A Unified Approach for All Pairs Approximate Shortest Paths in Weighted Undirected Graphs

Maor Akav
Bar-Ilan University, Ramat Gan, Israel

Liam Roditty
Bar-Ilan University, Ramat Gan, Israel

Abstract

Let $G = (V, E)$ be a weighted undirected graph with n vertices and m edges, and let $d_G(u, v)$ be the length of the shortest path between u and v in G. In this paper we present a unified approach for obtaining algorithms for all pairs approximate shortest paths in weighted undirected graphs. For every integer $k \geq 2$ we show that there is an $\tilde{O}(n^2 + kn^{2-3/k}m^{2/k})$ expected running time algorithm that computes a matrix M such that for every $u, v \in V$:

$$d_G(u, v) \leq M[u, v] \leq (2 + \frac{k-2}{k})d_G(u, v).$$

Previous algorithms obtained only specific approximation factors. Baswana and Kavitha [FOCS 2006, SICOMP 2010] presented a 2-approximation algorithm with expected running time of $\tilde{O}(n^2 + m\sqrt{n})$ and a 7/3-approximation algorithm with expected running time of $\tilde{O}(n^2 + m^{2/3}n)$. Their results improved upon the results of Cohen and Zwick [SODA 1997, JoA 2001] for graphs with $m = o(n^2)$. Kavitha [FSTTCS 2007, Algorithmica 2012] presented a 5/2-approximation algorithm with expected running time of $\tilde{O}(n^{9/4})$.

For $k = 2$ and $k = 3$ our result gives the algorithms of Baswana and Kavitha. For $k = 4$, we get a 5/2-approximation algorithm with $\tilde{O}(n^{\frac{5}{4}}m^{\frac{1}{2}})$ expected running time. This improves upon the running time of $\tilde{O}(n^{9/4})$ due to Kavitha, when $m = o(n^2)$.

Our unified approach reveals that all previous algorithms are a part of a family of algorithms that exhibit a smooth tradeoff between approximation of 2 and 3, and are not sporadic unrelated results. Moreover, our new algorithm uses, among other ideas, the celebrated approximate distance oracles of Thorup and Zwick [STOC 2001, JACM 2005] in a non standard way, which we believe is of independent interest, due to their extensive usage in a variety of applications.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Graph algorithms, Approximate All Pairs of Shortest Paths, Distance Oracles

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.4

1 Introduction

Computing All Pairs of Shortest Paths (APSP) is one of the most fundamental problems in Computer Science. The fastest known algorithms for APSP in weighted graphs run in $\min\{\tilde{O}(mn), n^3/\exp(\sqrt{\log n})\}$ [27, 19, 20].

In unweighted undirected graphs the fastest known APSP algorithms run in $\tilde{O}(\min\{mn, n^{\omega}\})$ time[1] [21], where $\omega < 2.373$ is the exponent of square matrix multiplication [28, 17, 24], n is the number of vertices and m is the number of edges. For an extension of this result to undirected graphs with integral weights see [22] and to directed graphs see [30]. Fast Matrix Multiplication (FMM) algorithms hide large constants and are thus far from being practical. A fundamental research question is whether one can obtain fast "combinatorial" algorithms, that can be implemented.

[1] $\tilde{O}$ notation hides polylogarithmic factors

Aingworth, Chekuri, Indyk and Motwani [1] initiated the research on efficient APSP algorithms in unweighted undirected graphs, that settle for an approximated solution and do not use FMM. An all pairs approximate shortest paths (APASP) algorithm has (α, β)-approximation if every distance X is estimated by the algorithm to be at least X and at most $\alpha X + \beta$, where α is the multiplicative approximation and β is the additive approximation. Aingworth et al. [1] presented a $(1, 2)$-approximation that runs in $\tilde{O}(n^{2.5})$.

Dor, Halperin and Zwick [10] improved and generalized the results of [1]. For every even $k > 2$, they presented a generalized scheme with $\tilde{O}(\min\{n^{2-\frac{2}{k+2}}m^{\frac{2}{k+2}}, n^{2+\frac{2}{3k-2}}\})$ running time and an additive approximation of k. For $k = 2$, the running time is $\tilde{O}(\min\{n^{\frac{3}{2}}m^{\frac{1}{2}}, n^{\frac{7}{3}}\})$ and the additive approximation is 2. They also obtained an algorithm with $\tilde{O}(n^2)$ running time and $(3, 0)$-approximation. Berman and Kasiviswanathan [6] improved this result and obtained an algorithm with the same running time of $\tilde{O}(n^2)$ and $(2, 1)$-approximation. The question of obtaining efficient APASP algorithms for *unweighted* undirected graphs was considered by many subsequent works [11, 5, 3, 23, 16, 2].

Cohen and Zwick [9] extended the results of Dor et al. [10] to *weighted* undirected graphs. They obtained an $\tilde{O}(n^2)$ time algorithm with a $(3, 0)$-approximation, an $\tilde{O}(n^{\frac{3}{2}}m^{\frac{1}{2}})$ time algorithm with a $(2, 0)$-approximation and also $\tilde{O}(n^{\frac{7}{3}})$ time algorithm with $(7/3, 0)$-approximation. Their analysis of the $(3, 0)$-approximation algorithm allowed them to obtain a general scheme with a running time of $\tilde{O}(n^{2-1/k}m^{1/k})$ that for every pair $u, v \in V$ computes an additive approximation of $2\sum_{i=1}^{k-1} w_i$, where w_i is the ith heaviest edge on the shortest path between u and v. (For a similar result see also [12].) This is still a $(3, 0)$-approximation in the worst case.

Baswana and Kavitha [4] presented several algorithms that perform better than the algorithms of Cohen and Zwick when $m = o(n^2)$. They presented a $(2, 0)$-approximation algorithm with expected running time of $\tilde{O}(n^2+m\sqrt{n})$ and a $(7/3, 0)$-approximation algorithm with expected running time of $\tilde{O}(n^2 + m^{2/3}n)$. Notice that both algorithms have the same running times as those of Cohen and Zwick when $m = \Theta(n^2)$. Kavitha [15] presented a $(5/2, 0)$-approximation algorithm with expected running time of $\tilde{O}(n^{9/4})$. Baswana and Kavitha [4] presented also a $(2, W)$-approximation algorithm, where W is the largest edge weight, with expected running time of $\tilde{O}(\min\{n^2, m\sqrt{n}\})$. This is still a $(3, 0)$-approximation in the worst case. A deterministic algorithm with the same bounds was presented by Berman and Kasiviswanathan [6].

It stems from all the previous results mentioned above that for $m = \Theta(n^2)$ there is a $(2 + (k - 2)/k, 0)$-approximation algorithm with an expected running time of $\tilde{O}(n^{2+1/k})$, for every $k \in \{2, 3, 4\}$. For $k \in \{2, 3\}$, there is a $(2 + (k - 2)/k, 0)$-approximation algorithm with expected running time of $\tilde{O}(n^2 + n^{2-\frac{3}{k}}m^{\frac{2}{k}})$. Therefore, in light of the existing results and the lack of progress in improving them for more than 15 years, a natural research question that arises is, whether there is a general scheme of algorithms with $(2+(k-2)/k, 0)$-approximation and $\tilde{O}(n^{2+1/k})$ running time, for every $k > 4$ or even more generally, is there a general scheme of algorithms with $(2+(k-2)/k, 0)$-approximation that work better for sparse graphs and have a running time of $\tilde{O}(n^2 + n^{2-\frac{3}{k}}m^{\frac{2}{k}})$, for every $k > 3$. Obtaining the latter scheme requires first to improve the $(5/2, 0)$-approximation algorithm with expected running time of $\tilde{O}(n^{9/4})$ of Kavitha [15]. In this paper we answer the more general question positively and prove:

▶ **Theorem 1.** *For every integer $k \geq 2$, there is an APASP algorithm with expected running time of $\tilde{O}(n^2 + m^{\frac{2}{k}}n^{2-\frac{3}{k}})$ and multiplicative approximation of $2 + \frac{k-2}{k}$.*

For $k = 2, 3$ we get the results of Baswana and Kavitha [4]. For $k = 4$, we improve upon the result of Kavitha [15] and get a $(5/2, 0)$-approximation algorithm with $\tilde{O}(n^2 + n^{\frac{5}{4}}m^{\frac{1}{2}})$ expected running time, which is better than $\tilde{O}(n^{9/4})$ when $m = o(n^2)$.

The main contribution of Theorem 1 is in unifying all previous algorithms into one general systematic scheme which reveals for the previous known algorithms with 2, 7/3 and 5/2 approximation, that rather than being accidental ad-hoc results, they are part of a family of algorithms that exhibits a smooth tradeoff between time and approximation.

Another aspect of Theorem 1 is that the current state of the art in conditional lower bounds only rules out the existence of a $(2-\varepsilon)$-approximation algorithm whose running time is better than the running time of FMM. No conditional lower bounds are currently known for approximation factors in the range between 2 and 3. Our result shed a light on this range from the upper bound perspective.

From the technical perspective our result is obtained by a combination of several observations with one important key ingredient that might be useful for generalizing also other algorithms that are based on distance computation. Very roughly speaking, the algorithms for $k = 2, 3$ of [4] and for $k = 4$ of [15] are obtained by using a case analysis in which one of the cases uses the following approach. For every $u, v \in V$ there is some edge $(a, b) \in E$ on a shortest path between u and v that is used to get an estimation to the distance between u and v based on approximate distance queries between u and b and v and a. We develop a generalization of this idea which is based on the query procedure of the *approximate distance oracles* (ADO) of Thorup and Zwick [26]. However, this key ingredient in its own is not enough to obtain Theorem 1. A tighter analysis of the algorithm of [4] for the case that $k = 2$ and a careful combination of it with our generalization technique is required.

The ADO of Thorup and Zwick [26] plays a pivotal role in many results related to approximating distances. Formally, Thorup and Zwick showed that for any integer $k \geq 1$ it is possible to preprocess a weighted undirected graph in $O(kmn^{1/k})$ expected time and to create ADO of size $O(kn^{1+1/k})$. For every $u, v \in V$ a query returns in $O(k)$ time a $(2k-1, 0)$-approximation. Many of the subsequent works on ADO were focused on improving the preprocessing time. (For more details see for example [4, 29, 23].) Baswana and Kavitha [4] showed that for $k > 2$ it is possible to compute ADO in $\tilde{O}(\min\{n^2, kmn^{1/k}\})$ expected running time. Other aspects of ADO were considered as well. For more details see for example [18, 29, 7, 8, 13, 14].

The rest of this paper is organized as follows. In the next section we present notations and review previous works. In Section 3 we present the main ingredients needed for obtaining the generalization. In Section 4 we present a couple of additional ingredients whose combination with the generalization presented in Section 3 yields the proof of Theorem 1.

2 Preliminaries

Let $G = (V, E)$ be a weighted undirected graph with $n = |V|$ vertices and $m = |E|$ edges. Let $\mathbf{w} : E \to \mathbb{R}^+$ be a weight function on the edges of G. For a vertex $u \in V$, let $N(u)$ be the set of neighbours of u including u itself. Let E_u be the set of incident edges of u. Let $E_u(i)$ be the i lightest edges[2] that are incident to u and let $N(u, i) = \{v \mid (u, v) \in E_u(i)\}$. Let $E(i) = \cup_{u \in V} E_u(i)$. Let $u, v \in V$ and let $d_G(u, v)$ be the distance between u and v in G, that is, the length of the shortest path between u and v. Let $S \subseteq V$. Let $p_S(u) = \arg\min_{w \in S} d_G(u, w)$ and let $d_G(u, S) = d_G(u, p_S(u))$. If $S = \emptyset$ then $d_G(u, S) = \infty$. Notice that $p_S(s) = s$ for every $s \in S$. Let $E_S(u) = \{(u, v) \in E_u \mid \mathbf{w}(u, v) < d_G(u, S)\}$ and let $E_S = \cup_{u \in V} E_S(u)$. Notice that in the degenerate case that $S = \emptyset$ we have $E_S = E$, since $d_G(u, S) = \infty$.

[2] Ties are broken arbitrarily. If i is not integral we take the $\lceil i \rceil$ lightest edges.

Several variants of the next two Lemmas were used in many of the previous papers [10, 9, 26]. For our needs we use the formulation of Baswana and Kavitha [4]:

▶ **Lemma 2** ([4]). *Given a weighted undirected graph $G = (V, E)$ and a set $S \subseteq V$ the following holds:*

1. *If $d_G(u, v) < d_G(u, S)$ then all the edges on a shortest path between u and v are in $G(V, E_S)$ and $d_{G(V,E_S)}(u, v) = d_G(u, v)$.*
2. *$d_{G(V,E_S \cup E_{p_S(u)})}(u, p_S(u)) = d_G(u, p_S(u))$.*

▶ **Lemma 3** ([4]). *If $v \in V$ is added to S with probability p then $\mathbb{E}[|E_S|]$ is $O(n/p)$.*

For every $u \in V$, the *ball* of u with respect to S is $B_S(u) = \{w \mid d_G(u, w) < d_G(u, S)\}$. An example to a ball is presented in The next definition of *overlapping* balls is used implicitly in many of the previous research on ADO and APASP. This definition was stated explicitly by Kavitha [15].

▶ **Definition 4** ([15]). *Balls $B_S(u)$ and $B_S(v)$ overlap if $d_G(u, S) + d_G(v, S) > d_G(u, v)$.*

The next Corollary from [15] is crucial for our new algorithms.

▶ **Corollary 5** ([15]). *If $P(u, v)$ is a shortest path and $B_S(u)$ and $B_S(v)$ overlap then $P(u, v)$ can be divided into a portion $P(u, a)$ in $B_S(u)$, a portion $P(v, b)$ in $B_S(v)$, and an edge (a, b)*[3].

Let $S_0 \subseteq V$, $S_{i+1} \subseteq S_i$, where $i \in [0, k]$ and $k \geq 1$. The set S_{i+1} is constructed by picking every vertex of S_i independently at random with probability $p \cdot c \log n$, for some constant c. We denote these $k + 1$ vertex sets with $\mathcal{S}_k^p$ and call them a *regular* hierarchy if $S_0 = V$ and $S_k = \emptyset$. Later we also define *augmented* and *mixed* hierarchies. We will usually refer to vertices in S_i, for $i > 0$, as pivots. For every $u \in V$ let $p_i(u) = p_{S_i}(u)$.

Thorup and Zwick [26] introduced approximate distance oracles (ADO). Given $k \geq 1$, an ADO of size $\tilde{O}(n^{1+1/k})$ can be constructed in $\tilde{O}(mn^{1/k})$ time. A query between any $u, v \in V$ returns in $O(k)$ time a $(2k - 1)$-multiplicative approximation of the distance. At the core of the ADO is the definition of a *bunch* $B_i(u)$ for every vertex $u \in V$ and $i \in [0, k - 1]$. For a given $i \in [0, k - 1]$ and $u \in V$ a bunch $B_i(u)$ is defined as follows: $B_i(u) = \{w \in S_i \setminus S_{i+1} \mid d_G(u, w) < d_G(u, p_{i+1}(u))\}$. Notice that if we translate the definition of bunches to the terminology of balls then for every $u \in V$ and $i \in [0, k - 1]$, $B_i(u) = S_i \cap B_{S_{i+1}}(u)$. In particular, for $i = 0$ we have that the bunch $B_0(u)$ is simply the ball $B_{S_1}(u)$. Thorup and Zwick [26] proved the following Lemma on the size of the bunches.

▶ **Lemma 6** ([26]). *Let $p = qc \log n$, where c is a large constant and $q \in (0, 1/(c \log n))$. For every $i \in [0, k-2]$ the size of $B_i(u)$ is $O(q^{-1} \log n)$ with high probability. The size of $B_{k-1}(u)$ is $O(n(q \log n)^{k-1})$, whp.*

The different bound on $|B_{k-1}(u)|$ is not really relevant for the ADO of Thorup and Zwick, as they set $q = n^{-1/k}$ and both bounds coincide at the same value. In our case, however, it raises a non trivial obstacle on the way to obtain efficient algorithms for sparse graphs. We will elaborate more on this issue later on.

The ADO is composed of an hierarchy $\mathcal{S}_k^p$, where $p = n^{-1/k} c \log n$, and $S_k = \emptyset$. For every $u \in V$ and $i \in [0, k]$, $p_i(u)$ and $B_i(u)$ are computed and saved in the data structure. Thorup and Zwick defined also *clusters* $C_i(w) = \{u \mid w \in B_i(u)\}$, for every $w \in S_i \setminus S_{i+1}$. The query algorithm is presented in Algorithm 1.

[3] Notice that in the degenerate case of a path of two edges either $u = a$ or $b = v$

Algorithm 1 DIST(u, v).

```
$i \leftarrow 0$;
while $p_i(u) \notin B_i(v)$ do
    $i \leftarrow i + 1$;
    swap $u$ and $v$;
end
return $d(u, p_i(u)) + d(v, p_i(u))$;
```

Algorithm 2 DIST(u, v, r).

```
$i \leftarrow r$;
while $p_i(u) \notin B_i(v)$ do
    $i \leftarrow i + 1$;
    swap $u$ and $v$;
end
return $d(u, p_i(u)) + d(v, p_i(u))$;
```

For the purpose of our new algorithms we generalize the query of Thorup and Zwick's ADO (see Algorithm 2) by adding an additional input value r. The search is for the first $i \geq r$, for which $p_i(u) \in B_i(v)$, alternating between u and v after each iteration. Let $f(u, v, r)$ be the value of i when DIST(u, v, r) ends.

▶ **Lemma 7.** *Let $u, v \in V$, $r \in [0, k-1]$ and $f = f(u, v, r)$. If $f-r$ is even then $d_G(u, p_f(u)) \leq d_G(u, p_r(u)) + (f-r)d_G(u, v)$. If $f-r$ is odd then $d_G(v, p_f(v)) \leq d_G(u, p_r(u)) + (f-r)d_G(u, v)$.*

Proof. Consider every $i \in [r, f-1]$. If $i - r$ is even then $p_i(u) \notin B_i(v)$ and from bunch definition together with the triangle inequality we get $d_G(v, p_{i+1}(v)) \leq d_G(v, p_i(u)) \leq d_G(u, v) + d_G(u, p_i(u))$. Similarly, if $i - r$ is odd then $p_i(v) \notin B_i(u)$ and $d_G(u, p_{i+1}(u)) \leq d_G(u, p_i(v)) \leq d_G(u, v) + d_G(v, p_i(v))$. This implies that for even $i-r$ we have $d_G(v, p_{i+1}(v)) \leq (i+1-r) \cdot d_G(u, v) + d_G(u, p_r(u))$ and for odd $i - r$ we have $d_G(u, p_{i+1}(u)) \leq (i+1-r) \cdot d_G(u, v) + d_G(u, p_r(u))$.

Thus, if $f - r$ is even we have $d_G(u, p_f(u)) \leq (f-r) \cdot d_G(u, v) + d_G(u, p_r(u))$ and if $f - r$ is odd we have $d_G(v, p_f(v)) \leq (f-r) \cdot d_G(u, v) + d_G(u, p_r(u))$.

As before we consider two subcases. The first is that r is even and the second is that r is odd.

In case that r is even then we let $f = f(a, u, r)$. ◀

3 A generalized scheme for APASP algorithms

3.1 Improving the pivots data

Baswana and Kavitha [4] presented a simple algorithm that given an hierarchy $\mathcal{S}_k^p$ computes the distance between every pair of vertices $(s, v) \in S_i \times V$ in the graph $(V, E_{S_{i+1}} \cup E_s)$, where $i \in [0, k-1]$, in expected running time of $\tilde{O}(n^{2+1/k})$.

In our new PIVOT-DIST algorithm we compute the bunches of every $u \in V$ with respect to hierarchy $\mathcal{S}_k^p$. We also change the way distances are computed for the pairs $(s, v) \in S_i \times V$ in the graph $(V, E_{S_{i+1}} \cup E_s)$ by using more edges and in particular edges which are not necessarily in E.

We define a new graph $G_{i+1}(s) = (V, H_{i+1}(s))$, for every $i \in [0, k-1]$ and $s \in S_i$ and run Dijkstra's algorithm for s in $G_{i+1}(s)$. The graph $G_{i+1}(s)$ is constructed as follows. The edge set $H_{i+1}(s)$ is initialized with $E_{S_{i+1}}$. The first change is that we add to $H_{i+1}(s)$ the set of edges $E(1/(p^{i+1}))$.

The second change is that for every $s \in S_i$ the set $H(s)$ contains an edge between s and every $u \in V$. The weight of every $(s, v) \in H(s) \setminus E_s$ is initially set to ∞. Every $(s, v) \in H(s) \cap E_s$ already has a weight. Next, we ensure in a loop that the weight of $(s, v) \in H(s)$ is at most $\min\{\mathbf{w}(s, v), \min_{u \in C_i(s) \cap N(v)}\{d_G(s, u) + \mathbf{w}(u, v)\}, \min_{u \in N(v) \wedge p_i(u) = s}\{d_G(s, u) + \mathbf{w}(u, v)\}\}$ and add $H(s)$ to $H_{i+1}(s)$.

As mentioned above we perform these computations using algorithm PIVOT-DIST. The additional data computed by PIVOT-DIST for the pairs $(s, v) \in S_i \times V$ enables us later on to perform more queries when computing the approximate distance between pairs $u, v \in V$.

Algorithm 3 PIVOT-DIST($\mathcal{S}_k^p$).

```
    $M[i,j] \leftarrow \infty$, for every $(i,j) \in [n] \times [n]$;
    for $i \leftarrow 0$ to $k-1$ do
        // Phase 1: Adding shortcut edges
        foreach $s \in S_i \setminus S_{i+1}$ do
            $H(s) \leftarrow \{(s,v) \mid v \in V\}$;
            foreach $e \in H(s) \setminus E_s$ do $\mathbf{w}(e) \leftarrow \infty$;
        end
        // Phase 2: Computing $d_G(u, S_i)$, $p_i(u)$, $B_i(u)$ and updating edge weights.
        foreach $u \in V$ do
            compute $d_G(u, S_i)$, $p_i(u)$ and $B_i(u)$;
            $M[p_i(u), u] \leftarrow d_G(u, S_i)$, $M[u, p_i(u)] \leftarrow d_G(u, S_i)$;
            $\mathbf{w}(p_i(u), u) = d_G(u, S_i)$;
(1)         foreach $(u,v) \in E_u$ do
             $\mathbf{w}(p_i(u), v) \leftarrow \min\{\mathbf{w}(p_i(u), v), d_G(p_i(u), u) + \mathbf{w}(u,v)\}$;
            foreach $s \in B_i(u)$ do $M[s,u] \leftarrow d_G(u,s)$, $M[u,s] \leftarrow d_G(u,s)$;
(2)         foreach $(u,v) \in E_u$ do
                foreach $s \in B_i(u)$ do $\mathbf{w}(s,v) \leftarrow \min\{\mathbf{w}(s,v), d_G(s,u) + \mathbf{w}(u,v)\}$;
            end
        end
        // Phase 3: Computing shortest paths for the vertices of $S_i$
        foreach $s \in S_i$ do
            $G_{i+1}(s) = (V, E_{S_{i+1}} \cup E(1/(p^{i+1})) \cup H(s))$;
            run Dijkstra's algorithm from $s$ in $G_{i+1}(s)$;
            foreach $u \in V$ do
                $M[s,u] \leftarrow \min\{M[s,u], d_{G_{i+1}(s)}(s,u)\}$,
                 $M[u,s] \leftarrow \min\{M[u,s], d_{G_{i+1}(s)}(s,u)\}$
            end
        end
    end
    return $M$;
```

A pseudocode of PIVOT-DIST($\mathcal{S}_k^p$) for computing distance information for $(s, v) \in S_i \times V$, for every $i \in [0, k-1]$, $s \in S_i$ and $v \in V$, is presented in Algorithm 3. A matrix M of size $n \times n$ is initialized with ∞ in every entry. We iterate on i from 0 to $k-1$. For every i we have three phases. In the first phase we iterate on the set S_i. For every $s \in S_i \setminus S_{i+1}$ the set $H(s)$ is initialized with edges between s and every $v \in V$. For every $(s, v) \in H(s) \setminus E_s$ we set $\mathbf{w}(s, v)$ to ∞. In the second phase for every $u \in V$ we compute $p_i(u)$, $d_G(u, S_i)$ and $B_i(u)$. We then set $M[p_i(u), u] = d_G(u, S_i)$ and $\mathbf{w}(p_i(u), u) = d_G(u, S_i)$. Notice that here we update the weight of edges in $H(s)$, some might be in the original graph and some might not. In line (1) we scan the edges of u and for each $(u, v) \in E_u$ we set $\mathbf{w}(p_i(u), v) = \min\{\mathbf{w}(p_i(u), v), d_G(p_i(u), u) + \mathbf{w}(u, v)\}$. We set $M[s, u] = d_G(u, s)$, for every

$s \in B_i(u)$. Notice that by this we ensure that we have for every $u \in V$ the distance between u and every vertex in its bunches, according to the bunch definition of Thorup and Zwick's ADO. In line (2) we scan the edges of u and the vertices of $B_i(u)$ and for every $(u,v) \in E_u$ and $s \in B_i(u)$ we set $\mathbf{w}(s,v) = \min\{\mathbf{w}(s,v), d_G(s,u) + \mathbf{w}(u,v)\}$.

In the third phase we iterate on the vertices of S_i. We compute the distance from every $s \in S_i$ to every vertex in V in the graph $G_{i+1}(s)$ by running Dijkstra's algorithm from s and update M accordingly. The matrix M is returned as the output. Next, we analyze the running time of Algorithm 3.

▶ **Lemma 8.** *PIVOT-DIST($\mathcal{S}^p_k$) runs in $\tilde{O}(n^2 + (k-1)n^2p^{-1} + mnp^{k-1})$ expected time.*

Proof. Initializing M takes $O(n^2)$ time. In the first phase of the loop on i we scan $S_i \setminus S_{i+1}$ and for every $s \in S_i \setminus S_{i+1}$ we add an edge between s and every vertex of V to the set $H(s)$. We then update the weight. This costs $O(|S_i \setminus S_{i+1}|n)$. The total cost of the first phase for every $i \in [0, k-1]$ is $O(n^2)$.

In the second phase we compute $p_i(u)$, $d_G(u, S_i)$ and $B_i(u)$ for every $u \in V$. It is easy to compute $p_i(u)$ and $d_G(u, S_i)$, for every $u \in V$, in $\tilde{O}(m)$ time by connecting a dummy vertex only to the vertices of S_i and running Dijkstra's algorithm from the dummy vertex.

Thorup and Zwick [26] showed that computing $B_i(u)$ for every $u \in V$ takes $\tilde{O}(\sum_{u\in V} |B_i(u)| \cdot |N(u)|)$ time (Section 4.3 in [26]). For $i \in [0, k-2]$ this results in a running time of $\tilde{O}(m \cdot p^{-1})$, since from Lemma 6 we have $B_i(u) = \tilde{O}(1/p)$. The cost of computing $B_{k-1}(u)$, for every $u \in V$, is $\tilde{O}(m \cdot np^{k-1})$, since from Lemma 6 we have $B_{k-1}(u) = \tilde{O}(np^{k-1})$.

For every i, the cost of line (1) is $\tilde{O}(\sum_{u\in V} |N(u)|) = \tilde{O}(m)$ and the cost of line (2) is $\tilde{O}(\sum_{u\in V} |B_i(u)| \cdot |N(u)|)$. Thus, the total cost of the second phase is $\tilde{O}(km \cdot p^{-1} + m \cdot np^{k-1})$.

In the third phase we run Dijkstra's algorithm from every $s \in S_i$ in $G_{i+1}(s)$. The set of edges of $G_{i+1}(s)$ is $E_{S_{i+1}} \cup E(1/(p^{i+1})) \cup H(s)$. The set $H(s)$ is of size $O(n)$. The set $E(1/(p^{i+1}))$ is of size $O(n/(p^{i+1}))$.

Consider now the set $E_{S_{i+1}}$, for $i < k-1$. The probability of a vertex to be in S_{i+1} is $\tilde{O}(p^{i+1})$. Applying Lemma 3 we get that the expected size of the set $E_{S_{i+1}}$ is $O(n/(p^{i+1}))$.

We get that the cost of the third phase for every inetger $i \in [0, k-2]$ is $\tilde{O}(|S_i| \cdot n/p^{i+1})$. Since the expected size of S_i is $\tilde{O}(np^i)$ we get a bound of $\tilde{O}(\sum_{i=0}^{k-2}(n \cdot p^i \cdot n/p^{i+1})$=$\tilde{O}(n^2 + (k-1)n^2p^{-1})$.

When $i = k-1$ we cannot apply Lemma 3 to bound the size of E_{S_k} since $S_k = \emptyset$, thus, we bound the cost of running Dijkstra's algorithm from every $s \in S_{k-1}$ in $G_k(s)$ with $\tilde{O}(|S_{k-1}|m) = \tilde{O}(n \cdot p^{k-1}m)$. We get a running time of $O(n^2 + (k-1)n^2p^{-1} + mnp^{k-1})$. ◀

In the next Lemma we summarize several properties of the matrix M returned by PIVOT-DIST. These properties are ensured by phase 2.

▶ **Lemma 9.** *Let $i \in [0, k-1]$, let $s \in S_i$ and let $u \in V$.*
1. *If $s \in \{p_i(u)\} \cup B_i(u)$ then $M[s,u] = d_G(s,u)$*
2. *If $s \in \{p_i(u)\} \cup B_i(u)$ and $(u,v) \in E$ then $M[s,v] \le d_G(s,u) + \mathbf{w}(u,v)$*

Proof. The proof easily follows from the execution of phase 2. ◀

We finish this section with a Lemma that summarizes several additional properties that follow from the structure of the graph $G_{i+1}(s)$, where $s \in S_i$ and $i \in [0, k-1]$. These properties are ensured by phase 3 of PIVOT-DIST.

▶ **Lemma 10.** *Let $u, v \in V$ and let $P(u,v)$ be a shortest path between u and v with at least two edges. Let $s \in S_i$ and let $p_i(u) = s$, where $i \in [0, k-1]$.*

1. *If $P(u,v)$ is in $G_{i+1}(s)$ then $M[s,v] \leq d_G(s,u) + d_G(u,v)$.*
2. *If $P(u,v)$ is not in $G_{i+1}(s)$ then $i \leq k-2$.*
3. *If $P(u,v)$ is not in $G_{i+1}(s)$ and $B_{S_{i+1}}(u)$ and $B_{S_{i+1}}(v)$ overlap, where $(a,b) \in P(u,v)$ is the edge that connects them, then $(a,b) \notin E_a(1/p^{i+1}) \cup E_b(1/p^{i+1}) \cup E_{S_{i+1}}$.*

Proof.

1. The claim follows since $P(u,v)$ is in $G_{i+1}(p_i(u))$, $(p_i(u), u) \in H(p_i(u))$ and $\mathbf{w}(p_i(u), u) = d_G(p_i(u), u)$.
2. Assume towards a contradiction that $i = k-1$. The graph $G_k(p_{k-1}(u))$ contains the edges E_{S_k} and since $S_k = \emptyset$ we have $E_{S_k} = E$. Thus, $G_k(p_{k-1}(u))$ contains E and in particular, $P(u,v)$ is $G_k(p_{k-1}(u))$, a contradiction.
3. Since $d_G(u,a) < d_G(u, p_{i+1}(u))$ and $d_G(v,b) < d_G(v, p_{i+1}(v))$ it follows from Lemma 2 that $P(u,a)$ and $P(v,b)$ are in $E_{S_{i+1}}$ and thus in $G_{i+1}(p_i(u))$. Since $P(u,v)$ is not in $G_{i+1}(p_i(u))$ it must be that $(a,b) \notin E_a(1/p^{i+1}) \cup E_b(1/p^{i+1}) \cup E_{S_{i+1}}$[4]. ◀

3.2 A general scheme

We turn now to describe our new and general scheme for APASP algorithm. We denote this algorithm with APASP. Our new algorithm is obtained by using PIVOT-DIST($\mathcal{S}_k^p$) to obtain better estimations in several important cases. These better estimations are then combined with a procedure similar to the query of Thorup and Zwick distance oracles. The main challenge is in the analysis of the approximation factor.

The algorithm gets as an input a weighted undirected graph G and an hierarchy $\mathcal{S}_k^p$. The algorithm returns a matrix M of approximate distances.

The algorithm works as follows. We start by initializing the matrix M. For every $(i,j) \in E$ we set $M[i,j]$ to $\mathbf{w}(i,j)$ and for every $(i,j) \notin E$ we set $M[i,j]$ to ∞. Next, we run PIVOT-DIST($\mathcal{S}_k^p$) and update matrix M with the result. Finally, we scan for every $u, v \in V$ the vertices $p_i(u)$, for every $i \in [0, k-1]$ and the vertices in $B_i(u)$, for every $i \in [0, k-2]$. We update $M[u,v]$ if we find a vertex w such that $M[u,w] + M[v,w] < M[u,v]$. Notice that we are not scanning the vertices of $B_{k-1}(u)$ deliberately. This caveat is because the size of $B_{k-1}(u)$ is $\tilde{O}(np^{k-1})$, for every $u \in V$, thus scanning these bunches is prohibited if we like to obtain an algorithm with an efficient running time in sparse graphs. A careful case analysis of odd and even values of k allows us to avoid scanning $B_{k-1}(u)$ without affecting the approximation factor. A pseudocode of APASP is presented in Algorithm 4.

We now analyze the approximation of the algorithm.

▶ **Lemma 11.** *The output M of APASP$(G, \mathcal{S}_k^p)$ satisfies $d_G(u,v) \leq M[u,v] \leq 2d_G(u,v) + \frac{k-2}{k} \cdot d_G(u,v)$.*

Proof. Let $u, v \in V$. Let $P(u,v)$ be a shortest path between u and v. If $P(u,v)$ has a single edge then $M[u,v] = \mathbf{w}(u,v)$ since we set $M[u,v]$ to $\mathbf{w}(u,v)$ in APASP. Thus, we can assume that $P(u,v)$ has at least 2 edges. Let $i \in [0, k-1]$ be the largest index such that $d_G(u, S_i) + d_G(v, S_i) \leq d_G(u,v)$. Such an index must exist since $S_0 = V$ which implies that $d_G(u, S_0) + d_G(v, S_0) = 0 \leq d_G(u,v)$.

[4] If $P(u,v)$ has two edges then the claim hold since either $v = b$ or $u = a$.

Algorithm 4 APASP$(G, \mathcal{S}_k^p)$.

```
    foreach $(i,j) \in [n] \times [n]$ do $M[i,j] \leftarrow \infty$;
    foreach $(i,j) \in E$ do $M[i,j] \leftarrow \mathbf{w}(i,j)$;
    $M \leftarrow \min\{M, \text{PIVOT-DIST}(\mathcal{S}_k^p)\}$;
    foreach $u \in V$ do
        for $i \leftarrow 0$ to $k-1$ do
            foreach $v \in V$ do
(1)             $M[u,v] \leftarrow \min\{M[u,v], M[p_i(u),u] + M[p_i(u),v], M[v,u]\}$;
                if $i \leq k-2$ then
                    foreach $w \in B_i(u)$ do
(2)                     $M[u,v] \leftarrow \min\{M[u,v], M[w,u] + M[w,v], M[v,u]\}$
                    end
                end
            end
        end
    end
    return $M$;
```

Assume, wlog, that $d_G(u, S_i) \leq d_G(v, S_i)$. If $P(u,v)$ is in $G_{i+1}(p_i(u))$ it follows from Lemma 9(i) and from Lemma 10(i) that after updating M with the result of PIVOT-DIST $M[p_i(u), u] = d_G(p_i(u), u)$ and $M[p_i(u), v] \leq d_G(p_i(u), u) + d_G(u, v)$. In line (1) of APASP we update $M[u,v]$ if needed, therefore, it is guaranteed that $M[u,v] \leq 2d_G(p_i(u), u) + d_G(u,v) \leq 2d_G(u,v)$.

Consider now the case that $P(u,v)$ is not in $G_{i+1}(p_i(u))$. From Lemma 10(ii) it follows that $i < k-1$. Since i is the largest index for which we have $d_G(u, S_i) + d_G(v, S_i) \leq d_G(u,v)$ it follows that $d_G(u, S_{i+1}) + d_G(v, S_{i+1}) > d_G(u,v)$, which implies that $B_{S_{i+1}}(u)$ and $B_{S_{i+1}}(v)$ overlap. Let $(a,b) \in P(u,v)$ be the edge that connects $B_{S_{i+1}}(u)$ and $B_{S_{i+1}}(v)$. From Lemma 10(iii) it follows that $(a,b) \notin E_a(1/p^{i+1})$ and $(a,b) \notin E_b(1/p^{i+1})$. Therefore, for the rest of the proof we can assume that $P(u,v)$ has at least 2 edges, $i < k-1$, $P(u,a)$ is in $G(V, E_{S_{i+1}})$, $P(v,b)$ is in $G(V, E_{S_{i+1}})$ and $(a,b) \notin E_a(1/p^{i+1}) \cup E_b(1/p^{i+1}) \cup E_{S_{i+1}}$.

Next, we will prove two different bounds on $M[u,v]$.

▷ Claim 12. $M[u,v] \leq \min\{3d_G(u,v) - 2d_G(b,v), 3d_G(u,v) - 2d_G(a,u)\}$.

Proof. Let $j > i+1$ be the smallest index for which $(a,b) \in E(1/p^j)$. If $j \geq k$ we set j to k. Assume that $(a,b) \in E_a(1/p^j)$ and $(a,b) \in E_b(1/p^{j'})$, where $j' \geq j$. If $j = k$ we set j' to k. By definition, the sets $N(a, 1/p^{j-1})$ and $N(b, 1/p^{j-1})$ are of size $1/p^{j-1}$. Since vertices of V are in S_{j-1} with probability at least $p^{j-1} c \log n$, it follows that, whp, S_{j-1} contains at least one vertex from $N(a, 1/p^{j-1})$ and $N(b, 1/p^{j-1})$.

Let $x \in N(a, 1/p^{j-1}) \cap S_{j-1}$ and let $y \in N(b, 1/p^{j-1}) \cap S_{j-1}$. Since $(a,x) \in E_a(1/p^{j-1})$ and $(a,b) \notin E_a(1/p^{j-1})$ we have $\mathbf{w}(a,x) \leq \mathbf{w}(a,b)$. Similarly, since $(b,y) \in E_b(1/p^{j-1})$ and $(a,b) \notin E_b(1/p^{j-1})$ we have $\mathbf{w}(b,y) \leq \mathbf{w}(a,b)$. Thus, we get that $d_G(u, p_{j-1}(u)) \leq d_G(u,a) + \mathbf{w}(a,b) \leq d_G(u,b)$ and $d_G(v, p_{j-1}(v)) \leq d_G(v,b) + \mathbf{w}(a,b) \leq d_G(v,a)$.

Now since $(a,b) \in E_a(1/p^j)$ it follows that the path $P(u,v)$ is in $G_j(p_{j-1}(u))$ and $G_j(p_{j-1}(v))$, when $j < k$. When $j = k$ we have $E_{S_k} = E$ and $P(u,v)$ is in $G_k(p_{k-1}(u))$. It follows from Lemma 10(i) that $M[p_{j-1}(u), v] \leq d_G(p_{j-1}(u), u) + d_G(u,v)$ and $M[p_{j-1}(v), u] \leq d_G(p_{j-1}(v), v) + d_G(u,v)$.

Thus, after line (1) of APASP is executed for the pair u, v and for the pair v, u we have $M[u,v] = M[v,u] \leq \min\{2d_G(u, p_{j-1}(u)) + d_G(u,v), 2d_G(v, p_{j-1}(v)) + d_G(u,v)\}$. We get:

$$\begin{aligned} M[v,u] = M[u,v] &\leq \min\{2d_G(u,p_{j-1}(u)) + d_G(u,v), 2d_G(v,p_{j-1}(v)) + d_G(u,v)\} \\ &\leq \min\{d_G(u,v) + 2d_G(u,b), d_G(u,v) + 2d_G(v,a)\} \\ &= \min\{3d_G(u,v) - 2d_G(b,v), 3d_G(u,v) - 2d_G(a,u)\}, \end{aligned}$$

since $d_G(u,p_{j-1}(u)) \leq d_G(u,b)$ and $d_G(v,p_{j-1}(v)) \leq d_G(v,a)$ and since $d_G(u,b) = d_G(u,v) - d_G(b,v)$ and $d_G(v,a) = d_G(u,v) - d_G(a,u)$. ◁

We now turn to prove a second bound on $M[u,v]$.

▷ Claim 13. $M[u,v]$ is bounded either by (i) $2 \cdot (k-1)d_G(u,a) + d_G(u,v)$ or by (ii) $2 \cdot (k-1)d_G(v,b) + d_G(u,v)$

Proof. We consider the pair of vertices u and a and the pair of vertices v and b.

Let r be the largest index such that $a \notin B_{S_r}(u)$ and $u \notin B_{S_r}(a)$. Let r' be the largest index such that $b \notin B_{S_{r'}}(v)$ and $v \notin B_{S_{r'}}(b)$. Since $S_0 = V$ it follows that $B_{S_0}(x) = \{x\}$, for every $x \in V$. Thus, we have $a \notin B_{S_0}(u)$ and $u \notin B_{S_0}(a)$ and also $b \notin B_{S_0}(v)$ and $v \notin B_{S_0}(b)$.

We assume that $r \geq r'$ and show that $M[u,v] \leq 2 \cdot (k-1)d_G(u,a) + d_G(u,v)$. If $r' \geq r$ then a symmetric proof shows that $M[u,v] \leq 2 \cdot (k-1)d_G(v,b) + d_G(u,v)$. In the degenerate case that $P(u,v)$ has only two edges and $b = v$ we set $r' = r$. The proof below works for this case as well and hence $M[u,v] \leq 2 \cdot (k-1)d_G(u,a) + d_G(u,v)$. If $a = u$ a symmetric proof shows that $M[u,v] \leq 2 \cdot (k-1)d_G(v,b) + d_G(u,v)$.

Recall that we are in the case that $d_G(u,a) < d_G(u, p_{i+1}(u))$ and $d_G(v,b) < d_G(v, p_{i+1}(v))$, thus we have $a \in B_{S_{i+1}}(u)$ and $b \in B_{S_{i+1}}(v)$. This implies that $r \leq i < k-1$ and $r' \leq i < k-1$, since $i < k-1$.

From the definition of r it follows that either $a \in B_{S_{r+1}}(u)$ or $u \in B_{S_{r+1}}(a)$. If $a \in B_{S_{r+1}}(u)$ then $d_G(u,a) < d_G(u, S_{r+1})$ and it follows from Lemma 2 that $P(u,a) \in E_{S_{r+1}}$. Similarly, if $u \in B_{S_{r+1}}(a)$ then $d_G(u,a) < d_G(a, S_{r+1})$ and it follows from Lemma 2 that $P(u,a) \in E_{S_{r+1}}$. From symmetrical arguments we get that $P(v,b) \in E_{S_{r'+1}}$. Since $E_{S_j} \subseteq E_{S_{j+1}}$, for every $j \in [0, k-1]$, we have and $P(u,a) \cup P(v,b) \subseteq E_{S_{q+1}}$, for every $q \geq r$.

Next, we distinguish between odd and even values of k. Assume first that k is odd. This implies that $k-1$ is even. We now consider two subcases. The first is that r is even and the second is that r is odd. In case that r is even then we let $f = f(u,a,r)$.

From Lemma 7 it follows that if $f - r$ is even then we have $d_G(u, p_f(u)) \leq d_G(u, p_r(u)) + (f-r)d_G(u,a)$ and if $f-r$ is odd then $d_G(a, p_f(a)) \leq d_G(u, p_r(u)) + (f-r)d_G(u,a)$.

We proceed by considering these two scenarios.

1. **Even $f - r$.** In this case $d_G(u, p_f(u)) \leq d_G(u, p_r(u)) + (f-r)d_G(u,a)$ and $p_f(u) \in B_f(a)$. It follows from Lemma 9(ii) that after Phase 2 of PIVOT-DIST $M[p_f(u), b] \leq d_G(p_f(u), a) + \mathbf{w}(a,b)$. When executing in Phase 3 of PIVOT-DIST Dijkstra's algorithm from $p_f(u)$ in $G_{f+1}(p_f(u))$ with edge set $E_{S_{f+1}} \cup E(1/(p^{f+1})) \cup H(p_f(u))$ the weight of the edge $(p_f(u), b) \in H(p_f(u))$ is $M[p_f(u), b] \leq d_G(p_f(u), a) + \mathbf{w}(a,b)$ and $P(v,b) \subseteq E_{S_{f+1}}$. Thus, we get that $M[p_f(u), v] \leq d_G(p_f(u), a) + d_G(a,v) \leq d_G(p_f(u), u) + d_G(u,v)$. In line (1) of APASP we update $M[u,v]$ so that it is at most $M[p_f(u), u] + M[p_f(u), v]$. From Lemma 9(i) it follows that $M[p_f(u), u] = d_G(p_f(u), u)$ and since $M[p_f(u), v] \leq d_G(p_f(u), u) + d_G(u,v)$ we get that $M[u,v] \leq 2 \cdot d_G(p_f(u), u) + d_G(u,v)$. (See
Combining this with the fact that $d_G(u, p_f(u)) \leq d_G(u, p_r(u)) + (f-r)d_G(u,a)$ we get that for even $f - r$ we have $M[u,v] \leq 2 \cdot (d_G(u, p_r(u)) + (f-r)d_G(u,a)) + d_G(u,v)$. Recall also that $f \leq k-1$. In the degenerate case of $r = 0$ we have $p_0(u) = u$ and $M[u,v] \leq 2 \cdot (k-1)d_G(u,a) + d_G(u,v)$.

For the case that $r \geq 1$ since $a \notin B_{S_r}(u)$ we have $d_G(u,p_r(u)) \leq d_G(u,a)$. Using this we get $M[u,v] \leq 2 \cdot ((f-r+1)d_G(u,a)) + d_G(u,v) \leq 2 \cdot (k-1)d_G(u,a) + d_G(u,v)$.

2. **Odd $f-r$.** In this case $d_G(a,p_f(a)) \leq d_G(u,p_r(u)) + (f-r)d_G(u,a)$ and $p_f(a) \in B_f(u)$. It follows from Lemma 9(ii) that after Phase 2 of PIVOT-DIST $M[p_f(a),b] \leq d_G(p_f(a),a) + \mathbf{w}(a,b)$. When executing in Phase 3 of PIVOT-DIST Dijkstra's algorithm from $p_f(a)$ in $G_{f+1}(p_f(a))$ with edge set $E_{S_{f+1}} \cup E(1/(p^{f+1})) \cup H(p_f(a))$ the weight of the edge $(p_f(a),b) \in H(p_f(a))$ is $M[p_f(a),b] \leq d_G(p_f(a),a) + \mathbf{w}(a,b)$ and $P(v,b) \subseteq E_{S_{f+1}}$. Thus, we get that $M[p_f(a),v] \leq d_G(p_f(a),a) + d_G(a,v)$.
   Consider the execution of line (2) in APASP for vertices v, u and a and the index f. Notice that this line is only executed for indices $i \leq k-2$. In our case we have $p_f(a) \in B_f(u)$ and since r is even and $f-r$ is odd, f must be odd. Since $k-1$ is even we have $f \leq k-2$. Thus line (2) is executed and we update $M[u,v]$ so that it is at most $M[p_f(a),u] + M[p_f(a),v]$. Since $p_f(a) \in B_f(u)$ it follows from Lemma 9(i) that $M[p_f(a),u] = d_G(p_f(a),u) \leq d_G(u,a) + d_G(p_f(a),a)$ and since $M[p_f(a),v] \leq d_G(p_f(a),a) + d_G(a,v)$ we get that $M[u,v] \leq 2 \cdot d_G(p_f(a),a) + d_G(u,v)$.
   Combining this with the fact that $d_G(a,p_f(a)) \leq d_G(u,p_r(u)) + (f-r)d_G(u,a)$ we get that for odd $f-r$ we have $M[u,v] \leq 2 \cdot (d_G(u,p_r(u)) + (f-r)d_G(u,a)) + d_G(u,v)$. Recall also that $f \leq k-1$.
   In the degenerate case of $r=0$ we have $p_0(u) = u$ and $M[u,v] \leq 2 \cdot (k-1)d_G(u,a) + d_G(u,v)$.
   For the case that $r \geq 1$ since $a \notin B_{S_r}(u)$ we have $d_G(u,p_r(u)) \leq d_G(u,a)$. Using this we get $M[u,v] \leq 2 \cdot ((f-r+1)d_G(u,a)) + d_G(u,v) \leq 2 \cdot (k-1)d_G(u,a) + d_G(u,v)$.

Consider now the second case in which r is odd and let $f = f(a,u,r)$.

1. **Odd $f-r$.** In this case $d_G(u,p_f(u)) \leq d_G(a,p_r(a)) + (f-r)d_G(u,a)$ and $p_f(u) \in B_f(a)$. The proof is identical to the proof for even r, $f = f(u,a,r)$ and even $f-r$.
2. **Even $f-r$.** In this case $d_G(a,p_f(a)) \leq d_G(a,p_r(a)) + (f-r)d_G(u,a)$ and $p_f(a) \in B_f(u)$. The proof is identical to the proof for even r, $f = f(u,a,r)$ and odd $f-r$.

Assume now that k is even. This implies that $k-1$ is odd.

1. **Odd $f-r$.** In this case $d_G(u,p_f(u)) \leq d_G(a,p_a(u)) + (f-r)d_G(u,a)$ and $p_f(u) \in B_f(a)$. The proof is identical to the proof for even r, $f = f(u,a,r)$ and even $f-r$.
2. **Even $f-r$.** In this case $d_G(a,p_f(a)) \leq d_G(a,p_r(a)) + (f-r)d_G(u,a)$ and $p_f(a) \in B_f(u)$. The proof is identical to the proof for even r, $f = f(u,a,r)$ and odd $f-r$.

Consider now the second case in which r is odd and let $f = f(u,a,r)$.

1. **Even $f-r$.** In this case $d_G(u,p_f(u)) \leq d_G(u,p_r(u)) + (f-r)d_G(u,a)$ and $p_f(u) \in B_f(a)$. The proof is identical to the proof for even r, $f = f(u,a,r)$ even $f-r$.
2. **Odd $f-r$.** In this case $d_G(a,p_f(a)) \leq d_G(u,p_r(u)) + (f-r)d_G(u,a)$ and $p_f(a) \in B_f(u)$. The proof is identical to the proof for even r, $f = f(u,a,r)$ and odd $f-r$. ◁

We now combine the two bounds to complete the proof. From Claim 13 it follows that either $M[u,v] \leq 2 \cdot (k-1)d_G(u,a) + d_G(u,v)$ or $M[u,v] \leq 2 \cdot (k-1)d_G(v,b) + d_G(u,v)$. Assume that $M[u,v] \leq 2 \cdot (k-1)d_G(u,a) + d_G(u,v)$. From Claim 12 it follows that $M[u,v] \leq \min\{3d_G(u,v) - 2d_G(b,v), 3d_G(u,v) - 2d_G(a,u)\}$. Thus, $M[u,v] \leq \min\{2 \cdot (k-1)d_G(u,a) + d_G(u,v), 3d_G(u,v) - 2d_G(a,u)\}$. Let $X = 2 \cdot (k-1)d_G(u,a) + d_G(u,v)$ and let $Y = 3d_G(u,v) - 2d_G(a,u)$. When $d_G(u,a) = d_G(u,v)/k$, we have $X = Y = 2d_G(u,v) + \frac{k-2}{k} \cdot d_G(u,v)$. When $d_G(u,a) < d_G(u,v)/k$ we have $X < 2d_G(u,v) + \frac{k-2}{k} \cdot d_G(u,v)$. When $d_G(u,a) > d_G(u,v)/k$ we have $Y < 2d_G(u,v) + \frac{k-2}{k} \cdot d_G(u,v)$. Since $M[u,v] \leq \min\{X,Y\}$ we get that $M[u,v] \leq 2d_G(u,v) + \frac{k-2}{k} \cdot d_G(u,v)$.

Consider the degenerate case in which $P(u, v)$ has only two edges, and assume that $b = v$. From Claim 13 it follows that $M[u, v] \leq 2 \cdot (k-1)d_G(u, a) + d_G(u, v)$. From Claim 12 it follows that $M[u, v] \leq 3d_G(u, v) - 2d_G(a, u)$. Thus, from the same arguments as above, $M[u, v] \leq 2d_G(u, v) + \frac{k-2}{k} \cdot d_G(u, v)$. ◀

We now turn to analyze the running time of the algorithm.

▶ **Lemma 14.** *The expected running time of* APASP$(G, \mathcal{S}_k^p)$ *is* $\tilde{O}(n^2 + (k-1)n^2p^{-1} + mnp^{k-1})$

Proof. Initializing M takes $O(n^2)$ time. From Lemma 8 it follows that the call to PIVOT-DIST takes $\tilde{O}(n^2 + (k-1)n^2p^{-1} + mnp^{k-1})$ time. Finally, for every $u \in V$, $i \in [0, k-2]$ and $v \in V$ we do $|B_i(u)| + 1$ read operations to matrix M. This takes $\tilde{O}(n^2 + k(n^2p^{-1}))$ time, since $|B_i(u)| = \tilde{O}(p^{-1})$, for $i \in [0, k-2]$. ◀

We can now prove:

The approach presented so far powerful enough on its own to obtain the following non trivial generalization, which already improves the general scheme presented by Cohen and Zwick [9].

▶ **Theorem 15.** *For every integer $k \geq 2$, there is an APASP algorithm with expected running time of $\tilde{O}(kn^{2-1/k}m^{1/k})$ and multiplicative approximation of $2 + \frac{k-2}{k}$.*

Proof. We run APASP with the hierarchy $\mathcal{S}_k^p$. From Lemma 14 it follows that the running time is $\tilde{O}(n^2 + (k-1)n^2p^{-1} + mnp^{k-1})$. Setting $n^2p^{-1} = mnp^{k-1}$ we get $p = \left(\frac{n^2}{mn}\right)^{1/k}$. Thus, the running time is $\tilde{O}(kn^{2-1/k}m^{1/k})$. From Lemma 11 it follows that the multiplicative approximation is $2 + \frac{k-2}{k}$. ◀

4 A faster scheme

In this section we first present augmented hierarchies. This is basically the first part of the 2-approximation algorithm of Baswana and Kavitha [4]. We provide a tighter analysis that relaxes the requirements for getting a 2-approximation. We then present the idea of mixed hierarchies, a combination of a regular hierarchy with the last set of the augmented hierarchy. This allows us to connect our algorithm from Section 3, that worked with a base set V, to the first part of the 2-approximation algorithm, by forcing the second set of a mixed hierarchy to be the last set of the augmented hierarchy. We end by proving Theorem 1.

4.1 Augmented hierarchy with $\hat{S}_k \neq \emptyset$

Thorup and Zwick [25] showed that if a set $S \subseteq V$ is constructed by a careful recursive sampling procedure then the maximum size of every cluster is bounded as well. They proved:

▶ **Lemma 16** ([25])**.** *Given a parameter p, we can compute a set S of size $\tilde{O}(np)$ in $\tilde{O}(mp^{-1})$ expected time such that, $|C_S(w)| = O(1/p)$ for every vertex $w \in V \setminus S$, and $|B_S(v)| = O(1/p)$ for every $v \in V$.*

Following Baswana and Kavitha [4] we will extend the vertex hierarchy to an *augmented hierarchy*, denoted with $\hat{\mathcal{S}}_k^p$. Let $\mathcal{S}_k^p$ be a regular hierarchy with $S_0 = V$ and $S_k = \emptyset$. Let S be a set computed using Lemma 16 with parameter p^k. In $\hat{\mathcal{S}}_k^p$ we have $\hat{S}_i = S_i \cup S$, for every $i \in [0, k]$. In particular, we have $\hat{S}_k = S$. We also set $\hat{S}_{k+1} = \emptyset$. We refer to S as the augmenting set of the hierarchy.

The pivots data that we compute for an augmented hierarchy $\hat{\mathcal{S}}_k^p$ is computed using algorithm AUG-PIVOT-DIST. Algorithm AUG-PIVOT-DIST differs from PIVOT-DIST by an additional special phase that is added between the first phase and the second phase and by avoiding the explicit computation of bunches in phase 2. The special phase is devoted for the set $\hat{S}_k$. For every $u \in V$, we compute $B_{\hat{S}_k}(u)$ and $C_{\hat{S}_k}(u)$. For every $a \in B_{\hat{S}_k}(u) \cup C_{\hat{S}_k}(u)$ and $i \in [0, k]$ we scan the edges of a and update the weight of the edge between $p_i(u)$ and every $b \in N(a)$ so that it will be at most $d_G(p_i(u), u) + d_G(u, a) + \mathbf{w}(a, b)$.

Next, we analyze the running time of AUG-PIVOT-DIST.

▶ **Lemma 17.** *If S is computed using Lemma 16 with parameter p^k then the expected running time of Algorithm* AUG-PIVOT-DIST*($\hat{\mathcal{S}}_k^p$) is $\tilde{O}(n^2 + (k-1)n^2p^{-1} + kmp^{-k})$*

Proof. Phase 1 remains unchanged, therefore, its cost remains $\tilde{O}(n^2)$ as in Lemma 8 .

In the special phase we compute for every $u \in V$ the ball $B_S(u)$ in $\tilde{O}(\sum_{u \in V} deg(u)|B_S(u)|)$ time as was shown by Baswana and Kavitha [4]. Since $|B_S(u)| = O(p^{-k})$ this part takes $\tilde{O}(mp^{-k})$ time. Since clusters are simply the inverse of balls we can compute them at the same cost. For every $a \in B_{\hat{S}_k}(u) \cup C_{\hat{S}_k}(u)$ and for every $i \in [0, k-1]$ we scan the edges of a and update for every $b \in N(a)$ the value of $\mathbf{w}(p_i(u), b)$, if needed. The total cost of this is $O(k \sum_{a \in V} |C_{\hat{S}_k}(a) \cup B_{\hat{S}_k}(a)| deg(a)) = O(kmp^{-k})$. In phase 2 we do not compute bunches as before and thus the cost is only $\tilde{O}(m)$ for the computation of $p_i(u)$ and $d_G(u, S_i)$. In phase 3 we no longer have to split the analysis of the case that $i = k-1$ from the analysis of the general case. This is due to the fact that $\hat{S}_k \neq \emptyset$. Since a vertex is in $\hat{S}_k$ with probability less than p^k we can apply Lemma 3 and get that the expected size of E_{S_k} is $O(n/(p^k))$. Therefore, phase 3 takes $\tilde{O}(n^2 + (k-1)n^2p^{-1})$ time. ◀

We use AUG-PIVOT-DIST for computing APASP in an algorithm denoted with AUG-APASP. The running time analysis of AUG-APASP stems from Lemma 17 and Lemma 14.

▶ **Corollary 18.** AUG-APASP*($G, \hat{\mathcal{S}}_k^p$) has $\tilde{O}(n^2 + kn^2p^{-1} + kmp^{-k})$ expected running time.*

In order to analyse the approximation produced by AUG-APASP we introduce the following definition which allows us to relax the condition required for proving a 2 approximation.

▶ **Definition 19.** *Let $u, v \in V$ and let $S \subseteq V$. We say that u and v are covered by balls $B_S(u)$ and $B_S(v)$ if there is a shortest path P between u and v such that $P \subseteq B_S(u) \cup B_S(v)$.*

The next Lemma follows easily from the above definition.

▶ **Lemma 20.** *If $B_S(u)$ and $B_S(v)$ do not cover $u, v \in V$ then $d_G(u, S) + d_G(v, S) \leq d_G(u, v)$*

Proof. As u and v are not covered by $B_S(u)$ and $B_S(v)$ every shortest path $P(u, v)$ has a vertex $w \notin B_S(u) \cup B_S(v)$. Thus, $d_G(u, S) \leq d_G(u, w)$ and $d_G(v, S) \leq d_G(v, w)$. Since $w \in P(u, v)$ we get $d_G(u, v) = d_G(u, w) + d_G(v, w)$ and $d_G(u, S) + d_G(v, S) \leq d_G(u, v)$. ◀

We are now ready to bound the approximation produced by AUG-APASP.

▶ **Lemma 21.** *Let $\hat{\mathcal{S}}_k^p$ be an augmented hierarchy with $\hat{S}_0 = V$ and $\hat{S}_k = S$. If $B_{\hat{S}_k}(u)$ and $B_{\hat{S}_k}(v)$ do not cover $u, v \in V$ then* AUG-APASP*($G, \hat{\mathcal{S}}_k^p$) returns a matrix M that satisfies: $d_G(u, v) \leq M[u, v] \leq 2d_G(u, v)$.*

Proof. Let $u, v \in V$. Let $i \in [0, k]$ be the smallest index such that $B_{\hat{S}_i}(u)$ and $B_{\hat{S}_i}(v)$ cover u and v. Such an index must exist since u and v are covered by $B_{\hat{S}_k}(u)$ and $B_{\hat{S}_k}(v)$. Since $B_{\hat{S}_0}(v) = \{v\}$ and $B_{\hat{S}_0}(u) = \{u\}$ and since we can assume that any shortest path between u

and v is of at least two edges we have that u and v are not covered by $B_{\hat{S}_0}(v)$ and $B_{\hat{S}_0}(u)$. Thus, $i > 0$. From Lemma 20 it follows that $d_G(u, \hat{S}_{i-1}) + d_G(v, \hat{S}_{i-1}) \leq d_G(u, v)$. Assume, wlog, that $d_G(u, \hat{S}_{i-1}) \leq d_G(v, \hat{S}_{i-1})$. Let $P(u, v)$ be a shortest path between u and v such that $P(u, v) \subseteq B_{\hat{S}_i}(u) \cup B_{\hat{S}_i}(v)$.

We can partition $P(u, v)$ into three portions. A portion $P(u, a)$ between u and a in $B_{\hat{S}_i}(u)$, a portion $P(b, v)$ between v and b in $B_{\hat{S}_i}(v)$, and an edge (a, b). Since $d_G(u, a) < d_G(u, p_i(u))$ it follows from Lemma 2 that $P(u, a)$ is in $G(V, E_{\hat{S}_i})$ and $d_{(V,E(\hat{S}_i))}(u, a) = d_G(u, a)$. Similarly, $P(v, b)$ is in $G(V, E_{\hat{S}_i})$ and $d_{(V,E(\hat{S}_i))}(v, b) = d_G(v, b)$.

Now $B_{\hat{S}_i}(u) \subseteq B_{\hat{S}_k}(u)$. This implies that $a \in B_{\hat{S}_k}(u)$ and $\mathbf{w}(p_{i-1}(u), b)$ was updated in the special phase of AUG-PIVOT-DIST such that $\mathbf{w}(p_{i-1}(u), b) \leq d_G(p_{i-1}(u), u) + d_G(u, a) + \mathbf{w}(a, b)$. After running Dijkstra's algorithm in phase 3 of AUG-PIVOT-DIST for $p_{i-1}(u)$ in $G_i(p_{i-1}(u))$ we have $M[p_{i-1}(u), v] \leq d_G(p_{i-1}(u), u) + d_G(u, v)$. Thus, after updating the M with the result of AUG-PIVOT-DIST we have $M[p_{i-1}(u), v] \leq d_G(p_{i-1}(u), u) + d_G(u, v)$ and $M[p_{i-1}(u), u] = d_G(p_{i-1}(u), u)$. In line (1) of AUG-APASP we update $M[u, v]$ if needed, therefore, it is guaranteed that $M[u, v] \leq 2d_G(p_{i-1}(u), u) + d_G(u, v) \leq 2d_G(u, v)$. ◀

4.2 A mixed hierarchy

Let S be the augmenting set of an augmented hierarchy. We define a *mixed hierarchy* as follows: $\bar{S}_0 = V$, $\bar{S}_1 = S$. For $i \in [2, k-1]$ the set $\bar{S}_i$ is constructed by picking every vertex of $\bar{S}_{i-1}$ independently at random with probability $p \cdot c \log n$, for some constant c. The set $\bar{S}_k$ is empty. We denote a mixed hierarchy with $\bar{\mathcal{S}}_k^{p,q}$, where q is the parameter used to create S by Lemma 16.

We update PIVOT-DIST to handle a mixed hierarchy. We add a special phase as in AUG-PIVOT-DIST for the set $\bar{S}_1$. Since $|\bar{S}_0| = n$ and $|E_{\bar{S}_1}| = O(nq^{-1})$ we do not execute phase 3 for the vertices of $\bar{S}_0 = V$ to avoid an additional cost of $\tilde{O}(n^2q^{-1})$ time. In phase 3 the set $E(1/(p^{i+1}))$ is changed to $E(1/(qp^i))$, so that it reflects the size of $\bar{S}_1$. The updated algorithm is called MIX-PIVOT-DIST.

Next, we anaylze the running time of MIX-PIVOT-DIST.

▶ **Lemma 22.** *MIX-PIVOT-DIST($\bar{\mathcal{S}}_k^{p,q}$) has an expected running time of $\tilde{O}(n^2 + (k-2)n^2p^{-1} + mq^{-1} + mnqp^{k-2})$.*

Proof. The analysis of the first phase remains as in Lemma 8, and therefore it is $O(n^2)$. The special phase costs $\tilde{O}(mq^{-1})$ as in Lemma 17. In the second phase we compute $p_i(u)$, $d_G(u, \bar{S}_i)$ and $B_i(u)$ for every $u \in V$. It takes $\tilde{O}(m)$ time to compute $p_i(u)$ and $d_G(u, \bar{S}_i)$, for every $u \in V$. Computing $B_i(u)$ for every $u \in V$, where $i \in [2, k-2]$ costs $\tilde{O}(m \cdot p^{-1})$. Computing $B_1(u)$ for every $u \in V$ costs $\tilde{O}(m \cdot q^{-1})$ since q is the parameter used for $\bar{S}_1$. Computing $B_{k-1}(u)$, for every $u \in V$, costs $\tilde{O}(m \cdot nqp^{k-2})$, since from Lemma 6 we have $B_{k-1}(u) = \tilde{O}(nqp^{k-2})$.

Line (1) costs $\tilde{O}(\sum_{u \in V} |N(u)|) = \tilde{O}(m)$ and line (2) costs $\tilde{O}(\sum_{u \in V} |B_i(u)| \cdot |N(u)|) = \tilde{O}(m \cdot (p^{-1} + q^{-1}))$.

In the third phase we run Dijkstra's algorithm from every $s \in \bar{S}_i$ in $G_{i+1}(s)$, as beofre. The set of edges of $G_{i+1}(s)$ is $E_{S_{i+1}} \cup E(1/(qp^i)) \cup H(s)$. The set $H(s)$ is of size $O(n)$. The set $E(1/(qp^i))$ is of size $O(n/(qp^i))$. Consider now the set $E_{\bar{S}_{i+1}}$, for $0 < i < k-1$. The probability of a vertex to be in S_{i+1} is $\tilde{O}(qp^i)$. Applying Lemma 3 we get that the expected size of the set $E_{S_{i+1}}$ is $O(n/(qp^i))$. We get that the cost of the third phase for every $i \in [1, k-2]$ is $\tilde{O}(|\bar{S}_i| \cdot n/qp^i)$. Since the expected size of $\bar{S}_i$ is $\tilde{O}(nqp^{i-1})$ we get a bound of $\tilde{O}(\sum_{i=1}^{k-2}(n \cdot qp^{i-1} \cdot n/qp^i)$=$\tilde{O}(n^2 + (k-2)n^2p^{-1})$.

When $i = k-1$ we cannot apply Lemma 3 to bound the size of $E_{\bar{S}_k}$ since $\bar{S}_k = \emptyset$, thus, we bound the cost of running Dijkstra's algorithm from every $s \in \bar{S}_{k-1}$ in $G_k(s)$ with $\tilde{O}(|\bar{S}_{k-1}|m) = \tilde{O}(n \cdot qp^{k-2}m)$. We get a running time of $O(n^2 + (k-2)n^2p^{-1} + mnqp^{k-2})$. ◀

MIX-APASP is algorithm APASP in which MIX-PIVOT-DIST is called instead of PIVOT-DIST. We do not execute Line (2) of APASP when $i = 0$, from the same reason we have not executed phase 3 for the vertices of $\bar{S}_0 = V$ in MIX-PIVOT-DIST. The analysis of the running time of MIX-APASP is relatively straightforward and stems from Lemma 22 and Lemma 14.

▶ **Corollary 23.** *MIX-APASP$(G, \hat{\mathcal{S}}_k^{p,q})$ runs in $\tilde{O}(n^2 + kn^2p^{-1} + mq^{-1} + mnqp^{k-2})$ expected time.*

We prove a variant of Lemma 11 for MIX-APASP $(G, \bar{\mathcal{S}}_k^{p,q})$. The difference with respect to Lemma 11 is that the Lemma is proved only for vertices $u, v \in V$ that are not covered by $B_{\bar{S}_1}(u)$ and $B_{\bar{S}_1}(v)$. This change is required since we are not computing shortest paths for the vertices of $\bar{S}_0$ in phase 3 of MIX-PIVOT-DIST.

▶ **Lemma 24.** *MIX-APASP$(G, \bar{\mathcal{S}}_k^{p,q})$ returns a matrix M that satisfies: $d_G(u,v) \leq M[u,v] \leq 2d_G(u,v) + \frac{k-2}{k} \cdot d_G(u,v)$, for every $u, v \in V$ that are not covered by $B_{\bar{S}_1}(u)$ and $B_{\bar{S}_1}(v)$.*

Proof. Let $u, v \in V$. Let $P(u,v)$ be a shortest path between u and v. Let $i \in [0, k-1]$ be the largest index such that $d_G(u, \bar{S}_i) + d_G(v, \bar{S}_i) \leq d_G(u,v)$. Such an index must exist since $\bar{S}_0 = V$, which implies that $d_G(u, \bar{S}_0) + d_G(v, \bar{S}_0) = 0 \leq d_G(u,v)$. Since u and v are not covered by $B_{\bar{S}_1}(u)$ and $B_{\bar{S}_1}(v)$, it follows from Lemma 20 that $d_G(u, \bar{S}_1) + d_G(v, \bar{S}_1) \leq d_G(u,v)$, as well and $i \geq 1$. This implies that the analysis of the case that $P(u,v)$ is in $G_{i+1}(p_i(u))$ remains the same and is not affected by the fact that in MIX-PIVOT-DIST no data is computed in phase 3 for the set $\bar{S}_0$.

The analysis of the case that the path $P(u,v)$ is not in $G_{i+1}(p_i(u))$ is not using the third phase data of $\bar{S}_0$ and remains the same. Thus, for the rest of the proof we can assume that $0 < i < k-1$, $P(u,a)$ is in $G(V, E_{\bar{S}_{i+1}})$, $P(v,b)$ is in $G(V, E_{\bar{S}_{i+1}})$ and $(a,b) \notin E_a(1/qp^i) \cup E_b(1/qp^i) \cup E_{\bar{S}_{i+1}}$.

The proof of Claim 12 remains the same since we only use data for pivots from $\bar{S}_{j-1}$, where $j > i+1$ and $i > 0$. Thus, not computing data for $\bar{S}_0$ in phase 3 of MIX-PIVOT-DIST has no affect.

The proof of Claim 13 remains also the same from the following reason. Recall that we focus on the pair of vertices u and a and the pair of vertices v and b. In the proof we use r which is defined to be the largest index such that $a \notin B_{\bar{S}_r}(u)$ and $u \notin B_{\bar{S}_r}(a)$ and r' which is defined to be the largest index such that $b \notin B_{\bar{S}_{r'}}(v)$ and $v \notin B_{\bar{S}_{r'}}(b)$. Recall also that since $\bar{S}_0 = V$ we have $B_{\bar{S}_0}(x) = \{x\}$, for every $x \in V$ and thus $a \notin B_{\bar{S}_0}(u)$ and $u \notin B_{\bar{S}_0}(a)$ and similarly $b \notin B_{\bar{S}_0}(v)$ and $v \notin B_{\bar{S}_0}(b)$.

From the definition of r it follows that either $a \in B_{\bar{S}_{r+1}}(u)$ or $u \in B_{\bar{S}_{r+1}}(a)$. If $a \in B_{\bar{S}_{r+1}}(u)$ then $d_G(u,a) < d_G(u, \bar{S}_{r+1})$ and it follows from Lemma 2 that $P(u,a) \in E_{\bar{S}_{r+1}}$. Similarly, if $u \in B_{\bar{S}_{r+1}}(a)$ then $d_G(u,a) < d_G(a, \bar{S}_{r+1})$ and it follows from Lemma 2 that $P(u,a) \in E_{\bar{S}_{r+1}}$.

From symmetrical arguments we get that $P(v,b) \in E_{\bar{S}_{r'+1}}$. Since $E_{\bar{S}_j} \subseteq E_{\bar{S}_{j+1}}$, for every $j \in [0, k-1]$, we have and $P(u,a) \subseteq E_{\bar{S}_{q+1}}$, and $P(v,b) \subseteq E_{\bar{S}_{q'+1}}$ for every $q \geq r$ and $q' \geq r'$, respectively.

Because we do not compute in the third phase data for the set $\bar{S}_0$ we deal separately with the special case that $r = r' = 0$. In such a case we have $a \in B_{\bar{S}_1}(u)$ or $u \in B_{\bar{S}_1}(a)$ and also $b \in B_{\bar{S}_1}(v)$ or $v \in B_{\bar{S}_1}(b)$[5]. Now since u and v are not covered by $B_{\bar{S}_1}(u)$ and $B_{\bar{S}_1}(v)$ it must

[5] If there are only two edges of the path the same proof holds with $b = v$

Algorithm 5 MAIN-APASP(G, k').

```
create a set $S$ with prob. $n^{-\beta}$ using Lemma 16;
create an augmented hierarchy $\hat{S}_{\log n}^{n^{-\beta/\log n}}$ with $S$ as the augmenting set;
create a mixed hierarchy $\bar{\mathcal{H}}_{k'}^{n^{-\gamma/(k'-2)}, n^{-\beta}}$ with $\bar{H}_1 = S$;
$M_1 \leftarrow$ AUG-APASP$(G, \hat{\mathcal{S}}_{\log n}^{n^{-\beta/\log n}})$;
$M_2 \leftarrow$ MIX-APASP$(G, \bar{\mathcal{H}}_{k'}^{n^{-\gamma/(k'-2)}, n^{-\beta}})$;
$M \leftarrow \min\{M_1, M_2\}$;
return $M$;
```

be that either $a \notin B_{\bar{S}_1}(u)$ or $b \notin B_{\bar{S}_1}(v)$. Assume, wlog, that $a \notin B_{\bar{S}_1}(u)$. From the definition of r it must be that $u \in B_{\bar{S}_1}(a)$, as otherwise $r > 0$. Because $u \in B_{\bar{S}_1}(a)$ we update $\mathbf{w}(p_1(u), b)$ in the special phase of MIX-PIVOT-DIST so that it is at most $d_G(p_1(u), u) + d_G(u, a) + \mathbf{w}(a, b)$.

Since $r' = 0$ we have that $P(v, b) \in E_{\bar{S}_2}$. Thus, at the third phase of MIX-PIVOT-DIST we update $M[p_1(u), v]$, due to the bound of $\mathbf{w}(p_1(u), b)$, so that it is at most $d_G(p_1(u), u) + d_G(u, v)$. Now in line (1) of MIX-APSP $M[u, v]$ is updated so that it is at most $2d_G(p_1(u), u) + d_G(u, v)$. Since $a \notin B_{\bar{S}_1}(u)$ we have $d_G(p_1(u), u) \leq d_G(u, a)$ and we get that $M[u, v] \leq 2d_G(u, a) + d_G(u, v)$. This is exactly bound (i) in Claim 13 for $k = 2$ and thus holds for every $k \geq 2$. Thus, we can assume that $r > 0$ and not computing data for $\bar{S}_0$ in phase 3 of MIX-PIVOT-DIST has no affect on the proof of Claim 13 since $f \geq r$. ◀

4.3 Combining augmented hierarchies and mixed hierarchies

We now describe our main APASP algorithm MAIN-APASP. The input is a weighted undirected graph G and an integer k'. We use two hierarchies. An augmented hierarchy $\hat{\mathcal{S}}_k^p$, $k = \log n$ levels, set $\hat{S}_0$ is V and the probability p is $\tilde{O}(n^{-\beta/\log n})$. The hierarchy is augmented with a set S computed by Lemma 16 with parameter $n^{-\beta}$. Notice that $|\hat{S}_{\log n}| = \tilde{O}(n^{1-\beta})$, and for every $v \in V$ we have $|B_{\hat{S}_{\log n}}(v)| = |C_{\hat{S}_{\log n}}(v)| = O(n^\beta)$. The second hierarchy is a mix hierarchy $\bar{\mathcal{H}}_{k'}^{p,q}$ with k' levels, $\bar{H}_1 = \hat{S}_{\log n}$ and $\bar{H}_{k'} = \emptyset$. Since $\hat{S}_{\log n}$ is formed using Lemma 16 with parameter $n^{-\beta}$ we have $q = n^{-\beta}$. The probability p is $\tilde{O}(n^{-\gamma/(k'-2)})$.

Next, we run AUG-APASP$(G, \hat{\mathcal{S}}_{\log n}^{n^{-\beta/\log n}})$. Then we run MIX-APASP$(G, \bar{\mathcal{H}}_{k'}^{n^{-\gamma/(k'-2)}, n^{-\beta}})$. MAIN-APASP is presented in Algorithm 5. We now turn to analyse the running time.

▶ **Lemma 25.** *MAIN-APASP(G, k') runs in of $\tilde{O}(n^2 + m^{\frac{2}{k'}} n^{2-\frac{3}{k'}})$ expected time.*

Proof. It follows from Lemma 16 that S can be computed in $\tilde{O}(mn^\beta)$ expected running time. It follows from Corollary 18 that the expected running time of AUG-APASP$(G, \hat{\mathcal{S}}_{\log n}^{n^{-\beta/\log n}})$ is $\tilde{O}(n^2 + n^2 \cdot n^{\beta/\log n} \log n + mn^\beta \log n) = \tilde{O}(n^2 + mn^\beta)$. It follows from Corollary 23 that the expected running time of MIX-APASP$(G, \bar{\mathcal{H}}_{k'}^{n^{-\gamma/(k'-2)}, n^{-\beta}})$ is $\tilde{O}(n^2 + (k'-2)n^2 n^{\gamma/(k'-2)} + mn^{1-\beta-\gamma})$. We first express n^β as a function of n^γ using the equation $n^2 n^{\gamma/(k'-2)} = mn^{1-\beta-\gamma}$ which balances the two main terms in the running time of MIX-APASP.

$$n^2 n^{\gamma/(k'-2)} = mn^{1-\beta-\gamma} \longrightarrow n^\beta = \frac{mn^{1-\gamma}}{n^2 n^{\gamma/(k'-2)}} \longrightarrow n^\beta = \frac{m}{n^{1+\gamma(1+1/(k'-2))}}$$

Next, we substitute n^β as a function of n^γ in the running time of AUG-APASP.

$$\tilde{O}(n^2 + mn^\beta) \xrightarrow[n^\beta = \frac{m}{n^{1+\gamma(1+1/(k'-2))}}]{} \tilde{O}(n^2 + \frac{m^2}{n^{1+\gamma(1+1/(k'-2))}})$$

We now compute the value of n^γ using the equation $n^{2+\gamma/(k'-2)} = \frac{m^2}{n^{1+\gamma(1+1/(k'-2))}}$ which balances the two main terms in the running times of MIX-APASP and AUG-APASP.

$$n^{\gamma(1+2/(k'-2))} = \frac{m^2}{n^3} \longrightarrow n^\gamma = \left(\frac{m^2}{n^3}\right)^{\frac{k'-2}{k'}}$$

To obtain the running time of APASP we substitute n^γ in $n^{2+\gamma/(k'-2)}$:

$$n^{2+\gamma/(k'-2)} \xrightarrow[n^\gamma=\left(\frac{m^2}{n^3}\right)^{\frac{k'-2}{k'}}]{} n^2\left(\frac{m^2}{n^3}\right)^{\frac{1}{k'}} = n^{2-\frac{3}{k'}}m^{\frac{2}{k'}}$$

Thus, we get that the running time of APASP is $\tilde{O}(n^2 + m^{\frac{2}{k'}}n^{2-\frac{3}{k'}})$. ◀

We now bound the approximation of MAIN-APASP.

▶ **Lemma 26.** *Algorithm MAIN-APASP(G, k') returns a matrix M that satisfies: $d_G(u,v) \leq M[u,v] \leq 2d_G(u,v) + \frac{k'-2}{k'} \cdot d_G(u,v)$, where $k' \geq 2$.*

Proof. Let $u, v \in V$. Recall that $\bar{H}_1 = \hat{S}_{\log n}$. If u and v are covered by $B_{\bar{H}_1}(u)$ and $B_{\bar{H}_1}(v)$ then it follows from Lemma 21 that Algorithm AUG-APASP returns a matrix M_1 that satisfies: $d_G(u,v) \leq M_1[u,v] \leq 2d_G(u,v)$. If u and v are not covered by $B_{\bar{H}_1}(u)$ and $B_{\bar{H}_1}(v)$ then it follows from Lemma 24 that Algorithm MIX-APASP returns a matrix M that satisfies:

$$d_G(u,v) \leq M_2[u,v] \leq 2d_G(u,v) + \frac{k'-2}{k'} \cdot d_G(u,v).$$

Since $M = \min\{M_1, M_2\}$ the claim follows. ◀

Proof of Theorem 1

Proof. The proof follows from Lemma 25 and Lemma 26 . ◀

References

1 D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999.

2 Maor Akav and Liam Roditty. An almost 2-approximation for all-pairs of shortest paths in subquadratic time. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1–11. SIAM, 2020.

3 S. Baswana, V. Goyal, and S. Sen. All-pairs nearly 2-approximate shortest paths in $o(n^2\text{poly}\log n)$ time. *Theor. Comput. Sci.*, 410(1):84–93, 2009.

4 S. Baswana and T. Kavitha. Faster algorithms for all-pairs approximate shortest paths in undirected graphs. *SIAM J. Comput.*, 39(7):2865–2896, 2010.

5 Surender Baswana, Akshay Gaur, Sandeep Sen, and Jayant Upadhyay. Distance oracles for unweighted graphs: Breaking the quadratic barrier with constant additive error. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, pages 609–621, 2008.

6 P. Berman and S. P. Kasiviswanathan. Faster approximation of distances in graphs. In *Proc. WADS*, pages 541–552, 2007.

7 Shiri Chechik. Approximate distance oracles with constant query time. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 654–663, 2014.

8 Shiri Chechik. Approximate distance oracles with improved bounds. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 1–10, 2015.

9 E. Cohen and U. Zwick. All-pairs small-stretch paths. *J. Algorithms*, 38(2):335–353, 2001.

10 D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM J. Comput.*, 29(5):1740–1759, 2000.

11 M. Elkin. Computing almost shortest paths. *ACM Transactions on Algorithms*, 1(2):283–323, 2005.

12 Michael Elkin, Yuval Gitlitz, and Ofer Neiman. Almost shortest paths and PRAM distance oracles in weighted graphs. *CoRR*, abs/1907.11422, 2019.

13 Michael Elkin, Ofer Neiman, and Christian Wulff-Nilsen. Space-efficient path-reporting approximate distance oracles. *Theor. Comput. Sci.*, 651:1–10, 2016.

14 Michael Elkin and Seth Pettie. A linear-size logarithmic stretch path-reporting distance oracle for general graphs. *ACM Trans. Algorithms*, 12(4):50:1–50:31, 2016.

15 Telikepalli Kavitha. Faster algorithms for all-pairs small stretch distances in weighted graphs. *Algorithmica*, 63(1-2):224–245, 2012.

16 Mathias Bæk Tejs Knudsen. Additive spanners and distance oracles in quadratic time. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 64:1–64:12, 2017.

17 François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014.

18 Manor Mendel and Assaf Naor. Ramsey partitions and proximity data structures. In *FOCS*, pages 109–118, 2006.

19 S. Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theor. Comput. Sci.*, 312(1):47–74, 2004.

20 Seth Pettie and Vijaya Ramachandran. A shortest path algorithm for real-weighted undirected graphs. *SIAM J. Comput.*, 34(6):1398–1431, 2005.

21 R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *JCSS*, 51:400–403, 1995.

22 A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proc. FOCS*, pages 605–614, 1999.

23 Christian Sommer. All-Pairs Approximate Shortest Paths and Distance Oracle Preprocessing. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 55:1–55:13, 2016.

24 A. Stothers. On the complexity of matrix multiplication. *Ph.D. Thesis, U. Edinburgh*, 2010.

25 Mikkel Thorup and Uri Zwick. Compact routing schemes. In *SPAA*, pages 1–10, 2001.

26 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 2005.

27 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 664–673, 2014.

28 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 887–898. ACM, 2012.

29 Christian Wulff-Nilsen. Approximate distance oracles with improved preprocessing time. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 202–208, 2012.

30 U. Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.