - v'$ path in G from the collection $\mathcal{P}$ of shortest paths (see Section 3), and denote $h = |P|$. If $h \leq \Gamma$, then there is an arc $\langle u', v' \rangle \in E'$ of weight $\omega'(\langle u', v' \rangle) = \omega(P(u', v')) = d_G(u', v')$, and so $d_{G'}(u', v') \leq d_G(u', v')$.

Otherwise, with high probability, there are virtual vertices $u' = u_0', u_1', \ldots, u_{q-1}', u_q' = v'$ on the path P, such that the hop-distance in P (and so in G, too) between every pair of these consecutive virtual vertices is at most Γ. Hence all the arcs $\langle u_0', u_1' \rangle, \langle u_1', u_2' \rangle, \ldots, \langle u_{q-1}', u_q' \rangle \in E'$, and for each index $i \in [0, q-1]$, we have $\omega'(\langle u_i', u_{i+1}' \rangle) = d_G(u_i', u_{i+1}')$. Hence the path $\langle u' = u_0', u_1', \ldots, u_q' = v' \rangle$ is contained in G', and its length is $\omega(P) = d_G(u', v')$. Hence $d_{G'}(u', v') \leq d_G(u', v')$, whp. □

Observe also that for every virtual vertex $v' \in V'$ and for every vertex $v \in V \setminus V'$, we store all the distances $d_G(v', v)$, such that v is reachable from v' via a Γ-limited path.

For a virtual vertex $u' \in V'$, and a vertex $v \in V \setminus V'$, which is not reachable from u' via a Γ-limited path, note that, whp, the shortest $u' - v'$ path P in G contains a virtual vertex v' within Γ hops from v, and moreover, $d_{G'}(u', v) = d_G(u', v') + d_G(v', v)$.

To complete the algorithm, we compute offline all distances $r - v'$ in G', for all $v' \in V'$. To store the virtual graph itself, we need only $O(k^2) = O(nk)$ memory. Now, for every vertex v, we select the vertex $v' \in V'$ such that v is reachable from v' via a Γ-limited path, that minimizes $d_G(r, v') + d_G(v', v)$. By the above argument, this is equal to $d_G(r, v)$.

If we are interested in distances from s designated sources $r_1, \ldots, r_s$, then we use $k \geq s$, and include all these sources in V'. In the last step of the algorithm, instead of computing the distance from r to v (for every $v \in V$), we do it for every r_i, $i \in [s]$. The number of passes and the memory requirement of the algorithm remain unchanged.

To retrieve the actual paths, we store for every vertex v not just the distance estimate from each virtual vertex v' whose B-F exploration reached v, but also the arc $\langle u, v \rangle \in E$ connecting v to the parent u through which the exploration of v' reached v. (This still requires an expected $O(n \cdot k)$ memory.)

Also, when computing an SPT T' in G' rooted at r (or SPTs T'_i rooted at r_i, for every $i \in [s]$, in the case of multiple sources), we store for every virtual vertex v', its parent u' in T'. Also, for every incoming arc $\langle u', v' \rangle \in E'$, the virtual vertex v' stores its incoming G-parent $u \in V$. This enables us to retrieve the actual shortest path trees in G, while maintaining the same guarantees on space and the number of phases.

Consider now the case that negative-weight cycles may appear in the graph G. Consider the negative weight cycle $C = \langle u_0, u_1, \ldots, u_{q-1}, u_q = u_0 \rangle$ in G with minimum number of hops, reachable from the root r. (In the case of multiple sources, the cycle needs to be reachable from one of the sources.)

If $q < \Gamma$, then let v' denote the closest selected (aka virtual) vertex to a vertex of C. Suppose wlog that $d_G(v', C) = d_G(v', u_0)$. Then the distance estimate of u_0 will keep decreasing after Γ iterations of the B-F from the selected vertices. Thus, to detect such cycles C, we modify the algorithm so that it will conduct 2Γ iterations of B-F, instead of Γ ones. However, we will only keep Γ-limited distance estimates. However, in the additional Γ iterations we will check if an estimate of some vertex $v \in V$ decreases below $d^{(\Gamma)}(r, v)$. If this happens, then the algorithm reports that the graph contains a negative-weight cycle.

Indeed, in the cycle C as above, the distance estimate of u_0 will necessarily decrease in one of these additional Γ iterations of B-F. However, if the graph contains no negative-weight cycle, then as we have seen, whp, for every vertex $v \in V$, its distance estimate after Γ iterations is already equal to $d_G(r, v)$, and thus it will not decrease any more. Hence, whp, no negative-weight cycle will be reported, if there is no such a cycle in G.

If $q \geq \Gamma$, then we replace the path $\langle u_0, \ldots, u_{q-1} \rangle$ in the cycle C by the shortest q-limited path P from u_0 to u_{q-1}, such that $P \in \mathcal{P}$. (Note that this quantity is well-defined, even when negative-weight cycles are present.) We obtain a possibly different negative-weight cycle $\tilde{C} = \langle u_0, u'_1, u'_2, \ldots, u'_{q-2}, u_{q-1}, u_q = u_0 \rangle$, with the same number of hops. (Recall that C is the negative-weight cycle with minimum number of hops.)

Moreover, whp, every Γ hops in the cycle $\tilde{C}$ contain a selected vertex $v' \in V'$. There are two possibilities. Either the entire cycle has length $q \leq 2\Gamma$, and then it may contain just one selected vertex. If this is the case, then the estimate of v' will decrease when we will conduct the additional Γ iterations of B-F, and the algorithm will report that there is a negative-weight cycle. However, if $q > 2\Gamma$, then, whp, the cycle contains at least two selected vertices $v'_1, v'_2 \in V'$, and moreover, the selected vertices $v'_1, v'_2, \ldots, v'_p$, for some $p \geq 2$, in $\tilde{C}$ form a cycle C' in G'. (Because, whp, all hop-distances between consecutive selected vertices on the cycle $\tilde{C}$ are at most Γ.) It follows that there is a negative-weight cycle $C' = \langle v'_1, v'_2, \ldots, v'_p, v'_1 \rangle$ in G'. This cycle will be detected offline by a B-F exploration in G'.

Hence, whp, if the graph G contains a negative-weight cycle, then it will be detected by the algorithm; however, when there is no negative-weight cycle, no distance estimate will decrease within the additional Γ iterations of B-F, and thus the algorithm will not report that there is a negative-weight cycle.

We summarize this analysis below.

Theorem 7.7. *For any n-vertex directed weighted graph $G = (V, E)$, with possibly negative edge weights, and any s designated sources $r_1, \ldots, r_s$, and any parameter k, $s \leq k \leq n-1$, whp, our randomized algorithm computes exact s-sources shortest paths using $O(\frac{n}{k} \log n)$ passes and expected $O(nk)$ space. If there is a negative-weight cycle reachable from one of the sources, then the algorithm (whp) reports one, and, whp, when there is no negative-weight cycle, the algorithm reports that there is no such a cycle, and computes shortest paths.*

APPENDICES

A SOME PROOFS

Proof of Theorem 3.1: For a pair $u, v \in V$ of vertices, let $\pi(u, v) = (u = x_0, x_1, \ldots, x_\ell = v)$ be the shortest $u - v$ path with the smallest number of hops in $G \cup G^{(k)} = (V, E \cup H, \hat{\omega})$, $\hat{\omega}(e) = \omega^{(k)}(e)$ for $e \in H$, and $\hat{\omega}(e) = \omega(e)$ for $e \in E \setminus H$. (We assume that $\pi(u, v)$ has finite length, i.e., for every index i, $0 \leq i \leq \ell - 1$, we have $\hat{\omega}(x_i, x_{i+1}) < \infty$.)

We argue that for any index i, $0 \leq i \leq \ell/4 - 1$, we have $S_G[k](x_{4i}) \cap S_G[k](x_{4i+4}) = \emptyset$.

Observe that $x_{4i+2} \notin S_G[k](x_{4i})$, because otherwise we could obtain a shortest $u - v$ path in $G \cup G^{(k)}$ with fewer edges than in $\pi(u, v)$ by replacing the two edges $(x_{4i}, x_{4i+1}), (x_{4i+1}, x_{4i+2})$ in $\pi(u, v)$ by (x_{4i}, x_{4i+2}). Analogously, $x_{4i+2} \notin S_G[k](x_{4i+4})$, too.

Since $d_G(x_{4i}, x_{4i+2}), d_G(x_{4i+2}, x_{4i+4}) < \infty$, it follows that $|S_G[k](x_{4i})| = |S_G[k](x_{4i+4})| = k$.

Hence for any $y \in S_G[k](x_{4i})$,

$$d_G(x_{4i}, y) \leq d_G(x_{4i}, x_{4i+2}). \tag{5}$$

Also, for any $y \in S_G[k](x_{4i+4})$,

$$d_G(x_{4i+4}, y) \leq d_G(x_{4i+4}, x_{4i+2}). \tag{6}$$

So, if $S_G[k](x_{4i}) \cap S_G[k](x_{4i+4}) \neq \emptyset$, then there exists a vertex y that satisfies both Equations (5) and (6). But then one could get a shortest $u - v$ path in $G \cup G^{(k)}$ with fewer edges than in $\pi(u, v)$ by replacing the four edges of the subpath $(x_{4i}, \ldots, x_{4i+4})$ of $\pi(u, v)$ by the two edges $(x_{4i}, y), (y, x_{4i+4})$, contradiction.

Consider the disjoint union $\bigcup_{i=0}^{\ell'} S_G[k](x_{4i})$, for the maximum ℓ' such that $4\ell' \leq \ell$. It contains $(\ell' + 1)k \leq n$ vertices. (One can assume that G is connected, and thus, for every vertex $v \in V$, we have $|S_G[k](v)| = k$.) Thus $4n/k \geq 4(\ell' + 1) > \ell$, as required. □

Proof of Lemma 3.2: Let $\Gamma(v) = \{u_1, u_2, \ldots, u_d\}$, $d = deg(v)$, denote the neighborhood of v (in G). In what follows we write $d^{(i)}$ as a shortcut for $d_G^{(i)}$. In super-round i, the vertex v learns from its neighors the sets ${S'}_G^{(i)}[k](u_1), \ldots, {S'}_G^{(i)}[k](u_d)$. Consider a tuple $\sigma = (x, d^{(i+1)}(x, v), h^{(i+1)}(x, v), u) \in {S'}_G^{(i+1)}[k](v)$, $u \in \Gamma(v)$, and let $\pi(x, v)$ be the shortest $(i+1)$-limited $x - v$ path, $\omega(\pi(x, v)) = d^{(i+1)}(x, v)$, $|\pi(x, v)| = h^{(i+1)}(x, v)$.

We argue that this tuple is in $\mathcal{S}^{(i+1)}(v)$ as well. To this end, we first show that the tuple $\sigma_u = (x, d^{(i)}(x, u), h^{(i)}(x, u), p(u))$, where $p(u)$ is the predecessor of u in $\pi(x, v)$, must belong to $\mathcal{S}^{(i)}(u) = {S'}_G^{(i)}[k](u)$. (The equality is by the induction hypothesis.)

Indeed, suppose for contradiction that

$$S'^{(i)}_G[k](u) = \{(x_1, d^{(i)}(x_1,u), h^{(i)}(x_1,u), p_1(u)), \ldots, (x_k, d^{(i)}(x_k,u), h^{(i)}(x_k,u), p_k(u))\}$$

does not contain the tuple σ_u, where for every j, $1 \le j \le k$, $p_j(u)$ is the predecessor of u in $\pi(x_j,u)$.

Then, for each j, $1 \le j \le k$, either $d^{(i)}(x_j,u) < d^{(i)}(x,u)$, or $d^{(i)}(x_j,u) = d^{(i)}(x,u)$ but $h^{(i)}(x_j,u) < h^{(i)}(x,u)$, or both $d^{(i)}(x_j,u) = d^{(i)}(x,u)$ and $h^{(i)}(x_j,u) = h^{(i)}(x,u)$, but the path $\pi(u,x_j)$ is lexicographically smaller than $\pi(u,x)$, i.e., $\pi(u,x_j) <_u \pi(u,x)$.

Denote $\pi(x_j,v) = \pi(x_j,u) \circ (u,v)$, for every $1 \le j \le k$. (Here $\circ$ stands for concatenation.) It follows that for each j, $1 \le j \le k$, the tuple $(x_j, d^{(i+1)}(x_j,v), h^{(i+1)}(x_j,v), u)$ is better than $(x, d^{(i+1)}(x,v), h^{(i+1)}(x,v), u)$ (with respect to $<_v$). This is a contradiction to the assumption that $\sigma \in S'^{(i+1)}_G[k](v)$.

Hence $\sigma_u = (x, d^{(i)}(x,u), h^{(i)}(x,u), p(u)) \in S'^{(i)}_G[k](u) = \mathcal{S}^{(i)}(u)$. (The equality is by the induction hypothesis.) Thus, the vertex v receives this tuple from u on super-round i. As a result, the vertex v computes the tuple $\hat{\sigma} = (x, \delta^{(i+1)}_u(x,v), h^{(i+1)}_u(x,v), u)$, with $(\delta^{(i+1)}_u(x,v), h^{(i+1)}_u(x,v)) = (d^{(i+1)}(x,v), h^{(i+1)}(x,v))$. (Note that $\hat{\sigma} = \sigma$.)

Consider a comparison that v conducts between $\sigma = \hat{\sigma}$ and some other tuple

$$\sigma' = (x', d^{(i+1)}(x',v), h^{(i+1)}(x',v), p'(v)) = \hat{\sigma}' = \left(x', \delta^{(i+1)}_{p'(v)}(x',v), h^{(i+1)}_{p'(v)}(x',v), p'(v)\right),$$

where $p'(v)$ is the parent of v in this tuple. (The vertex v has computed $\hat{\sigma}'$ in this super-round.) Assume that $\sigma <_v \sigma'$, i.e., that $(d^{(i+1)}(x,v), h^{(i+1)}(x,v)) < (d^{(i+1)}(x',v), h^{(i+1)}(x',v))$, or the two pairs are equal, but $\pi(x,v) <_v \pi(x',v)$, where $\pi(x,v)$ (respectively, $\pi(x',v)$) is the shortest $(i+1)$-limited $x - v$ (respectively, $x' - v$) path with the smallest number of hops.

If $x = x'$, then v received this tuple from two distinct neighbors u and u', and it prefers σ, because either $(\delta^{(i+1)}_u(x,v), h^{(i+1)}_u(x,v)) < (\delta^{(i+1)}_{u'}(x,v), h^{(i+1)}_{u'}(x,v))$, or the two pairs are equal but $Id(u) < Id(u')$. (Otherwise there were a contradiction to $\sigma <_v \sigma'$.)

So we assume that $x \ne x'$. If $(d^{(i+1)}(x,v), h^{(i+1)}(x,v)) < (d^{(i+1)}(x',v), h^{(i+1)}(x',v))$, then obviously v prefers σ over σ'. So it remains only to consider the case that the two pairs are equal. Denote by $u = p(v)$ and $u' = p'(v)$ the predecessors of v in $\pi(x,v)$ and $\pi(x',v)$, respectively. If $u \ne u'$, then $Id(u) < Id(u')$ (recall that $\sigma <_v \sigma'$), and so v prefers σ over σ' (as it knows both u and u'). Finally, consider the case that $u = u'$. Then u has sent both the tuples σ_u and σ'_u to v, with an indication that $\sigma_u <_u \sigma'_u$. (The tuples σ and σ' are computed by v from the tuples σ_u and σ'_u, respectively.) So v concludes that $\sigma <_v \sigma'$ in this case, too.

To summarize, based on the information that v receives, it can compare the tuples correctly. Since it receives all the tuples of $S'^{(i+1)}_G[k](v)$ (along with possibly some other tuples), it therefore computes correctly the set of k smallest tuples with respect to $<_v$, i.e., the set $\mathcal{S}^{(i+1)}(v)$ that it computes is equal to $S'^{(i+1)}_G[k](v)$. □

Proof of Lemma 3.3: If $\omega(P^{(\ell)}(x,y)) < \omega(P'^{(\ell)}(x,y))$, then $P'^{(i')}(u',v')$ is not the shortest i'-bounded $u' - v'$ path, contradiction. Hence $\omega(P^{(\ell)}(x,y)) = \omega(P'^{(\ell)}(x,y))$.

Write $P^{(\ell)}(x,y) = (x = x_0, x_1, \ldots, x_\ell = y)$, $P'^{(\ell)}(x,y) = (x = x'_0, x'_1, \ldots, x'_\ell = y)$. Let $h < \ell$ be the largest index such that $x_h \ne x'_h$. (Higher-index vertices coincide.) Assume without loss of generality (henceforth, we will write "wlog") that $Id(x_h) < Id(x'_h)$. Replace $P'^{(\ell)}(x,y)$ in $P'^{(i')}(u',v')$ by $P^{(\ell)}(x,y)$. We obtain a shortest i-limited $u' - v'$ path with weight at most $\omega(P'^{(i')}(u,v))$, but which is lexicographically smaller. This is a contradiction to the assumption that $P'^{(i')}(u,v) \in \mathcal{P}$. (Recall that $\mathcal{P}$ was obtained via B-F explorations that return lexicographically smallest paths, when considering them from tail to head.) □

Proof of Lemma 3.4: For a single path P, $|P| \geq \sqrt{n}$, the probability for it not to contain a vertex of V' as an internal vertex is at most

$$\left(1 - \frac{c \ln n}{\sqrt{n}-1}\right)^{|V(P)|-2} = \left(1 - \frac{c \ln n}{\sqrt{n}-1}\right)^{|P|-1} \leq \left(1 - \frac{c \ln n}{\sqrt{n}-1}\right)^{\sqrt{n}-1} \leq n^{-c}.$$

By union-bound, the probability for some $P \in \mathcal{P}$ with $|P| \geq \sqrt{n}$ not to contain an internal selected vertex is at most $n^{-(c-3)}$. Hence the assertion of the lemma holds with probability at least $1 - n^{-(c-3)}$. □

Proof of Lemma 3.10: Consider a vertex $u' \in S'_{G'}[k](v')$. Let $\pi_G(u', v')$ be the shortest $u' - v'$ path in G with the smallest number of hops. Let $\pi'(u', v') = (u' = u'_0, u'_1, \ldots, u'_h = v')$ be the vertices of $V' \cap \pi_G(u', v')$, in the order of their appearrence on $\pi_G(u', v')$. By a previous argument, for every $i \in [0, h-1]$, $(u'_i, u'_{i+1}) \in E'$. (This can be seen by repeatedly applying Lemma 3.4 to $\pi_G(u', v')$. First, we consider a subpath $(u' = u^{(0)}, u^{(1)}, \ldots, u^{(\lceil\sqrt{n}\rceil)})$ of the first $\lceil\sqrt{n}\rceil$ edges from G of $\pi_G(u, v)$. By Lemma 3.4, this subpath contains an internal vertex $u'_1 = u^{(j_1)} \in V'$, for some $j_1 \leq \sqrt{n}$. Then we consider a $\lceil\sqrt{n}\rceil$-long subpath of $\pi_G(u, v)$ that starts at u'_1. By applying to it Lemma 3.4 again, we obtain u'_2, etc.)

Observe that $h - 1 < k$, as there are $h - 1$ vertices $u'_1, \ldots, u'_{h-1} \in V'$, which are all closer to v' than u' in G'. (Or, if zero weights are allowed, then some of the u'_i may be at the same distance from v' as u', but the number of hops in the shortest $v' - u'$ path in G with the smallest number of hops is larger than in the respective path between v' and u'_i.) Hence if $h - 1 \geq k$, then $u' \notin S'_{G'}[k](v')$, contradiction.

Hence u' is reachable from v' in G' within $h \leq k$ hops, and moreover, $d^{(k)}_{G'}(u', v') = d_{G'}(u', v') < \infty$. (Recall that $u' \in S'_{G'}[k](v')$, and the latter set contains only vertices reachable from v' in G'.)

If $u' \notin S'^{(k)}_{G'}[k](v')$, then since $d^{(k)}_{G'}(u', v') < \infty$, it follows that there exist vertices $u'_1, \ldots, u'_k \in V'$, $u' \notin \{u'_1, \ldots, u'_k\} = S'^{(k)}_{G'}[k](v')$, such that for every $i \in [k]$,

$$d^{(k)}_{G'}(v', u'_i) \leq d^{(k)}_{G'}(v', u') = d_{G'}(v', u') < \infty.$$

Also, $d_{G'}(v', u'_i) \leq d^{(k)}_{G'}(v', u'_i)$, i.e., $d_{G'}(v', u'_i) \leq d_{G'}(v', u')$, for every $i \in [k]$. Moreover, if there is an equality $(d_{G'}(v', u'_i) = d_{G'}(v', u'))$, then $\pi_G(v', u'_i) <_{v'} \pi_G(u', v')$.

But this is a contradiction to the assumption that $u' \in S'_{G'}[k](v')$. Hence

$$S'_{G'}[k](v') \subseteq S'^{(k)}_{G'}[k](v'). \tag{7}$$

Now we prove the opposite direction, i.e., $S'^{(k)}_{G'}[k](v') \subseteq S'_{G'}[k](v')$.

If $|S'_{G'}[k](v')| = k$, then Equation (7) implies that the two sets are equal, because $|S'^{(k)}_{G'}[k](v')| \leq k$.

So assume that $|S'_{G'}[k](v')| < k$. Hence for every $u' \notin S'_{G'}[k](v')$, we have $d_{G'}(v', u') = \infty$, and thus $d^{(k)}_{G'}(v', u') \geq d_{G'}(v', u') = \infty$ as well. Suppose for contradiction that there exists a vertex $u' \in S'^{(k)}_{G'}[k](v') \setminus S'_{G'}[k](v')$. Then $u' \notin S'_{G'}[k](v')$ implies $d^{(k)}_{G'}(v', u') \geq d_{G'}(v', u') = \infty$, and thus $u' \notin S'^{(k)}_{G'}[k](v')$ as well, contradiction. (Recall that, by definition, the latter set contains only reachable vertices.) □

B UPCAST

Consider a problem in which we are given a tree τ of hop-diameter D, rooted at a vertex rt, and some m distinct messages distributed in its vertices. (A single vertex may hold more than one message.) We want to *upcast* all these messages to the root rt of τ in the $CONGEST(b \log n)$ model.

When $b = 1$, this problem was analyzed in Reference [48], Chapter 4, where it was shown that this task can be performed in $O(D + m)$ time. Here we extend this result to a general b, and show that in general, $O(D + m/b)$ time is sufficient. The argument follows closely that of Reference [48]; it is provided here for the sake of completeness.

The algorithm is a trivial one: whenever a vertex v has some q messages that it still did not send to its parent $p(v) = w$, it sends some arbitrary $\min\{q, b\}$ of them to w (assuming that the edge (v, w) is available for sending messages at this point).

For a vertex v, let $M(v)$ denote the set of messages initially stored in the subtree τ_v rooted at v. Let $m(v) = |M(v)|$. Let $\hat{L}(v)$ denote the depth of τ_v, i.e., the maximum hop-distance between a leave z in τ_v and its root v.

We start from the following simple lemma.

Lemma B.1. *Given a vertex v and two positive integers t, h,* suppose *that for all i, $1 \leq i \leq h$, after $t + i$ rounds, the vertex v has at least $\min\{i \cdot b, m(v)\}$ messages. Then after $t + h + 1$ rounds, the parent $w = p(v)$ of v has $\min\{h \cdot b, m(v)\}$ messages received from v.*

The proof of this lemma is by a straightforward induction on h. We omit it.

Lemma B.2. *For any vertex w, and for all i, $1 \leq i \leq \lceil \frac{m(w)}{b} \rceil$, after $\hat{L}(w) + i - 1$ round, at least $\min\{m(w), b \cdot i\}$ messages are at w.*

Proof. The proof is by induction on $\hat{L}(w)$. The base case is when w is a leaf; it is immediate.

Step: Suppose that the lemma holds for every child v_j of w. Denote $\ell = \hat{L}(w)$, $\ell_j = \hat{L}(v_j)$, and $m_j = m(v_j)$, for all $1 \leq j \leq p$, where p is the number of children of w. Define $\gamma_j = \min\{i, \lceil \frac{m_j}{b} \rceil\} \leq \lceil \frac{m_j}{b} \rceil$.

By the induction hypothesis for v_j, for every index i', $1 \leq i' \leq \lceil \frac{m_j}{b} \rceil$, after $\ell_j + i' - 1 = (\ell_j - 1) + i'$ rounds, v_j has $\min\{i' \cdot b, m_j\}$ items.

Use Lemma B.1 with $t = \ell_j - 1$, $h = \gamma_j$. Since $\gamma_j \leq \lceil \frac{m_j}{b} \rceil$, the assumption of Lemma B.1 holds with these t and h. Hence after $(\ell_j - 1) + (\gamma_j + 1) = \ell_j + \gamma_j$ rounds, the parent w of v_j has received at least $\min\{\gamma_j \cdot b, m(v_j)\} = m_j$ messages from v_j.

If there exists a child v_j of w with $\gamma_j \cdot b \leq m_j$, then w has received just from this child v_j at least $\gamma_j \cdot b \geq i \cdot b \geq \min\{m(w), i \cdot b\}$ messages by time

$$\ell_j + \gamma_j \leq \hat{L}(w) - 1 + \gamma_j \leq \hat{L}(w) - 1 + \left\lceil \frac{m_j}{b} \right\rceil \leq \hat{L}(w) - 1 + \left\lceil \frac{m(w)}{b} \right\rceil,$$

as required.

Otherwise, every child v_j of w satisfies $\gamma_j \cdot b > m_j$. Hence $\gamma_j = \min\{i, \lceil \frac{m_j}{b} \rceil\} > \frac{m_j}{b}$. It follows that $i \geq \lceil \frac{m_j}{b} \rceil = \min\{i, \lceil \frac{m_j}{b} \rceil\} = \gamma_j$.

Then, after $\max\{\ell_j + \gamma_j\} \leq \max\{\ell_j\} + \max\{\gamma_j\} \leq \ell - 1 + i$ rounds, the parent w has received from each v_j at least $\min\{m_j, b \cdot \lceil \frac{m_j}{b} \rceil\} = m_j$ messages. Hence, by this time, w has received all $m(w)$ messages from all its children. (Except for the messages that were originally stored at w, and they are kept being stored there.) □

Using the lemma with $w = r$ and $i = \lceil m(r)/b \rceil$, we conclude that all messages are collected at the root within $D + \lceil m(rt)/b \rceil$ rounds. They can, of course, be also disseminated to all vertices of the graph via an analogous pipelined broadcast (see Reference [48], Chapter 4) within the same time.

C LARGE BANDWIDTH

In this section we analyze our algorithm in the *CONGEST*$(b \log n)$ model, where $1 \leq b \leq n$ is the bandwidth parameter, i.e., when every edge can relay up to b messages of size $O(\log n)$ each round.

We start with the single-source case (Section C.1), and then analyze the case of multiple sources (Section C.2).

We remark that the benchmark exact SPT algorithm in the $CONGEST(b \log n)$ model is still the Bellman-Ford, which requires $O(n)$ time for single source, and $O(n\lceil \frac{s}{b} \rceil)$ for s sources. Note that if really huge messages of $b = O(|E|)$ size are allowed, one can solve the problem (as well as any other problem) in $O(D)$ time by collecting the entire topology in a single vertex, solving the problem locally, and disseminating the solution. For more general bandwidth parameter b, the same solution works in $O(D + |E|/b)$ time, for the single-source case, and in $O(D + (|E| + ns)/b)$ time, for the case of s sources. This solution is, however, generally very problematic, as it requires heavy local computation.

C.1 Single Source

Consider the construction of the k-shortcut hopset $G'^{(k)}$ via a B-F in G to depth $O(\frac{\log n}{q} \cdot k)$, where every vertex v forwards k smallest estimates of its distances from vertices of V' that it knows in every super-round. When the bandwidth is b (words, i.e., $O(b \log n)$ bits), each super-round can be implemented in $\lceil k/b \rceil$ time, i.e., this step requires $O(\frac{\log n}{q} \cdot k\lceil \frac{k}{b} \rceil)$ time.

The upcast and pipelined broadcast over the BFS tree τ of G to disseminate the hopset $G'^{(k)}$ can be now implemented in $O(D + \frac{nqk}{b})$ time. (We disseminate $m = nqk$ messages over a tree of depth D, with edges of bandwidth b. See Lemma B.2 in Appendix B.)

The B-F in $G' \cup G'^{(k)}$ is conducted for $h' = O(\frac{nq}{k})$ iterations. The first part of every iteration involves dissemination of $O(n \cdot q)$ estimates of distances of virtual vertices from the designated root vertex r over the BFS tree τ. This step requires $O(D + \lceil \frac{nq}{b} \rceil)$ time. The second part of every iteration is a B-F in G to depth $O(\frac{\log n}{q})$ from vertices of V'. Each vertex forwards just one single smallest estimate of distance from r that it knows. Here the larger bandwidth does not help, and this step still requires $O(\frac{\log n}{q})$ time, exactly as in the unit-bandwidth case.

Hence the overall running time is

$$T = O\left(\left(D + \left\lceil \frac{nq}{b} \right\rceil + \frac{\log n}{q}\right) \cdot \frac{nq}{k} + \frac{\log n}{q} \cdot k \cdot \left\lceil \frac{k}{b} \right\rceil + \frac{nqk}{b}\right). \tag{8}$$

In this expression we always require $k \le nq$. We will also have $b \le nq$, because the last term nqk/b is typically dominated by other terms.

For the range of small D, i.e., $D = O(\sqrt{\frac{n \log n}{b}})$, we will use two settings of the parameters, depending on the value of b. In both settings we will set $q = \sqrt{\frac{b \log n}{n}}$. When b is relatively small, i.e., $b \le (n \log n)^{1/3}$, we set $k = (n \log n)^{1/6} \cdot b^{1/2}$. We have $b \le k = (n \log n)^{1/6} \cdot b^{1/2}$, and $k = (n \log n)^{1/6} \cdot b^{1/2} \le nq = \sqrt{n \log n \cdot b}$. The running time of the algorithm becomes

$$T = O\left(\left(\frac{nq}{b} + \frac{\log n}{q}\right) \cdot \frac{nq}{k} + \frac{\log n}{q} \cdot \frac{k^2}{b} + \frac{nqk}{b}\right) = O\left(\frac{(n \log n)^{5/6}}{b^{1/2}}\right). \tag{9}$$

(Note that $\frac{nq}{b} \cdot k = \sqrt{\frac{n \log n}{b}} \cdot (n \log n)^{1/6} \cdot b^{1/2} = (n \log n)^{2/3} \le \frac{(n \log n)^{5/6}}{b^{1/2}}$, for $b \le (n \log n)^{1/3}$.)

When $b \ge (n \log n)^{1/3}$, we set $k = \sqrt{nq} = (n \log n \cdot b)^{1/4}$. Here $k \le b \le nq$. Indeed, $(n \log n \cdot b)^{1/4} \le b \le \sqrt{n \log n \cdot b}$, for this range of b. (Recall that we assume that b is at most linear in n.) Hence, by (8), the running time is (as $\lceil \frac{k}{b} \rceil = 1$)

$$T = O\left(\left(\frac{nq}{b} + \frac{\log n}{q}\right) \cdot \frac{nq}{k} + \frac{\log n}{q} \cdot k + \frac{nqk}{b}\right).$$

Then

$$T = O\left(\frac{(n\log n)^{3/4}}{b^{1/4}}\right). \tag{10}$$

Observe that the two bounds (9) and (10) agree when $b = \Theta((n\log n)^{1/3})$ and are equal to $O((n\log n)^{2/3})$. Also, for $b \approx n$, the bound (10) gives time $\tilde{O}(\sqrt{n})$.

For the case of large D, i.e., $D \geq \sqrt{\frac{n\log n}{b}}$, we will assume $b \leq k \leq nq$. Assuming $D \geq \max\{\lceil\frac{nq}{b}\rceil, \frac{\log n}{q}\}$, by Equation (8), we then have

$$T = O\left(D\cdot\frac{nq}{k} + \frac{\log n}{q}\cdot\frac{k^2}{b} + \frac{nqk}{b}\right).$$

We set $q = \frac{\log n}{D}$, $k = (\frac{n\log n\cdot b}{D})^{1/3}$. Note that it indeed holds that $D \geq \max\{\lceil\frac{nq}{b}\rceil, \frac{\log n}{q}\}$. The condition $k \leq nq$ implies $D \leq \frac{n\log n}{b^{1/2}}$, i.e., $b \leq (\frac{n\log n}{D})^2$. The condition $b \leq k$ implies $b \leq (\frac{n\log n}{D})^{1/2}$, i.e., $D \leq \frac{n\log n}{b^2}$. The time becomes

$$T = O\left(\left(\frac{D}{b}\right)^{1/3}(n\log n)^{2/3}\right), \tag{11}$$

assuming $b \leq (n\log n)^{1/3}$. (Observe that $nqk/b = ((n\log n)/D)^{4/3}/b^{2/3} \leq (D/b)^{1/3}(n\log n)^{2/3}$ iff $D \geq (n\log n)^{2/5}/b^{1/5}$. However, $D \geq \sqrt{\frac{n\log n}{b}}$, and $\sqrt{\frac{n\log n}{b}} \geq \frac{(n\log n)^{2/5}}{b^{1/5}}$, for $b \leq (n\log n)^{1/3}$.) Also, the conditions $D \geq \sqrt{\frac{n\log n}{b}}$ and $b \leq (\frac{n\log n}{D})^{1/2}$ imply $D \geq (n\log n\cdot D)^{1/4}$, i.e., $D \geq (n\log n)^{1/3}$.

We summarize this analysis in the next theorem.

Theorem C.1. *Our algorithm computes single-source exact SPT in CONGEST$(b\log n)$ model in time:*

1. *If $b \leq (n\log n)^{1/3}$ and $D = O(\sqrt{\frac{n\log n}{b}})$, then in $O(\frac{(n\log n)^{5/6}}{b^{1/2}})$ time.*
2. *If $b \geq (n\log n)^{1/3}$ and $D = O(\sqrt{\frac{n\log n}{b}})$, then in $O((\frac{(n\log n)^{3/4}}{b^{1/4}})$ time. In particular, for $b = \Theta(n)$, $D = O(\sqrt{\log n})$, the running time is $\tilde{O}(\sqrt{n})$.*
3. *If $b \leq (n\log n)^{1/3}$ and $D \geq (n\log n)^{1/3}$, and also $\sqrt{\frac{n\log n}{b}} \leq D \leq \frac{n\log n}{b^2}$, then in $O((D/b)^{1/3}(n\log n)^{2/3})$ time. In particular, for $b \approx D \approx (n\log n)^{1/3}$, the time is $O((n\log n)^{2/3})$.*

So for $D = n/\alpha$, for a parameter α, we use $b = (\alpha\log n)^{1/2}$, and get running time $O(\frac{n\cdot\log^{1/2} n}{\alpha^{1/2}})$, i.e., it is sublinear in n for $\alpha = \omega(\log n)$ (that is, for almost entire range of D, specifically, $D = o(n/\log n)$).

The bound 1 of Theorem C.1 is always better than the (linear) bound of the Bellman-Ford algorithm. Comparing it will the topology-collecting algorithm (that requires $O(D + |E|/b)$) time, our algorithm has smaller running time whenever $|E| = \omega(n\log n)$. When $|E| = O(n\log n)$, our algorithm has still smaller running time than the topology-collecting algorithm if $b = o(n^{1/3}/\log^{5/3} n)$. Only if the graph is very sparse ($|E| = o(n\log n)$) and the bandwidth b satisfies $\Omega(n^{1/3}/\log^{5/3} n) = b = O((n\log n)^{1/3})$, the topology-collecting algorithm has smaller running time than our algorithm, by a polylogarithmic in n factor. (But, of course, our algorithm avoids heavy local computations, while the topology-collecting algorithm heavily relies on them.)

The bound 2 of Theorem C.1 is also always better than the linear bound of Bellman-Ford. Here our running time loses to the topology-collecting algorithm only when $b = \omega(\frac{|E|^{4/3}}{n\log n})$ and

$D \leq \sqrt{\frac{n \log n}{b}} = \frac{n \log n}{|E|^{2/3}}$. In particular, if $|E| = \omega((n \log n)^{3/2})$, then our bound is always better than that of the topology-collecting algorithm.

The bound 3 of Theorem C.1 is sublinear in n as long as $D \leq \frac{n}{\log^2 n} \cdot b$, i.e., also almost in the entire range of parameters. This bound is also better than that of the topology-collecting algorithm whenever $|E| = \omega(n \log n)$. Even when the graph is very sparse (i.e., $|E| = O(n \log n)$), our bound loses (by at most a polylogarithmic in n factor) to that of the topology-collecting algorithm only when the diameter D gets close (within a polylogarithmic in n factor) to its upper bound $\frac{n \log n}{b^2}$.

C.2 Multiple Sources

As we have seen, the construction of the k-shortcut hopset $G'^{(k)}$ requires $O(\frac{\log n}{q} \cdot k \cdot \lceil \frac{k}{b} \rceil)$ time in the *CONGEST*$(b \log n)$ model. The upcast and pipelined broadcast of its edges over the BFS tree τ of G are performed within an additional $O(D + \frac{nqk}{b})$ time.

Like in Section 6, we assume that $s \leq nq$.

After the hopset $G'^{(k)}$ was constructed, the algorithm conducts a B-F in $G' \cup G'^{(k)}$ for $h' = O(nq/k)$ iterations. (So $k \leq nq$ is another constraint.) The first part of each iteration of the B-F involves collecting and disseminating $O(nqs)$ estimates over τ, i.e., it requires $O(D + \frac{nqs}{b})$ time.

In the second part of each iteration, a B-F in G to depth $O((\log n)/q)$ is conducted. Here in each step the algorithm needs to relay up to s estimates. When the bandwidth is b, each such a step requires $O(\lceil \frac{s}{b} \rceil)$ rounds. Hence, overall, the second part of each iteration requires $O(\frac{\log n}{q} \cdot \lceil \frac{s}{b} \rceil)$ time.

Thus, each iteration of the B-F in $G' \cup G'^{(k)}$ is executed within $O(D + \frac{nqs}{b} + \frac{\log n}{q} \cdot \lceil \frac{s}{b} \rceil)$ time. Hence the entire B-F in $G' \cup G'^{(k)}$ requires $O((D + \frac{nqs}{b} + \frac{\log n}{q} \cdot \lceil \frac{s}{b} \rceil) \cdot \frac{nq}{k})$ time.

The final step of the algorithm, where vertices of $V \setminus V'$ learn their distances, requires $O(\frac{\log n}{q} \cdot \lceil \frac{s}{b} \rceil)$ time. This term is dominated by the running times of other steps of the algorithm.

Hence the total running time of the algorithm is given by

$$T = O\left(\frac{\log n}{q} \cdot k \cdot \left\lceil \frac{k}{b} \right\rceil + D + \frac{nqk}{b} + \left(D + \frac{nqs}{b} + \frac{\log n}{q} \cdot \left\lceil \frac{s}{b} \right\rceil\right) \cdot \frac{nq}{k}\right). \tag{12}$$

To analyze this expression, we consider a number of regimes of parameters. **Case 1:** Here we restrict the parameters to satisfy $s \leq b \leq k (\leq nq)$. (The parenthesized inequality always holds.) Then

$$T = O\left(\left(D + \frac{nqs}{b} + \frac{\log n}{q}\right) \cdot \frac{nq}{k} + \frac{\log n}{q} \cdot \frac{k^2}{b} + \frac{nqk}{b}\right). \tag{13}$$

Consider first the subcase of small diameter, i.e., $D \leq \sqrt{\frac{s}{b} n \log n}$. Set $q = \sqrt{\frac{b \log n}{ns}}$, $k = \frac{(n \log n)^{1/6} \cdot b^{1/2}}{s^{1/6}}$. It follows that

$$T = O\left(\frac{(n \log n)^{5/6}}{b^{1/2}} \cdot s^{1/6}\right). \tag{14}$$

This bound applies when $s \leq (n \log n)^{1/4}$, $s \leq b \leq (\frac{n \log n}{s})^{1/3}$. (The right-hand inequality follows from $b \leq k$.) Note that for $s = b \leq (n \log n)^{1/4}$, $D \leq \sqrt{n \log n}$, we have here $T = O(\frac{(n \log n)^{5/6}}{b^{1/3}})$.

For large diameter, i.e., $D \geq \max\{\sqrt{\frac{ns \log n}{b}}, (n \log n)^{1/3} \cdot s^{2/3}\}$, we set $q = \frac{\log n}{D}$, $k = (\frac{n \log n \cdot b}{D})^{1/3}$. Note that

$$\frac{nqk}{b} = \frac{n}{b} \frac{\log n}{D} \cdot \left(\frac{n \log n \cdot b}{D}\right)^{1/3} = \left(\frac{n \log n}{D}\right)^{4/3} / b^{2/3}.$$

This expression is dominated by $(n\log n)^{2/3}(D/b)^{1/3}$, as long as $D \geq \frac{(n\log n)^{2/5}}{b^{1/5}}$. Since $D \geq \sqrt{\frac{n\log n\cdot s}{b}}$, it is sufficient to show that $s^{5/6}(n\log n)^{1/3} \geq b$. But we also have $b \leq (\frac{n\log n}{D})^{1/2}$. (This follows from $b \leq k = (\frac{n\log n\cdot b}{D})^{1/3}$.) Hence

$$b \leq \left(\frac{n\log n\cdot\sqrt{b}}{\sqrt{n\log n\cdot s}}\right) = \left(\frac{n\log n\cdot b}{s}\right)^{1/4},$$

and so $b \leq (\frac{n\log n}{s})^{1/3}$. Thus, $s^{5/6}(n\log n)^{1/3} \geq ((n\log n)/s)^{1/3} \geq b$ follows.

We conclude that

$$T = O((n\log n)^{2/3}(D/b)^{1/3}), \tag{15}$$

subject to $D \geq \max\{\sqrt{\frac{n\log n\cdot s}{b}}, (n\log n)^{1/3}s^{2/3}\}$. (The inequality $D \geq (n\log n)^{1/3}\cdot s^{2/3}$ is because $D \geq \sqrt{\frac{n\log n\cdot s}{b}}$ and $b \leq k = (\frac{nb\log n}{D})^{1/3}$ imply $b \leq (\frac{n\log n}{D})^{1/2}$.) We also need $s \leq b \leq (\frac{n\log n}{D})^{1/2}$ to hold, for (15) to be valid.

So given b and D, the bound (15) holds for s sources, as long as $s \leq \min\{b, \frac{D^2 b}{n\log n}\}$. This bound generalizes Theorem C.1, bound 3.

Another regime of parameters that gives rise to meaningful bounds is the following one.

Case 2: $b \leq s$, $b \leq k(\leq nq)$.

Here the running time (see Equation (12)) becomes

$$T = O\left(\left(D + \frac{nqs}{b} + \frac{\log n}{q}\cdot\frac{s}{b}\right)\cdot\frac{nq}{k} + \frac{\log n}{q}\cdot\frac{k^2}{b} + \frac{nqk}{b}\right). \tag{16}$$

First, we consider the subcase of small diameter, i.e., $D \leq \frac{s}{b}\sqrt{n\log n}$. We set $q = \sqrt{\frac{\log n}{n}}$, $k = (n\log n)^{1/6}\cdot s^{1/3}$, and obtain

$$\begin{aligned} T &= O\left(\sqrt{n\log n}\cdot\frac{s}{b}\frac{\sqrt{n\log n}}{(n\log n)^{1/6}\cdot s^{1/3}} + \sqrt{n\log n}\cdot\frac{1}{b}\cdot(n\log n)^{1/3}\cdot s^{2/3} + \frac{\sqrt{n\log n}}{b}(n\log n)^{1/6}\cdot s^{1/3}\right) \\ &= O\left((n\log n)^{5/6}\cdot\frac{s^{2/3}}{b} + (n\log n)^{2/3}\cdot\frac{s^{1/3}}{b}\right) \\ &= O\left((n\log n)^{5/6}\cdot\frac{s^{2/3}}{b}\right). \end{aligned} \tag{17}$$

This bound applies for $b \leq \min\{s, (k =)(n\log n)^{1/6}\cdot s^{1/3}\}$, and $D \leq \frac{s}{b}\sqrt{n\log n}$.

For $s = b$, this bound agrees with the bound (14), i.e., both bounds give $T = O((n\log n)^{5/6}/b^{1/3})$. Both these bounds (14) and (17) are non-trivial, i.e., they are smaller than $O(\min\{n\lceil\frac{s}{b}\rceil, D + \frac{|E|+ns}{b}\}$ almost in the entire range of parameters. See the discussion following Theorem C.2.

Now we consider the case of large diameter, i.e., $D \geq \frac{s}{b}\sqrt{n\log n}$. (We also have $b \leq s$, $b \leq k(\leq nq)$.) Here we have

$$T = O\left(D\cdot\frac{nq}{k} + \frac{\log n}{q}\cdot\frac{k^2}{b} + \frac{nqk}{b}\right).$$

We set $q = \sqrt{\frac{\log n}{n}}$, $k = (Db)^{1/3}$, and get

$$T = T_1 = O\left(\frac{D^{2/3}\sqrt{n\log n}}{b^{1/3}}\right), \tag{18}$$

subject to $b \le s \le b \cdot \frac{D}{\sqrt{n \log n}}$, and $\max\{\frac{s}{b}\sqrt{n \log n}, b^2\} \le D \le \frac{(n \log n)^{3/2}}{b}$.

Another option (still for $b \le s$, $b \le k \le nq$, $D \ge \frac{s}{b}\sqrt{n \log n}$) is to set $q = \frac{s \log n}{D \cdot b}$. This guarantees $\frac{\log n}{q} \ge nq$, i.e., $T = O(D \cdot \frac{nq}{k} + \frac{\log n}{q} \cdot \frac{k^2}{b})$. (The third term $\frac{nqk}{b}$ is dominated by $\frac{\log n}{q} \cdot \frac{k^2}{b}$.) Set $k = (\frac{n \log n}{Db})^{1/3} \cdot s^{2/3}$. We get

$$T = T_2 = O\left((Ds)^{1/3}\left(\frac{n \log n}{b}\right)^{2/3}\right). \tag{19}$$

The condition $b \le k$ gives rise to $b \le \min\{s, s^{1/2}(\frac{n \log n}{D})^{1/4}\}$. The condition $k \le nq$ (together with the above) gives rise to $\frac{s}{b}\sqrt{n \log n} \le D \le n \log n \cdot \frac{\sqrt{s}}{b}$.

Note that $T_2 \le T_1$ whenever $D \ge \frac{s}{b}\sqrt{n \log n}$, i.e., whenever both bounds apply, T_2 is better than T_1. Hence T_1 is meaningful only for $D > n \log n \frac{\sqrt{s}}{b}$, or for $s \ge b \ge s^{1/2}(\frac{n \log n}{D})^{1/4}$. Moreover, in fact, if $b \le s^{1/2}(\frac{n \log n}{D})^{1/4}$, then $D \ge n \log n \frac{\sqrt{s}}{b}$ (or else T_2 is applicable). But this implies $b \le b^{1/4} \cdot s^{3/8}$, i.e., $b \le \sqrt{s}$. Then, however, the condition $D \ge \frac{\sqrt{s}}{b} n \log n$ cannot hold. So for T_1 to be meaningful, it must hold that $b \ge s^{1/2}(\frac{n \log n}{D})^{1/4}$. However, using T_2 with $b = s^{1/2}(\frac{n \log n}{D})^{1/4}$ (even when b is, in fact, larger than that value) gives rise to

$$T_2 = O\left((Ds)^{1/3}\left(\frac{(n \log n)^{3/4} \cdot D^{1/4}}{s^{1/2}}\right)^{2/3}\right) = O(\sqrt{Dn \log n}).$$

However, T_1 applies for $b^2 \le D$, and in this range $D^{2/3}\sqrt{n \log n}/b^{1/3} \ge \sqrt{Dn \log n}$. Hence T_2 is never worth than T_1.

Yet another bound for the case $b \le s$ and large diameter D arises when we consider the case $s \ge b \ge k$, $nq \ge k$. Here, by Equation (12), the running time behaves as

$$T = O\left(\left(D + \frac{s}{b}(nq + \frac{\log n}{q})\right)\frac{nq}{k} + \frac{\log n}{q} \cdot k + \frac{nqk}{b}\right).$$

When D is large, i.e., $D \ge \max\{nq, \frac{\log n}{q}\} \ge \frac{s}{b}\sqrt{n \log n}$, the bound becomes

$$T = O\left(D \cdot \frac{nq}{k} + \frac{\log n}{q} \cdot k + \frac{nqk}{b}\right). \tag{20}$$

We set $q = \sqrt{\frac{\log n}{n}}$, $k = \sqrt{D}$, and obtain $T = T_3 = O(\sqrt{n \log n \cdot D})$, subject to $s^2 \ge b^2 \ge D \ge \frac{s}{b}\sqrt{n \log n}$. (This follows from $s \ge b \ge k = \sqrt{D}$.) It also follows that $b \ge s^{1/3}(n \log n)^{1/6}$, and $s \ge \max\{b, \frac{\sqrt{n \log n}}{b}\} \ge (n \log n)^{1/4}$.

Recall that T_1 applies only when $D \ge b^2$, i.e., in the complimentary range. Indeed, for $D = b^2$, we have $T_1, T_3 = O(\sqrt{Dn \log n})$. **Case 3** (Larger bandwidth): Here we consider the case $b \ge s$, $b \ge k$, $nq \ge k$.

First, we consider the subcase of small diameter, i.e., $D \le \sqrt{\frac{s}{b} n \log n}$. We set $q = \sqrt{\frac{b \log n}{ns}}$, $k = (\frac{n \log n \cdot b}{s})^{1/4}$, and, by Equation (12), obtain running time

$$T = O\left(\left(D + \frac{nqs}{b} + \frac{\log n}{q}\right) \cdot \frac{nq}{k} + \frac{\log n}{q} \cdot k + \frac{nqk}{b}\right) \tag{21}$$

$$= O\left(\left(\frac{nqs}{b}+\frac{\log n}{q}\right)\cdot\frac{nq}{k}+\frac{\log n}{q}\cdot k+\frac{nqk}{b}\right)$$
$$= O\left(\sqrt{\frac{ns\log n}{b}}\cdot\sqrt{\frac{n\log n\cdot b}{s}}\cdot\left(\frac{s}{n\log n\cdot b}\right)^{1/4}\right)$$
$$+ O\left(\sqrt{\frac{ns\log n}{b}}\cdot\left(\frac{n\cdot\log n\cdot b}{s}\right)^{1/4}+\sqrt{\frac{n\log n\cdot b}{s}}\cdot\frac{1}{b}\left(\frac{n\log n\cdot b}{s}\right)^{1/4}\right)$$
$$= O\left((n\log n)^{3/4}\cdot\left(\frac{s}{b}\right)^{1/4}\right). \tag{22}$$

The condition $b\geq k$ implies $b\geq\max\{s,(\frac{n\log n}{s})^{1/3}\}\geq(n\log n)^{1/4}$. The condition $k\leq nq$ always holds. We always have $T=O((n\log n)^{3/4})$ in this range.

For $s=O(1)$, and $b=(n\log n)^{1-\epsilon}$, for a small constant $\epsilon>0$, we obtain $D\leq\sqrt{\frac{s}{b}\cdot n\log n}=(n\log n)^{\epsilon/2}$, and $b\geq\max\{1,(n\log n)^{1/3}\}=(n\log n)^{1/3}$. Here $T=(n\log n)^{1/2+\epsilon/4}$. With $b=n/polylog(n)$, we can have $T=\tilde{O}(\sqrt{n})$, for $D\leq polylog(n)$, $s=O(1)$.

For $s=b=(n\log n)^{1/4}$, we get here $T=(n\log n)^{3/4}$, for $D\leq\sqrt{n\log n}$.

For $s=1$, we obtain here $T=O(\frac{(n\log n)^{3/4}}{b^{1/4}})$, for $b\geq(n\log n)^{1/3}$. (This is the bound 2 of Theorem C.1.) Formerly (see Equation (9)), we had $T=O(\frac{(n\log n)^{5/6}}{b^{1/2}})$, for $b\leq(n\log n)^{1/3}$. (Both bounds apply for $D\leq\sqrt{(n\log n)/b}$.) These bounds agree at $b=(n\log n)^{1/3}$, and give $T=O((n\log n)^{2/3})$.

Now, consider the case of large D, i.e., $D\geq\max\{\frac{nqs}{b},\frac{\log n}{q}\}\geq\sqrt{\frac{s}{b}\cdot n\log n}$. First, we set q so that $D\geq\frac{nqs}{b}\geq\frac{\log n}{q}$. Specifically, $q=\sqrt{\frac{b\log n}{ns}}$, $k=\sqrt{\frac{Db}{s}}$. Then , by Equation (21),

$$T=O\left(D\cdot\frac{nq}{k}+\frac{\log n}{q}\cdot k+\frac{nqk}{b}\right)=O(\sqrt{Dn\log n}). \tag{23}$$

For this bound to hold, we need $bs\geq D\geq\sqrt{\frac{s}{b}\cdot n\log n}$, $b\geq s$. (The inequality $bs\geq D$ follows from $b\geq k=\sqrt{\frac{Db}{s}}$.) This also means that $b^2\geq bs\geq D$, i.e., $b\geq\sqrt{D}$. (Recall that we also have the bound $T=T_3=O(\sqrt{Dn\log n})$, for $s^2\geq b^2\geq D\geq\frac{s}{b}\sqrt{n\log n}$. See Equation (20).)

Another option is to set $q=b\sqrt{\frac{\log n}{Dn}}$, $k=b$. (We are still in the regime $b\geq s$.) Then

$$\begin{aligned} T &= O\left(D\cdot\frac{nq}{k}+\frac{\log n}{q}\cdot k+\frac{nqk}{b}\right) \\ &= O\left(\sqrt{Dn\log n}+\sqrt{\frac{n\log n}{D}}\cdot b\right). \end{aligned} \tag{24}$$

Assuming $D\geq b$, the latter expression is $O(\sqrt{Dn\log n})$ (subject to $D\geq b\geq s$). We also require $q=\Omega(\frac{\log n}{n})$, and so $b\geq\sqrt{\frac{D\log n}{n}}$, i.e., $\frac{nb^2}{\log n}\geq D\geq b\geq s$. In addition, the conditions $D\geq\frac{\log n}{q}$, $D\geq\frac{nqs}{b}$ (under which (24) applies) imply $D\geq\max\{\frac{n\log n}{b^2},s^{2/3}\cdot(n\log n)^{1/3}\}$.

Hence the bound $T=O(\sqrt{Dn\log n})$ holds in three ranges:

(1) For $s\geq b\geq\sqrt{D}$, $D\geq\frac{s}{b}\sqrt{n\log n}$.
(2) For $b^2\geq bs\geq D\geq\sqrt{\frac{s}{b}\cdot n\log n}$, $b\geq s$.

(3) For $\frac{nb^2}{\log n} \geq D \geq \max\{\frac{n\log n}{b^2}, s^{2/3}(n\log n)^{1/3}\}$, $b \geq s$.

The next theorem summarizes the bounds that we have for the case of relatively small D.

Theorem C.2. *Our algorithm computes exact shortest paths for $S \times V$, $|S| = s$, CONGEST$(b \log n)$ model for the case of relatively small diameter D in running time T given by:*

When $D \leq \sqrt{\frac{s}{b} n \log n}$, then

(1) $T = O(\frac{(n\log n)^{5/6}}{b^{1/2}} \cdot s^{1/6})$, for $s \leq O((n\log n)^{1/4})$, $s \leq b \leq (\frac{n\log n}{s})^{1/3}$ (see Equation (14)); and
(2) $T = O((n\log n)^{3/4} \cdot (s/b)^{1/4})$, for $b \geq \max\{s, ((n\log n)/s)^{1/3}\} \geq (n\log n)^{1/4}$ (see Equation (22)).

When $D \leq \frac{s}{b} \cdot \sqrt{n\log n}$, and $b \leq \min\{s, s^{1/3}(n\log n)^{1/6}\}$, then (see Equation (17))

$$T = O\left((n\log n)^{5/6} \cdot \frac{s^{2/3}}{b}\right). \tag{25}$$

Observe that for $b = s$, the bounds 1 (of Theorem C.2) and Equation (25) agree, and give

$$T = O\left((n\log n)^{5/6} \cdot \frac{1}{s^{1/3}}\right) = O\left((n\log n)^{5/6} \cdot \frac{1}{b^{1/3}}\right).$$

The bound (25) is non-trivial (i.e., smaller than $O(\min\{n\lceil\frac{s}{b}\rceil, D + \frac{ns+|E|}{b}\})$ in the entire range of parameters.

Even if the graph is relatively sparse, i.e., $|E| = O(ns)$, still the bound 1 of Theorem C.2 is non-trivial (i.e., it is $o(ns/b)$) if $s = \omega(\log n)$, or (more generally) if $b = o(\frac{n^{1/3}\cdot s^{5/3}}{\log^{5/3} n})$. In other words, it loses to the topology-collecting algorithm only if $\omega(\frac{n^{1/3}\cdot s^{5/3}}{\log^{5/3} n}) = b = O((\frac{n\log n}{s})^{1/3})$. For $\omega(ns) = |E| = o(n(s+b))$, the range in which this bound is non-trivial is even larger than that. If $|E| = \Omega(n(s+b))$, then the trivial algorithm requires at least $\Omega(n\lceil s/b\rceil) = \Omega(n)$ time, while the bound 1 is always sublinear. (Recall that $s \leq (n\log n)^{1/4}$.) Hence for relatively dense graphs, this bound is always non-trivial.

The situation is similar with respect to bound 2. Even if the graph is relatively sparse, i.e., $|E| = O(ns)$, still the bound 2 of Theorem C.2 is non-trivial for $s \cdot \frac{n^{1/3}}{\log n} \geq b \geq \max\{s, (\frac{n\log n}{s})^{1/3}\}$. For $\omega(ns) = |E| = o(n(s+b))$, the range in which this bound is non-trivial is even larger than that. If $|E| = \Omega(n(s+b))$, then the trivial algorithm requires at least $\Omega(n\lceil s/b\rceil) = \Omega(n)$ time, while the bound 2 is always sublinear. (Recall that $b \geq s$.) Hence for relatively dense graphs, this bound is always non-trivial.

The bound 1 of Theorem C.2 generalizes the single-source bound 3 of Theorem C.1. The bound 2 of Theorem C.2 generalizes the single-source bound 2 of Theorem C.1. The bound (25) generalizes the multi-source unit-bandwidth bound (for small diameter) of Theorem 6.1.

In the next theorem we summarize the bounds that we have for the case of relatively large D.

Theorem C.3. *Our algorithm computes exact shortest paths for pairs $S \times V$, $|S| = s$, in CONGEST$(b \log n)$ model for the case of relatively large diameter D in running time T given by:*

(1) $T_0 = T = O((n\log n)^{2/3}(D/b)^{1/3})$, subject to $D \geq max\{\sqrt{\frac{s}{b} n\log n}, (n\log n)^{1/3} \cdot s^{2/3}\}$, $s \leq b \leq (\frac{n\log n}{D})^{1/2}$. (Equivalently, given D, b, the bound holds for $s \leq \min\{b, \frac{D^2 b}{n\log n}\}$. See inequality (15).)

(2) $T = T_2 = O((Ds)^{1/3} \cdot (\frac{n \log n}{b})^{2/3})$, *subject to* $b \leq \min\{s, s^{1/2}(\frac{n \log n}{D})^{1/4}\}$, $\frac{s}{b} \cdot \sqrt{n \log n} \leq D \leq n \log n \cdot \frac{\sqrt{s}}{b}$. *Note that* T_2 *is better than* T_1 *whenever* T_2 *is applicable, and thus* T_1 *is meaningful only when* $s \geq b \geq s^{1/2}((n \log n)/D)^{1/4}$. *(See inequality (19).)*

(3) *The bound* $T = T_3 = O(\sqrt{Dn \log n})$, *which applies in the three following domains:*

(a) *For* $s \geq b \geq \sqrt{D}$, $D \geq \frac{s}{b}\sqrt{n \log n}$.

(b) *For* $sb \geq D \geq \sqrt{\frac{s}{b} \cdot n \log n}$, $b \geq s$.

(c) *For* $D \geq \max\{\frac{n \log n}{b^2}, s^{2/3}(n \log n)^{1/3}\}$, $b \geq s$.

The bounds 1, 3b, and 3c of Theorem C.3 apply for $s \leq b$. Observe that in 1 and 3b we have $s \leq b$ and $D \geq \sqrt{\frac{s}{b} \cdot n \log n}$. In 3b, $sb \geq \sqrt{\frac{s}{b} \cdot n \log n}$ implies $sb^3 \geq n \log n$, i.e., $b \leq ((n \log n)/s)^{1/3}$. In bound 1, $s \leq b \leq ((n \log n)/s)^{1/3}$ implies $sb^3 \leq s\frac{n \log n}{s} = n \log n$. Hence the boundary of these two domains is $sb^3 = n \log n$. This (together with $b \geq s$) implies $b \geq (n \log n)^{1/4}$. In bound 1, $sb = \frac{n \log n}{b^3} \cdot b \geq D \geq \sqrt{\frac{s}{b} \cdot n \log n}$. Since $s = \frac{n \log n}{b^3}$, the right-hand side is equal to $\frac{n \log n}{b^2}$, i.e., $D = \frac{n \log n}{b^2}$. Hence the intersection of these two domains is $D = \frac{n \log n}{b^2}$. For this value of the diameter, the two bounds indeed agree, as $T_0 = T_3 = \frac{n \log n}{b}$.

Similarly, the bounds 1 and 3c also intersect when $D = \frac{n \log n}{b^2}$ (i.e., $b = \sqrt{\frac{n \log n}{D}}$) and $s = b$. Then

$$\left(\frac{D}{b}\right)^{1/3} (n \log n)^{2/3} = \left(\frac{D^{3/2}}{\sqrt{n \log n}}\right)^{1/3} (n \log n)^{2/3} = \sqrt{Dn \log n},$$

i.e., these two bounds agree on the boundary as well.

The bound 1 of Theorem C.3 generalizes the single-source bound 2 from Theorem C.1. The bound 2 of Theorem C.3 generalizes the large-diameter multi-source bound of Theorem 6.1.

Next we compare the bounds of Theorem C.3 with the trivial bound of $O(\min\{n\lceil\frac{s}{b}\rceil, D + \frac{ns+|E|}{b}\})$. First, observe that this trivial bound can be improved only when $D \leq \frac{|E|+ns}{b}$, as these problems require $\Omega(D)$ time.

The bound 1 improves the trivial bound (moreover, it is $o(ns/b)$) for $D = o(\frac{n}{\log^2 n} \cdot \frac{s^3}{b^2})$. It is $o(n\lceil s/b\rceil) = o(n)$ (because $s \leq b$ in bound 1) for $D = o(\frac{n \cdot b}{\log^2 n})$.

The bound 2 improves the trivial one for $D = o(\frac{n \cdot s^2}{b \cdot \log^2 n})$. Finally, the bound 3 improves the trivial one for $D = o(\frac{n}{\log n} \cdot \frac{s^2}{b^2})$.

ACKNOWLEDGMENTS

The author thanks anonymous referees of the STOC'17 conference for helpful remarks and Keren Censor-Hillel, Sebastian Krinninger, Danupon Nanongkai, and Ofer Neiman for helpful discussions.

REFERENCES

[1] Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. 1999. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.* 28, 4 (1999), 1167–1181. DOI: https://doi.org/10.1137/S0097539796303421

[2] Ingo Althöfer, Gautam Das, David P. Dobkin, and Deborah Joseph. 1990. Generating sparse spanners for weighted graphs. In *Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory (SWAT'90)*. 26–37.

[3] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. 2012. Graph sketches: Sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'12)*. 5–14. DOI: https://doi.org/10.1145/2213556.2213560

[4] Udit Agarwal and Vijaya Ramachandran. 2019. Distributed weighted all pairs shortest paths through pipelining. *In International Parallel and Distributed Processing Symposium, (IPDPS'19)*. IEEE, 23–32. https://doi.org/10.1109/IPDPS.2019.00014

[5] Udit Agarwal, Vijaya Ramachandran, Valerie King, and Matteo Pontecorvi. 2018. A deterministic distributed algorithm for exact weighted all-pairs shortest paths in Õ(n 3/2) rounds. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing (PODC'18)*. 199–205. DOI: https://doi.org/10.1145/3212734.3212773

[6] Baruch Awerbuch. 1989. Distributed shortest paths algorithms (extended abstract). In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*. 490–500. DOI: https://doi.org/10.1145/73007.73054

[7] S. Baswana. 2006. Faster Streaming algorithms for graph spanners. *CoRR, abs/cs/0611023* (2006).

[8] Richard Bellman. 1958. On a routing problem. *Quart. Appl. Math.* 16 (1958), 87–90.

[9] D. Bertsekas and R. Gallager. 1992. *Data Networks*. Prentice-Hall International, London.

[10] Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. 2016. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. In *31st International Symposium on Distributed Computing (DISC'17)*, Andréa W. Richa (Ed.), Vol. 91. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 7:1–7:16. DOI: 10.4230/LIPIcs.DISC.2017.7

[11] Aaron Bernstein and Danupon Nanongkai. 2018. Distributed exact weighted all-pairs shortest paths in near-linear time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC'19)*. ACM, 334–342.

[12] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. 2015. Algebraic methods in the congested clique. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing (PODC'15)*. 143–152. DOI: https://doi.org/10.1145/2767386.2767414

[13] Edith Cohen. 2000. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *J. ACM* 47, 1 (2000), 132–166. DOI: https://doi.org/10.1145/331605.331610

[14] M. Elkin. 2004. An unconditional lower bound on the time-approximation tradeoff of the minimum spanning tree problem. In *Proceedings of the 36th ACM Symposium on Theory of Computing (STOC'04)*. 331–340.

[15] Michael Elkin. 2004. Distributed approximation: A survey. *SIGACT News* 35, 4 (2004), 40–57. DOI: https://doi.org/10.1145/1054916.1054931

[16] Michael Elkin. 2011. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. *ACM Trans. Algor.* 7, 2 (2011), 20. DOI: https://doi.org/10.1145/1921659.1921666

[17] Michael Elkin and Ofer Neiman. 2016. Hopsets with constant hopbound, and applications to approximate shortest paths. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS'16)*.

[18] Michael Elkin and Ofer Neiman. 2017. Efficient algorithms for constructing very sparse spanners and emulators. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'17)*. 652–669. DOI: https://doi.org/10.1137/1.9781611974782.41

[19] M. Elkin and O. Neiman. 2019. Linear-size hopsets with small hopbound, and distributed routing with low memory. In *Proceedings of the 31st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'19)*.

[20] Michael Elkin and Jian Zhang. 2006. Efficient algorithms for constructing (1+epsilon, beta)-spanners in the distributed and streaming models. *Distrib. Comput.* 18, 5 (2006), 375–385. DOI: https://doi.org/10.1007/s00446-005-0147-2

[21] Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. 2012. Networks cannot compute their diameter in sublinear time. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'12)*. 1150–1162.

[22] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. 2004. On graph problems in a semi-streaming model. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming*. 531–543.

[23] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. 2005. Graph distances in the streaming model: The value of space. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*. 745–754.

[24] Sebastian Forster and Danupon Nanongkai. 2018. A faster distributed single-source shortest paths algorithm. In *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science (FOCS'18)*. 686–697. DOI: https://doi.org/10.1109/FOCS.2018.00071

[25] Lester Ford Jr. 1956. In *Network Flow Theory*. RAND Corporation, Santa Monica, CA.

[26] Greg N. Frederickson. 1985. A single source shortest path algorithm for a planar distributed network. In *Proceedings of the 2nd Symposium of Theoretical Aspects of Computer Science (STACS'85)*. 143–150. DOI: https://doi.org/10.1007/BFb0024003

[27] R. G. Gallager. 1976. Distributed minimum hop algorithms. *Technical Report, Lab. for Information and Decision Systems* (1976).

[28] R. G. Gallager. 1976. A shortest path routing algorithm with automatic resynch. *Technical Report LIDS-P-1175* (1976).

[29] Mohsen Ghaffari and Fabian Kuhn. 2013. Distributed minimum cut approximation. In *Proceedings of the 27th International Symposium on Distributed Computing (DISC'13)*. 1–15. DOI: https://doi.org/10.1007/978-3-642-41527-2_1

[30] Juan A. Garay, Shay Kutten, and David Peleg. 1998. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM J. Comput.* 27, 1 (1998), 302–316. DOI: https://doi.org/10.1137/S0097539794261118

[31] Mohsen Ghaffari and Jason Li. 2018. Improved distributed algorithms for exact shortest paths. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC'18)*. 431–444. DOI: https://doi.org/10.1145/3188745.3188948

[32] Venkatesan Guruswami and Krzysztof Onak. 2016. Superlinear lower bounds for multipass graph processing. *Algorithmica* 76, 3 (2016), 654–683. DOI: https://doi.org/10.1007/s00453-016-0138-7

[33] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. 2016. An almost-tight distributed algorithm for computing single-source shortest paths. *Proceedings of the Annual ACM SIGACT Symposium on Theory of Computing (STOC'16)*.

[34] Chien-Chung Huang, Danupon Nanongkai, and Thatchaphol Saranurak. 2017. Distributed exact weighted all-pairs shortest paths in $\tilde{O}(n^{5/4})$ Rounds. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS'17)*. 168–179. DOI: https://doi.org/10.1109/FOCS.2017.24

[35] Stephan Holzer and Nathan Pinsker. 2015. Approximation of distances and shortest paths in the broadcast congest clique. In *Proceedings of the 19th International Conference on Principles of Distributed Systems (OPODIS'15)*. 6:1–6:16. DOI: https://doi.org/10.4230/LIPIcs.OPODIS.2015.6

[36] Stephan Holzer, David Peleg, Liam Roditty, and Roger Wattenhofer. 2014. Distributed 3/2-approximation of the diameter. In *Proceedings of the 28th International Symposium on Distributed Computing (DISC'14)*. 562–564.

[37] Stephan Holzer and Roger Wattenhofer. 2012. Optimal distributed all pairs shortest paths and applications. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC'12)*. 355–364. DOI: https://doi.org/10.1145/2332432.2332504

[38] J. M. Jaffe. 1985. Distributed multi-destination routing: The constraints of local information. *SIAM J. Comput.* 14 (1985), 875–888.

[39] Shay Kutten and David Peleg. 1998. Fast distributed construction of small k-dominating sets and applications. *J. Algor.* 28, 1 (1998), 40–66. DOI: https://doi.org/10.1006/jagm.1998.0929

[40] Philip N. Klein and Sairam Subramanian. 1997. A randomized parallel algorithm for single-source shortest paths. *J. Algor.* 25, 2 (1997), 205–220. DOI: https://doi.org/10.1006/jagm.1997.0888

[41] Christoph Lenzen and Boaz Patt-Shamir. 2013. Fast routing table construction using small messages. In *Proceedings of the Symposium on Theory of Computing Conference (STOC'13)*. 381–390. DOI: https://doi.org/10.1145/2488608.2488656

[42] Zvi Lotker, Boaz Patt-Shamir, and David Peleg. 2006. Distributed MST for constant diameter graphs. *Distrib. Comput.* 18, 6 (2006), 453–460. DOI: https://doi.org/10.1007/s00446-005-0127-6

[43] 2006. List of open problems in sublinear algorithms: Problem 14. Retrieved from http://sublinear.info/14.

[44] E. F. Moore. 1957. The shortest path through a maze. In *Proceedings of the International Symposium on the Theory of Switching*. 285–292.

[45] Danupon Nanongkai. 2014. Distributed approximation algorithms for weighted shortest paths. In *Proceedings of the Symposium on Theory of Computing (STOC'14)*. 565–573. DOI: https://doi.org/10.1145/2591796.2591850

[46] Danupon Nanongkai and Hsin-Hao Su. 2014. Almost-tight distributed minimum cut algorithms. In *Proceedings of the 28th International Symposium on Distributed Computing (DISC'14)*. 439–453. DOI: https://doi.org/10.1007/978-3-662-45174-8_30

[47] David Peleg. 1990. Time-optimal leader election in general networks. *J. Parallel Distrib. Comput.* 8, 1 (1990), 96–99. DOI: https://doi.org/10.1016/0743-7315(90)90074-Y

[48] D. Peleg. 2000. *Distributed Computing: A Locality-Sensitive Approach*. SIAM.

[49] D. Peleg and V. Rubinovich. 1999. A near-tight lower bound on the time complexity of distributed MST construction. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*. 253–261.

[50] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. 2012. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.* 41, 5 (2012), 1235–1265. DOI: https://doi.org/10.1137/11085178X

[51] Hanmao Shi and Thomas H. Spencer. 1999. Time-work tradeoffs of the single-source shortest paths problem. *J. Algor.* 30, 1 (1999), 19–32. DOI: https://doi.org/10.1006/jagm.1998.0968

[52] Jeffrey D. Ullman and Mihalis Yannakakis. 1991. High-probability parallel transitive-closure algorithms. *SIAM J. Comput.* 20, 1 (1991), 100–125. DOI: https://doi.org/10.1137/0220006

Received May 2019; revised January 2020; accepted March 2020