Fast 2-Approximate All-Pairs Shortest Paths

Michal Dory* Sebastian Forster† Yael Kirkpatrick‡ Yasamin Nazari§
Virginia Vassilevska Williams ¶ Tijn de Vos†

Abstract

In this paper, we revisit the classic approximate All-Pairs Shortest Paths (APSP) problem in undirected graphs. For unweighted graphs, we provide an algorithm for 2-approximate APSP in $\tilde{O}(n^{2.5-r} + n^{\omega(r)})$ time, for any $r \in [0, 1]$. This is $O(n^{2.032})$ time, using known bounds for rectangular matrix multiplication $n^{\omega(r)}$ [Le Gall, Urrutia, SODA 2018]. Our result improves on the $\tilde{O}(n^{2.25})$ bound of [Roditty, STOC 2023], and on the $\tilde{O}(m\sqrt{n} + n^2)$ bound of [Baswana, Kavitha, SICOMP 2010] for graphs with $m \geq n^{1.532}$ edges.

For weighted graphs, we obtain $(2+\epsilon)$-approximate APSP in $\tilde{O}(n^{3-r} + n^{\omega(r)})$ time, for any $r \in [0, 1]$. This is $O(n^{2.214})$ time using known bounds for $\omega(r)$. It improves on the state of the art bound of $O(n^{2.25})$ by [Kavitha, Algorithmica 2012]. Our techniques further lead to improved bounds in a wide range of density for weighted graphs. In particular, for the sparse regime we construct a distance oracle in $\tilde{O}(mn^{2/3})$ time that supports 2-approximate queries in constant time. For sparse graphs, the preprocessing time of the algorithm matches conditional lower bounds [Patrascu, Roditty, Thorup, FOCS 2012; Abboud, Bringmann, Fischer, STOC 2023]. To the best of our knowledge, this is the first 2-approximate distance oracle that has subquadratic preprocessing time in sparse graphs.

We also obtain new bounds in the near additive regime for unweighted graphs. We give faster algorithms for $(1+\epsilon, k)$-approximate APSP, for $k = 2, 4, 6, 8$.

We obtain these results by incorporating fast rectangular matrix multiplications into various combinatorial algorithms that carefully balance out distance computation on layers of sparse graphs preserving certain distance information.

*University of Haifa

†Department of Computer Science, University of Salzburg. This work is supported by the Austrian Science Fund (FWF): P 32863-N. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 947702).

‡MIT, this material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No 2141064

§VU Amsterdam. This work was partially conducted while the author was a postdoc at University of Salzburg.

¶MIT, Supported by NSF Grants CCF-2129139 and CCF-2330048 and BSF Grant 2020356.

Contents

1 Introduction

The All-Pairs Shortest Paths (APSP) problem is one of the most fundamental problems in graph algorithms. In this problem, the goal is to compute the distances between all pairs of vertices in a graph. It is well-known that APSP can be solved in $O(n^3)$ time in directed weighted graphs with n vertices using the Floyd-Warshall algorithm (see [CLRS94]), or in $\tilde{O}(mn)$ time using Dijkstra's algorithm, where m is the number of edges in the graph (see also [Tho99, Pet04, PR05]). A slightly subcubic algorithm for APSP with running time $n^3/2^{\Omega(\log n)^{1/2}}$ was given by Williams [Wil14]. A natural hypothesis in graph algorithms (see [RZ11, Vas18]) is that $n^{3-o(1)}$ time is required to solve APSP in weighted graphs; this is known as the APSP hypothesis. Subcubic equivalences[1] between the APSP problem and many other problems such as finding a negative weight triangle or finding the radius of the graph [Vas15, WW18, AGW23] significantly strengthened the belief in the APSP hypothesis.

All the above algorithms solve the problem in weighted, directed graphs. If the graphs are unweighted and undirected, APSP can be solved faster, in $\tilde{O}(n^\omega)$ time, using fast matrix multiplication [Sei95, GM97], where $\omega < 2.372$ is the exponent of square matrix multiplication [CW87, Wil12, Gal14, AW21, DWZ23]. For weighted, directed graphs with bounded weights, Zwick showed an algorithm that takes $O(n^{2.529})$ time [Zwi02, GU18][2]. In $\tilde{O}((n^\omega/\epsilon)\cdot \log W)$ time it is also possible to obtain a $(1+\epsilon)$-approximation for APSP in weighted directed graphs [Zwi98], where W is the maximum edge weight and the weights are scaled such that the smallest non-zero weight is 1.

However, the above complexities can be high for large graphs, and it is desirable to have faster algorithms. While it may be difficult to get faster algorithms for exact APSP, a natural question is whether we can get faster *approximation* algorithms for APSP. We say that an algorithm gives an (α, β)-approximation for APSP if for any pair of vertices u, v it returns an estimate $\delta(u,v)$ of the distance between u and v such that $d(u,v) \le \delta(u,v) \le \alpha \cdot d(u,v) + \beta$, where $d(u,v)$ is the distance between u and v. If $\beta = 0$, we get a purely multiplicative approximation that we refer to as α-approximation, α is sometimes called the *stretch* of the algorithm. If $\alpha = 1$, we get a purely additive approximation, and call it a $+\beta$-approximation.

Dor, Halperin, and Zwick [DHZ00] showed that obtaining a $(2-\epsilon)$-approximation for APSP is at least as hard as Boolean matrix multiplication. This implies that we cannot get algorithms with running time below $O(n^\omega)$ for approximate APSP with approximation ratios below 2, as it would lead to algorithms for matrix multiplication with the same running time. Their reduction holds even in the case that the graphs are unweighted and undirected. In the directed case the same result holds for *any* approximation. This makes the special case of a 2-approximation in undirected graphs a very interesting special case, as this is the first approximation ratio where we can beat the $O(n^\omega)$ bound. Since in directed graphs any approximation for APSP requires $\Omega(n^\omega)$ time, we will focus from now on *undirected* graphs.

2-Approximate APSP. While 2-approximation algorithms for APSP have been extensively studied [ACIM99, DHZ00, CZ01, BK10, Kav12, DKRW+22, Dür23, Rod23] (see Table 1 for a summary), it is still unclear what the best running time is that can be obtained for this problem. This question is also open even in the simpler case of unweighted and undirected graphs. The study of 2-approximate APSP was treated by the seminal work of Aingworth, Chekuri, Indyk, and Motwani [ACIM99] that showed an additive +2-approximation algorithm for APSP that takes $\tilde{O}(n^{2.5})$ time. Their algorithm works in undirected unweighted graphs. While the running time of the algorithm is above $O(n^\omega)$, it is a simple *combinatorial* algorithm, where we use the widespread informal terminology that an algorithm is combinatorial if it does not use fast matrix multiplication techniques. The running time was later improved by Dor, Halperin, and Zwick that showed a +2-approximation algorithm in $\tilde{O}(\min\{n^{3/2}m^{1/2}, n^{7/3}\})$ time [DHZ00]. This algorithm is still the fastest known combinatorial algorithm for a +2-approximation. Very recent results show that using fast matrix multiplication techniques, one can get faster algorithms for a +2-approximation, leading to an $O(n^{2.260})$ running time [DKRW+22, Dür23]. All the above mentioned algorithms give an additive +2-approximation in unweighted undirected graphs, which also implies a multiplicative 2-approximation. This holds since for pairs u, v where $d(u,v) = 1$ we can learn the exact distance in $O(m)$ time, and once $d(u,v) \ge 2$, an additive +2-approximation is also a multiplicative 2-approximation. However, if our goal is to obtain a multiplicative approximation, it may be possible to get a faster algorithm.

Algorithms for multiplicative 2-approximate APSP are studied in [CZ01, BK10, Kav12, Rod23]. In particular, Cohen and Zwick [CZ01] showed that a 2-approximation for APSP can be computed in $\tilde{O}(n^{3/2}m^{1/2})$ time also in weighted graphs. A faster algorithm with running time of $\tilde{O}(m\sqrt{n} + n^2)$ was shown by Baswana and Kavitha [BK10], this algorithm also works in weighted graphs. While in dense graphs the running time is $\tilde{O}(n^{2.5})$, this algorithm is more efficient in sparse graphs. In

[1] Two problems are subcubically equivalent if one problem can be solved in time $O(n^{3-\epsilon})$ for some $\epsilon > 0$ if and only if the other can be solved in time $O(n^{3-\delta})$ for some $\delta > 0$.

[2] The running time of this algorithm depends on the exponent of rectangular matrix multiplication, and becomes $O(n^{2.529})$ with the rectangular matrix multiplication algorithm from [GU18].

particular, for graphs with $O(n^{3/2})$ edges, the running time becomes $\tilde{O}(n^2)$. A faster algorithm for denser graphs was shown by Kavitha that showed a $(2+\epsilon)$-approximation for weighted APSP in $\tilde{O}(n^{2.25})$ time [Kav12].[3] This algorithm exploits fast matrix multiplication techniques. Very recently, Roditty showed a combinatorial algorithm for 2-approximate APSP in unweighted undirected graphs in $\tilde{O}(n^{2.25})$ time [Rod23]. To conclude, currently the fastest 2-approximation algorithms take $\tilde{O}(\min\{m\sqrt{n}+n^2, n^{2.25}\})$ time, and it is still unclear what is the best running time that can be obtained for the problem. In particular, the following question is still open.

Question 1. *Can we get a 2-approximation for APSP in $\tilde{O}(n^2)$ time?*

It is worth mentioning that a slightly higher approximation of $(2,1)$ can be obtained in $\tilde{O}(n^2)$ time in unweighted undirected graphs [BK10, BK07, Som16, Knu17]. In weighted undirected graphs, a multiplicative 3-approximation can be obtained in $\tilde{O}(n^2)$ time [CZ01, BK10]. Furthermore, [BK10] also gives a $(2, W_{u,v})$-approximation for weighted undirected graphs in $\tilde{O}(n^2)$ time, where $W_{u,v}$ is the maximum weight of an edge in the shortest path between u and v. In other words, for any pair of vertices u, v the algorithm returns an estimate $\delta(u,v) \leq 2d(u,v) + W_{u,v}$.

Distance Oracles. Note that $\Omega(n^2)$ is a natural barrier for APSP, as just writing the output of the problem takes $\Omega(n^2)$ time. However, $\Omega(n^2)$ time or space may be too expensive for large graphs, and if we do not need to write the output explicitly, we may be able to obtain algorithms with *subquadratic* time or space. This has motivated the study of *distance oracles*, which are compact data structures that allow fast query of (possibly approximate) distances between any pair of vertices. The study of near-optimal approximate distance oracles was initiated by the seminal work of Thorup and Zwick [TZ05] that showed that for any integer $k \geq 2$, a distance oracle of size $O(kn^{1+1/k})$ can be constructed in $O(kmn^{1/k})$ time, such that for any pair of vertices we can obtain a $(2k-1)$-approximation of the distance between them in query time $O(k)$. As an important special case, it gives 3-approximate distance oracles of size $O(n^{3/2})$ in $O(m\sqrt{n})$ time. The construction is for weighted undirected graphs. Note that the distance oracle uses *subquadratic* space, and the construction time is *subquadratic* for sparse graphs. As shown later these distance oracles can be built also in $\tilde{O}(\min\{kmn^{1/k}, n^2\})$ time [BK10]. The construction time was further improved in [Wul12], where the query time and size were further improved in [Wul13, Che14, Che15].

Distance oracles of size $\tilde{O}(n^{5/3})$ that provide a $(2,1)$-approximation in unweighted undirected graphs are studied in [BGS09, PR14, Som16, Knu17]. This tradeoff between stretch and space is optimal assuming the hardness of set intersection [PR14, PRT12]. The fastest of them run in $\min\{\tilde{O}(n^2, mn^{2/3})\}$ time [BGS09, Som16, Knu17].[4]

Recently, slightly subquadratic algorithms with nearly 2-approximations were given for undirected unweighted graphs. In particular, a distance oracle with stretch $(2(1+\epsilon), 5)$ and slightly subquadratic running time is given by Akav and Roditty [AR20]. In a follow-up work, Chechik and Zhang constructed a $(2,3)$-approximate distance oracle of size $\tilde{O}(n^{5/3})$ in $\tilde{O}(m+n^{1.987})$ time. They also study other trade-offs between the stretch and running time, and in particular show that $(2+\epsilon, c(\epsilon))$-approximate distance oracle of size $\tilde{O}(n^{5/3})$ can be constructed in $\tilde{O}(m+n^{5/3+\epsilon})$ time, where $c(\epsilon)$ is a constant depending exponentially on $1/\epsilon$. The preprocessing time of this distance oracle nearly matches a recent conditional lower bound by Abboud, Bringmann, and Fischer [ABF23], who showed that $m^{5/3-o(1)}$ time is required for a $(2+o(1))$-approximation, conditional on the 3-SUM conjecture. While there are subquadratic constructions of distance oracles with nearly 2-approximations, all existing algorithms for 2-approximations take at least $\Omega(n^2)$ time [DHZ00, CZ01, BK10, Kav12], which raises the following question.

Question 2. *Can we construct a 2-approximate distance oracle in subquadratic time?*

1.1 Our Results

Throughout we write $n^{\omega(r)}$ for the time for multiplying an $n \times n^r$ matrix by an $n^r \times n$ matrix (see Section 2). Rectangular matrix multiplication is an active research field, with the bounds on $\omega(r)$ being improved in recent years [Gal23, GU18, Gal12]. We provide how the recent work by Vassilevska Williams, Xu, Xu, and Zhou [WXXZ23] affects our running times in Appendix A. For the rest of this paper, we use [GU18], the last published paper on the topic, for the sake of replicability. We balance the terms using [Bra].

Unweighted 2-Approximate APSP. Our first contribution is a faster algorithm for 2-approximate APSP in unweighted undirected graphs.[5]

[3] In the introduction we assume that $1/\epsilon = n^{o(1)}$ for simplicity of presentation. Similarly, when we discuss weighted graphs, we assume polynomial weights.

[4] The running time of $\tilde{O}(mn^{2/3})$ is implicit in [BGS09].

[5] All our randomized algorithms are always correct; the randomness only affects the running time.

THEOREM 1.1. *There exists a randomized algorithm that, given an unweighted, undirected graph $G = (V, E)$, computes 2-approximate APSP. With high probability, the algorithm takes $\tilde{O}(n^{2.5-r} + n^{\omega(r)})$ time, for any $r \in [0, 1]$. Using known upper bounds for rectangular matrix multiplication, this is $O(n^{2.032})$ time.*

Our running time improves over the previous best running time of $\tilde{O}(\min\{m\sqrt{n} + n^2, n^{2.25}\})$ [BK10, Kav12, Rod23] as long as $m \geq n^{1.532}$, and gets closer to an $O(n^2)$ running time. In particular, if we have a rectangular matrix multiplication bound of $\omega(0.5) = 2$, then we obtain a $\tilde{O}(n^2)$ time algorithm. Currently, we know $\omega(0.5) < 2.043$ [WXXZ23] – a very recent improvement on [GU18]: $\omega(0.5) < 2.044183$.

While the fastest version of our algorithm exploits fast rectangular matrix multiplication, our approach also leads to an alternative simple combinatorial 2-approximation algorithm in $\tilde{O}(n^{2.25})$ time, matching the very recent result by Roditty [Rod23]. See Section 3.2 for the details.

Algorithms for Unweighted APSP		
Reference	Time	Approx.
[ACIM99]	$n^{2.5}$	+2
[DHZ00]	$n^{7/3}$	+2
[DKRW+22]	$n^{2.287}$	+2
[Dür23]	$n^{2.260}$	+2
[Rod23]	$n^{2.25}$	2
This work	$n^{2.032}$	2

Algorithms for Weighted APSP		
Reference	Time	Approx.
[CZ01]	$n^{3/2}m^{1/2}$	2
[BK10]	$m\sqrt{n} + n^2$	2
[Kav12]	$n^{2.25}$	$2+\epsilon$
This work	$n^{2.214}$	$2+\epsilon$
This work	$mn^{2/3}$	2

Table 1: Algorithms for 2-Approximate APSP. The last result in the right table is a 2-approximate distance oracle.

Weighted 2-Approximate APSP. For weighted undirected graphs we show the following.

THEOREM 1.2. *There exists a randomized algorithm that, given an undirected graph $G = (V, E)$ with non-negative integer weights bounded by W, computes $(2+\epsilon)$-approximate APSP. With high probability the algorithm takes $O(n^{3-r} + n^{\omega(r)}(1/\epsilon)^{O(1)} \log W)$ time, for any $r \in [0, 1]$. Using known upper bounds for rectangular matrix multiplication, this is $O(n^{2.214}(1/\epsilon)^{O(1)} \log W)$ time.*

We remark that with standard scaling techniques the result generalizes to non-integer weights. Hence we state all our weighted results only for integers.

To the best of our knowledge, this is the first improvement for weighted $(2+\epsilon)$-approximate APSP since the work of Kavitha [Kav12] that showed an algorithm with $\tilde{O}(n^{2.25})$ running time, that also exploited fast matrix multiplication techniques. We note that our algorithm and the algorithm of [Kav12] give a $(2+\epsilon)$-approximation, and currently the fastest algorithms that give a 2-approximation for weighted APSP take $\tilde{O}(\min\{n^\omega, m\sqrt{n} + n^2\})$ time [Zwi02, BK10]. The fastest combinatorial algorithm for the problem is the $\tilde{O}(m\sqrt{n} + n^2)$ time algorithm of Baswana and Kavitha [BK10]. This algorithm takes $\tilde{O}(n^{2.5})$ time in dense graphs, but it is faster for sparser graphs.

Our approach can also be combined with the approach of [BK10] to obtain faster algorithms for sparser graphs, giving the following. See Table 2 for some specific choices for the value of m.

THEOREM 1.3. *There exists a randomized algorithm that, given an undirected graph $G = (V, E)$ with non-negative, integer weights bounded by W and a parameter $r \in [0, 1]$, computes $(2+\epsilon)$-approximate APSP. With high probability, the algorithm runs in $\tilde{O}(mn^{1-r} + n^{\omega(r)}(1/\epsilon)^{O(1)} \log W)$ time.*

In particular, our algorithm is faster than [BK10] for $m \geq n^{\omega(0.5)-0.5}$. With the current value of $\omega(0.5)$ our algorithm is faster for $m \geq n^{1.545}$ [GU18].

Weighted 2-Approximate Distance Oracle. The results mentioned above are fast for graphs that are relatively dense. For sparser graphs we show a simple combinatorial algorithm that gives the following.

THEOREM 1.4. *There exists a combinatorial algorithm that, given a weighted graph $G = (V, E)$, constructs a distance oracle that answers 2-approximate distance queries in constant time, and uses $\tilde{O}(mn^{2/3})$ space with preprocessing time $\tilde{O}(mn^{2/3})$.*

For sparse graphs ($m = o(n^{4/3})$), we have *subquadratic* running time and space. To the best of our knowledge this is the first 2-approximate distance oracle with subquadratic construction time for sparse graphs. Moreover, when $m = O(n)$ our bounds match the conditional lower bound of $\tilde{\Omega}(m^{5/3})$ preprocessing time for 2-approximations, conditional on the set intersection conjecture [PRT12] or the 3-SUM conjecture [ABF23], where [PRT12] shows the stronger $\tilde{\Omega}(m^{5/3})$ *space* lower bound. Moreover, for any stretch strictly below 2, there is a $\tilde{\Omega}(n^2)$ lower bound (conditional on the set intersection conjecture) [PR14]. In [PR14] they also show a polynomial time construction of a 2-approximate distance oracle for weighted graphs of size $\tilde{O}(n^{4/3}m^{1/3})$, which is subquadratic space for sparse graphs. The proof of Theorem 1.4 can be found in Section 4.3.

We can also obtain a $(2, W_{u,v})$-approximate distance oracle with a better construction time than our 2-approximate construction. In particular, we can obtain a *subquadratic* preprocessing time of $\tilde{O}(nm^{2/3})$ when $m \leq n^{3/2}$, see Appendix B for full details. This result is also implied by an algorithm of Baswana, Goyal, and Sen [BGS09]. They focus on $(2, 1)$-distance oracles for unweighted graphs using the same algorithm. We observe that 1) the same algorithm can lead to subquadratic preprocessing time (in sparse graphs), and 2) this leads to a $(2, W_{u,v})$-approximation for weighted graphs. We make these observations explicit for a wider range of parameter settings and for completeness give an analysis in Appendix B.

m	This work	Previous results
$n^{1.0}$	$n^{1.667}$	n^2 [BK10]
$n^{1.1}$	$n^{1.767}$	n^2 [BK10]
$n^{1.2}$	$n^{1.867}$	n^2 [BK10]
$n^{1.3}$	$n^{1.967}$	n^2 [BK10]
$n^{1.4}$	$n^{2.009}$	n^2 [BK10]
$n^{1.5}$	$n^{2.032}$	n^2 [BK10]

m	This work	Previous results
$n^{1.6}$	$n^{2.062}$	$n^{2.1}$ [BK10]
$n^{1.7}$	$n^{2.097}$	$n^{2.2}$ [BK10]
$n^{1.8}$	$n^{2.134}$	$n^{2.25}$ [Kav12]
$n^{1.9}$	$n^{2.173}$	$n^{2.25}$ [Kav12]
$n^{2.0}$	$n^{2.214}$	$n^{2.25}$ [Kav12]

Table 2: Comparison of the novel running time bounds of our approximation algorithms for weighted graphs and prior work, for $1/\epsilon = n^{o(1)}$. For sparser graphs ($m \leq n^{1.3}$), we use our 2-approximate distance oracle (Theorem 1.4), and for denser graphs ($m \geq n^{1.4}$) we use our $(2+\epsilon)$-approximate APSP (Theorem 1.3). Theorem 1.4 gives the fastest running time when $m \leq n^{4/3}$, and Theorem 1.3 gives the fastest running time when $m \geq n^{1.545}$. For $n^{1.334} < m < n^{1.545}$, we do not improve on [BK10].

Near-Additive APSP. We also study algorithms that give a near-additive approximation in unweighted undirected graphs. We show the following.

Theorem 1.5. *There exists a deterministic algorithm that, given an unweighted, undirected graph $G = (V, E)$ and an even integer $k \geq 2$, computes $(1+\epsilon, k)$-approximate APSP with running time $\tilde{O}\left(n^{2+(1-r)\frac{2}{k+2}} + n^{\omega(r)}(1/\epsilon)\right)$, for any choice of $r \in [0, 1]$.*

As an important special case, we get a $(1+\epsilon, 2)$-approximation in $O(n^{2.152})$ time when $1/\epsilon = n^{o(1)}$. While the running time is higher compared to our multiplicative 2-approximation, it improves over previous additive or near-additive approximation algorithms. As mentioned above, currently the fastest algorithm for a +2-approximation takes $O(n^{2.260})$ time [Dür23], and it exploits fast matrix multiplication. The fastest combinatorial +2-approximation algorithm takes $\tilde{O}(\min\{n^{3/2}m^{1/2}, n^{7/3}\})$ time [DHZ00]. If we consider near-additive approximation algorithms, Berman and Kasiviswanathan showed a $(1+\epsilon, 2)$-approximation in $n^{2.24+o(1)}$ time based on fast matrix multiplication [BK07].

For $k \to \log n$, our running time goes to $\tilde{O}(n^2)$. In particular, when $1/\epsilon = n^{o(1)}$ we obtain a $(1+\epsilon, 4)$-approximation in $O(n^{2.119})$ time, a $(1+\epsilon, 6)$-approximation in $O(n^{2.098})$ time, and a $(1+\epsilon, 8)$-approximation in $O(n^{2.084})$ time. The state of the art for these approximations is $O(n^{2.2})$, $O(n^{2.125})$, and $O(n^{2.091})$ [DHZ00] respectively, using the purely additive algorithm by Dor, Halperin, and Zwick [DHZ00] that gives a deterministic $+k$-approximation for APSP in $\tilde{O}(\min\{n^{2-\frac{2}{k+2}}m^{\frac{2}{k+2}}, n^{2+\frac{2}{3k-2}}\})$ time, for even $k > 2$. See also Table 3 in Section 3.3 for comparison of our results and prior work. We remark that the algorithm of [DHZ00] becomes faster than ours once the additive term is at least 10. Our work shows that for smaller additive terms, a $(1+\epsilon, \beta)$-approximation can be obtained faster than a purely additive $+\beta$-approximation. To the best of our knowledge, previously such results were known only for the special case of $\beta = 2$ [BK07].

We remark that except [BK07] previous algorithms for near-additive approximations [Coh00, Elk05, EGN22] had larger additive terms compared to the ones we study here. In particular, Elkin, Gitlitz and Neiman [EGN22] showed an algorithm

that computes a $(1+\epsilon,\beta)$-approximation in $\tilde{O}(mn^{1/k}+n^{2+1/(2^k-1)})$ time, where $\beta = O(k/\epsilon)^{O(k)}$. While β is a constant when ϵ and k are constants, it is larger compared to the constants we consider here.

Independent work Independently, Saha and Ye [SY23] achieved similar results. They also obtain 2-approximate APSP for unweighted graphs in $O(n^{2.032})$ time (Theorem 1.1), moreover they give a deterministic algorithm for this problem in $O(n^{2.073})$ time. In the (near) additive regime (Theorem 1.5), they both give faster running times and do not incur the multiplicative $(1+\epsilon)$ error. They include additional results for additive approximations in weighted graphs, but do not have our results for 2-approximate APSP on weighted graphs (Theorem 1.3 and Theorem 1.4).

1.2 Technical Overview

Unweighted 2-Approximate APSP We start by describing our 2-approximation algorithm for APSP in unweighted undirected graphs running in $O(n^{2.032})$ time. At a high-level, we divide all the shortest paths in the graph into 2 types: the *sparse* paths and the *dense* paths. A path is sparse if the degrees of all vertices in the path are at most $\sqrt{n}$, and it is dense otherwise.

Dealing with sparse paths. In order to compute the distances between all pairs of vertices u, v where the shortest path between u and v is sparse we use the following approach. All the sparse paths are contained in a subgraph $G' = (V, E')$ of the input graph $G = (V, E)$ obtained by taking all edges adjacent to vertices of degree $\leq \sqrt{n}$. Note that this graph has only $O(n^{3/2})$ edges. To estimate the distances, we run the algorithm of Baswana and Kavitha [BK10] on the graph G', and exploit the fact that this algorithm is efficient for sparse graphs. This gives a 2-approximation for the distances in the sparse graph G' in $\tilde{O}(|E'|\sqrt{n}+n^2) = \tilde{O}(n^2)$ time.

Dealing with dense paths in $\tilde{O}(n^{2.5})$ time. We are left with dense shortest paths, i.e. paths that have at least one vertex with degree larger than $\sqrt{n}$. As a warm-up, we start by describing a simple algorithm that obtains +2-approximations for the lengths of all these paths in $\tilde{O}(n^{2.5})$ time, following the classic algorithm of Aingworth, Chekuri, Indyk and Motwani [ACIM99], and later we explain how we get a faster algorithm. We denote by H the high-degree vertices that are vertices with degree larger than $\sqrt{n}$. We start by computing a *hitting set* S, a small set of vertices such that each vertex in H has a neighbor in S. It is easy to find a hitting set of size $\tilde{O}(\sqrt{n})$, for example by adding each vertex to the set with probability $\tilde{O}(1/\sqrt{n})$. Since each high-degree vertex has degree at least $\sqrt{n}$, with high probability all high-degree vertices will have neighbors in S. The algorithm then proceeds as follows.

1. We compute distances from S to all other vertices.
2. We set $\delta(u,v) = \min_{a\in S}\{d(u,a)+d(a,v)\}$.

If the shortest path between u and v is dense, then after Step 2 the value $\delta(u,v)$ is a +2-approximation for $d(u,v)$. The reason is that there exists a high-degree vertex x in the shortest $u-v$ path, and x has a neighbor $a \in S$. Then $d(u,a)+d(a,v) \leq d(u,x)+d(x,a)+d(a,x)+d(x,v) = d(u,v)+2$. See Figure 1 for illustration. Hence by computing distances from S to all other vertices, we can get an additive +2-approximation for all dense paths.

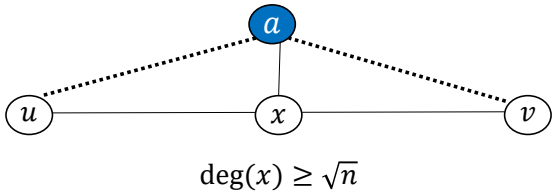

Figure 1: Illustration of the stretch analysis.

The running time is $\tilde{O}(n^{2.5})$. First, computing distances from all vertices in S takes $O(|S|m) = \tilde{O}(n^{2.5})$ time since $|S| = \tilde{O}(\sqrt{n}), m = O(n^2)$ and computing the distances from one vertex in S takes $O(m)$ time by computing a BFS tree. Second, in Step 2, for each one of the n^2 pairs of vertices we compute distances through all possible vertices in S which takes $\tilde{O}(n^{2.5})$ time.

Implementing Step 1 faster. Our goal is to implement the above approach faster. First, note that Step 1 takes $O(|S|m)$ time. To obtain a faster algorithm, our goal is to run this step on a graph where $O(|S|m) = \tilde{O}(n^2)$. To do so, we divide the dense paths to $O(\log n)$ different levels. We say that a path is 2^i-dense if the maximum degree of a vertex in the path is between $[2^i, 2^{i+1}]$. Since 2^i-dense paths have a vertex with degree at least 2^i, we can find a hitting set S_i of size $\tilde{O}(n/2^i)$ such that each 2^i-dense path will have a neighbor in S_i. Our goal is to repeat the above algorithm but on a sparser graph. Let $G_i = (V, E_i)$ be a subgraph of G that has all edges adjacent to vertices of degree at most 2^{i+1}. To deal with 2^i-dense paths we work as follows.

1. We compute distances from S_i to all other vertices in the graph G_i.
2. We set $\delta(u, v) = \min_{a \in S_i}\{\delta(u, a) + \delta(a, v)\}$.

In Step 2, the distance estimates $\delta(u, a), \delta(a, v)$ are the estimates computed in Step 1. By definition, all the vertices of the 2^i-dense paths and their edges to their neighbors are included in the graph G_i, so it is enough to compute distances in this graph in order to obtain +2-approximation for the lengths of 2^i-dense paths. Since we worked on a sparser graph the running time for computing the BFS trees in level i is now $O(|S_i||E_i|) = \tilde{O}((n/2^i) \cdot n2^{i+1}) = \tilde{O}(n^2)$. Summing up over all $O(\log n)$ levels gives $\tilde{O}(n^2)$ running time for computing the BFS trees. After this step, we are guaranteed that if the shortest path between u and v is 2^i-dense, then there is a vertex $a \in S_i$ such that $\delta(u, a) + \delta(a, v) \leq d(u, v) + 2$, where $\delta(u, a), \delta(a, v)$ are the distances computed from a in G_i.

Implementing Step 2 faster. By now we have computed all the relevant distances from the sets S_i, there is a remaining challenge. In order to estimate the distance of each pair u and v we need distance estimates going through all possible vertices $a \in S_i$ (Step 2), which takes $\tilde{O}(n^{2.5})$ time for all pairs, as in the worst case $|S_i| = \tilde{O}(\sqrt{n})$. To implement this step faster, we exploit fast matrix multiplication. Note that Step 2 is essentially equivalent to matrix multiplication when the operations are minimum and plus, such multiplications are called *distance products*. While it is well-known that APSP can be computed using matrix multiplication, usually it requires multiplication of square matrices which takes $\tilde{O}(n^\omega)$ time. The trick in our case is that since we only want to compute distances through a small set S_i, we can use fast *rectangular* matrix multiplication, as first exploited by Zwick [Zwi02]. Since $|S_i| = \tilde{O}(\sqrt{n})$ we only need to multiply matrices with dimensions $n \times n^{0.5}$ and $n^{0.5} \times n$, which can be done in just $O(n^{2.045})$ time using the rectangular matrix multiplication algorithms by Le Gall and Urrutia [GU18]. Fast matrix multiplication algorithms do not directly apply to distance products, however there is a well-known reduction that shows that we can get $(1+\epsilon)$-approximation for distance products in the same time [Zwi02], see Section 2 for the details.

Conclusion. Using the ideas described we can get a $(2+\epsilon)$-approximation in $O(n^{2.045})$ time. To remove the ϵ term in the stretch, we exploit the fact that in unweighted graphs we can get an additive $+\log n$-approximation in $\tilde{O}(n^2)$ time [DHZ00]. Note that for pairs of vertices at distance larger than $\log n$, an additive $+\log n$-approximation is already a multiplicative 2-approximation. Hence we can focus our attention on pairs of vertices at distance at most $\log n$ from each other. We show that this allows us to turn any $(2+\epsilon)$-approximation for unweighted graphs to a 2-approximation, as long as the algorithm depends polynomially on $1/\epsilon$ (see Section 2.1 for the details).

Moreover, we can improve the running time to $O(n^{2.032})$ by a better balancing of our two approaches, for dealing with sparse and dense paths. In particular, in our discussion so far we defined sparse paths to be ones where the maximum degree is at most $\sqrt{n}$, which led to hitting sets of size $\tilde{O}(\sqrt{n})$. To obtain a faster algorithm we want to have a smaller hitting set of size $O(n^r)$ for an appropriate choice or r, and then the sparse paths are paths where the maximum degree is at most $O(n^{1-r})$. With these parameters, computing distances in the sparse graph using [BK10] takes $\tilde{O}(n^{2.5-r})$ time, while dealing with dense paths takes $\tilde{O}(n^{\omega(r)})$ time. Balancing these two terms gives an $O(n^{2.032})$ time algorithm. Full details appear in Section 3.

We remark that we can also implement Step 1 using the $(1+\epsilon)$-approximate multi-source shortest paths algorithm by Elkin and Neiman [EN22] that is based on fast rectangular matrix multiplication. This allows computing $(1+\epsilon)$-approximate distances from $\tilde{O}(n^r)$ sources in $\tilde{O}(n^{\omega(r)})$ time, which leads to the same overall running time. If we do so we can have only one set S as in the original description of the algorithm. In our paper we implement Step 1 using the combinatorial algorithm discussed above that takes $\tilde{O}(n^2)$ time, which shows that currently the bottleneck in the algorithm is Step 2, and other parts of the algorithm can be computed in $\tilde{O}(n^2)$ time.

A combinatorial algorithm. Our approach also leads to a simple combinatorial 2-approximation algorithm that takes $\tilde{O}(n^{2.25})$ time. To do so, we just change the threshold of sparse and dense paths. We say that a path is sparse if all the vertices in the path have degree at most $n^{3/4}$, and it is dense otherwise. Computing 2-approximations for sparse paths will

now take $\tilde{O}(n^{2.25})$ time by [BK10]. In the dense case, since the dense paths now have a vertex of degree at least $n^{3/4}$, we can construct a smaller hitting set S of size $\tilde{O}(n^{1/4})$, and then we can implement Steps 1 and 2 directly via a combinatorial algorithm in $\tilde{O}(n^{2.25})$ time (by the same approach described above but replacing the size of S with $\tilde{O}(n^{1/4})$). See Section 3.2 for the details.

We remark that Roditty [Rod23] recently obtained the same result (a combinatorial 2-approximation in $\tilde{O}(n^{2.25})$ time) via a different approach. At a high-level, his approach is based on a detailed case analysis of the +4-approximation algorithm of Dor, Halperin, and Zwick [DHZ00], showing that for close-by pairs a better approximation can be obtained.

Near-additive approximations. We can use the same approach also in order to obtain near-additive approximations. Note that in the dense case, our algorithm actually computed a $(1+\epsilon, 2)$-approximation, where the $(1+\epsilon)$ term comes from using fast matrix multiplication. Hence, if on the sparse graph we run the $+k$-additive approximation algorithm by Dor, Halperin, and Zwick [DHZ00] instead of the multiplicative 2-approximation algorithm of Baswana and Kavitha [BK10], we can get $(1+\epsilon, k)$-approximations for all the distances. See Section 3.3 for the details.

Weighted 2-Approximate APSP The techniques above do not generalize well to the weighted setting. For weighted graphs, we use a different approach, based on the set-up of *bunches* and *clusters* as introduced by Thorup and Zwick [TZ05]. For a parameter $p \in [\frac{1}{n}, 1]$ we sample each vertex with probability p, and if sampled we add it to a set S. With high probability, we have $|S| = \tilde{O}(pn)$. Now, for each vertex $u \in V$, we define the pivot of u to be the closest vertex in S to u, i.e., $p(u)$ is an arbitrary vertex in the set $\{v \in S : d(u,v) = d(u,S)\}$, and we define the bunch of u by $B(u) := \{v \in V : d(u,v) < d(u,p(u))\} \cup \{p(u)\}$. For a vertex $v \in V$, we define the cluster of u as the inverse bunch: $C(v) := \{u \in V : v \in B(u)\}$. Thorup and Zwick [TZ05] show how to compute pivots, bunches, clusters, and distances $d(u,v)$ for all $u \in V, v \in B(u)$ in time $\tilde{O}(\frac{m}{p})$. Moreover, in follow up work [TZ01] they show that with different techniques, that is, a more involved construction of S, we can have that both bunches and clusters are bounded by $\tilde{O}(\frac{1}{p})$ with high probability. The running time remains $\tilde{O}(\frac{m}{p})$.

Warm-up: 3-approximate APSP. To see how we use this to compute approximate shortest paths, we consider a pair of vertices $u, v \in V$. If either $v \in B(u)$ or $u \in B(v)$, we have the exact distance, so assume this is not the case. In particular if $v \notin B(u)$, then $d(u,p(u)) \leq d(u,v)$. We compute shortest paths from S to V, in particular obtaining $d(p(u),v)$. With this additional information we get a 3-approximation almost directly [TZ05]:

$$d(u,p(u)) + d(p(u),v) \leq d(u,p(u)) + d(p(u),u) + d(u,v) \leq 3d(u,v),$$

where the first inequality holds by the triangle inequality.

2-approximate APSP. By a closer inspection, we can improve this analysis to a 2-approximation for each pair of vertices, whose shortest path interacts in a particular way with the bunches. To be precise, let $u, v \in V$ be a pair of vertices and let π be a shortest path between them. Suppose it contains a vertex x, such that $x \notin B(u)$ and $x \notin B(v)$, see the right case in Figure 2. That means that $d(u,p(u)) \leq d(u,x)$ and $d(v,p(v)) \leq d(x,v)$, so at least one of the two is at most $d(u,v)/2$. Without loss of generality, say that $d(u,p(u)) \leq d(u,v)/2$. Hence, after computing shortest paths from S, we can obtain a 2-approximation as follows: $d(u,p(u)) + d(p(u),v) \leq d(u,p(u)) + d(u,p(u)) + d(u,v) \leq 2d(u,v)$, where the first inequality holds by the triangle inequality.

It remains to give an algorithm that guarantees a 2-approximation for the case that for every vertex x on the shortest path π we have $x \in B(u)$ or $x \in B(v)$, the left case in Figure 2. Note that this case also contains the special case that the bunches overlap. We will refer to it as the 'adjacent case', since the two bunches $B(u)$ and $B(v)$ are adjacent in the sense that they are connected by an edge. In previous work, this adjacent case is often a bottleneck in the running time, and dealt with in various ways. As seen above, by not distinguishing it at all, we obtain a 3-approximation [TZ05]. By only considering the cases where the bunches have at least one vertex in common, Baswana, Goyal, and Sen [BGS09] obtain a $(2,1)$-approximation. In Appendix B we generalize this result to $(2, W_{u,v})$-approximate APSP in weighted graphs, where $W_{u,v}$ is the maximum weight of an edge on a shortest $u-v$ path. For a 2-approximation, Kavitha [Kav12] and Baswana and Kavitha [BK10] each use a multilevel approach, the latter of which we detail later. More recently, in distributed [CDKL21, DP22] and dynamic [DFNV22] 2-approximate APSP algorithms, the adjacent case is computed explicitly. To be precise, they compute

$$\delta_{\text{adjacent}}(u,v) = \min\{d(u,u') + w(u',v') + d(v',v) : \{u',v'\} \in E, u' \in B(u), v' \in B(v)\},$$

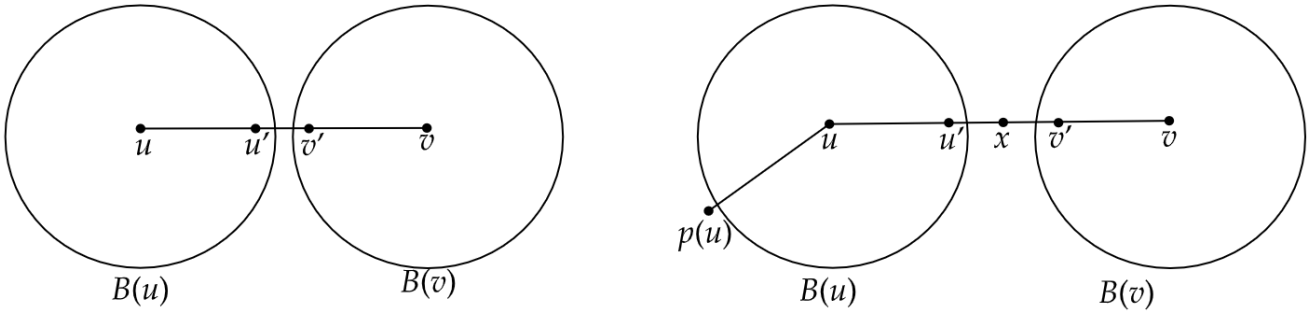

Figure 2: Two different possible interactions between the shortest path between u and v, and the bunches of u and v.

which in the case that $\pi \subseteq B(u) \cup B(v)$ returns the exact distance between u and v. Using these algorithms directly does not lead to fast algorithms in our, centralized, setting, since they are tailored for their respective models. Our work is inspired by this approach, and gives two novel ways to compute $\delta_{\text{adjacent}}(u,v)$; one for sparse graphs and one for dense graphs.

Sparse case. Rather than fixing $u, v \in V$ and trying to compute $\delta_{\text{adjacent}}(u,v)$, we fix an edge $\{u',v'\} \in E$, and see for which pairs $u, v \in V$ this contributes to $\delta_{\text{adjacent}}(u,v)$. More precisely, by definition of bunches and clusters we have $u' \in B(u) \iff u \in C(u')$, so we can compute δ_{adjacent} as follows: for all $\{u',v'\} \in E$, for all $u \in C(u')$, and for all $v \in C(v')$:

1. Initialize $\delta_{\text{adjacent}}(u,v) \leftarrow d(u,u') + w(u',v') + d(v',v)$ if no such entry exists.
2. Otherwise: $\delta_{\text{adjacent}}(u,v) \leftarrow \min\{\delta_{\text{adjacent}}(u,v), d(u,u') + w(u',v') + d(v',v)\}$.

We note that Step 1 and 2 can both be done in constant time, so computing δ_{adjacent} takes time

$$\sum_{\{u',v'\}\in E} \sum_{u\in C(u')} \sum_{v\in C(v')} O(1) = \sum_{\{u',v'\}\in E} O(|C(u')| \cdot |C(v')|) = \sum_{\{u',v'\}\in E} \tilde{O}(\tfrac{1}{p^2}),$$

where the last equality holds since clusters have size at most $\tilde{O}(\frac{1}{p})$. This means it takes $\tilde{O}(\frac{m}{p^2})$ time in total to compute δ_{adjacent}.

Together with the running time for computing bunches and clusters, $\tilde{O}(\frac{m}{p})$, and the running time for computing shortest paths from S, $\tilde{O}(|S|m) = \tilde{O}(pnm)$ using Dijkstra, we obtain $\tilde{O}(\frac{m}{p^2} + pnm)$. For $p = n^{-1/3}$, we obtain running time $\tilde{O}(mn^{2/3})$.

We note that for each pair of vertices $u, v \in V$, we still have to take the minimum between the estimate through the pivot, $d(u,p(u)) + d(p(u),v)$, and $\delta_{\text{adjacent}}(u,v)$. Doing this explicitly would take n^2 time. Instead, we provide a distance oracle, and perform this minimum in constant time when the pair u, v is queried.

Dense case. We adapt our algorithm in two ways for the dense case. We use a different approach to compute δ_{adjacent}, and we compute shortest paths from the set of pivots S differently.

First, we show how to compute δ_{adjacent} in $\tilde{O}(\frac{n^2}{p})$ time. For each node u, we run Dijkstra twice on a graph with $\tilde{O}(\frac{n}{p})$ edges, whose size comes from the fact that for each node the bunches have size $\tilde{O}(\frac{1}{p})$. On the first graph we obtain estimates from u to v' for every v' that neighbors the bunch of u, i.e., $\exists u' \in B(u)$ such that $\{u',v'\} \in E$. And on the second graph we obtain estimates $\delta_{\text{adjacent}}(u,v)$ for all $v \in V$. For more details see Section 4.4.

Second, for computing shortest paths from S, we can do something (much) more efficient than computing multiple Dijkstra's by using recent results on approximate multi-source shortest paths (MSSP). Elkin and Neiman [EN22] provide efficient $(1+\epsilon)$-MSSP results using rectangular matrix multiplication. For example, we can let the number of pivots be as big as $n^{0.8}$, while the running time stays below $O(n^{2.23})$. This means that the sizes of bunches drop dramatically to $\tilde{O}(n^{0.2})$, making the above very efficient. To be more precise, we need to balance the running time to compute δ_{adjacent}, $\tilde{O}(\frac{n^2}{p})$, with the running time to compute shortest paths from S, which has size $\tilde{O}(pn)$. If we use Dijkstra for the latter (for a graph with $m = n^2$), we need to balance $\frac{n^2}{p}$ and pn^3, obtaining running time $\tilde{O}(n^{2.5})$ for $p = 1/\sqrt{n}$.

To see how this trade-off improves using [EN22], we denote $p = n^{r-1}$. Now we have $|S| = \tilde{O}(pn) = \tilde{O}(n^r)$, and we can compute $(1+\epsilon)$-approximate shortest paths from S in $\tilde{O}(n^{\omega(r)})$ time. We obtain total running time $\tilde{O}(\frac{n^2}{p} + n^{\omega(r)}) = O(n^{2.214})$, using [GU18] for an upper bound on $\omega(r)$.

We note that the $(1+\epsilon)$-factor carries over to our stretch analysis, making it a $(2+\epsilon)$-approximation, see Lemma 4.2.

A Density Sensitive Algorithm. Next, we describe how we can generalize the ideas from the dense case to a wider density range. Our goal is to combine our approach from the dense case with the 2-approximate APSP algorithm of Baswana and Kavitha [BK10]. Similar to the dense case, they create a set of pivots S of size $\tilde{O}(pn)$. They compute shortest paths from S to V using Dijkstra in $\tilde{O}(pnm)$ time, and use this for an estimate through the pivot.

Baswana and Kavitha [BK10] do not consider the adjacent case explicitly, but consider $O(\log n)$ additional levels, each with a gradually growing set of pivots S_i. For each of those sets, they compute shortest paths in a sparser graph, where the distances to some essential vertices equal the distances in the original graph. They show that on at least one of the levels, a distance estimate through the pivot of that level gives a 2-approximation. For details we refer to Section 4.5.

By computing shortest paths on the sparser graph, they avoid the expensive computation of shortest paths from S_i to all of V. Instead, for each level, they require $\tilde{O}(m/p)$ time to construct the sparser graph, and $\tilde{O}(n^2)$ time to compute shortest paths from S_i. Combining this with the first step, their algorithm takes $\tilde{O}(pnm + m/p + n^2)$ time. Setting $p = 1/\sqrt{n}$ balances the terms and gives $\tilde{O}(m\sqrt{n} + n^2)$ time.

We modify their algorithm in two ways. First of all, we remark that parameterizing the size of the (smallest) set of pivots S, by $|S| = \tilde{O}(pn)$ already gives us the following result, which allows us to get a better trade-off.

Theorem 1.6. *There exists a randomized algorithm that, given an undirected graph with non-negative edge weights $G = (V, E)$ and parameters $p \in (\frac{1}{n}, 1]$, $\epsilon \geq 0$, computes $(2+\epsilon)$-approximate APSP. With high probability, the algorithm takes $\tilde{O}(n^2 + m/p + T(\tilde{O}(pn)))$ time, where $T(s)$ is the time to compute $(1+\epsilon)$-MSSP from s sources.*

Secondly, we use the fast MSSP algorithm of Elkin and Neiman [EN22] to compute the shortest paths from S faster. If we write $p = n^{q-1}$, for some $r \in [0, 1]$, then [EN22] gives $T(pn) = T(n^r) = \tilde{O}(m^{1+o(1)} + n^{\omega(r)})$. The total running time for 2-approximate APSP is then $\tilde{O}(mn^{1-r} + n^{\omega(r)})$, for any $r \in [0, 1]$. For $m = n^2$ this recovers our result, Theorem 1.2, for dense graphs. See Theorem 1.3 and Table 2 for our results for graphs with different densities.

1.3 Additional Related Work

Approximation between 2 and 3. In addition to the 2-approximate APSP algorithms mentioned above, approximate APSP algorithms with approximations between 2 and 3 in weighted undirected graphs are studied in [CZ01, BK10, Kav12, AR21]. In particular, [CZ01, BK10] studied algorithms with approximation 7/3, and [Kav12] studied an algorithm with approximation 5/2. These results were generalized by Akav and Roditty [AR21] who showed an algorithm with approximation $2 + \frac{k-2}{k}$ and running time $\tilde{O}(m^{2/k}n^{2-3/k} + n^2)$ for any $k \geq 2$.

Algorithms using fast matrix multiplication. Algorithms for fast rectangular matrix multiplication are studied in [Cop82, LR83, Cop97, HP98, KZHP08, Gal12, GU18], and have found numerous applications in algorithms (see e.g. [Zwi02, RS11, Yus09, YZ04, KRSV07, KSV06, SM10, WWWY14, BRSW+21, BN19, VFN22, BHGW+21, WX20, GR21]).

In the context of APSP, the state of the art algorithm by Zwick for computing APSP in directed graphs with bounds weights is based on rectangular matrix multiplication [Zwi02]. In addition, Kavitha [Kav12] used fast rectangular matrix multiplication as one of the ingredients in her $(2+\epsilon)$-approximation algorithm for weighted APSP. Additive +2-approximations for APSP based on fast matrix multiplication are studied in [DKRW+22, Dür23]. The latter algorithms are based on Min-Plus product of rectangular bounded difference matrices. The $(1+\epsilon, 2)$-approximation of Berman and Kasiviswanathan [BK07] is also based on fast rectangular matrix multiplication. In addition, Elkin and Neiman showed $(1+\epsilon)$-approximation for multi-source shortest paths based on rectangular matrix multiplication [EN22]. For the problem of computing shortest paths for $S \times T$, for $|S|, |T| = O(n^{0.5})$, Dalirrooyfard, Jin, Vassilevska Williams, and Wein [DJWW22] provide a 2-approximation in $\tilde{O}(m + n^{(1+\omega)/8})$ time for weighted graphs, by leveraging *sparse* rectangular matrix multiplication. Dynamic algorithms for shortest paths and spanners based on rectangular matrix multiplication are studied in [BN19, VFN22, BHGW+21], and an algorithm for approximating the diameter based on rectangular matrix multiplication is studied in [BRSW+21].

1.4 Discussion

In this work we showed fast algorithms for 2-approximate APSP, many intriguing questions remain open. First, our algorithm for 2-approximate APSP in unweighted undirected graphs takes $O(n^{2.032})$ time, and an interesting direction for future work is to try to obtain an $O(n^2)$ time algorithm for this problem.

Second, many of our algorithms are based on fast rectangular matrix multiplication, and it would be interesting to develop also fast combinatorial algorithms for these problems. Currently the fastest combinatorial algorithm for 2-approximate APSP in unweighted undirected graphs is the recent algorithm by [Rod23] that takes $\tilde{O}(n^{2.25})$ time. For weighted undirected graphs the fastest combinatorial 2-approximate APSP algorithm takes $\tilde{O}(m\sqrt{n}+n^2)$ time, which is $\tilde{O}(n^{2.5})$ for dense graphs, where we show a non-combinatorial $(2+\epsilon)$-approximation algorithm that takes $O(n^{2.214})$ time. Narrowing the gaps between the combinatorial and non-combinatorial algorithms, or proving conditional hardness results for combinatorial algorithms is an interesting direction for future research. We remark that such a gap exists also for the case of additive +2-approximations, where the fastest combinatorial algorithm takes $\tilde{O}(\min\{n^{3/2}m^{1/2}, n^{7/3}\})$ time [DHZ00], and the fastest non-combinatorial algorithm takes $O(n^{2.260})$ time [Dür23].

2 Preliminaries

Throughout, we use $n = |V|$ and $m = |E|$. When writing log-factors, we round them to the closest integer. We use $d_G(u,v)$ for the distance between u and v in G, where we omit G if it is clear from the context. We write $\delta(u,v)$ for distance estimates between u and v. We denote SSSP for the Single Source Shortest Path Problem, MSSP for the Multi Source Shortest Path Problem, and APSP for the All Pairs Shortest Path Problem. SSSP takes the source as part of the input, and MSSP takes the set of sources as part of the input. We say that an algorithm gives an (α,β)-approximation for SSSP/MSSP/APSP if for any pair of vertices u,v it returns an estimate $\delta(u,v)$ of the distance between u and v such that $d(u,v) \le \delta(u,v) \le \alpha \cdot d(u,v) + \beta$, where $d(u,v)$ is the distance between u and v. If $\beta = 0$, we get a purely multiplicative approximation that we refer to as α-approximation. If $\alpha = 1$, we get a purely additive approximation, and call it a $+\beta$-approximation. If $\alpha = 1+\epsilon$, we refer to it as a near-additive approximation.

All our randomized algorithms are always correct, and provide running time guarantees 'with high probability', which means with probability $1 - n^{-c}$ for any constant c.

When we refer to the APSP problem, we are required to output all n^2 distances. If we drop this requirement, and only want to give a data structure subject to distance queries, we call it a *distance oracle*. For distance oracles, there are three complexities to consider: preprocessing time, space, and query time. Both the preprocessing time and the required space can be less than n^2. In our algorithms, we provide constant query time.

Rectangular Matrix Multiplication and Shortest Paths Let A be an $n_1 \times n_2$ matrix, and B be an $n_2 \times n_3$ matrix, then we define the *distance product*, also called *min-plus product*, $A \star B$ by

$$(A \star B)_{ij} = \min_{1 \le k \le n_2} \{A_{ik} + B_{kj}\},$$

for $1 \le i \le n_1$ and $1 \le j \le n_3$.

Moreover we say a matrix $A' \in \mathbb{R}^{n_1 \times n_2}$ is a $(1+\epsilon)$-approximation of a matrix $A \in \mathbb{R}^{n_1 \times n_2}$ if $A_{ij} \le A'_{ij} \le (1+\epsilon)A_{ij}$, for all $1 \le i \le n_1$ and $1 \le j \le n_2$.

Distance product form a *semiring*, i.e., a ring without guaranteed additive inverses. Although the results for fast matrix multiplication hold for *rings*, it turns out they can be leveraged for distance products as well [AGM97]. In particular, we have the following result for approximate distance products.

Theorem 2.1. ([Zwi02]) *Let W be a positive integer, and $\epsilon > 0$ be a parameter. Let A be an $n \times n^r$ matrix and B be an $n^r \times n$ matrix, whose entries are all in $\{0,1,\dots,W\} \cup \{\infty\}$. Then there is an algorithm that computes a $(1+\epsilon)$-approximation to $A \star B$ in time $\tilde{O}(n^{\omega(r)}/\epsilon \log W)$.*

Here $\omega(r)$ denotes the time constant for rectangular matrix multiplication, i.e., the constant such that in $n^{\omega(r)}$ time we can multiply an $n \times n^r$ with an $n^r \times n$ matrix. This algorithm is a deterministic reduction to rectangular matrix multiplication, for which the state of the art [GU18] is also deterministic. Note that [GU18] does not provide a closed form for $\omega(r)$. Throughout, we use the tool of van den Brand [Bra] to balance $\omega(r)$ with other terms to obtain our numerical results.

Backurs, Roditty, Segal, Vassilevska Williams, and Wein [BRSW+21] leverage these results to obtain multi-source approximate shortest paths from $\sqrt{n}$ sources, given that the distances are short. Elkin and Neiman [EN22] show how to exploit rectangular matrix multiplication to compute distances from an arbitrary set of sources S.

Theorem 2.2. ([EN22]) *There exists a deterministic algorithm that, given a parameter $\epsilon > 0$, an undirected graph $G = (V,E)$ with integer weights bounded by W and a set of sources S of size $|S| = O(n^r)$, computes $(1+\epsilon)$-approximate distances for $S \times V$ in $\tilde{O}(m^{1+o(1)} + n^{\omega(r)}(1/\epsilon)^{O(1)} \log W)$ time.*

2.1 From $2+\epsilon$ to 2-Approximate APSP Dor, Halperin, and Zwick [DHZ00] provide a $+\log n$-approximate APSP for unweighted graphs in $\tilde{O}(n^2)$ time. Using this result, we can reduce the problem of an unweighted 2-approximation to an unweighted $(2+\epsilon)$-approximation.

LEMMA 2.1. *Given an algorithm that computes $(2+\epsilon)$-approximate APSP on unweighted graphs in $\tau(n,m)\,\mathrm{poly}(1/\epsilon)$ time, we obtain an algorithm that computes 2-approximate APSP in $\tilde{O}(n^2+\tau(n,m))$ time. The reduction is deterministic and combinatorial.*

Proof. Set $\epsilon = 1/\log n$, and let $\delta(u,v)$ denote the output of the $(2+\epsilon)$-approximate APSP algorithm. Let $\delta'(u,v)$ denote the output of the $+\log n$ approximation of [DHZ00] (see also Theorem 3.5). We output $\hat{d}(u,v) = \min\{\lfloor\delta(u,v)\rfloor, \delta'(u,v)\}$. Clearly this takes $\tilde{O}(n^2+\tau(n,m))$ time in total, so it remains to show that $\hat{d}(\cdot,\cdot)$ is a 2-approximation.

First of all we notice that since distances in an unweighted graph are integers, we have that $d(u,v) \le \delta(u,v)$ implies that $d(u,v) \le \lfloor\delta(u,v)\rfloor$. Since also $d(u,v) \le \delta'(u,v)$, we can conclude $d(u,v) \le \hat{d}(u,v)$.

To show that $\hat{d}(u,v) \le 2d(u,v)$, we distinguish the cases $d(u,v) < \log n$ and $d(u,v) \ge \log n$. If $d(u,v) < \log n$, then from $\delta(u,v) \le (2+\epsilon)d(u,v)$ we obtain

$$\begin{aligned}\hat{d}(u,v) &\le \lfloor\delta(u,v)\rfloor \le \lfloor(2+\epsilon)d(u,v)\rfloor \\ &= 2d(u,v) + \lfloor\epsilon d(u,v)\rfloor = 2d(u,v) + \lfloor\tfrac{d(u,v)}{\log n}\rfloor \\ &\le 2d(u,v).\end{aligned}$$

Now if $d(u,v) \ge \log n$, we have that $\hat{d}(u,v) \le \delta'(u,v) \le d(u,v) + \log n \le 2d(u,v)$. ∎

3 Approximate APSP Algorithms for Unweighted Graphs

We obtain our three unweighted APSP results, 2-approximate (Theorem 1.1), combinatorial 2-approximate (Theorem 3.4), and $(1+\epsilon,k)$-approximate for even $k \ge 2$ (Theorem 1.5), through a more general framework. In this section, we develop said framework, from which these theorems follow almost immediately.

The goal of our framework is to split the graph into two cases: a sparse graph and a dense graph. On the sparse graph, we just run an existing approximate APSP algorithm that performs well on sparse graphs (denoted by Algorithm $\mathcal{A}$ in Theorem 3.1 below). For the dense graph, we use more ingenuity. We further split it into poly $\log n$ density regimes, where in each regime the bottleneck is to compute APSP through a known set S, i.e., for each u,v to find a shortest path of the form (u,x,v) for some $x \in S$ (this task is done by Algorithm $\mathcal{B}$). If we solve this problem exactly, we obtain a +2-approximation. If we solve it approximately, this carries over into the overall approximation factor. For example, if we solve it up to a multiplicative factor $(1+\epsilon)$, we obtain total approximation $(1+\epsilon, 2(1+\epsilon))$ for the dense graph. For our approximations, we only use algorithms $\mathcal{B}$ that are either exact or $(1+\epsilon)$-approximate. The formal statement of the framework is as follows.

THEOREM 3.1. *Let $\mathcal{A}$ be an algorithm that computes $(mult_A, add_A)$-approximate APSP on unweighted graphs with running time $\tau_A(n,m)$, and let $\mathcal{B}(S)$ be an algorithm that computes $(mult_B, add_B)$-approximate all-pairs shortest paths on weighted graphs, among the $u-v$ paths of the form (u,x,v) for some x in a given set S, with running time $\tau_B(n,|S|)$. Then there exists an algorithm that, given an unweighted, undirected graph $G=(V,E)$ and a parameter $r \in [0,1]$, computes approximate APSP with running time $\tau_A(n,n^{2-r}) + \tilde{O}(\tau_B(n,\tilde{O}(n^r)))$, where for each pair of vertices we have either a $(mult_A, add_A)$ or a $(mult_B, add_B + 2mult_B)$ approximation.*

Besides possibly Algorithm $\mathcal{A}$ and $\mathcal{B}$, the procedure is deterministic and combinatorial.

Note that $\tau_A(n,n^{2-r}), \tau_B(n,\tilde{O}(n^r)) \ge n^2$ as they both need n^2 time to write their output.

In Section 3.1, Section 3.2, and Section 3.3, we obtain a 2-approximation, a combinatorial 2-approximation, and a near-additive approximation respectively, by using different algorithms for $\mathcal{A}, \mathcal{B}$ and balancing the parameter r accordingly.

Next, we proceed by describing the algorithm satisfying Theorem 3.1, followed by a correctness proof and running time analysis. Pseudo-code can be found in Algorithm 1. To desecribe the algrithm, we recall the notion of hitting sets. A set S is said to be a *hitting set* for the vertices that have at least one neighbor in S. The following result provides a hitting set S for the vertices with degree at least s. Such a set can easily be obtained by random sampling or with a deterministic algorithm [ACIM99, DHZ00].

LEMMA 3.1. ([DHZ00]) *There exists a deterministic algorithm* `HittingSet`(G,s) *that, given an undirected graph $G=(V,E)$ and a parameter $1 \le s \le n$, computes a set $S \subseteq V$ of size $O(\frac{n}{s}\log n)$ such that all vertices of degree at least s have at least one neighbor in S. The algorithm takes $O(m+n)$ time.*

Algorithm. Given the parameter $r \in [0,1]$, we say a vertex is *light* if it has at most n^{1-r} incident edges. We look at the *sparse graph*, where each vertex keeps at most n^{1-r} edges to its neighbors. On this graph we run Algorithm $\mathcal{A}$. We show that if for two vertices the shortest path only consists of light vertices, then this provides a $(\text{mult}_A, \text{add}_A)$-approximation. We also run the following procedure, on the entire input graph. This part ensures that if the shortest path between two vertices contains at least one dense vertex, we obtain a $(\text{mult}_B, \text{add}_B + 2)$-approximation. We look at $O(\log n)$ levels. At level i, the goal is to get an approximation for a shortest path with maximum degree in $[2^i, 2^{i+1}]$. We let i be all the integer values between $(1-r)\log n$ and $\log n$. By abuse of notation, we write $i = (1-r)\log n, (1-r)\log n + 1, \ldots, \log n$. At level i we do the following.

1. Let $S_i \subseteq V$ be defined by $\texttt{HittingSet}(G, 2^i)$. We set $G_i = (V, E_i)$ to be the graph with $E_i := \{\{u,v\} \in E : \deg(u) \leq 2^{i+1} \text{ or } \deg(v) \leq 2^{i+1}\}$.

2. We compute multi-source shortest paths from S_i on G_i, by running Dijkstra from each vertex in S_i. We store these results in the graph $G_i' := (V, E_i')$. So $E_i' = S_i \times V$, and $w_{E_i'}(a,v) := d_{G_i}(a,v)$.

3. Now we want to compute shortest paths through S_i on the graph induced by edges in step (2), hereto we call algorithm $\mathcal{B}(G_i', S_i)$.

Algorithm 1: Our algorithm to compute APSP using algorithms $\mathcal{A}$ and $\mathcal{B}$

Let $G' = (V, E')$, where $E' := \{\{u,v\} \in E : \deg(u) \leq n^{1-r} \text{ or } \deg(v) \leq n^{1-r}\}$;
Let $\delta_A(\cdot,\cdot)$ be the result of Algorithm $\mathcal{A}(G')$;
for $i = (1-r)\log n, \ldots, \log n$ **do**
  Let $G_i = (V, E_i)$, where $E_i := \{\{u,v\} \in E : \deg(u) \leq 2^{i+1} \text{ or } \deg(v) \leq 2^{i+1}\}$;
  $S_i := \texttt{HittingSet}(G, 2^i)$;
  for $a \in S_i$ **do**
    Run Dijkstra from a on G_i
  end
  Let $G_i' := (V, E_i')$ be a weighted graph, where $E_i' = S_i \times V$, and $w_{E_i'}(a,v) := d_{G_i}(a,v)$ for $a \in S_i, v \in V$.;
  Let $\delta_i(\cdot,\cdot)$ be the result of Algorithm $\mathcal{B}(G_i', S_i)$;
end
$\delta_B(u,v) := \min\{\delta_i(u,v) : (1-r)\log n \leq i \leq \log n\}$, for $u, v \in V$.;
$\delta(u,v) := \min\{\delta_A(u,v), \delta_B(u,v)\}$, for $u, v \in V$.;
return $\delta(\cdot,\cdot)$

Correctness. Given $u, v \in V$, we have to show that the returned approximation $\delta(u,v)$ is a 2-approximation. We distinguish two (non-disjoint) cases:

a) There exists a shortest path from u to v solely consisting of light vertices: vertices of degree at most n^{1-r}. In this case, we show we obtain a $(\text{mult}_A, \text{add}_A)$-approximation through δ_A.

   In this case, this shortest path is fully contained in the sparse graph, hence running algorithm $\mathcal{A}$ on it provides the given approximation δ_A.

b) There exists a shortest path from u to v containing at least one heavy vertex: a vertex with degree at least n^{1-r}. In this case, we show we obtain a $(\text{mult}_B, \text{add}_B + 2\text{mult}_B)$-approximation through δ_B.

   Let $i \in \{(1-r)\log n, (1-r)\log n + 1, \ldots \log n\}$ be the maximal index such that there exists a vertex x on the shortest path from u to v with degree in $[2^i, 2^{i+1}]$. We show that δ_i provides the desired approximation. By definition of S_i, x has at least one neighbor in S_i, denote this neighbor by a, see Figure 3.

   Now consider the distance from u to v in G_i. Since G_i contains the shortest path in G, we have $d_{G_i}(u,v) = d_G(u,v)$. Further we know that $d(u,a) + d(a,v) \leq d_{G_i}(u,v) + 2$, since a is neighboring x on the shortest path between u and v. In particular, this means that $\min_{a' \in S_i} d_{G_i}(u,a') + d_{G_i}(a',v) \leq d_{G_i}(u,v) + 2 = d_G(u,v) + 2$. So we can focus on computing shortest paths through S_i. In the second step of the algorithm, we have computed the (exact) distances S_i: $d_{G_i}(a',y)$ for all $y \in V$, in particular to u and v. To compute $\min_{a' \in S_i} d_{G_i}(u,a') + d_{G_i}(a',v)$, we use Algorithm $\mathcal{B}(G_i', S_i)$. which then

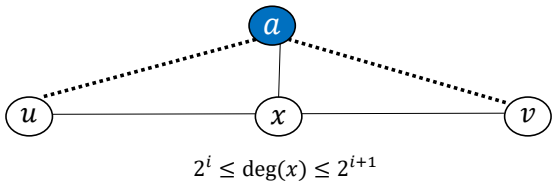

Figure 3: Illustration of the stretch analysis.

provides an estimate $\delta_i(u,v)$ for the distance from u to v through S_i in G_i. Hence, the approximation factors aggregate as follows:

$$\begin{aligned}\delta_i(u,v) &\le \text{mult}_B(d_{G_i}(u,a) + d_{G_i}(a,v)) + \text{add}_B \\ &\le \text{mult}_B(d_{G_i}(u,x) + 1 + d_{G_i}(x,v) + 1) + \text{add}_B \\ &= \text{mult}_B \cdot d_{G_i}(u,v) + 2 \cdot \text{mult}_B + \text{add}_B \\ &= \text{mult}_B \cdot d(u,v) + 2 \cdot \text{mult}_B + \text{add}_B.\end{aligned}$$

If Algorithm $\mathcal{A}$ is correct with probability at least $1 - p_A$, and Algorithm $\mathcal{B}$ is correct with probability at least $1 - p_B$, then our algorithm is correct with probability at least $1 - p_A - p_B \cdot \log n$. Note that in all applications below we have $p_A = 0 = p_B$, i.e., both algorithms are always correct.

Running time. We run algorithm $\mathcal{A}(G')$, where G' has n vertices and $O(n^{2-r})$ edges, hence this takes $\tau_A(n, n^{2-r})$ time. Then, for each i, we perform three steps. First, we compute a hitting set in $O(m+n)$ time. Second, we run Dijkstra from $|S_i| = \tilde{O}(\frac{n}{2^i})$ vertices on a graph with $O(n2^{i+1})$ edges, taking $\tilde{O}(n^2)$ time. Third, we run algorithm $\mathcal{B}(G'_i, S_i)$, where we know every shortest path goes through S_i with $|S_i| = O(\frac{n}{2^i}\log n) = \tilde{O}(n^r)$, hence taking $\tau_B(n, \tilde{O}(n^r))$ time.

Any probabilistic guarantees on the running time of Algorithm $\mathcal{A}$ and $\mathcal{B}$ simply carry over.

3.1 2-Approximate APSP for Unweighted Graphs

We employ the theorem of the previous section to obtain a multiplicative 2-approximation. Hereto we use the following result of Baswana and Kavitha [BK10], which gives an efficient algorithm for sparse graphs.[6]

THEOREM 3.2. ([BK10]) *There exists a randomized, combinatorial algorithm that, given an undirected graph with non-negative edge weights $G = (V, E)$, computes 2-approximate APSP. With high probability, the algorithm takes $\tilde{O}(n^2 + m\sqrt{n})$ time.*

The algorithm of Theorem 3.2 can be retrieved from our more general algorithm for weighted graphs in Section 4.

THEOREM 3.3. *There exists a randomized algorithm that, given an unweighted, undirected graph $G = (V, E)$, computes 2-approximate APSP. With high probability, the algorithm takes $\tilde{O}(n^{2.5-r} + n^{\omega(r)})$ time, for any $r \in [0,1]$. Using known upper bounds for rectangular matrix multiplication, this is $O(n^{2.032})$ time.*

Proof. By Lemma 2.1, it is sufficient to provide a $(2+\epsilon)$-approximation, given that the running time only depends polynomially on $1/\epsilon$. We utilize Theorem 3.1 and for that we need to specify which algorithms to use for $\mathcal{A}$ and $\mathcal{B}$, and analyze the tradeoffs on the approximation and running time.

For algorithm $\mathcal{A}$ we use Theorem 3.2 ([BK10]) on a graph with n vertices and n^{2-r} edges, which results in a 2-approximation in $\tilde{O}(n^2 + n^{5/2-r})$ time w.h.p. Algorithm $\mathcal{B}$ has to provide shortest $u - v$ paths, that are minimal among all paths of the form $u - x - v$, for x in a given set S. We use rectangular matrix multiplication for this: multiplying the $V \times S$ with the $S \times V$ edge weight matrices gives exactly the paths of length 2. We obtain $(1 + \epsilon/2)$-approximate shortest paths through a set of $\tilde{O}(n^r)$ vertices in time $\tilde{O}(m^{1+o(1)} + n^{\omega(r)}(1/\epsilon)^{O(1)})$ [Zwi02] (see also Theorem 2.1).

[6] We note that [BK10] states their result with an *expected* running time. We can easily make this 'with high probability' as follows. We run the algorithm $\log n$ time, stopping whenever we exceed the running time we aim for by more than a factor $\log n$. By a Chernoff bound, at least one of them finishes within this time w.h.p.

By Theorem 3.1, for each pair of vertices we either obtain a stretch of $(2,0)$, or a stretch of $(1+\epsilon/2,(1+\epsilon/2)\cdot 2)$. Note that for any $d(u,v)\geq 2$ we have $(1+\epsilon/2)d(u,v)+(1+\epsilon/2)\cdot 2\leq (1+\epsilon)d(u,v)+2$, so we have a $(1+\epsilon,2)$-approximation, hence a $(2+\epsilon)$-approximation in total.

Also by Theorem 3.1, w.h.p. we obtain a running time of $\tilde{O}(n^2)+\tilde{O}(n^2+n^{5/2-r})+\tilde{O}(m^{1+o(1)}+n^{\omega(r)}(1/\epsilon)^{O(1)}))=\tilde{O}((n^{5/2-r}+n^{\omega(r)})(1/\epsilon)^{O(1)})=\tilde{O}(n^{2.03184039}(1/\epsilon)^{O(1)})$ for $r=0.46815961$, where we use [Bra] to balance the terms. □

3.2 2-Approximate Combinatorial APSP for Unweighted Graphs The algorithm of the previous section uses matrix multiplication as a subroutine. In this section, we present a simple combinatorial algorithm for the same problem, matching the very recent result by Roditty [Rod23].

THEOREM 3.4. *There exists a* combinatorial *algorithm that, given an unweighted, undirected graph $G=(V,E)$, computes 2-approximate APSP. With high probability, the algorithm takes $\tilde{O}(n^{2.25})$ time.*

Proof. Again, we use Theorem 3.1. For algorithm $\mathcal{A}$ we use Theorem 3.2 to obtain a 2-approximation in $\tilde{O}(n^2+n^{5/2-r})$ time w.h.p. Algorithm $\mathcal{B}$ has to provide shortest $u-v$ paths, that are minimal among all paths of the form $u-x-v$, for x in a given set S. In particular, we can consider the graph $(V,S\times V)$, which has $n\cdot\tilde{O}(n^r)=\tilde{O}(n^{1+r})$ edges. Running Dijkstra from each node gives the (exact) result in $\tilde{O}(n^{2+r})$ time.

By Theorem 3.1, for each pair of vertices we either obtain a stretch of $(2,0)$, or a stretch of $(0,2)$, hence a $(2,0)$-approximation in total.

Also by Theorem 3.1, with high probability, we obtain a running time of $\tilde{O}(n^{2.5-r}+n^{2+r})=\tilde{O}(n^{2.25})$, for $r=0.25$.

Since both Theorem 3.2 and Dijkstra are combinatorial, the final result is combinatorial. □

3.3 $(1+\epsilon,k)$-Approximate APSP for Unweighted Graphs As a third application of Theorem 3.1, we give an algorithm for computing $(1+\epsilon,k)$-approximate APSP for even $k\geq 2$. Hereto, we use the following result of Dor, Halperin, and Zwick [DHZ00], which gives an efficient algorithm for sparse graphs.

THEOREM 3.5. ([DHZ00]) *i) There exists a deterministic algorithm that, given an unweighted, undirected graph $G=(V,E)$, computes $+2$-approximate APSP in $\tilde{O}(\min\{n^{3/2}m^{1/2},n^{7/3})$ time.*

ii) There exists a deterministic algorithm that, given an unweighted, undirected graph $G=(V,E)$ and an even integer $k\geq 4$, computes $+k$-approximate APSP in $\tilde{O}(\min\{n^{2-\frac{2}{k+2}}m^{\frac{2}{k+2}},n^{2+\frac{2}{3k-2}}\})$ time.

In particular, this gives a $+\log n$-approximation in $\tilde{O}(n^2)$ time. Also note that $n^{3/2}m^{1/2}=n^{2-\frac{2}{k+2}}m^{\frac{2}{k+2}}$ for $k=2$. So we can also say that for even $k\geq 2$ there exists a $+k$-approximate APSP algorithm that runs in $n^{2-\frac{2}{k+2}}m^{\frac{2}{k+2}}$ time.

THEOREM 3.6. *There exists a deterministic algorithm that, given an unweighted, undirected graph $G=(V,E)$ and an even integer $k\geq 2$, computes $(1+\epsilon,k)$-approximate APSP with running time $\tilde{O}\left(n^{2+(1-r)\frac{2}{k+2}}+n^{\omega(r)}(1/\epsilon)\right)$, for any choice of $r\in[0,1]$.*

Proof. Again, we use Theorem 3.1. For algorithm $\mathcal{A}$, we use the result of Theorem 3.5 for sparse graphs. We apply it to a graph with n vertices and n^{2-r} edges, which leads to the running time $\tilde{O}(n^{2+(1-r)\frac{2}{k+2}})$.

For algorithm $\mathcal{B}$, we use $(1+\epsilon/2)$-approximate rectangular matrix multiplication in time $\tilde{O}(n^{\omega(r)})(1/\epsilon)^{O(1)}$.

By Theorem 3.1, for each pair of vertices we either obtain a stretch of $(0,k)$, or a stretch of $(1+\epsilon/2,(1+\epsilon/2)2)$, Note that for any $d(u,v)\geq 2$ we have $(1+\epsilon/2)d(u,v)+(1+\epsilon/2)\cdot 2\leq(1+\epsilon)d(u,v)+2$, so we have a $(1+\epsilon,2)$-approximation hence a $(1+\epsilon,k)$-approximation in total.

Also by Theorem 3.1, we obtain a running time of $\tilde{O}(n^{2+(1-r)\frac{2}{k+2}}+n^{\omega(r)}(1/\epsilon))$, for any choice of $r\in[0,1]$ [Zwi02] (see also Theorem 2.1).

Since both [DHZ00] and [Zwi02] are deterministic, the final result is deterministic. □

To give optimal results, we pick k as a function of r. Since there is no closed form for (the state of the art of) $\omega(r)$, we balance it for specific k using [Bra].

$k=2$: we have $\tilde{O}(n^{2+(1-r)/2}+n^{\omega(r)}(1/\epsilon))=\tilde{O}(n^{2.15195331}/\epsilon)$, for $r=0.69609339$.

$k = 4$: we have $\tilde{O}(n^{2+(1-r)/3} + n^{\omega(r)}(1/\epsilon)) = \tilde{O}(n^{2.11900756}/\epsilon)$, for $r = 0.64297733$.

$k = 6$: we have $\tilde{O}(n^{2+(1-r)/4} + n^{\omega(r)}(1/\epsilon)) = \tilde{O}(n^{2.0981921}/\epsilon)$, for $r = 0.60723159$.

$k = 8$: we have $\tilde{O}(n^{2+(1-r)/5} + n^{\omega(r)}(1/\epsilon)) = \tilde{O}(n^{2.08383115}/\epsilon)$, for $r = 0.58084427$.

$k \geq 10$: The exponent will go to 2, as k goes to $\log n$. However, for $k \geq 10$, we do not improve upon the results of Dor, Halperin, and Zwick [DHZ00].

k	This work	Previous results (for $m = n^2$)
2	$n^{2.152}$	$n^{2.24+o(1)}$ [BK07]
4	$n^{2.119}$	$n^{2.2}$ [DHZ00]
6	$n^{2.098}$	$n^{2.125}$ [DHZ00]
8	$n^{2.084}$	$n^{2.091}$ [DHZ00]

Table 3: Comparison of our $(1+\epsilon, k)$-approximation and prior work.

Discussion. For future work, a further possibility would be to use a $+k$-approximation for Algorithm $\mathcal{B}$ in Theorem 3.1, which in total gives a $+(k+2)$-approximation. However, that requires an efficient $+k$-approximation for MSSP, to the best of our knowledge no such algorithm exists at this time. Note that the $+k$ algorithm of [DHZ00] is APSP and does not give a faster MSSP. Hence reducing $+(k+2)$ to $+k$ with that algorithm is only *slower*.

4 Weighted $(2+\epsilon)$-Approximate APSP

The techniques of the previous section do not generalize well to the weighted setting. In this section we present an alternative approach for weighted graphs. First, we review some standard definitions and results on bunches and clusters in Section 4.1. Then we continue with two different approaches to some subroutines to obtain efficient algorithms for sparse and dense graphs, in Section 4.3 and Section 4.4 respectively. Finally, we show that we can generalize the latter result to obtain faster algorithms in a wider density range in Section 4.5. Moreover, in Appendix B, we provide a faster algorithm for a $(2, W)$-approximation. This can be seen as an adaptation of Section 4.3.

4.1 Bunches and Clusters

We use the concepts bunches and clusters as defined by Thorup and Zwick [TZ05]. Given a parameter $p \in [\frac{1}{n}, 1]$, called the *cluster sampling rate*, we define S to be a set of sampled vertices, where each vertex is part of S with independent probability p. The *pivot* $p(v)$ of a vertex v is defined as the closest vertex s in S, i.e., $p(v) := \arg\min_{s \in S} d(s, v)$. With equal distances, we can break ties arbitrarily. Now we define the bunch $B(u)$ of u by $B(u) := \{v \in V : d(u, v) < d(u, p(u)\}$. If the set S is not clear from context, we write $B(u, S)$. Clusters are the inverse bunches: $C(v) := \{u \in V : d(u, v) < d(u, p(u)\} = \{u \in V : v \in B(u)\}$. By our random choice of S, we have with high probability that $|B(u)| \leq O(\frac{\log n}{p})$.

Thorup and Zwick [TZ05] showed how to compute this efficiently, in $\tilde{O}(\frac{m}{p})$ time. It is immediate that the *total* load of the bunches and clusters is the same: $\sum_{u \in V} |B(u)| = \sum_{v \in V} |C(v)|$. Hence the *average* size of each cluster is $\tilde{O}(\frac{1}{p})$. However, the maximal load over all clusters can still be big. In a later work, Thorup and Zwick [TZ01] showed that by refining the set S, we can bound the bunch and cluster sizes simultaneously.

Lemma 4.1. ([TZ01],[TZ05]) *There exists an algorithm* ComputeBunches(G, p) *that, given a weighted graph* G *and a parameter* $p \in [\frac{1}{n}, 1]$, *computes*

- *a set of vertices* $S \subseteq V$ *of size* $\tilde{O}(pn)$;
- *pivots* $p(u) \in A$ *such that* $d(u, p(u)) = d(u, S)$ *for all* $u \in V$;
- *the bunches and clusters with respect to* S;
- *distances* $d(u, v)$ *for all* $u \in V$ *and* $v \in B(u) \cup \{p(u)\}$.

With high probability, we have that both the bunches and clusters have size at most $\tilde{O}(\frac{1}{p})$. *The algorithm runs in time* $\tilde{O}(\frac{m}{p})$.

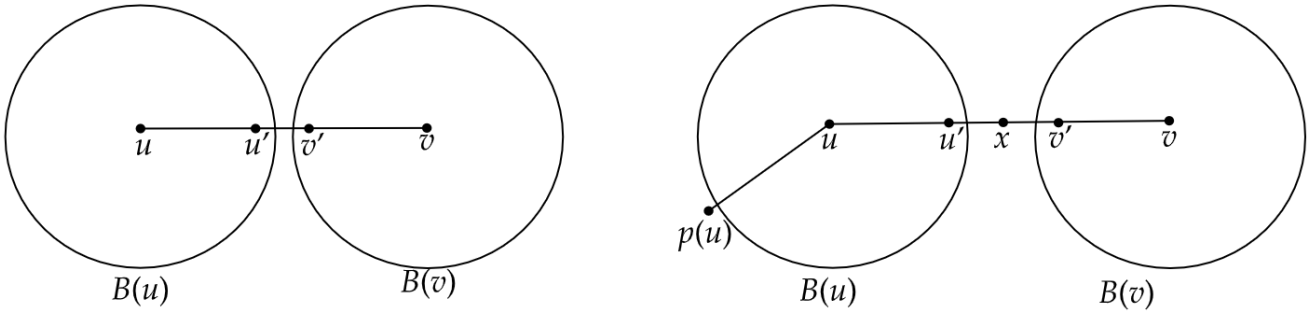

Figure 4: Two different possible interactions between the shortest path between u and v, and the bunches of u and v.

4.2 Structure of the Algorithm At a high-level, our algorithm is as follows. We compute a suitable set of centers with bounded bunch and cluster size using Lemma 4.1. Then for each pair of vertices whose bunches are adjacent (we depict this 'adjacent case' on the left in Figure 4) we store a distance estimate through the edge connecting them. This case also includes the case that the bunches overlap or even when $u \in B(v)$. In both cases there is still an edge connecting the bunches, it is just contained *within* one of the bunches already. We show that we maintain the exact distance in the adjacent case, and otherwise we give a 2-approximation by going through the pivot. We provide pseudo-code in Algorithm 2.

Algorithm 2: A $(2+\epsilon)$-approximate APSP algorithm for unweighted graphs

Input: Unweighted graph $G = (V, E)$, parameter $p \in [\frac{1}{n}, 1]$
ComputeBunches(G, p);
Compute $(1+\epsilon/2)$-approximate MSSP from S, denoted by δ_S;
Compute $\delta_{\text{adjacent}}(u, v) = \min\{d(u, u') + w(u', v') + d(v', v) : \{u', v'\} \in E, u' \in B(u), v' \in B(v)\}$;
foreach $u, v \in V$ **do**
  $\delta(u, v) \leftarrow \min\{d(u, p(u)) + \delta_S(p(u), v), d(v, p(v)) + \delta_S(p(v), u), \delta_{\text{adjacent}}(u, v)\}$
end
return δ

Lemma 4.2. *The distance estimate δ returned by Algorithm 2 satisfies $d(u, v) \le \delta(u, v) \le (2+\epsilon)d(u, v)$ for every $u, v \in V$.*

Proof. First, note that $\delta_S(x, y) \ge d(x, y)$ for any $x, y \in V$, and all other distance estimates making up $\delta(u, v)$ correspond to actual paths in the graph, hence $d(u, v) \le \delta(u, v)$. Next, let π be the shortest path from u to v. We distinguish two cases.

Case 1. There exists $x \in \pi$ such that $x \notin (B(u) \cup B(v))$ (the right case in Figure 4).

Since $x \notin B(u)$, we have $d(u, x) \ge d(u, p(u))$, and since $x \notin B(v)$, we have $d(x, v) \ge d(v, p(v))$. Because $d(u, x) + d(x, v) = d(u, v)$, we have either $d(u, x) \le \frac{d(u,v)}{2}$ or $d(x, v) \le \frac{d(u,v)}{2}$. Without loss of generality, assume $d(u, x) \le \frac{d(u,v)}{2}$. Then we have:

$$\begin{aligned}
\delta(u, v) &\le d(u, p(u)) + \delta_S(v, p(u)) \\
&\le d(u, p(u)) + (1+\epsilon/2)d(v, p(u)) \\
&\le d(u, p(u)) + (1+\epsilon/2)(d(u, p(u)) + d(u, v)) \\
&\le (2+\epsilon/2)d(u, x) + (1+\epsilon/2)d(u, v) \\
&\le (2+\epsilon)d(u, v),
\end{aligned}$$

where the third inequality holds by the triangle inequality.

Case 2. There is no $x \in \pi$ such that $x \notin (B(u) \cup B(v))$ (the left case in Figure 4).
In other words, $\pi \subseteq (B(u) \cup B(v))$. This means there are vertices $u' \in B(u)$ and $v' \in B(v)$ such that $\{u', v'\}$ is an edge on the shortest path. Note that there is always at least one such edge, since $u \ne v$ and we allow $u' = u$ and $v' = v$. Since

$$\delta_{\text{adjacent}}(u, v) = \min\{d(u, u') + w(u', v') + d(v', v) : u' \in B(u) \text{ and } v' \in B(v)\},$$

we get $\delta(u,v) = \delta_{\text{adjacent}}(u,v) = d(u,v)$ in this case.
□

In the following two sections, we give two different approaches to execute Algorithm 2, giving efficient algorithms for sparse and dense graphs respectively.

4.3 An Efficient Distance Oracle for Sparse Graphs In this section, we provide an efficient algorithm for 2-approximate APSP in sparse graphs. We use the structure of the previous section and use combinatorial subroutines to obtain a combinatorial algorithm. Note that as opposed to most of our other results (with an exception of Appendix B), this is a *distance oracle* and not explicit APSP. For $m = \tilde{O}(n)$, our bounds match the conditional lower bound of $\tilde{\Omega}(m^{5/3})$ preprocessing time for 2-approximations, conditional on the set intersection conjecture [PRT12] or the 3-SUM conjecture [ABF23], where [PRT12] shows the stronger $\tilde{\Omega}(m^{5/3})$ *space* lower bound.

THEOREM 4.1. *There exists a combinatorial algorithm that, given a weighted graph $G = (V, E)$, constructs a distance oracle that answers 2-approximate distance queries in constant time, and uses $\tilde{O}(mn^{2/3})$ space with preprocessing time $\tilde{O}(mn^{2/3})$.*

Proof. Algorithm details and running time. Let $p \in [\frac{1}{n}, 1]$ be a parameter to be set later. We use Algorithm 2, below we specify certain steps and the running time of this algorithm.

- line 1 takes $\tilde{O}(\frac{m}{p})$ time by Lemma 4.1.
- For line 2, we use Dijkstra in $O(m|S|)$ time to obtain exact shortest paths ($\epsilon = 0$). By Lemma 4.1 we have that $|S| = \tilde{O}(pn)$, hence this takes $\tilde{O}(pnm)$ time.
- By definition of bunches and clusters we have $u' \in B(u) \iff u \in C(u')$, so we can compute line 3 as follows: for all $\{u', v'\} \in E$, for all $u \in C(u')$, and for all $v \in C(v')$:
  1. Initialize $\delta_{\text{adjacent}}(u,v) \leftarrow d(u,u') + w(u',v') + d(v',v)$ if no such entry exists.
  2. Otherwise: $\delta_{\text{adjacent}}(u,v) \leftarrow \min\{\delta_{\text{adjacent}}(u,v), d(u,u') + w(u',v') + d(v',v)\}$.

  We note that Step 1 and 2 can both be done in constant time, so line 3 takes
  $$\sum_{\{u',v'\}\in E} \sum_{u\in C(u')} \sum_{v\in C(v')} O(1) = \sum_{\{u',v'\}\in E} O(|C(u')| \cdot |C(v')|) = \sum_{\{u',v'\}\in E} \tilde{O}(\tfrac{1}{p^2}),$$
  where the last equality holds by Lemma 4.1. This means it takes $\tilde{O}(\frac{m}{p^2})$ time in total.
- Instead of executing the for-loop of line 4, we execute line 5 in the query. This clearly takes constant time for a fixed pair $u, v \in V$.

Together we obtain a running time of $\tilde{O}(\frac{m}{p} + pnm + \frac{m}{p^2}) = \tilde{O}(pnm + \frac{m}{p^2})$. Balancing $pnm = \frac{m}{p^2}$ gives $p = n^{-1/3}$, so total running time $\tilde{O}(mn^{2/3})$.

Correctness. This holds by Lemma 4.2. As detailed above, we have $\epsilon = 0$, obtaining a 2-approximation.

Space. We need $O(|S|n) = \tilde{O}(pn^2)$ space for the distances from S, and $\tilde{O}(\frac{m}{p^2})$ space for the adjacent data structure. All other space requirements are clearly smaller. Inserting $p = n^{-1/3}$, we obtain total space requirement $\tilde{O}(mn^{2/3} + n^{5/3}) = \tilde{O}(mn^{2/3})$. □

4.4 $(2+\epsilon)$-Approximate APSP for Dense Graphs In this section, we provide an efficient algorithm for $(2+\epsilon)$-approximate APSP for dense graphs. In this case, by dense we mean $m = n^2$. The algorithm of this section already improve on the state of the art for a wider range of m, but we defer the case of $m = o(n^2)$ to Section 4.5, where we obtain better results for this regime.

THEOREM 4.2. *There exists a randomized algorithm that, given an undirected graph $G = (V, E)$ with non-negative integer weights bounded by W, computes $(2+\epsilon)$-approximate APSP. With high probability the algorithm takes $O(n^{3-r} + n^{\omega(r)}(1/\epsilon)^{O(1)} \log W)$ time, for any $r \in [0,1]$. Using known upper bounds for rectangular matrix multiplication, this is $O(n^{2.214}(1/\epsilon)^{O(1)} \log W)$ time.*

Proof. Again, we will use a parameter $p \in [\frac{1}{n}, 1]$. For ease of notation, we let $r \in [0,1]$ such that $p = n^{r-1}$. We use Algorithm 2, below we specify certain steps and the running time of this algorithm. *Algorithm details and running time.*

- line 1 takes $\tilde{O}(\frac{m}{p})$ time by Lemma 4.1.
- For line 2, we use [EN22] this takes time $\tilde{O}(m^{1+o(1)} + n^{\omega(r)}(1/\epsilon)^{O(1)} \log W)$.
- We execute line 3 as follows, in $\tilde{O}(\frac{n^2}{p})$ time:
  a) For all $v' \in V$, run Dijkstra on the graph $G_{v'} = (V, E_{v'})$, where $E_{v'}$ consists of all edges incident to v', and for each vertex it contains edges connecting the vertex to the bunch. More formally, $E_{v'} := \{\{v', x\} \in E : x \in V\} \cup \bigcup_{x \in V} \{\{x, y\} : y \in B(x)\}$, with weights $w_{G_{v'}}(v', x) := w_G(v', x)$ and $w_{G_{v'}}(x, y) := d_G(x, y)$ respectively. This second term has size $\tilde{O}(\frac{n}{p})$, since each bunch has size $\tilde{O}(\frac{1}{p})$. Hence computing SSSP with Dijkstra takes $\tilde{O}(\frac{n}{p} + n) = \tilde{O}(\frac{n}{p})$ time.
  b) For all $u \in V$, run Dijkstra on the graph $G'_u := (V, E'_u)$, where E'_u consists of the distances computed in the previous step and again the edges between vertices and their bunch. More formally, $E'_u := (\{u\} \times V) \cup \bigcup_{x \in V} \{\{x, y\} : y \in B(x)\}$, with weights $w_{G'_u}(u, v') := d_{G_{v'}}(v', u)$ and $w_{G'_u}(x, y) := d_G(x, y)$ respectively. For each u, the graph G'_u has $\tilde{O}(\frac{n}{p})$ edges, hence computing shortest paths takes $\tilde{O}(\frac{n}{p})$ time using Dijkstra. Denote the output of this step by $\delta_{\text{adjacent}}(u, v)$.

  Correctness. From Step a we obtain an edge $\{u, v'\}$ for each $v' \in V$ such that there exists $u' \in B(u)$ with $\{u', v'\} \in E$. This edge has weight $w(u, v') = \min_{u' \in B(u)} d(u, u') + w(u', v')$. By Step b we combine this with the shortest path from v' to v for $v' \in B(v)$. So in total we get $\delta_{\text{adjacent}}(u, v) = \min\{d(u, u') + w(u', v') + d(v', v) : \{u', v'\} \in E, u' \in B(u), v' \in B(v)\}$.
- The for-loop of line 4 takes n^2 time.

In total we have $\tilde{O}(\frac{n^2}{p} + n^{\omega(r)}(1/\epsilon)^{O(1)} \log W) = \tilde{O}(n^{3-r} + n^{\omega(r)}(1/\epsilon)^{O(1)} \log W)$. For $r = 0.7868671417236328$, we get $\tilde{O}(n^{2.21313612}(1/\epsilon)^{O(1)} \log W) = O(n^{2.214}(1/\epsilon)^{O(1)} \log W)$, using [Bra].

Correctness. By Lemma 4.2 we obtain $(2+\epsilon)$-approximate APSP. ∎

4.5 A Parameterized APSP Algorithm for Weighted Graphs This section generalizes the 2-approximation of Baswana and Kavitha [BK10], such that there is a parameter in the running time controlling from how many sources we need to compute shortest paths. We then use fast matrix multiplication results to compute MSSP [EN22] to do this part efficiently, and balance the parameters. We follow the algorithms and proofs of [BK10], making adjustments where necessary.

We start by creating a hierarchy of $k = (1-r)\log n$ sets: $V = S_0 \supseteq S_1 \supseteq \cdots \supseteq S_k$. We refer to this as an *r-hierarchy*. For notational purposes we also have a set S_{k+1}, which we define to be the empty set: $S_{k+1} = \emptyset$. We create these by starting with a hierarchy created by subsampling: we start with all vertices $S_0 := V$ and each subsequent $S'_i \subseteq S'_{i-1}$ is created by selecting each element of S'_{i-1} with probability $\frac{1}{2}$. Now $|S'_k| = \tilde{O}(n \cdot (\frac{1}{2})^{(1-r)\log n}) = \tilde{O}(n^r)$ w.h.p. This creates $V = S'_0 \supseteq S'_1 \supseteq \cdots \supseteq S'_k$. Now we use Lemma 4.1 to compute a set of pivots S of size $|S| = \tilde{O}(n^r)$, and we set $S_i = S'_i \cup S$. By adding the set S, we guarantee that the size of each cluster is bounded by at most $\tilde{O}(n^{1-r})$, by Lemma 4.1. We note that w.h.p. $|S_i| \leq |S'_i| + |S| = \tilde{O}(\frac{n}{2^i}) + \tilde{O}(n^r) = \tilde{O}(\frac{n}{2^i})$. In particular, the last set of sources S_k as size $|S_k| \leq |S'_k| + |S| = \tilde{O}(n^r)$. Using Lemma 4.1, it is easy to see that we can compute an r-hierarchy in $\tilde{O}(mn^{1-r})$ time w.h.p., including pivots, bunches and clusters for each S_i.

Later, we compute shortest paths from the last level S_k, and show that either this gives a 2-approximation, or that we can obtain a 2-approximation through the lower levels. The latter is done with Algorithm 3. Here, for each level, we compute shortest paths from S_i in a sparser graph, where the distances to some essential vertices equal the distances in the original graph. This suffices for correctness: on at least one of the levels, a distance estimate through the pivot of that level gives a 2-approximation, see Lemma 4.3.

Notation. We denote $d(v, A)$ for the distance from a vertex $v \in V$ to a set $A \subseteq V$, i.e., $d(v, A) := \min_{u \in A} d(v, u)$. Next, we define $E_A(v)$ as the set of edges with weight less than $d(v, A)$, i.e., $E_A(v) := \{\{u, v\} \in E : w(u, v) \leq d(v, A)\}$. And finally, we define $E_A := \sum_{v \in V} E_A(v)$. We recall that we use $B(u, A)$ to denote the bunch of u w.r.t. some set of pivots A.

Algorithm 3: A subroutine for computing APSP [BK10, Algorithm 8]

Input: An r-hierarchy $V = S_0 \supseteq S_1 \supseteq \cdots \supseteq S_k \supseteq S_{k+1} = \emptyset$, for $k = (1-r)\log n$
foreach $i \in [0,k]$ *and* $u \in V$ **do**
  Compute pivot $p_i(u)$ of u w.r.t. S_i;
  $\delta(u, p_i(u)) \leftarrow d(u, p_i(u))$
end
foreach $u \in V \setminus S_k$ **do**
  Compute $B(u, S_k)$;
  foreach $x \in B(u, S_k)$ *and* $0 \le i \le k$ **do**
    foreach *neighbor* y *of* x **do**
      $\delta(p_i(u), y) \leftarrow \min\{\delta(p_i(u), y), \delta(u, p_i(u)) + \delta(u, x) + w(x, y)\}$
    end
  end
end
foreach $i \in [0, k-1]$ *and* $s \in S_i$ **do**
  Run Dijkstra from s on $(V, E_{S_{i+1}} \cup \{s\} \times V)$, with $w(s,v) = \delta(s,v)$, and update $\delta(s,v)$ for all $v \in V$ accordingly
end
return δ

Independent of our choice of r, this algorithm ensures a 2-approximation for certain vertices.

Lemma 4.3. *[[BK10] Theorem 7.3] Let $u, v \in V$ be any two vertices, and let δ be the output of Algorithm 3. If $d(u, p_k(u)) + d(v, p_k(v)) > d(u,v)$, then*

$$\min_{0 \le i \le k} \{\delta(u, p_i(u)) + \delta(p_i(u), v), \delta(v, p_i(v)) + \delta(p_i(v), u)\} \le 2d(u,v).$$

This holds for any choice of $r \in [0,1]$.

The running time however does depend on r. By computing shortest paths on the sparser graph, we avoid the expensive computation of shortest paths from S_i to all of V. Instead, for each level, we require $\tilde{O}(mn^{1-r})$ time to construct the sparser graph, and $\tilde{O}(n^2)$ time to compute shortest paths from S_i.

Lemma 4.4. *[[BK10] Lemma 7.4] Given $r \in [0,1]$, Algorithm 3 takes $\tilde{O}(mn^{1-r} + n^2)$ time w.h.p.*

Next, we combine this subroutine with shortest paths from the last level to obtain a 2-approximation, see Algorithm 4. This corresponds to Algorithm 9 of [BK10], where we parameterize the hierarchy.

Algorithm 4: A 2-approximate APSP algorithm

$k \leftarrow (1-r)\log n$;
Compute an r-hierarchy $V = S_0 \supseteq S_1 \supseteq \cdots \supseteq S_k \supseteq S_{k+1} = \emptyset$;
Run Algorithm 3 w.r.t. $S_0, S_1, \ldots, S_k, S_{k+1} = \emptyset$;
Compute $(1+\epsilon/2)$-approximate MSSP from S_k in (V, E)
foreach $u, v \in V$ **do**
  if $v \notin B(u, S_k)$ *and* $u \notin B(v, S_k)$ **then**
    $\delta(u,v) \leftarrow \min_{0 \le i \le k}\{\delta(u, p_i(u)) + \delta(p_i(u), v), \delta(v, p_i(v)) + \delta(p_i(v), u)\}$
  end
end
return δ

We show that independent of our choice for r, this gives a 2-approximation. In case $\epsilon = 0$, this is [BK10, Lemma 7.5], we adapt this proof to allow for approximate shortest paths.

Lemma 4.5. *Algorithm 4 computes $(2+\epsilon)$-approximate APSP for any choice of $r \in [0,1]$.*

Proof. First of all, if $u \in B(v, S_k)$ or $v \in B(u, S_k)$ the exact distance is known by $\delta(u,v)$. So for the rest of the proof we assume otherwise. Now if $d(u, p_k(u)) + d(v, p_k(v)) > d(u,v)$, then we obtain a 2-approximation by Lemma 4.3.

We are left with the case that $d(u,p_k(u))+d(v,p_k(v)) \le d(u,v)$. Without loss of generality, let $d(u,p_k(u)) \le d(u,v)/2$. By Algorithm 3, we have $\delta(u,p_k(u)) = d(u,p_k(u))$. Furthermore, by line 4, we have $\delta(p_k(u),v) \le (1+\epsilon/2)d(p_k(u),v)$. In total we obtain by line 5 and the triangle inequality that

$$\begin{aligned}\delta(u,v) &\le \delta(u,p_k(u)) + \delta(p_k(u),v) \le d(u,p_k(u)) + (1+\epsilon/2)d(p_k(u),v)\\ &\le d(u,p_k(u)) + (1+\epsilon/2)d(u,p_k(u)) + (1+\epsilon/2)d(u,v) \le (2+\epsilon)d(u,v).\end{aligned}$$

Since all distance estimates $\delta(u,v)$ correspond to paths in the graph, we trivially have $d(u,v) \le \delta(u,v)$. ∎

Next, we show how the running time depends on r.

Lemma 4.6. *For $r \in [0,1]$, Algorithm 4 takes $\tilde{O}(n^2 + mn^{1-r} + T(\tilde{O}(n^r)))$ time w.h.p., where $T(s)$ is the time to compute $(1+\epsilon)$-approximate MSSP from s sources in a graph with n vertices and m edges.*

Proof. We can compute an r-hierarchy in $\tilde{O}(mn^{1-r})$ time w.h.p. (follows directly from the definition and Lemma 4.1). Algorithm 3 takes $\tilde{O}(mn^{1-r}+n^2)$ time (Lemma 4.4). Next, in line 4, we need to compute MSSP from $|S_k| = \tilde{O}(n^r)$, for which we denote the running time as $T(\tilde{O}(n^r))$. Finally, the for-loop of line 5 takes $O(n^2k) = \tilde{O}(n^2)$ time. Adding all running times, we obtain $\tilde{O}(n^2 + mn^{1-r} + T(\tilde{O}(n^r)))$ time w.h.p. ∎

Together Lemma 4.5 and Lemma 4.6 give Theorem 1.6.

Theorem 4.3. *There exists a randomized algorithm that, given an undirected graph with non-negative edge weights $G=(V,E)$ and parameters $p \in (\frac{1}{n},1]$, $\epsilon \ge 0$, computes $(2+\epsilon)$-approximate APSP. With high probability, the algorithm takes $\tilde{O}(n^2 + m/p + T(\tilde{O}(pn)))$ time, where $T(s)$ is the time to compute $(1+\epsilon)$-MSSP from s sources.*

$(2+\epsilon)$-Approximate APSP for Weighted Graphs Baswana and Kavitha [BK10] proceed by setting $r = 1/2$ (or equivalently $p = 1/\sqrt{n}$). For the MSSP computations they use Dijkstra (hence $\epsilon = 0$) in $\tilde{O}(n^{1-r}m) = \tilde{O}(m\sqrt{n})$ time, see also Theorem 3.2. Instead, we keep r as a parameter, and use fast matrix multiplication to obtain $(1+\epsilon)$-approximate MSSP.

Theorem 4.4. *There exists a randomized algorithm that, given an undirected graph $G=(V,E)$ with non-negative, integer weights bounded by W and a parameter $r \in [0,1]$, computes $(2+\epsilon)$-approximate APSP. With high probability, the algorithm runs in $\tilde{O}(mn^{1-r} + n^{\omega(r)}(1/\epsilon)^{O(1)} \log W)$ time.*

Proof. This follows directly from Theorem 1.6, combined with the $(1+\epsilon)$-approximate MSSP algorithm of [EN22] (see Theorem 2.2). ∎

For dense graphs, i.e., $m = n^2$, we can balance the terms using [Bra]. If we do so, we recover Theorem 1.2. Results for other densities are obtained in a similar fashion, see Table 2 for the results.

Theorem 4.5. *There exists a randomized algorithm that, given an undirected graph $G=(V,E)$ with non-negative integer weights bounded by W, computes $(2+\epsilon)$-approximate APSP. With high probability the algorithm takes $O(n^{3-r} + n^{\omega(r)}(1/\epsilon)^{O(1)} \log W)$ time, for any $r \in [0,1]$. Using known upper bounds for rectangular matrix multiplication, this is $O(n^{2.214}(1/\epsilon)^{O(1)} \log W)$ time.*

Proof. For $m = n^2$, w.h.p. the running time of Theorem 1.3 becomes $\tilde{O}(n^{3-r} + n^{\omega(r)}(1/\epsilon)^{O(1)} \log W) = \tilde{O}(n^{2.21313612}(1/\epsilon)^{O(1)} \log W)$ for $r = 0.78686388$. ∎

References

[ABF23] Amir Abboud, Karl Bringmann, and Nick Fischer. "Stronger 3-SUM Lower Bounds for Approximate Distance Oracles via Additive Combinatorics". In: *Proc. of the 55th Annual ACM Symposium on Theory of Computing (STOC 2023)*. 2023. DOI: 10.48550/arXiv.2211.07058. arXiv: 2211.07058 (cit. on pp. 4, 6, 19).

[ACIM99] Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. "Fast Estimation of Diameter and Shortest Paths (Without Matrix Multiplication)". In: *SIAM J. Comput.* 28.4 (1999). Announced at SODA 1996, pp. 1167–1181. DOI: 10.1137/S0097539796303421. URL: https://doi.org/10.1137/S0097539796303421 (cit. on pp. 3, 5, 7, 13).

[AGM97] Noga Alon, Zvi Galil, and Oded Margalit. "On the Exponent of the All Pairs Shortest Path Problem". In: *J. Comput. Syst. Sci.* 54.2 (1997). Announced at FOCS 1991, pp. 255–262. DOI: 10.1006/jcss.1997.1388. URL: https://doi.org/10.1006/jcss.1997.1388 (cit. on p. 12).

[AGW23] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. "Subcubic Equivalences between Graph Centrality Problems, APSP, and Diameter". In: *ACM Trans. Algorithms* 19.1 (2023). Announced at SODA 2014, 3:1–3:30. DOI: 10.1145/3563393. URL: https://doi.org/10.1145/3563393 (cit. on p. 3).

[AR20] Maor Akav and Liam Roditty. "An almost 2-approximation for all-pairs of shortest paths in subquadratic time". In: *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, (SODA 2020)*. 2020, pp. 1–11. DOI: 10.1137/1.9781611975994.1 (cit. on p. 4).

[AR21] Maor Akav and Liam Roditty. "A Unified Approach for All Pairs Approximate Shortest Paths in Weighted Undirected Graphs". In: *Proceedings of the 29th Annual European Symposium on Algorithms (ESA 2021)*. Vol. 204. 2021, 4:1–4:18. DOI: 10.4230/LIPIcs.ESA.2021.4 (cit. on p. 11).

[AW21] Josh Alman and Virginia Vassilevska Williams. "A refined laser method and faster matrix multiplication". In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2021, pp. 522–539 (cit. on p. 3).

[BGS09] Surender Baswana, Vishrut Goyal, and Sandeep Sen. "All-pairs nearly 2-approximate shortest paths in $O(n^2 \operatorname{polylog} n)$ time". In: *Theoretical Computer Science* 410.1 (2009). Announced at STACS 2005, pp. 84–93. DOI: 10.1016/j.tcs.2008.10.018 (cit. on pp. 4, 6, 9, 28).

[BHGW+21] Thiago Bergamaschi, Monika Henzinger, Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. "New techniques and fine-grained hardness for dynamic near-additive spanners". In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2021, pp. 1836–1855 (cit. on p. 11).

[BK07] Piotr Berman and Shiva Prasad Kasiviswanathan. "Faster approximation of distances in graphs". In: *Algorithms and Data Structures: 10th International Workshop, WADS 2007, Halifax, Canada, August 15-17, 2007. Proceedings 10*. Springer. 2007, pp. 541–552 (cit. on pp. 4, 6, 11, 17).

[BK10] Surender Baswana and Telikepalli Kavitha. "Faster Algorithms for All-pairs Approximate Shortest Paths in Undirected Graphs". In: *SIAM Journal on Computing* 39.7 (2010). Announced at FOCS 2006, pp. 2865–2896. DOI: 10.1137/080737174 (cit. on pp. 3–9, 11, 15, 20–22, 28).

[BN19] Jan van den Brand and Danupon Nanongkai. "Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time". In: *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2019, pp. 436–455 (cit. on p. 11).

[Bra] Jan van den Brand. *Complexity Term Balancer*. www.ocf.berkeley.edu/~vdbrand/complexity/. Tool to balance complexity terms depending on fast matrix multiplication. (cit. on pp. 4, 12, 16, 20, 22).

[BRSW+21] Arturs Backurs, Liam Roditty, Gilad Segal, Virginia Vassilevska Williams, and Nicole Wein. "Toward Tight Approximation Bounds for Graph Diameter and Eccentricities". In: *SIAM J. Comput.* 50.4 (2021). Announced at STOC 2018, pp. 1155–1199. DOI: 10.1137/18M1226737. URL: https://doi.org/10.1137/18M1226737 (cit. on pp. 11, 12).

[CDKL21] Keren Censor-Hillel, Michal Dory, Janne H Korhonen, and Dean Leitersdorf. "Fast approximate shortest paths in the congested clique". In: *Distributed Computing* 34.6 (2021), pp. 463–487 (cit. on p. 9).

[Che14] Shiri Chechik. "Approximate distance oracles with constant query time". In: *Proceedings of the forty-sixth annual ACM symposium on Theory of computing (STOC)*. 2014, pp. 654–663 (cit. on p. 4).

[Che15] Shiri Chechik. "Approximate distance oracles with improved bounds". In: *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*. 2015, pp. 1–10 (cit. on p. 4).

[CLRS94] Thomas H Cormen, Charles Eric Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. Vol. 3. MIT press Cambridge, MA, USA, 1994 (cit. on p. 3).

[Coh00] Edith Cohen. "Polylog-time and near-linear work approximation scheme for undirected shortest paths". In: *Journal of the ACM (JACM)* 47.1 (2000), pp. 132–166 (cit. on p. 6).

[Cop82] Don Coppersmith. "Rapid multiplication of rectangular matrices". In: *SIAM Journal on Computing* 11.3 (1982), pp. 467–471 (cit. on p. 11).

[Cop97] Don Coppersmith. "Rectangular matrix multiplication revisited". In: *Journal of Complexity* 13.1 (1997), pp. 42–49 (cit. on p. 11).

[CW87] Don Coppersmith and Shmuel Winograd. "Matrix multiplication via arithmetic progressions". In: *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. 1987, pp. 1–6 (cit. on p. 3).

[CZ01] Edith Cohen and Uri Zwick. "All-Pairs Small-Stretch Paths". In: *Journal of Algorithms* 38.2 (2001). Announced at SODA 1997, pp. 335–353. DOI: 10.1006/jagm.2000.1117 (cit. on pp. 3–5, 11).

[DFNV22] Michal Dory, Sebastian Forster, Yasamin Nazari, and Tijn de Vos. "New Tradeoffs for Decremental Approximate All-Pairs Shortest Paths". In: *CoRR* abs/2211.01152 (2022). DOI: 10.48550/arXiv.2211.01152. arXiv: 2211.01152. URL: https://doi.org/10.48550/arXiv.2211.01152 (cit. on p. 9).

[DHZ00] Dorit Dor, Shay Halperin, and Uri Zwick. "All-Pairs Almost Shortest Paths". In: *SIAM Journal on Computing* 29.5 (2000). Announced at FOCS 1996, pp. 1740–1759. DOI: 10.1137/S0097539797327908 (cit. on pp. 3–6, 8, 9, 12, 13, 16, 17).

[DJWW22] Mina Dalirrooyfard, Ce Jin, Virginia Vassilevska Williams, and Nicole Wein. "Approximation Algorithms and Hardness for n-Pairs Shortest Paths and All-Nodes Shortest Cycles". In: *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*. IEEE, 2022, pp. 290–300. DOI: 10.1109/FOCS54457.2022.00034. URL: https://doi.org/10.1109/FOCS54457.2022.00034 (cit. on p. 11).

[DKRW+22] Mingyang Deng, Yael Kirkpatrick, Victor Rong, Virginia Vassilevska Williams, and Ziqian Zhong. "New Additive Approximations for Shortest Paths and Cycles". In: *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*. Ed. by Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 50:1–50:10. DOI: 10.4230/LIPIcs.ICALP.2022.50. URL: https://doi.org/10.4230/LIPIcs.ICALP.2022.50 (cit. on pp. 3, 5, 11).

[DP22] Michal Dory and Merav Parter. "Exponentially faster shortest paths in the congested clique". In: *ACM Journal of the ACM (JACM)* 69.4 (2022), pp. 1–42 (cit. on p. 9).

[Dür23] Anita Dürr. "Improved bounds for rectangular monotone min-plus product and applications". In: *Information Processing Letters* (2023), p. 106358 (cit. on pp. 3, 5, 6, 11, 12).

[DWZ23] Ran Duan, Hongxun Wu, and Renfei Zhou. "Faster Matrix Multiplication via Asymmetric Hashing". In: *FOCS 2023* (2023) (cit. on p. 3).

[EGN22] Michael Elkin, Yuval Gitlitz, and Ofer Neiman. "Almost Shortest Paths with Near-Additive Error in Weighted Graphs". In: *18th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2022, June 27-29, 2022, Tórshavn, Faroe Islands*. Ed. by Artur Czumaj and Qin Xin. Vol. 227. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 23:1–23:22 (cit. on p. 6).

[Elk05] Michael Elkin. "Computing almost shortest paths". In: *ACM Transactions on Algorithms (TALG)* 1.2 (2005), pp. 283–323 (cit. on p. 6).

[EN22] Michael Elkin and Ofer Neiman. "Centralized, Parallel, and Distributed Multi-Source Shortest Paths via Hopsets and Rectangular Matrix Multiplication". In: *Proc. of the 39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022*. Vol. 219. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 27:1–27:22. DOI: 10.4230/LIPIcs.STACS.2022.27. arXiv: 2004.07572 (cit. on pp. 8, 10–12, 20, 22).

[Gal12] Francois Le Gall. "Faster algorithms for rectangular matrix multiplication". In: *2012 IEEE 53rd annual symposium on foundations of computer science*. IEEE. 2012, pp. 514–523 (cit. on pp. 4, 11, 28).

[Gal14] Francois Le Gall. "Powers of tensors and fast matrix multiplication". In: *Proceedings of the 39th international symposium on symbolic and algebraic computation*. 2014, pp. 296–303 (cit. on p. 3).

[Gal23] François Le Gall. *Faster Rectangular Matrix Multiplication by Combination Loss Analysis*. 2023. arXiv: 2307.06535 [cs.DS] (cit. on pp. 4, 28).

[GM97] Zvi Galil and Oded Margalit. "All pairs shortest distances for graphs with small integer length edges". In: *Information and Computation* 134.2 (1997), pp. 103–139 (cit. on p. 3).

[GR21] Yong Gu and Hanlin Ren. "Constructing a Distance Sensitivity Oracle in $O(n^{2.5794}M)$ Time". In: *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*. Ed. by Nikhil Bansal, Emanuela Merelli, and James Worrell. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 76:1–76:20. DOI: 10.4230/LIPIcs.ICALP.2021.76. URL: https://doi.org/10.4230/LIPIcs.ICALP.2021.76 (cit. on p. 11).

[GU18] Francois Le Gall and Florent Urrutia. "Improved Rectangular Matrix Multiplication using Powers of the Coppersmith-Winograd Tensor". In: *Proc. of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*. Ed. by Artur Czumaj. SIAM, 2018, pp. 1029–1046. DOI: 10.1137/1.9781611975031.67. arXiv: 1708.05622 (cit. on pp. 3–5, 8, 11, 12, 28).

[HP98] Xiaohan Huang and Victor Y. Pan. "Fast Rectangular Matrix Multiplication and Applications". In: *J. Complex.* 14.2 (1998), pp. 257–299. DOI: 10.1006/jcom.1998.0476. URL: https://doi.org/10.1006/jcom.1998.0476 (cit. on p. 11).

[Kav12] Telikepalli Kavitha. "Faster Algorithms for All-Pairs Small Stretch Distances in Weighted Graphs". In: *Algorithmica* 63.1-2 (2012). Announced at FSTTCS 2007, pp. 224–245. DOI: 10.1007/s00453-011-9529-y (cit. on pp. 3–6, 9, 11).

[Knu17] Mathias Bæk Tejs Knudsen. "Additive Spanners and Distance Oracles in Quadratic Time". In: *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming, (ICALP 2017)*. 2017, 64:1–64:12. DOI: 10.4230/LIPIcs.ICALP.2017.64 (cit. on p. 4).

[KRSV07] Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. "Counting colors in boxes". In: *SODA*. 2007, pp. 785–794 (cit. on p. 11).

[KSV06] Haim Kaplan, Micha Sharir, and Elad Verbin. "Colored intersection searching via sparse rectangular matrix multiplication". In: *Proceedings of the twenty-second annual symposium on Computational geometry*. 2006, pp. 52–60 (cit. on p. 11).

[KZHP08] ShanXue Ke, BenSheng Zeng, WenBao Han, and Victor Y Pan. "Fast rectangular matrix multiplication and some applications". In: *Science in China Series A: Mathematics* 51 (2008), pp. 389–406 (cit. on p. 11).

[LR83] Grazia Lotti and Francesco Romani. "On the asymptotic complexity of rectangular matrix multiplication". In: *Theoretical Computer Science* 23.2 (1983), pp. 171–185 (cit. on p. 11).

[Pet04] Seth Pettie. "A new approach to all-pairs shortest paths on real-weighted graphs". In: *Theoretical Computer Science* 312.1 (2004), pp. 47–74 (cit. on p. 3).

[PR05] Seth Pettie and Vijaya Ramachandran. "A shortest path algorithm for real-weighted undirected graphs". In: *SIAM Journal on Computing* 34.6 (2005), pp. 1398–1431 (cit. on p. 3).

[PR14] Mihai Patrascu and Liam Roditty. "Distance Oracles beyond the Thorup-Zwick Bound". In: *SIAM Journal on Computing* 43.1 (2014). Announced at FOCS 2010, pp. 300–311. DOI: 10.1137/11084128X (cit. on pp. 4, 6).

[PRT12] Mihai Patrascu, Liam Roditty, and Mikkel Thorup. "A New Infinity of Distance Oracles for Sparse Graphs". In: *Proc. of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012)*. IEEE Computer Society, 2012, pp. 738–747. DOI: 10.1109/FOCS.2012.44 (cit. on pp. 4, 6, 19).

[Rod23] Liam Roditty. "New Algorithms for All Pairs Approximate Shortest Paths". In: *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*. Ed. by Barna Saha and Rocco A. Servedio. ACM, 2023, pp. 309–320. DOI: 10.1145/3564246.3585197. URL: https://doi.org/10.1145/3564246.3585197 (cit. on pp. 3–5, 9, 12, 16).

[RS11] Liam Roditty and Asaf Shapira. "All-pairs shortest paths with a sublinear additive error". In: *ACM Transactions on Algorithms (TALG)* 7.4 (2011), pp. 1–12 (cit. on p. 11).

[RZ11] Liam Roditty and Uri Zwick. "On Dynamic Shortest Paths Problems". In: vol. 61. 2. Announced at ESA 2004. 2011, pp. 389–401. DOI: 10.1007/s00453-010-9401-5 (cit. on p. 3).

[Sei95] Raimund Seidel. "On the all-pairs-shortest-path problem in unweighted undirected graphs". In: *Journal of computer and system sciences* 51.3 (1995), pp. 400–403 (cit. on p. 3).

[SM10] Piotr Sankowski and Marcin Mucha. "Fast dynamic transitive closure with lookahead". In: *Algorithmica* 56 (2010), pp. 180–197 (cit. on p. 11).

[Som16] Christian Sommer. "All-Pairs Approximate Shortest Paths and Distance Oracle Preprocessing". In: *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming, (ICALP 2016)*. Vol. 55. 2016, 55:1–55:13. DOI: 10.4230/LIPIcs.ICALP.2016.55 (cit. on p. 4).

[SY23] Barna Saha and Christopher Ye. "Faster Approximate All Pairs Shortest Paths". In: *CoRR* abs/2309.13225 (2023). To appear in SODA'24. DOI: 10.48550/arXiv.2309.13225. arXiv: 2309.13225. URL: https://doi.org/10.48550/arXiv.2309.13225 (cit. on p. 7).

[Tho99] Mikkel Thorup. "Undirected single-source shortest paths with positive integer weights in linear time". In: *Journal of the ACM (JACM)* 46.3 (1999), pp. 362–394 (cit. on p. 3).

[TZ01] Mikkel Thorup and Uri Zwick. "Compact routing schemes". In: *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA 2001*. Ed. by Arnold L. Rosenberg. ACM, 2001, pp. 1–10. DOI: 10.1145/378580.378581. URL: https://doi.org/10.1145/378580.378581 (cit. on pp. 9, 17).

[TZ05] Mikkel Thorup and Uri Zwick. "Approximate distance oracles". In: *Journal of the ACM (JACM)* 52.1 (2005), pp. 1–24 (cit. on pp. 4, 9, 17).

[Vas15] Virginia Vassilevska Williams. "Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk)". In: *10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2015 (cit. on p. 3).

[Vas18] Virginia Vassilevska Williams. "On some fine-grained questions in algorithms and complexity". In: *Proceedings of the ICM*. Vol. 3. World Scientific. 2018, pp. 3431–3472 (cit. on p. 3).

[VFN22] Jan Van Den Brand, Sebastian Forster, and Yasamin Nazari. "Fast deterministic fully dynamic distance approximation". In: *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2022, pp. 1011–1022 (cit. on p. 11).

[Wil12] Virginia Vassilevska Williams. "Multiplying matrices faster than Coppersmith-Winograd". In: *Proceedings of the forty-fourth annual ACM symposium on Theory of computing (STOC)*. 2012, pp. 887–898 (cit. on p. 3).

[Wil14] Ryan Williams. "Faster all-pairs shortest paths via circuit complexity". In: *Proceedings of the forty-sixth annual ACM symposium on Theory of computing (STOC)*. 2014, pp. 664–673 (cit. on p. 3).

[Wul12] Christian Wulff-Nilsen. "Approximate distance oracles with improved preprocessing time". In: *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms (SODA)*. SIAM. 2012, pp. 202–208 (cit. on p. 4).

[Wul13] Christian Wulff-Nilsen. "Approximate distance oracles with improved query time". In: *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms (SODA)*. SIAM. 2013, pp. 539–549 (cit. on p. 4).

[WW18] Virginia Vassilevska Williams and R Ryan Williams. "Subcubic equivalences between path, matrix, and triangle problems". In: *Journal of the ACM (JACM)* 65.5 (2018). Announced at FOCS 2010, pp. 1–38 (cit. on p. 3).

[WWWY14] Virginia Vassilevska Williams, Joshua R Wang, Ryan Williams, and Huacheng Yu. "Finding four-node subgraphs in triangle time". In: *Proceedings of the twenty-sixth annual ACM-SIAM symposium on discrete algorithms (SODA)*. SIAM. 2014, pp. 1671–1680 (cit. on p. 11).

[WX20] Virginia Vassilevska Williams and Yinzhan Xu. "Monochromatic triangles, triangle listing and APSP". In: *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2020, pp. 786–797 (cit. on p. 11).

[WXXZ23] Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. *New Bounds for Matrix Multiplication: from Alpha to Omega*. 2023. arXiv: 2307.07970 [cs.DS] (cit. on pp. 4, 5, 28).

[Yus09] Raphael Yuster. "Efficient algorithms on sets of permutations, dominance, and real-weighted APSP". In: *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2009, pp. 950–957 (cit. on p. 11).

[YZ04] Raphael Yuster and Uri Zwick. "Detecting short directed cycles using rectangular matrix multiplication and dynamic programming." In: *SODA*. Vol. 4. 2004, pp. 254–260 (cit. on p. 11).

[Zwi02] Uri Zwick. "All pairs shortest paths using bridging sets and rectangular matrix multiplication". In: *J. ACM* 49.3 (2002), pp. 289–317. DOI: 10.1145/567112.567114. URL: https://doi.org/10.1145/567112.567114 (cit. on pp. 3, 5, 8, 11, 12, 15, 16).

[Zwi98] Uri Zwick. "All pairs shortest paths in weighted directed graphs-exact and almost exact algorithms". In: *Proceedings 39th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 1998, pp. 310–319 (cit. on p. 3).

A Utilizing Recent Improvements on Rectangular Matrix Multiplication

Rectangular matrix multiplication is an active research field, with the bounds on $\omega(r)$ being improved in recent years [Gal23, GU18, Gal12]. Throughout this paper, we used [GU18], the last published paper on the topic, for the sake of replicability. However, more recent, concurrent work by Vassilevska Williams, Xu, Xu, and Zhou [WXXZ23] gives better bounds. In this section, we detail how this affects our running times.

Our result for 2-approximate APSP in unweighted graphs, Theorem 1.1, has running time $\tilde{O}(n^{2.5-r} + n^{\omega(r)}) = O(n^{2.031062336})$, for $r = 0.4689376644$.

Our results for $(2+\epsilon)$-approximate APSP in weighted graphs, Theorem 1.3, are given in Table 4.

m	Runnning time	Using [WXXZ23] for $\omega(r)$	with r
$n^{1.4}$	$n^{2.4-r} + n^{\omega(r)}$	$n^{2.008199835}$	0.3918001650
$n^{1.5}$	$n^{2.5-r} + n^{\omega(r)}$	$n^{2.031062336}$	0.4689376644
$n^{1.6}$	$n^{2.6-r} + n^{\omega(r)}$	$n^{2.061029532}$	0.5389704676
$n^{1.7}$	$n^{2.7-r} + n^{\omega(r)}$	$n^{2.095342149}$	0.6046578512
$n^{1.8}$	$n^{2.8-r} + n^{\omega(r)}$	$n^{2.132619229}$	0.6673807708
$n^{1.9}$	$n^{2.9-r} + n^{\omega(r)}$	$n^{2.171770761}$	0.7282292393
$n^{2.0}$	$n^{3-r} + n^{\omega(r)}$	$n^{2.212352011}$	0.7876479892

Table 4: Our $(2+\epsilon)$-approximate APSP results (Theorem 1.3) using [WXXZ23], for $1/\epsilon = n^{o(1)}$. Theorem 1.3 gives the fastest running time when $m \geq n^{1.544}$. For $m < n^{1.544}$, we do not improve on [BK10].

Our results for near-additive APSP in unweighted graphs, Theorem 1.5, are given in Table 5.

k	Runnning time	Using [WXXZ23] for $\omega(r)$	with r
2	$n^{2+(1-r)/2} + n^{\omega(r)}$	$n^{2.151353127}$	0.6972937458
4	$n^{2+(1-r)/3} + n^{\omega(r)}$	$n^{2.118511896}$	0.6444643104
6	$n^{2+(1-r)/4} + n^{\omega(r)}$	$n^{2.097785917}$	0.6088563325
8	$n^{2+(1-r)/5} + n^{\omega(r)}$	$n^{2.083460832}$	0.5826958380

Table 5: Our $(1+\epsilon, k)$-approximate APSP results (Theorem 1.5) using [WXXZ23], for $1/\epsilon = n^{o(1)}$.

B $(2, W_{u,v})$-Approximate APSP

In this section, we prove Theorem B.1, stated below, providing $(2, W_{u,v})$-approximate shortest paths. This is a generalized version of Baswana, Goyal, and Sen [BGS09], who provide $(2,1)$-APSP in *unweighted* graphs in $\tilde{O}(nm^{2/3} + n^2)$ time. We make three improvements in our generalization: 1) the algorithm allows for *weighted* graphs, 2) it achieves subquadratic time for $m \leq n^{3/2}$, since it is a distance oracle rather than explicit APSP, and 3) we achieve a faster running time for $m > n^{3/2}$, by having a wider choice in parameters.

The structure of the algorithm is similar to Section 4.3. We show that for vertices whose bunches not overlap, the path through the pivot is actually a $(2, W_{u,v})$-approximation (even if the bunches are adjacent, top left case in Figure 5). Then for bunches that do overlap (the bottom case in Figure 5), we create a data structure to store these distances.

We note that this result is mostly interesting in the regime where $m = O(n^{3/2})$, where we obtain *subquadratic* time and space. For denser graphs, we do not improve upon Baswana and Kavitha [BK10], who provide a $(2, W_{u,v})$-approximation in $\tilde{O}(n^2)$ time and space.

Theorem B.1. *Given a weighted graph G, we can compute a distance oracle that returns $(2, W_{u,v})$-approximate queries in constant time, where $W_{u,v}$ is the maximum weight on a shortest path from u to v, with one of the following guarantees:*

- *preprocessing time $\tilde{O}(nm^{2/3})$ and uses space $\tilde{O}(nm^{2/3})$, when $m \leq n^{3/2}$, and*
- *preprocessing time $\tilde{O}(mn^{1/2})$ and uses space $O(n^2)$, when $m > n^{3/2}$,*

or it has (optimizing for space) running time $\tilde{O}(mn^{2/3})$ and $\tilde{O}(n^{5/3})$ space.

Proof. Algorithm. Let $p \in [\frac{1}{n}, 1]$ be a parameter, to be chosen later.
`PreProcessing(`G, p`)`:

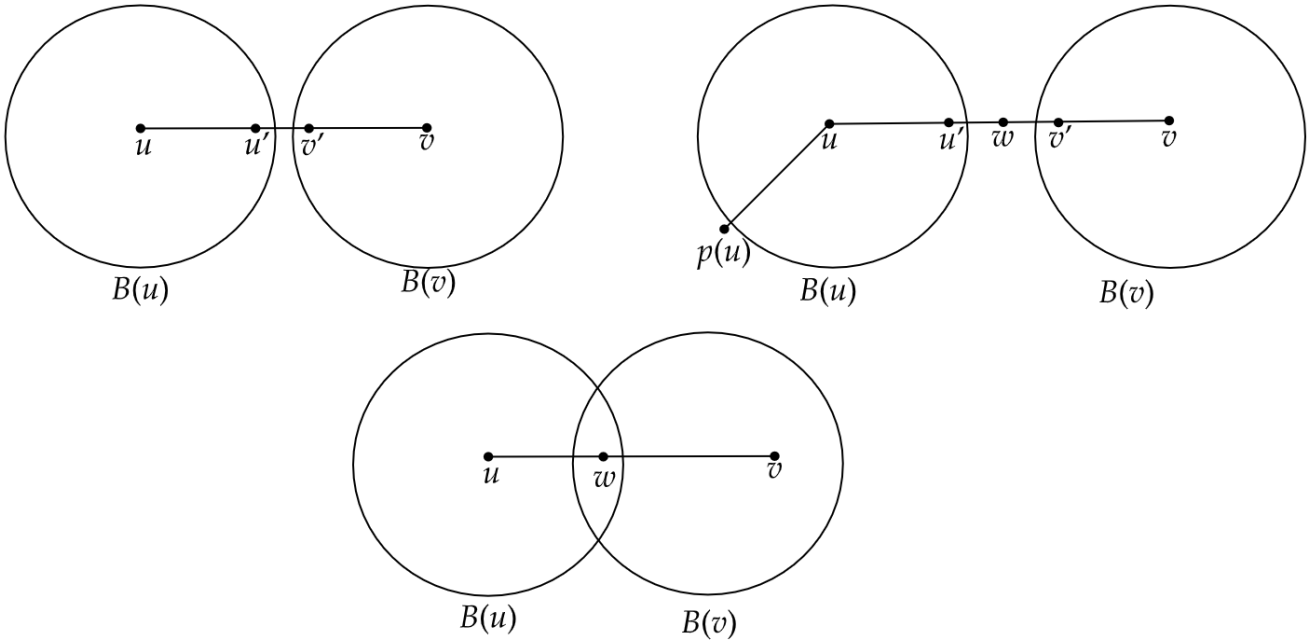

Figure 5: Three different possible interactions between the shortest path between u and v, and the bunches of u and v.

1. Run ComputeBunches(G, p)
2. Run Dijkstra for each vertex $u \in A$ to compute all distances $d(u, v)$ for $u \in A, v \in V$.
3. For $u \in V$, for $w \in B(u)$, for $v \in C(w)$:
   (a) Initialize $\delta_{\text{overlap}}(u, v) \leftarrow d(u, w) + d(w, v)$ if no such entry exists.
   (b) Otherwise: $\delta_{\text{overlap}}(u, v) \leftarrow \min\{\delta_{\text{overlap}}(u, v), d(u, w) + d(w, v)\}$.

`Query`(u, v):
Output $\delta(u, v)$ to be the minimum of

(a) $\min\{d(u, p(u)) + d(v, p(u)), d(u, p(v)) + d(v, p(v))\}$;

(b) $\delta_{\text{overlap}}(u, v)$;

Correctness. We will show that $\delta(u, v)$ gives a $(2, W_{u,v})$-approximation of $d(u, v)$. First, note that all distances making up $\delta(u, v)$ correspond to actual paths in the graph, hence $d(u, v) \leq \delta(u, v)$. Next, let π be the shortest path from u to v. We distinguish two cases.

Case 1. There exists $w \in \pi$ such that $w \in B(u) \cap B(v)$ (the bottom case in Figure 5).

We have that

$$\begin{aligned}\delta_{\text{overlap}}(u, v) &= \min\{d(u, x) + d(x, v) : x \in B(u) \text{ and } y \in C(x)\} \\ &= \min\{d(u, x) + d(x, v) : x \in B(u) \text{ and } x \in B(y)\}.\end{aligned}$$

In particular, this includes $x = w$, and w is on the shortest path, so by Query Step b, we have that $\delta(u, v) \leq d(u, v)$.

Case 2. There is no $w \in \pi$ such that $w \in B(u) \cap B(v)$.

This means there is either a vertex $w \in \pi$ such that $w \notin B(u) \cup B(v)$ (the top right case in Figure 5), or there is an edge $\{u', v'\}$ on π such that $u' \in B(u) \setminus B(v)$ and $v' \in B(v) \setminus B(u)$ (the top right case in Figure 5). The first case gives a 2-approximation by the same reasoning as in the proof of Lemma 4.2, where we use exact shortest path, hence $\epsilon = 0$. Here we only consider the second case. Since $v' \notin B(u)$, we have $d(u, v') \geq d(u, p(u))$, and since $u' \notin B(v)$, we have

$d(u', v) \geq d(v, p(v))$. Combining this, we obtain $d(u, p(u)) + d(v, p(v)) \leq d(u, v) + w(u', v')$. Without loss of generality, assume that $d(u, p(u)) \leq \frac{d(u,v)+w(u',v')}{2}$. By Query Step a we have:

$$\begin{aligned}\delta(u, v) &\leq d(u, p(u)) + d(v, p(u)) \\ &\leq d(u, p(u)) + d(u, p(u)) + d(u, v) \\ &\leq 2d(u, v) + w(u', v') \\ &\leq 2d(u, v) + W_{u,v}.\end{aligned}$$

Running time. Step 1 and 2 take $\tilde{O}(\frac{m}{p})$ and $\tilde{O}(pnm)$ time respectively, see Lemma 4.1. For Step 3, notice that Step 3a and 3b both take constant time. So Step 3 takes total time

$$\sum_{u \in V} \sum_{w \in B(u)} \sum_{v \in C(w)} O(1) = \tilde{O}(\tfrac{n}{p^2}),$$

since $|B(u)| = \tilde{O}(\frac{1}{p})$ and $|C(w)| = \tilde{O}(\frac{1}{p})$, by Lemma 4.1. We obtain total time $\tilde{O}(\frac{m}{p} + pnm + \frac{n}{p^2})$. We can balance this in three different ways:

- $\frac{m}{p} = pnm$, which implies $p = n^{-1/2}$ and gives running time $\tilde{O}(mn^{1/2} + n^2)$.
- $\frac{m}{p} = \frac{n}{p^2}$, which implies $p = \frac{n}{m}$ and gives running time $\tilde{O}(\frac{m^2}{n} + n^2)$.
- $pnm = \frac{n}{p^2}$, which implies $p = m^{-1/3}$ and gives running time $\tilde{O}(m^{4/3} + nm^{2/3})$.

Note that $\tilde{O}(mn^{1/2} + n^2)$ is always smaller than $\tilde{O}(\frac{m^2}{n} + n^2)$, since $mn^{1/2} \leq \frac{m^2}{n}$ for $m \geq n^{3/2}$ and $mn^{1/2} \leq n^2$ for $m \leq n^{3/2}$.

Further we see that $\tilde{O}(m^{4/3} + nm^{2/3})$ is smaller than $\tilde{O}(mn^{1/2} + n^2)$ when $m \leq n^{3/2}$, since then $m^{4/3} \leq n^2$ and $mn^{1/2} \leq n^2$.

Finally we notice that $nm^{2/3} \geq m^{4/3}$ when $m \leq n^{3/2}$, and that $mn^{1/2} \geq n^2$ for $m \geq n^{3/2}$. So our running time simplifies to

- $\tilde{O}(nm^{2/3})$, when $m \leq n^{3/2}$, and
- $\tilde{O}(mn^{1/2})$, when $m > n^{3/2}$.

Query time. We note that we have computed $B(u)$ and $B(v)$, so checking if $v \in B(u)$ or $u \in B(v)$ can be done in constant time. Further, we have also computed $d(u, v)$ if $v \in B(u)$. For Query Step a, notice that we computed distances from A to V, and for Query Step b we have already computed the value $\delta_{\text{overlap}}(u, v)$. So the whole query can be done in constant time.

Space. We need $O(|A|n) = \tilde{O}(pn^2)$ space for the distances from A, and $\tilde{O}(\frac{n}{p^2})$ space for the overlap data structure. All other space requirements are clearly smaller. Using $p = n^{-1/2}$ gives space $O(n^2)$, using $p = m^{-1/3}$ gives space $\tilde{O}(nm^{2/3} + n^2m^{-1/3}) = \tilde{O}(nm^{2/3})$.

To optimize the space usage, we set $pn^2 = \frac{n}{p^2}$, so $p = n^{-1/3}$, we obtain total space requirement $\tilde{O}(n^{5/3}) = \tilde{O}(mn^{2/3})$. This gives running time $\tilde{O}(mn^{1/3} + mn^{2/3} + n^{5/3}) = \tilde{O}(mn^{2/3})$. □