Improved Additive Approximation Algorithms for APSP

Ce Jin* Yael Kirkpatrick† Michał Stawarz‡ Virginia Vassilevska Williams§

Abstract. The All-Pairs Shortest Paths (APSP) is a foundational problem in theoretical computer science. Approximating APSP in undirected unweighted graphs has been studied for many years, beginning with the work of Dor, Halperin and Zwick [SICOMP'00]. Many recent works have attempted to improve these original algorithms using the algebraic tools of fast matrix multiplication. We improve on these results for the following problems.

- For +2-approximate APSP, the state-of-the-art algorithm runs in $O(n^{2.259})$ time [Dürr, IPL 2023; Deng, Kirkpatrick, Rong, Vassilevska Williams, and Zhong, ICALP 2022]. We give an improved algorithm in $O(n^{2.2255})$ time.
- For +4 and +6-approximate APSP, we achieve time complexities $O(n^{2.1462})$ and $O(n^{2.1026})$ respectively, improving the previous $O(n^{2.155})$ and $O(n^{2.103})$ achieved by [Saha and Ye, SODA 2024].

In contrast to previous works, we do not use the big hammer of bounded-difference $(\min,+)$-product algorithms. Instead, our algorithms are based on a simple technique that decomposes the input graph into a small number of clusters of constant diameter and a remainder of low degree vertices, which could be of independent interest in the study of shortest paths problems. We then use only standard fast matrix multiplication to obtain our improvements.

1 Introduction *All-Pairs Shortest Paths (APSP)* is a fundamental problem in computer science: given an edge-weighted graph $G=(V,E)$ with $|V|=n$ vertices, for every pair of vertices $u,v\in V$, compute their distance $d(u,v)$ in the graph G. The textbook Floyd–Warshall algorithm solves APSP in $O(n^3)$ time. The state-of-the-art algorithm by Williams runs in $n^3/2^{\Omega(\sqrt{\log n})}$ time [17]. A central hypothesis in fine-grained complexity asserts that no $O(n^{3-\varepsilon})$-time algorithms can solve APSP in edge-weighted graphs, for any $\varepsilon>0$ (see [16]).

For unweighted graphs, better algorithms for APSP are known. Seidel [14] gave an algorithm for undirected unweighted APSP in $\widetilde{O}(n^\omega)$ time, where $2\le\omega<2.3714$ is the exponent of fast square matrix multiplication [2]. There are also subcubic time APSP algorithms for directed unweighted graphs, and more generally for graphs with small integer weights; see e.g., [3, 15, 18].

In this paper, we focus on *undirected unweighted APSP*. There remains a gap between Seidel's $\widetilde{O}(n^\omega)$ time and the ideal $\widetilde{O}(n^2)$ time complexity (which would be nearly optimal). This gap can be explained by the *Boolean Matrix Multiplication (BMM) hypothesis* from fine-grained complexity, which asserts that multiplying two $n\times n$ matrices over the Boolean semi-ring cannot be solved in $O(n^{\omega-\varepsilon})$ time, for any $\varepsilon>0$[1]. Since APSP in undirected unweighted graphs is known to be at least as hard[2] as BMM [1], this suggests that Seidel's $\widetilde{O}(n^\omega)$ time complexity is likely nearly optimal.

Motivated by this situation, many works in the literature have considered approximate APSP in order to bypass this hardness. In this paper, we focus on *additive approximation*: in an unweighted undirected graph $G=(V,E)$, the *$+C$-APSP* problem asks to compute distance estimates $\tilde{d}(u,v)$ for every pair of vertices $u,v\in V$,

*MIT, cejin@mit.edu, supported by the Jane Street Graduate Research Fellowship, NSF grant CCF-2330048, and a Simons Investigator Award.

†MIT, yaelkirk@mit.edu, supported by NSF Grant No 2141064.

‡ETH Zurich, m.casi321@gmail.com.

§MIT, virgi@mit.edu, supported by NSF Grant CCF-2330048, BSF Grant 2020356 and a Simons Investigator Award.

[1]This hypothesis makes sense if $\omega>2$.

[2]It is in fact equivalent to BMM.

so that $d(u,v) \leq \tilde{d}(u,v) \leq d(u,v)+C$ always holds. It is known that +1-APSP is still as hard as BMM [1], so +2 is the smallest additive error that allows improvement over Seidel's $\widetilde{O}(n^\omega)$ time complexity. Dor, Halperin and Zwick [8] gave a combinatorial algorithm[3] for +2-APSP in $\widetilde{O}(n^{7/3})$ time, which is faster than Seidel's exact APSP algorithm (for the *current* value of ω). Over two decades later, Deng, Kirkpatrick, Rong, Vassilevska Williams, and Zhong [7] improved the time complexity to $O(n^{2.2867})$. Their key idea was to use Euler tours to design a black-box reduction to the bounded-difference $(\min,+)$-product problem, which is known to have sub-cubic time algorithms [5, 6].[4] Dürr [10] further improved the +2-APSP time complexity to $O(n^{2.259})$ by developing faster algorithms for rectangular bounded-difference $(\min,+)$-product.

For +2k-APSP ($k \geq 2$), Dor, Halperin and Zwick [8] gave faster combinatorial algorithms running in $\widetilde{O}(n^{2+\frac{1}{3k-1}})$ time. Saha and Ye [13] improved [8]'s +2k-APSP algorithms by refining the Euler-tour idea of [7] and the original analysis of [8]. (See Table 1.1.)

1.1 New Results In this paper, we give improved algorithms for $+2, +4$, and +6-APSP. In addition to quantitative improvements, our algorithms also simplify previous approaches in the sense that we no longer need to invoke the big hammer of bounded-difference $(\min,+)$-product algorithms. We prove the following two theorems.

THEOREM 1.1. *+2-APSP in an n-node unweighted undirected graph can be solved by a randomized algorithm in $O(n^{2.22548})$ time.*

Our algorithm for Theorem 1.1 uses rectangular matrix multiplication. If one only uses square matrix multiplication, the running time of Theorem 1.1 would be $\widetilde{O}(n^{2+\frac{\omega-1}{\omega+3}})$. Thus, the running time will beat Seidel's $\widetilde{O}(n^\omega)$ time bound as long as $\omega > \sqrt{5} \approx 2.236$.

If $\omega = 2$, our running time would be $\widetilde{O}(n^{2.2})$. In contrast, Dürr's [10] algorithm runs in $\widetilde{O}(n^{2+\frac{\omega-1}{2\omega}})$ time in terms of ω and hence if $\omega = 2$, its running time would be $\widetilde{O}(n^{2.25})$ and thus would still be slower than ours even with optimal matrix multiplication bounds.

THEOREM 1.2. *+2k-APSP in an n-node unweighted undirected graph can be solved by a deterministic algorithm in $O(n^{2+x/(k+1)})$ time, when x is the solution to $1+x = \omega(1-\frac{k-1}{k+1}x, 1-x, \frac{k}{k+1}x)$. (See the definition of the rectangular matrix multiplication exponent $\omega(\cdot,\cdot,\cdot)$ in Section 2.)*

The result statement of Saha and Ye [13] differs from ours only in the equation defining x. Theirs has the form $1+\mathbf{2}x = \omega(1-\frac{k-1}{k+1}x, 1-x, \mathbf{1}-\frac{\mathbf{k-2}}{\mathbf{k+1}}x)$, where the difference from ours appears in boldface.

The result of Theorem 1.2 gives a faster algorithm than the combinatorial +2k-approximation of Dor, Halperin and Zwick [8] for every k. Our algorithm uses only matrix multiplication, without the more complex algebraic tools used in the work of Saha and Ye [13]. However, the running time is faster than the best known one of Saha and Ye only for $k = 2, 3$ (additive +4 and +6 approximation). The results are summarized in Table 1.1, where all running times (ours and prior work) are computed using the code of [4] updated with the newest rectangular matrix multiplication bounds [2].

+2k-Additive Approximation for APSP				
2k	[8] (combinatorial)	[10]	[13]	This work
2	$n^{2+1/3} \leq n^{2.334}$	$n^{2.25899}$		$n^{2.22548}$
4	$n^{2.2}$		$n^{2.15492}$	$n^{2.14613}$
6	$n^{2.125}$		$n^{2.102926}$	$n^{2.102595}$
8	$n^{2+1/11} \leq n^{2.0910}$		$n^{2.077270}$	$n^{2.079072}$

Table 1.1: Comparison with previous results for +2k-APSP.

[3] As is typical in the literature, we informally refer to algorithms that do not use fast matrix multiplication as *combinatorial algorithms*.

[4] More specifically, [7] used fast algorithms that compute the $(\min,+)$-product $C[i,j] = \min_k\{A[i,k]+B[k,j]\}$ of two input integer matrices A, B, where A is column bounded-difference, i.e., $|A[i,j]-A[i+1,j]| \leq O(1)$ for all valid (i,j)'s, and B is row bounded-difference, i.e., $|B[i,j]-B[i,j+1]| \leq O(1)$ for all valid (i,j)'s.

1.2 Further Related Works A closely related problem is *multiplicative* approximate APSP. In unweighted undirected graphs, an $+2$-approximation for APSP automatically yields 2-multiplicative approximation, so it is an easier problem. The fastest known algorithm for 2-multiplicative approximate APSP runs in $O(n^{2.0319})$ time [9, 13], improving the previous $\widetilde{O}(n^{2.25})$-time algorithm by [12]. The main open question in this line of research is to achieve $O(n^{2+o(1)})$ time for 2-multiplicative approximate APSP. See [11] for very recent progress on this question.

1.3 Technical Overview For a few decades, the $+2k$-approximate APSP algorithm of Dor, Halperin, and Zwick [8] was the fastest known additive APSP approximation. The core idea of this algorithm is selecting a series of degree thresholds $1 = d_0 < d_1 < d_2 < \ldots < d_k$ and sampling hitting sets $S_1, \ldots, S_k$ of size $|S_i| = \tilde{O}(n/d_i)$ that hit the neighborhoods of all vertices of degree $\geq d_i$, for every i.

We first compute the distances out of every vertex in the smallest hitting set S_k, to compute a correct distance estimate $\tilde{d}(u,v) = d(u,v)$ for any $u \in S_k, v \in V$. Next, we compute the distances out of every vertex in S_{k-1}. However, we can't afford to use the full edge set, so instead we only search on the edge set E_{k-1}, consisting of edges adjacent to vertices of degree $< d_k$ and an edge from every vertex of degree $\geq d_k$ to a neighbor in S_k. Additionally, when running Dijkstra's algorithm from a vertex u, we include edges from u to every vertex $v \in S_k$ weighted by the current distance estimate computed between them in the previous round.

Now, for any $u \in S_{k-1}, v \in V$, if the shortest path between them contains only vertices of degree $< d_k$, then $\tilde{d}(u,v) = d(u,v)$ since the entire shortest path between them was included in the Dijkstra search. Otherwise, let w be the last vertex on the path from u to v of degree $\geq d_k$. There exists a vertex $s \in S_k$ such that $(s,w) \in E_{k-1}$. Thus, since the path from w to v is also included in E_{k-1}, our search out of u considers the path $u \to s \to w \rightsquigarrow v$ of weight $\tilde{d}(u,s) + 1 + d(w,v)$. Since $s \in S_k$ we have that $\tilde{d}(u,s) = d(u,s) \leq d(u,w) + 1$ and conclude that $\tilde{d}(u,v) \leq d(u,v) + 2$.

We can now iterate this idea. We run Dijkstra's out of every vertex $u \in S_{k-2}$ on the edge set E_{k-2} consisting of edges adjacent to vertices of degree $< d_{k-1}$ and edges connecting vertices of degree $\geq d_{k-1}$ to a neighbor of their in S_{k-1}, in addition to an edge out of u to every vertex v weighted by the current best distance estimate between the pair. The same argument shows that all distance estimates computed out of S_{k-2} will be within $+4$ of the true distance. Repeating this argument grows the additive error by $+2$ with every iteration, resulting in a $+2k$ error for distances computed out of $S_0 = V$.

Balancing the degree thresholds gives a running time of $\widetilde{O}(n^{2-1/(k+1)}m^{1/(k+1)})$, which is the best known running time for sparse approximate APSP. However, for dense graphs Dor, Halperin and Zwick introduce additional edges to the graph search that improve the additive approximation to $< 2k$, while keeping a running time of $\widetilde{O}(n^{2+1/(k+1)})$.

Recent work has sought to improve the dense approximate APSP algorithm using fast matrix multiplication. Deng et al. [7] noted that for a $+2$-approximation, we can avoid running Dijkstra's out of $V = S_{k-1}$ by considering two cases. For pairs of vertices such that the shortest path between them contains only vertices of degree $< d_1$, run the sparse approximate APSP algorithm of Dor, Halperin and Zwick. For the remaining pairs, they show how to compute the distances out of S_1 in $\widetilde{O}(n^2)$ time and then compute $\tilde{d}(u,v) = \min_{s \in S_1} d(u,s) + d(s,v) \leq d(u,x) + 2$ using the $(\min,+)$ product.

In general, the $(\min,+)$-product (likely) cannot be solved polynomially faster than brute force, as it is equivalent to the APSP problem. However, the authors of [7] showed that one can sort the vertices of the graph in a way (based on the Euler tour of a spanning tree) that the resulting matrices are *column/row-bounded-difference* matrices. For such matrices, there exists a subcubic time algorithm for computing their $(\min,+)$-product. The following work of Dürr [10] used faster rectangular $(\min,+)$-product to speed up the $+2$ additive approximation.

Saha and Ye [13] used this same idea to improve the general $+2k$-approximation. Their algorithm replaces the first two stages of Dijkstra's searches with a call to sparse approximate APSP and a $(\min,+)$-product of the matrices representing the distances between S_{k-1}, S_k and S_k, S_{k-2}.

In this paper, we introduce a new way to speed up the $(\min,+)$-product computation. Instead of using the subcubic, but still considerably slow and complicated, bounded-difference $(\min,+)$-product, we reduce the problem to a $(\min,+)$-product with entries bounded by a constant. Due to a standard reduction (e.g. [15]), such a product can be computed in fast matrix multiplication time.

To achieve this, we introduce a new graph decomposition technique which decomposes the graph into a small number of clusters of constant diameter and a remainder of low degree vertices. We then compute the $(\min,+)$-product on each cluster independently. As all vertices in the cluster are within constant distance of each other, we

can shift the values of the corresponding distance matrix such that the resulting matrix is bounded by a constant. This allows us to compute distances between pairs of vertices in the clusters. To extend our approximation to the vertices in the remainder, we run a sparse graph search out of every vertex on just the low degree edges adjacent to vertices in the remainder.

1.4 Organization In Section 2 we prove the general decomposition lemma which we will later use in all of our algorithms. In Section 3 we prove Theorem 1.1 in two stages, beginning with a slower, simpler, 'warm up' algorithm that already beats the current state of the art algorithm for +2-APSP. Finally, in Section 4 we prove Theorem 1.2 by extending the simpler algorithm of Section 3 to a general $+2k$-approximation. We conclude with open questions in Section 5.

2 Preliminaries Let $[n] = \{1, 2, \ldots, n\}$. Let $G = (V, E)$ be an undirected graph and $U \subseteq V$ be a vertex subset. Let $G[U]$ denote the subgraph of G induced by U. Let $\deg_G(u)$ denote the degree of vertex u in graph G. Let $d_G(u, v)$ denote the distance between vertices u and v in graph G, and let $P_G(u, v)$ denote the shortest path from u to v, including the endpoints u, v (if more than one shortest path exists, for convenience we pick one that maximizes $\max_{x \in P_G(u,v)} \deg_G(x)$). Let $\mathrm{diam}_G(U)$ denote the (weak) diameter of the vertex subset U, defined as $\mathrm{diam}_G(U) = \max_{u,v \in U} d_G(u, v)$. We omit the subscript G and simply write $\deg(u)$, $\mathrm{diam}(U)$, $d(u, v)$ and $P(u, v)$ when the underlying graph G is clear from the context.

For a path P, let $|P|$ denote the length of P (in unweighted graphs, $|P|$ equals the number of edges in P).

We call a distance estimate $\tilde{d}(u, v)$ an additive $+C$-approximation if for every pair $u, v \in V$ the estimate satisfies $d(u, v) \leq \tilde{d}(u, v) \leq d(u, v) + C$.

In our algorithms we use the following (combinatorial) algorithm for sparse additive approximate APSP by Dor, Halperin, and Zwick [8].

LEMMA 2.1 ([8]). *$+2k$-Approximate APSP on an n-node m-edge unweighted undirected graph can be solved in $\widetilde{O}(n^{2-1/(k+1)}m^{1/(k+1)})$ time.*

We will use this lemma on paths containing vertices of bounded degree, in which case we obtain the following corollary by considering the $O(nd)$ edges adjacent to vertices of degree $\leq d$.

COROLLARY 2.2. *$+2k$-Approximate APSP between pairs of points such that a shortest path between them uses only vertices of degree $\leq d$ can be solved in $\widetilde{O}(n^2 d^{1/(k+1)})$ time.*

A standard tool used in these algorithms is the *hitting set*, as defined in the following lemma.

LEMMA 2.3 (Hitting set, e.g., [1, Theorem 2.7]). *Given an n-node undirected graph $G = (V, E)$ and a degree threshold $1 \leq d \leq n$, one can deterministically construct in $O(n^2)$ time a hitting set $S \subseteq V$ of size $O(\frac{n \log n}{d})$ such that every node $u \in V$ of degree at least d in G is adjacent to some $s \in S$.*

Let $\mathrm{MM}(n_1, n_2, n_3)$ denote the time complexity of multiplying an $n_1 \times n_2$ matrix by an $n_2 \times n_3$ matrix. We denote by $\omega(\gamma_1, \gamma_2, \gamma_3)$ the exponent of $\mathrm{MM}(n^{\gamma_1}, n^{\gamma_2}, n^{\gamma_3})$, i.e. the minimum value c such that the product of an $n^{\gamma_1} \times n^{\gamma_2}$ matrix by an $n^{\gamma_2} \times n^{\gamma_3}$ matrix can be computed in $O(n^{c+\varepsilon})$ time for any $\varepsilon > 0$.

A common matrix product used in shortest path computation is the $(\min, +)$-product, defined as follows.

DEFINITION 2.4 ($(\min, +)$-matrix product). *The $(\min, +)$-product of two matrices A, B is defined as $C = A \star B$ where $C[i, j] \coloneqq \min_k A[i, k] + B[k, j]$.*

When the entries of the matrices are bounded by an integer L, a standard method which encodes the entries as polynomials of degree $O(L)$ allows to compute their $(\min, +)$-product in fast matrix multiplication time (see e.g. [15]).

LEMMA 2.5 ([15]). *Given an $n_1 \times n_2$ matrix A and an $n_2 \times n_3$ matrix B such that entries of both matrices are in $\{0, 1, \ldots, L, \infty\}$, computing $C = A \star B$ can be done in time $\widetilde{O}(L \cdot \mathrm{MM}(n_1, n_2, n_3))$.*

2.1 A Decomposition Lemma Next, we prove the following decomposition lemma, which is a key component in our new approximation algorithms. The lemma shows that for any threshold d we can decompose our graph into disjoint clusters of size greater than d with constant diameter. The remaining vertices that are not assigned to clusters will all have degree smaller than d. While the decomposition itself is quite simple, the bounded diameter of the resulting clusters making up the graph is crucial in allowing for faster computation of their approximate shortest paths.

LEMMA 2.6. *Let $1 \le d \le n$. Given an m-edge undirected unweighted graph $G = (V, E)$, in $O(m)$ time we can deterministically decompose V into a disjoint union $R \cup \bigcup_{i=1}^{h} H_i$ such that:*

- $|H_i| > d$ *for all* $i \in [h]$.
- $\mathrm{diam}(H_i) \coloneqq \max_{u,v \in H_i} d_G(u, v) \le 4$ *for all* $i \in [h]$.
- $\deg_G(u) < d$ *for all* $u \in R$.

Proof. We begin constructing the clusters iteratively. Initialize $U = V$. While there exists $u \in U$ such that $\deg_{G[U]}(u) \ge d$, create a new cluster H'_i containing u and all of u's neighbors in U. Remove the vertices of H'_i from U and iterate until no vertex in $G[U]$ has degree $\ge d$ and we have created h clusters $H'_1, \ldots, H'_h$. Note that by taking H'_i to be the neighborhood of a high degree vertex we guarantee that $|H'_i| > d$ and $\mathrm{diam}(H'_i) \le 2$.

Next, beginning with $i = 1$ and iterating over the clusters, we define H''_i to be the vertices in U that are adjacent to a vertex in H'_i, and remove H''_i from U. We return $H_i \coloneqq H'_i \cup H''_i$ and set R to be the remaining vertices in U at the end of this process.

When we can no longer create new clusters at the first stage, we have that every vertex $u \in U$ has $< d$ neighbors in U. Thus, for every $u \in U$, if $\deg_G(u) \ge d$, it must have a neighbor in H'_i for some $i \in [h]$ and so this vertex will have been added to H''_i and removed from U. We conclude that all vertices $u \in R$ have degree $\deg_G(u) < d$. Finally, since $\mathrm{diam}(H'_i) \le 2$ and all vertices in H''_i are adjacent to H'_i we have that $\mathrm{diam}(H_i) \le 4$ for every $i \in [h]$. Since $|H'_i| > d$ we also have that $|H_i| > d$. ∎

2.2 A Min-Plus Lemma Lastly, we prove the following lemma about computing the $(\min, +)$ product of a matrix representing the distances from a set of low diameter.

LEMMA 2.7. *Let integer parameter $L \ge 1$. Given input matrices $A \in \mathbb{Z}^{n_1 \times n_2}$ and $B \in \mathbb{Z}^{n_2 \times n_3}$ such that $|B[k, j] - B[k, j']| \le L$ for all $k \in [n_2]$ and all $j, j' \in [n_3]$, we can compute the $(\min, +)$-product of A and B in $\widetilde{O}(L \cdot \mathrm{MM}(n_1, n_2, n_3))$ time.*

Proof. For $k \in [n_2]$, let $\Delta_k \coloneqq \min_{j \in [n_3]} B[k, j]$, and define matrix $B' \in \mathbb{Z}^{n_2 \times n_3}$ by $B'[k, j] \coloneqq B[k, j] - \Delta_k$. Then, all entries of B' are in $\{0, 1, \ldots, L\}$. Define matrix $A' \in \mathbb{Z}^{n_1 \times n_2}$ by $A'[i, k] \coloneqq A[i, k] + \Delta_k$. Then the $(\min, +)$-product of A and B equals the $(\min, +)$-product of A' and B', so it suffices to compute the latter (and we denote the answer by $C \in \mathbb{Z}^{n_1 \times n_3}$).

For $i \in [n_1]$, let $m_i \coloneqq \min_{k \in [n_2]} A'[i, k]$. Then, since $0 \le B'[k, j] \le L$, we have $C[i, j] = \min_{k \in [n_2]} \{A'[i, k] + B'[k, j]\} \in [m_i, m_i + L]$ for all i, j. Thus, if $A'[i, k_0] > m_i + L$, then $A'[i, k_0]$ is useless since $A'[i, k_0] + B'[k_0, j] > C[i, j]$ for all j, so we can replace the entry $A'[i, k_0]$ by $+\infty$ without changing the $(\min, +)$-product of A' and B'. Then, every entry $A'[i, k]$ is either $+\infty$ or in $[m_i, m_i + L]$. Define the matrix $A'' \in \mathbb{Z}^{n_1 \times n_2}$ by $A''[i, k] \coloneqq A'[i, k] - m_i \in \{0, 1, \ldots, L, +\infty\}$. We compute the $(\min, +)$-product C'' between $A'' \in \{0, 1, \ldots, L, +\infty\}^{n_1 \times n_2}$ and $B' \in \{0, 1, \ldots, L\}^{n_2 \times n_3}$ in $\widetilde{O}(L \cdot \mathrm{MM}(n_1, n_2, n_3))$ time using Lemma 2.5.

Finally, return the answer matrix $C[i, j] = C''[i, j] + m_i$. ∎

3 Faster +2-Approximate APSP

3.1 A Warm-Up Algorithm Now we describe our algorithms for computing +2-Approximate APSP based on the decomposition given in Lemma 2.6. In this section we present a warm-up algorithm (Corollary 3.2) which already improves over the state-of-the-art algorithms [7, 10]. In the next section we will give further improvement using more technical ideas.

We use a standard argument (which was also used in [7, 10]) that assumes the nodes on the (true) shortest paths under consideration have maximum degree $\Theta(D)$. More specifically, we will prove the following main lemma:

LEMMA 3.1. *Let $G = (V, E)$ be an n-vertex undirected unweighted graph, and parameter $1 \le D \le n$. Let $d_D(u, v)$ denote the minimum length of any (not necessarily simple) path P from u to v such that $\max_{x \in P} \deg(x) \in [D, 2D]$. We can compute distance estimates $\tilde{d}(u, v)$ such that $d(u, v) \le \tilde{d}(u, v) \le d_D(u, v) + 2$ holds for all $(u, v) \in V^2$, by a deterministic algorithm with time complexity*

$$\widetilde{O}\left(\min_{1 \le d < D} \left\{ n^2 d + \frac{n}{d} \mathrm{MM}\left(n, \frac{n}{D}, d\right) \right\}\right).$$

We analyze the overall time complexity for +2-Approximate APSP obtained from Lemma 3.1:

COROLLARY 3.2 (Warm-Up +2-APSP). *+2-Approximate APSP has a deterministic algorithm in $O(n^{2.2548})$ time.*

Proof. We enumerate $1 \le D \le n$ that are powers of two, and compute distance estimates $\tilde{d}(u,v) \in [d(u,v), d(u,v)+2]$ for pairs (u,v) satisfying $\max_{x \in P(u,v)} \deg(x) \in [D, 2D]$ (note that $d_D(u,v) = d(u,v)$ holds for such u, v). Finally we combine the answers across all D. Then, for given D, we can without loss of generality assume the input graph has maximum degree at most $2D$, and hence number of edges at most Dn. We run either Lemma 3.1 or Corollary 2.2 (whichever is faster) to compute the distance estimates. The time complexity of Corollary 2.2 is $\widetilde{O}(n^2\sqrt{D})$. The overall time complexity for +2-APSP is thus

$$\widetilde{O}\left(\max_{1 \le D \le n} \min\left\{n^2\sqrt{D}, \min_{1 \le d < D}\left\{n^2 d + \frac{n}{d}\,\mathrm{MM}\left(n, \frac{n}{D}, d\right)\right\}\right\}\right).$$

Using current fastest algorithms for rectangular matrix multiplication [2], the time complexity can be bounded by $n^{2.2548}$ [4].

If we only use square matrix multiplication, then in the above expression it is optimal to pick $d = (n/D)^{1/(4-\omega)} \le n/D$, so the time complexity becomes

$$\widetilde{O}(\max_D \min\{n^2\sqrt{D}, n^2 d + \tfrac{n}{d} \cdot (n/d)(\tfrac{n}{D}/d)d^\omega\}) = \widetilde{O}(\max_D \min\{n^2\sqrt{D}, n^2(\tfrac{n}{D})^{1/(4-\omega)}\}) = \widetilde{O}(n^{2+\frac{1}{6-\omega}}).$$ ∎

Now we describe our algorithm for given degree parameter D:

Proof of Lemma 3.1. Without loss of generality, we can assume the input graph $G = (V, E)$ has maximum degree at most $2D$, and hence number of edges $m \le Dn$.

Deterministically construct a hitting set (Lemma 2.3) $S \subset V$ of size $O(\frac{n \log n}{D})$ such that every $x \in V$ with $\deg(x) \ge D$ is adjacent to some vertex $s_x \in S$. Thus, if $d_D(u,v)$ is realized by the path P which connects u, v and contains a vertex x of degree $\deg(x) \ge D$, then

$$d(u,v) \le \min_{s \in S}\{d(u,s) + d(s,v)\} \le d(u,s_x) + d(s_x,v) \le d(u,x) + 1 + d(x,v) + 1 = d_D(u,v) + 2. \tag{3.1}$$

We run BFS from every $s \in S$ on G to compute $d(s,u)$ for all $(s,u) \in S \times V$, in total time $O(|S|m) \le \widetilde{O}(n/D) \cdot Dn = \widetilde{O}(n^2)$.

Let $1 \le d < D$ be a tunable parameter. Run Lemma 2.6 with parameter d, and obtain the vertex partition $V = R \cup \bigcup_{i=1}^{h} H_i$, where each cluster H_i has size $|H_i| \ge d$ and (weak) diameter $\mathrm{diam}(H_i) \le 4$. For convenience, we assume without loss of generality that $|H_i| \le 2d$ for all i (by possibly breaking larger clusters into smaller ones, which does not increase their weak diameter), and we still have $h = O(n/d)$ clusters in total.

For each cluster H_i, we compute $\tilde{d}[V, H_i]$ as the $(\min, +)$ product between the distance matrices $d[V,S]$ and $d[S,H_i]$ using Lemma 2.7. Note that the parameter L in Lemma 2.7 can be bounded using the triangle inequality by $L = \max_{s \in S, v, v' \in H_i} |d(s,v) - d(s,v')| \le \max_{s \in S, v, v' \in H_i} d(v,v') = \mathrm{diam}(H_i) \le O(1)$. Thus, the total time for invoking Lemma 2.7 over all $h = O(n/d)$ clusters is

$$O\left(\frac{n}{d}\right) \cdot \widetilde{O}\big(L \cdot \mathrm{MM}(|V|, |S|, |H_i|)\big) = \widetilde{O}\left(\frac{n}{d}\,\mathrm{MM}(n, n/D, d)\right).$$

By Equation (3.1), $\tilde{d}(v,h)$ (i.e., the (v,h)-th entry in the $(\min,+)$-product between $d[V,S]$ and $d[S,H_i]$) is a +2-approximation of $d_D(v,h)$, so we have already computed the desired answers for pairs $(v,h) \in V \times \bigcup_{i=1}^{h} H_i$. It remains to compute answers for the pairs $(v,r) \in V \times R$.

We enumerate each $v \in V$, and compute answers for the pairs $v \times R$ using the following lemma:

LEMMA 3.3. *For $v \in V$, suppose for all $w \in V \setminus R$ we know distance estimates $\tilde{d}(v,w)$ such that $d(v,w) \le \tilde{d}(v,w) \le d_D(v,w) + 2$. Then, in $\widetilde{O}(nd)$ time, we can compute distance estimates $\tilde{d}(v,w)$ such that $d(v,w) \le \tilde{d}(v,w) \le d_D(v,w) + 2$, for all $w \in V$.*

Proof. Define a (weighted) auxiliary graph G_v on the same vertex set V as follows:

- For every $r \in R$, include all the neighboring edges of r into G_v. Since $\deg_G(r) < d$ by Lemma 2.6, we have added $\sum_{r \in R} \deg_G(r) \le |R|d \le nd$ edges.

- For every $h \in \bigcup_{i=1}^{h} H_i$, add an edge into G_v between v and h with edge weight $\tilde{d}(v,h)$. This step adds only $\leq n$ edges to G_v.

Then, use Dijkstra's algorithm to compute the distances from v to all vertices on G_v in $\widetilde{O}(|E(G_v)|) = \widetilde{O}(nd)$ time, and return these results as our distance estimates $\tilde{d}(v,r)$ for all $r \in R$. By construction of G_v, it is clear that $d_{G_v}(v,r)$ does not underestimate the true distance $d_G(v,r)$, so it remains to prove that they achieve $+2$-approximation.

Suppose $d_D(v,r)$ is realized by the path P from v to r with $\max_{x \in P} \deg(x) \geq D$. If all vertices on P are contained in R, then by construction of G_v, all edges on the path $P(v,r)$ are included in G_v, so $d_{G_v}(v,r) \leq d_D(v,r) < d_D(v,r) + 2$ as claimed. It remains to consider the case where P is not fully contained in R. Let h be the last vertex on P such that $h \notin R$ (we know h exists and $h \neq r \in R$). Let $r_1, r_2, \ldots, r$ be the vertices after h on the path P. Then, $r_1, r_2, \ldots, r \in R$ by definition of h. By construction of G_v, this means the edges on the suffix $P_{hr} = h \to r_1 \to r_2 \to \cdots \to r$ of the path P are included in G_v. Since G_v also contains edge (v,h) of weight $\tilde{d}(v,h)$, we obtain

$$d_{G_v}(v,r) \leq \tilde{d}(v,h) + |P_{hr}| \leq (d_D(v,h) + 2) + |P_{hr}| \leq d_D(v,r) + 2$$

as desired, where the second inequality follows from the input assumption and $h \in V \setminus R$, and the last inequality is justified as follows: Recall $\max_{x \in P} \deg(x) \in [D, 2D]$, whereas the vertices after h on path P, namely $r_1, r_2, \ldots, r$, all belong to R and hence have degree $< d < D$. Therefore, the prefix $v \rightsquigarrow h$ of the path P contains a node of degree in $[D, 2D]$, which implies $d_D(v,h) \leq d_D(v,r) - |P_{hr}|$ as required. □

The total time of applying the above Lemma 3.3 for all $v \in V$ is $\widetilde{O}(n^2 d)$. This finishes the proof of Lemma 3.1.□

3.2 A Faster Algorithm In the previous algorithm, we separately computed the $(\min,+)$-product between $d[V,S]$ and $d[S,H_i]$ for every cluster H_i. In this section, we obtain a speed-up by considering all clusters together. More specifically, the first step of our improved algorithm is to efficiently compute the $(\min,+)$-product between $d[\bigcup_{i=1}^{h} H_i, S]$ and $d[S, \bigcup_{i=1}^{h} H_i]$. This step uses the technique of false positives mod prime, which was also a crucial ingredient in the state-of-the-art bounded-difference $(\min,+)$-product algorithm [6]. This step can be formalized as the following technical lemma:

LEMMA 3.4. *Let integer parameter $L \geq 1$. Let $A \in \mathbb{Z}^{hd \times s}, B \in \mathbb{Z}^{s \times hd}$ be input matrices with entries in $[-U, U]$. Partition the indices $i \in \{1, 2, \ldots, hd\}$ into h contiguous groups each of size d (so that i belongs to the $\lceil i/d \rceil$-th group).*

Suppose $|A[i,k] - A[i',k]| \leq L$ holds for all $k \in [s]$ and all i, i' in the same group, and $|B[k,j] - B[k,j']| \leq L$ holds for all $k \in [s]$ and all j, j' in the same group.

Then, for any parameter $q \geq 1$, we can compute the $(\min,+)$-product of A and B by a randomized algorithm in time complexity

$$\widetilde{O}\left(h^2 s + qL \cdot \mathrm{MM}(hd, s, hd) + h^2 L \cdot \mathrm{MM}(d, s/q, d)\right) \cdot \mathrm{poly}\log(U).$$

Proof. Throughout, we use the shorthand $\hat{i} = \lceil i/d \rceil$ and $\hat{j} = \lceil j/d \rceil$ to denote the groups that contain indices i and j respectively.

Let $C \in \mathbb{Z}^{hd \times hd}$ denote the $(\min,+)$-product of A and B which we want to compute. We first efficiently compute an $O(L)$-additive approximation of C as follows: Define matrix $A' \in \mathbb{Z}^{h \times s}$ by

$$A'[\hat{i}, k] \coloneqq \left\lfloor \frac{1}{L} \min_{(\hat{i}-1)d < i \leq \hat{i}d} A[i,k] \right\rfloor,$$

and define matrix $A'' \in \mathbb{Z}^{hd \times s}$ by

$$A''[i,k] \coloneqq A[i,k] - L \cdot A'[\hat{i}, k]. \tag{3.2}$$

Then, observe that the input assumption on A implies

$$0 \leq A''[i,k] < 2L \tag{3.3}$$

for all $i \in [hd]$ and $k \in [s]$. We define matrices $B' \in \mathbb{Z}^{s \times h}$ and $B'' \in \mathbb{Z}^{s \times hd}$ analogously with

$$B''[k,j] \coloneqq B[k,j] - L \cdot B'[k,\hat{j}], \tag{3.4}$$

and

$$0 \le B''[k,j] < 2L \tag{3.5}$$

for all $k \in [s], j \in [hd]$. We compute $C' \in \mathbb{Z}^{h \times h}$ as the $(\min,+)$-product of A', B' by brute force in time $O(h^2 s)$. From Equations (3.3) and (3.5) we can observe that C' provides an $O(L)$-additive approximation of the true $(\min,+)$-product C in the sense that

$$0 \le C[i,j] - L \cdot C'[\hat{i},\hat{j}] < 4L \tag{3.6}$$

holds for all $i, j \in [hd]$. Thus, if k is a witness for $C[i,j]$ (i.e., $A[i,k] + B[k,j] = C[i,j]$), then

$$\begin{aligned} \left|A'[\hat{i},k] + B'[k,\hat{j}] - C'[\hat{i},\hat{j}]\right| &= \frac{1}{L}\left|-A''[i,k] - B''[k,j] + A[i,k] + B[k,j] - L \cdot C'[\hat{i},\hat{j}]\right| \\ &= \frac{1}{L}\left|-A''[i,k] - B''[k,j] + C[i,j] - L \cdot C'[\hat{i},\hat{j}]\right| \\ &< 4, \end{aligned} \tag{3.7}$$

where the last step follows from Equations (3.3), (3.5), and (3.6).

Pick a random prime $p \in \Theta(q \log U)$. Define an $hd \times s$ matrix $\tilde{A}$ where each entry is a bivariate monomial defined as

$$\tilde{A}[i,k] \coloneqq x^{A'[\hat{i},k] \bmod p} y^{A''[i,k]},$$

which has x-degree less than $p = O(q \log U)$ and y-degree less than $2L$ (by Equation (3.3)). Define $s \times hd$ matrix $\tilde{B}$ analogously. Compute $hd \times hd$ matrix $\tilde{C}$ as the product of $\tilde{A}$ and $\tilde{B}$ via Fast Matrix Multiplication and FFT (e.g., [15]) in $\widetilde{O}(\mathrm{MM}(hd, s, hd) \cdot (q \log U) \cdot (2L))$ time, so

$$\tilde{C}[i,j] = \sum_{k \in [s]} x^{(A'[\hat{i},k] + B'[k,\hat{j}]) \bmod p} y^{A''[i,k] + B''[k,j]}.$$

Recall every witness k for $C[i,j]$ satisfies Equation (3.7). In particular, k satisfies the following mod-p version of Equation (3.7):

$$A'[\hat{i},k] + B'[k,\hat{j}] \in C'[\hat{i},\hat{j}] + \{-3,\dots,3\} \pmod p. \tag{3.8}$$

Hence k contributes a term in the polynomial $\tilde{C}[i,j]$ with x-degree in $C'[\hat{i},\hat{j}] + \{-3,\dots,3\} \pmod p$. We say a triple $(\hat{i},k,\hat{j}) \in [h] \times [s] \times [h]$ is a *false positive* if it does not satisfy Equation (3.7), but satisfies Equation (3.8). Then, for every $(i,j) \in [hd] \times [hd]$, compute the polynomial

$$\tilde{C}[i,j] - \sum_{k:(\hat{i},k,\hat{j}) \text{ is a false positive}} x^{(A'[\hat{i},k] + B'[k,\hat{j}]) \bmod p} y^{A''[i,k] + B''[k,j]}. \tag{3.9}$$

We describe how to compute these polynomials later.

Consider each $(i,j) \in [hd] \times [hd]$. We enumerate every non-zero $x^{c'} y^{c''}$ term in the polynomial Equation (3.9) that satisfies $c' \in C'[\hat{i},\hat{j}] + \{-3,\dots,3\} \pmod p$. Then, this term must originate from some $x^{(A'[\hat{i},k] + B'[k,\hat{j}]) \bmod p} y^{A''[i,k] + B''[k,j]}$ where $A'[\hat{i},k] + B'[k,\hat{j}]$ equals the unique integer in $C'[\hat{i},\hat{j}] + \{-3,\dots,3\}$ that is congruent to c' modulo p (since the contribution from false positives has been removed), and $A''[i,k] + B''[k,j] = c''$. Then, the corresponding value of $A[i,k] + B[k,j]$ can be recovered via Equations (3.2) and (3.4) as

$$A[i,k] + B[k,j] = L \cdot (A'[\hat{i},k] + B'[k,\hat{j}]) + (A''[i,k] + B''[k,j]),$$

and we use it to update the candidate answer for $C[i,j]$. This algorithm correctly computes all $C[i,j]$ since all potential witnesses k for each $C[i,j]$ have been considered.

Now it remains to analyze the total time complexity for computing the polynomial in Equation (3.9) for all $(i,j) \in [hd] \times [hd]$; since the first term $\tilde{C}[i,j]$ is already computed, we focus on computing the second term (summation over k) in Equation (3.9). For $\hat{i}, \hat{j} \in [h], r \in \{-3, \dots, 3\}$, define set (which can be computed in $O(h^2 s)$ time by brute force enumeration)

$$F_{\hat{i},\hat{j},r} \coloneqq \Big\{ k \in [s] : (\hat{i}, k, \hat{j}) \text{ is a false positive and } A'[\hat{i}, k] + B'[k, \hat{j}] \equiv C'[\hat{i}, \hat{j}] + r \pmod{p} \Big\}.$$

Fix $(\hat{i}, \hat{j}) \in [h] \times [h]$. The second term in Equation (3.9) for any $(i,j) \in ((\hat{i}-1)d, \hat{i}d] \times ((\hat{j}-1)d, \hat{j}d]$ can be written as

$$\sum_{r \in \{-3,\dots,3\}} x^{(C'[\hat{i},\hat{j}]+r) \bmod p} \sum_{k \in F_{\hat{i},\hat{j},r}} y^{A''[i,k]} \cdot y^{B''[k,j]}.$$

We can compute this for all $(i,j) \in ((\hat{i}-1)d, \hat{i}d] \times ((\hat{j}-1)d, \hat{j}d]$ using matrix multiplications of dimension $d \times |F_{\hat{i},\hat{j},r}| \times d$ for $r \in \{-3, \dots, 3\}$ where each entry is a degree-$O(L)$ univariate polynomial in y, in time $\sum_{r \in \{-3,\dots,3\}} \widetilde{O}(\mathrm{MM}(d, |F_{\hat{i},\hat{j},r}|, d) \cdot L)$. Note that $k \in F_{\hat{i},\hat{j},r}$ holds only if p is a prime factor of the non-zero integer $A'[\hat{i}, k] + B'[k, \hat{j}] - C'[\hat{i}, \hat{j}] - r$, which happens with probability (over random prime $p \in \Theta(q \log U)$) at most $O(1/q)$ by the prime number theorem. Hence, by linearity of expectation and Markov's inequality, for fixed $(\hat{i}, \hat{j}) \in [h]^2$, we have $\sum_{r \in \{-3,\dots,3\}} |F_{\hat{i},\hat{j},r}| = O(s/q)$ with ≥ 0.99 success probability. In this successful case, the time for computing Equation (3.9) (which then allows to compute $C[i,j]$) for all $(i,j) \in ((\hat{i}-1)d, \hat{i}d] \times ((\hat{j}-1)d, \hat{j}d]$ becomes $\widetilde{O}(\mathrm{MM}(d, s/q, d) \cdot L)$. The total time over all $(\hat{i}, \hat{j}) \in [h]^2$ is $\widetilde{O}(h^2\, \mathrm{MM}(d, s/q, d) \cdot L)$. We repeat the whole process $O(\log h)$ times with independent random primes p, so that with high probability every $(\hat{i}, \hat{j}) \in [h]^2$ is successful at least once.

The total time is $\widetilde{O}(h^2 s + \mathrm{MM}(hd, s, hd) \cdot (q \log U) \cdot (2L) + h^2\, \mathrm{MM}(d, s/q, d) \cdot L)$ as claimed. ∎

Now we use Lemma 3.4 to prove the following lemma:

Lemma 3.5. *Let $G = (V, E)$ be an n-vertex undirected unweighted graph, and parameter $1 \leq D \leq n$. Let $d_D(u,v)$ denote the minimum length of any (not necessarily simple) path P from u to v such that $\max_{x \in P} \deg(x) \in [D, 2D]$. We can compute distance estimates $\tilde{d}(u,v)$ such that $d(u,v) \leq \tilde{d}(u,v) \leq d_D(u,v) + 2$ for all $(u,v) \in V^2$, by a randomized algorithm with time complexity*

$$\widetilde{O}\left(\min_{1 \leq d < D, q \geq 1} \left\{ n^2 d + \left(\frac{n}{d}\right)^2 \cdot \frac{n}{D} + q \cdot \mathrm{MM}\left(n, \frac{n}{D}, n\right) + \left(\frac{n}{d}\right)^2 \cdot \mathrm{MM}\left(d, \frac{n}{Dq}, d\right) \right\} \right).$$

Proof sketch. The first few steps are identical to the proof of Lemma 3.1: We have a hitting set $S \subset V$ of size $|S| = \widetilde{O}(n/D)$ and we compute $d(s,u)$ for all $(s,u) \in S \times V$. We obtain the vertex partition $V = R \cup \bigcup_{i=1}^{h} H_i$ where $|H_i| = \Theta(d), \mathrm{diam}(H_i) = L = O(1)$ and $\max_{r \in R} \deg(r) \leq d$. Here $1 \leq d < D$ and $h = O(n/d)$.

Then, we use Lemma 3.4 to compute the $(\min, +)$-product of the distance matrices $d[\bigcup_{i=1}^{h} H_i, S]$ and $d[S, \bigcup_{i=1}^{h} H_i]$. The time complexity of Lemma 3.4 is (where $q \geq 1$ is a tunable parameter)

$$\widetilde{O}\left(\left(\frac{n}{d}\right)^2 \cdot \frac{n}{D} + q \cdot \mathrm{MM}\left(n, \frac{n}{D}, n\right) + \left(\frac{n}{d}\right)^2 \cdot \mathrm{MM}\left(d, \frac{n}{Dq}, d\right) \right).$$

We now have $+2$-approximation of $d_D(u,v)$ for all $(u,v) \in \bigcup_{i=1}^{h} H_i \times \bigcup_{i=1}^{h} H_i = (V \setminus R) \times (V \setminus R)$. Based on these answers, use Lemma 3.3 to obtain $+2$-approximation of $d_D(u,v)$ for all $(u,v) \in V \times (V \setminus R)$, in total time $\widetilde{O}(n^2 d)$, in an analogous way to the last part of the proof of Lemma 3.1. Then, based on these answers, again use Lemma 3.3 to obtain $+2$-approximation of $d_D(u,v)$ for all $(u,v) \in V \times V$ in total time $\widetilde{O}(n^2 d)$. ∎

Finally we analyze the overall time complexity obtained from Lemma 3.5:

Proof of Theorem 1.1. As before, we enumerate $1 \leq D \leq n$ that are powers of two, and compute distance estimates $\tilde{d}(u,v) \in [d(u,v), d(u,v) + 2]$ for pairs (u,v) satisfying $\max_{x \in P(u,v)} \deg(x) \in [D, 2D]$ (note that $d_D(u,v) = d(u,v)$ holds for such u, v). Finally we combine the answers across all D. Then, for given D, we can without loss of generality assume the input graph has maximum degree at most $2D$, and hence number of

edges at most $O(Dn)$. We run either Lemma 3.5 or Corollary 2.2 (whichever is faster) to compute the distance estimates. The time complexity of Corollary 2.2 is $\widetilde{O}(n^2\sqrt{D})$. The overall time complexity for +2-APSP is thus

$$\widetilde{O}\left(\max_{1\le D\le n}\min\left\{n^2\sqrt{D}, \min_{1\le d<D, q\ge 1}\left\{n^2 d + \left(\frac{n}{d}\right)^2\cdot\frac{n}{D} + q\cdot \mathrm{MM}\left(n,\frac{n}{D},n\right) + \left(\frac{n}{d}\right)^2\cdot\mathrm{MM}\left(d,\frac{n}{Dq},d\right)\right\}\right\}\right).$$

We first show how to set the parameters d and q in terms of D when we only use square matrix multiplication. We will show that we can set d and q such that $n \ge Ddq$. If this is the case, for fixed D, the running time becomes

$$\min\left\{n^2\sqrt{D}, \min_{1\le d<D, q\ge 1}\left\{n^2 d + \left(\frac{n}{d}\right)^2\cdot\frac{n}{D} + q\cdot\frac{n^\omega}{D^{\omega-2}} + \frac{n^3}{Dd^{3-\omega}q}\right\}\right\}.$$

Now, assuming that $n \ge Dd$, set $q = (\frac{n}{Dd})^{\frac{3-\omega}{2}}$ to balance the latter two terms in the running time above. Verify that with this setting of q, $\frac{n}{Dd} \ge q$, as needed.

The time complexity is now

$$\min\left\{n^2\sqrt{D}, \min_{1\le d<D}\left\{n^2 d + \frac{n^3}{d^2 D} + \left(\frac{n^{3+\omega}}{D^{\omega-1}d^{3-\omega}}\right)^{1/2}\right\}\right\}.$$

Note that $n^2 d \ge \frac{n^3}{d^2 D}$ iff $d \ge (n/D)^{1/3}$, and $n^2 d \ge \left(\frac{n^{3+\omega}}{D^{\omega-1}d^{3-\omega}}\right)^{1/2}$ iff $d \ge (n/D)^{\frac{\omega-1}{5-\omega}} \ge (n/D)^{1/3}$. Hence to minimize the running time, we set $d = (n/D)^{\frac{\omega-1}{5-\omega}}$ which gives us a running time of

$$\min\left\{n^2\sqrt{D}, n^2\cdot(n/D)^{\frac{\omega-1}{5-\omega}}\right\}.$$

We verify that this setting of d also gives us that $n \ge Ddq$, as desired.

Finally, the running time is the maximum over all $D \le n$ of the above quantity. The worst-case running time is achieved when $n^2\sqrt{D} = n^2\cdot(n/D)^{\frac{\omega-1}{5-\omega}}$, i.e. when $D = n^{\frac{2\omega-2}{\omega+3}}$. Thus, the final runtime for +2 APSP in terms of ω is $\widetilde{O}(n^{2+\frac{\omega-1}{\omega+3}})$.

For the current value of ω, $\omega < 2.371339$ [2], the bound is $n^{2.25531}$. As ω goes to 2 the runtime goes to $n^{2.2}$, and the exponent $2+\frac{\omega-1}{\omega+3}$ is smaller than ω if $\omega > \sqrt{5} \approx 2.236$.

To bound the running time using rectangular matrix multiplication we proceed as follows. Let $D = n^b, d = n^\delta, q = n^z$ and minimize

$$\max\{2+b/2, 2+\delta, 3-2\delta-b, z+\omega(1,1-b,1), 2-2\delta+\omega(\delta,1-z-b,\delta)\}.$$

We use the code of [4] updated with the newest rectangular matrix multiplication bounds [2] and obtain that $b = 0.45095703, \delta = 0.22547851, z = 0.15981814$ gives running time $O(n^{2.225479})$. ∎

4 Faster $+2k$-Approximate APSP In this section we use the clustering technique introduced in Subsection 2.1 to construct a new algorithm for $+2k$-approximate APSP, resulting in a faster runtime for +4 and +6-approximate APSP. We note that for additive error of +8 and up this approach is no longer faster than that of Saha and Ye [13]. We prove Theorem 1.2.

THEOREM 1.2. *$+2k$-APSP in an n-node unweighted undirected graph can be solved by a deterministic algorithm in $O(n^{2+x/(k+1)})$ time, when x is the solution to $1+x = \omega(1-\frac{k-1}{k+1}x, 1-x, \frac{k}{k+1}x)$. (See the definition of the rectangular matrix multiplication exponent $\omega(\cdot,\cdot,\cdot)$ in Section 2.)*

Using standard techniques as explained in Corollary 3.2, we can assume the maximum degree on the true shortest path is $\Theta(D)$. We will use the following generalization of Lemma 3.1.

LEMMA 4.1. *Let $G = (V, E)$ be an n-vertex undirected unweighted graph, $U \subset V$ and parameter $1 \le D \le n$. Denote $d_D(u,v)$ as the minimum length of any (not necessarily simple) path P from u to v such that $\max_{x\in P}\deg(x) \in [D, 2D]$. We can compute distance estimates $\tilde{d}(u,v)$ such that $d(u,v) \le \tilde{d}(u,v) \le d_D(u,v)+2$ for all $u \in U, v \in V$, by a deterministic algorithm with time complexity*

$$\widetilde{O}\left(\min_{1\le d<D}\left\{|U|\cdot nd + \frac{n}{d}\,\mathrm{MM}\left(|U|,\frac{n}{D},d\right)\right\}\right).$$

The proof of this lemma follows directly from the proof of Lemma 3.1, by restricting the $(\min,+)$-products and graph search to the subset U.

Next, assuming we have computed a $+2$-approximation for distances out of a hitting set U which hits the neighborhood of all vertices of degree $\geq \delta$, we use the following lemma to extend these estimates to a $+2k$-approximation to all distances.

LEMMA 4.2. *Let $G = (V, E)$ be an n-vertex undirected unweighted graph and parameters $1 \leq \delta \leq D \leq n$. Let $U \subset V$ be a hitting set of size $|U| = \widetilde{O}(n/\delta)$ for the neighborhoods of vertices of degree $\geq \delta$. Given distance estimates $\tilde{d}(u,v)$ such that $d(u,v) \leq \tilde{d}(u,v) \leq d_D(u,v) + 2$ for every pair $u \in U, v \in V$, we can compute distance estimates $\tilde{d}(u,v)$ such that $d(u,v) \leq \tilde{d}(u,v) \leq d_D(u,v) + 2k$ for all $u, v \in V$, by a deterministic algorithm with time complexity*

$$O(n^2\delta^{1/(k-1)}).$$

Proof. Without loss of generality, assume G has maximum degree at most $2D$. Define the degree thresholds $1 = d_0 < d_1 < \ldots < d_{k-1} = \delta$ by $d_i = \delta^{i/(k-1)}$. For every $1 \leq i \leq k-2$, deterministically construct a hitting set S_i of size $|S_i| = \widetilde{O}(n/d_i)$ that hits the neighborhoods of all vertices of degree $\geq d_i$ (Lemma 2.3). Set $S_{k-1} = U$, and $S_0 = V$. For every $0 \leq i \leq k-2$, define the edge set E_i to include all edges adjacent to vertices of degree $< d_{i+1}$. For every vertex v of degree $\geq d_{i+1}$, include in E_i an edge connecting v to an arbitrary neighbor in S_{i+1}.

Recall that we initially have distance estimates $\tilde{d}(u,v)$ between $u \in U, v \in V$; initialize $\tilde{d}(u,v)$ to $+\infty$ for the remaining pairs. Beginning with $i = k-2$ and going down to $i = 0$, run the following Dijkstra's searches. For every vertex $u \in S_i$, add an edge from u to every $v \in V$ of weight $\tilde{d}(u,v)$. Run Dijkstra's algorithm out of u on the union of these edges and the edge set E_i. Update $\tilde{d}$ with the new distances computed.

Note that $|E_i| = O(n \cdot \delta^{(i+1)/(k-1)})$ and so running $|S_i|$ Dijkstra's searches (where $0 \leq i \leq k-2$) takes total time

$$\widetilde{O}(|S_i| \cdot (n + |E_i|)) = \widetilde{O}\left(\frac{n}{\delta^{i/(k-1)}} \cdot n\delta^{(i+1)/(k-1)}\right) = \widetilde{O}\left(n^2\delta^{1/(k-1)}\right).$$

Assuming $k = O(1)$, the total runtime of running all the searches is $\widetilde{O}(n^2\delta^{1/(k-1)})$. We are left to prove the bound on the approximation error. We do so using the following inductive claim.

CLAIM 4.3. *After searching out of S_i, $d(u,v) \leq \tilde{d}(u,v) \leq d_D(u,v) + 2(k-i)$ for every $u \in S_i, v \in V$.*

Proof. All distances computed in the Dijkstra's searches stem from true paths in the graph, so we always have $\tilde{d}(u,v) \geq d(u,v)$. We now show the upper bound inductively.

For $i = k-1$ we do not perform a search out of $S_{k-1} = U$ but rather use the provided distance estimates that are guaranteed to have $\tilde{d}(u,v) \leq d_D(u,v) + 2 = d_D(u,v) + 2(k-i)$ as required.

Now, for $0 \leq i \leq k-2$, assuming the claim holds for S_{i+1}, consider a vertex pair $u \in S_i, v \in V$, and the path P from u to v that realizes $d_D(u,v)$ with $\max_{x \in P} \deg(x) \in [D, 2D]$. Let w be the closest vertex to v on P with $\deg(w) \geq d_{i+1}$ and let $s \in S_{i+1}$ be a neighbor of w such that $(s,w) \in E_i$.

Since $\max_{x \in P} \deg(x) \in [D, 2D]$, whereas all vertices beyond w on P have degree $< d_{i+1} \leq d_{k-1} = \delta \leq D$, we have that $\max_{x \in P_{uw}} \deg(x) \in [D, 2D]$, where P_{uw} denotes the prefix of P up to node w. Appending node s to P_{uw} yields a path from u to s which has a highest degree in $[D, 2D]$ and has length $|P_{uw}| + 1 = d_D(u,v) - d(w,v) + 1$. Hence, $d_D(u,s) \leq d_D(u,v) - d(w,v) + 1$. Since $s \in S_{i+1}$, by the inductive hypothesis we conclude $\tilde{d}(u,s) \leq d_D(u,s) + 2(k-i-1) \leq d_D(u,v) - d(w,v) + 1 + 2(k-i-1)$.

When running Dijkstra's out of $u \in S_i$ we have an edge from u to s of weight $\tilde{d}(u,s)$. Furthermore, the edge $(w,s) \in E_i$, and the suffix of the path P from w to v is contained in E_i. Therefore,

$$\tilde{d}(u,v) \leq \tilde{d}(u,s) + 1 + d(w,v) \leq d_D(u,v) - d(w,v) + 1 + 2(k-i-1) + 1 + d(w,v) = d_D(u,v) + 2(k-i),$$

which proves the claim. ∎

Thus, after running Dijkstra's out of $V = S_0$ we have that $\tilde{d}(u,v) \leq d_D(u,v) + 2k$ for every pair of vertices $u, v \in V$. ∎

We can now combine Lemma 4.1, Lemma 4.2 and Corollary 2.2 to prove Theorem 1.2.

Proof of Theorem 1.2. Enumerate over $1 \leq D \leq n$ that are powers of two and compute distance estimates $\tilde{d}(u,v) \in [d(u,v), d(u,v) + 2k]$ for pairs u, v satisfying $\max_{x \in P(u,v)} \deg(x) \in [D, 2D]$ (note that $d_D(u,v) = d(u,v)$

holds for such u, v). Finally we combine the answers across all values of D by taking the minimum value of $\tilde{d}(u,v)$ computed for every pair, since we are guaranteed that for all values of D, the estimate computed is an upper bound to the true distance.

Thus, for a given D, we can assume the given input graph has maximum degree at most $2D$. We can compute our distance estimate in one of the following two ways. First, we can use Corollary 2.2 to compute $\tilde{d}(u,v)$ in time $\tilde{O}(n^2 D^{\frac{1}{k+1}})$.

Otherwise, we can set a parameter $\delta \leq D$ and deterministically construct a hitting set U of size $\tilde{O}\left(\frac{n}{\delta}\right)$ that hits the neighborhood of all vertices of degree $\geq \delta$ via Lemma 2.3. Using Lemma 4.1, compute distance estimates $\tilde{d}(u,v)$ such that $d(u,v) \leq \tilde{d}(u,v) \leq d_D(u,v) + 2$ for every $u \in U, v \in V$. Next, use Lemma 4.2 to extend these distance estimates to $+2k$-approximation for all pairs $u, v \in V$ in time $\tilde{O}(n^2 \delta^{\frac{1}{k-1}})$.

For every D, we can pick the faster of these two options. In total, our runtime comes out to

$$\tilde{O}\left(\max_{1 \leq D \leq n} \min\left\{n^2 D^{\frac{1}{k+1}}, n^2 \delta^{\frac{1}{k-1}} + \min_{1 \leq d \leq D}\left\{\frac{n^2 d}{\delta} + \frac{n}{d}\,\mathrm{MM}\left(\frac{n}{\delta}, \frac{n}{D}, d\right)\right\}\right\}\right).$$

This expression is maximized when the two runtimes are equal, at which point let $0 \leq x \leq 1$ be such that $D = \Theta(n^x)$. Set $\delta = n^{\frac{k-1}{k+1}x}$ and $d = n^{\frac{k}{k+1}x}$ to obtain a running time of

$$\widetilde{O}\left(n^{2+\frac{1}{k+1}x} + n^{1-\frac{k}{k+1}x}\,\mathrm{MM}\left(n^{1-\frac{k-1}{k+1}x}, n^{1-x}, n^{\frac{k}{k+1}x}\right)\right).$$

To minimize this expression, we find the value of x for which

$$2 + \frac{1}{k+1}x = 1 - \frac{k}{k+1}x + \omega\left(1 - \frac{k-1}{k+1}x, 1-x, \frac{k}{k+1}x\right).$$

Equivalently $1 + x = \omega(1 - \frac{k-1}{k+1}x, 1-x, \frac{k}{k+1}x)$. Thus, we obtain a $+2k$-approximate APSP algorithm running in time $\tilde{O}(n^{2+\frac{1}{k+1}x})$ for the value of x satisfying $1 + x = \omega(1 - \frac{k-1}{k+1}x, 1-x, \frac{k}{k+1}x)$. ∎

5 Open Problems A major open question is whether $+2$-approximate APSP can be solved in $n^{2+o(1)}$ time. However this is unknown even for the much easier 2-multiplicative approximate APSP problem, which can be solved in slightly superquadratic time using fast matrix multiplication techniques [9, 13].

The work of Dor, Halperin and Zwick [8] showed an $\tilde{O}(n^2)$ time algorithm for a $+\log n$ approximate APSP. It remains open to determine if there exists a constant C such that $+C$-approximate APSP can be solved in $n^{2+o(1)}$ time. One can also ask, can we solve $+2$-approximate APSP in $\tilde{O}(n^{f(\omega)})$ time, where function $f(\omega)$ satisfies $f(\omega) < \omega$ for all possible values of $\omega > 2$? In other words, is $+2$-approximate APSP easier than matrix multiplication?

In the scope of this work, we note that our $+2k$-approximation algorithm follows the idea of the *sparse* approximate APSP algorithm of [8] in that it computes *all* distances out of each hitting set. In reality, for the next step of the algorithm we are only interested in distances between a hitting set S_i and the next hitting set S_{i-1}. The dense approximate APSP algorithm of [8] makes use of this distinction to speed up their sparse algorithm. The work of Saha and Ye [13] also makes use of this fact when computing the bounded-difference $(\min, +)$-product between these hitting sets. However, in order to make use of this fact in our setting we would need to be able to adapt our decomposition lemma (Lemma 2.6) to decompose a subset of the graph, and not the entire vertex set. It remains open to determine if such a decomposition is possible.

References

[1] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani, *Fast estimation of diameter and shortest paths (without matrix multiplication)*, SIAM J. Comput., 28 (1999), pp. 1167–1181, https://doi.org/10.1137/S0097539796303421.

[2] J. Alman, R. Duan, V. Vassilevska Williams, Y. Xu, Z. Xu, and R. Zhou, *More asymmetry yields faster matrix multiplication*, in Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025, SIAM, 2025, pp. 2005–2039, https://doi.org/10.1137/1.9781611978322.63.

[3] N. Alon, Z. Galil, and O. Margalit, *On the exponent of the all pairs shortest path problem*, J. Comput. Syst. Sci., 54 (1997), pp. 255–262, https://doi.org/10.1006/JCSS.1997.1388.

[4] J. v. d. Brand, *Complexity term balancer*. www.ocf.berkeley.edu/~vdbrand/complexity/. Tool to balance complexity terms depending on fast matrix multiplication.

[5] K. Bringmann, F. Grandoni, B. Saha, and V. Vassilevska Williams, *Truly subcubic algorithms for language edit distance and RNA folding via fast bounded-difference min-plus product*, SIAM J. Comput., 48 (2019), pp. 481–512, https://doi.org/10.1137/17M112720X.

[6] S. Chi, R. Duan, T. Xie, and T. Zhang, *Faster min-plus product for monotone instances*, in STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, ACM, 2022, pp. 1529–1542, https://doi.org/10.1145/3519935.3520057.

[7] M. Deng, Y. Kirkpatrick, V. Rong, V. Vassilevska Williams, and Z. 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, vol. 229 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 50:1–50:10, https://doi.org/10.4230/LIPIcs.ICALP.2022.50.

[8] D. Dor, S. Halperin, and U. Zwick, *All-pairs almost shortest paths*, SIAM Journal on Computing, 29 (2000), pp. 1740–1759, https://doi.org/10.1137/S0097539797327908. Announced at FOCS 1996.

[9] M. Dory, S. Forster, Y. Kirkpatrick, Y. Nazari, V. Vassilevska Williams, and T. de Vos, *Fast 2-approximate all-pairs shortest paths*, in Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024, SIAM, 2024, pp. 4728–4757, https://doi.org/10.1137/1.9781611977912.169.

[10] A. Dürr, *Improved bounds for rectangular monotone min-plus product and applications*, Inf. Process. Lett., 181 (2023), p. 106358, https://doi.org/10.1016/J.IPL.2023.106358.

[11] M. Gupta, *Improved 2-approximate shortest paths for close vertex pairs*, CoRR, abs/2507.19859 (2025), https://arxiv.org/abs/2507.19859.

[12] L. 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, ACM, 2023, pp. 309–320, https://doi.org/10.1145/3564246.3585197.

[13] B. Saha and C. Ye, *Faster approximate all pairs shortest paths*, in Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024, SIAM, 2024, pp. 4758–4827, https://doi.org/10.1137/1.9781611977912.170.

[14] R. Seidel, *On the all-pairs-shortest-path problem in unweighted undirected graphs*, J. Comput. Syst. Sci., 51 (1995), pp. 400–403, https://doi.org/10.1006/JCSS.1995.1078.

[15] A. Shoshan and U. Zwick, *All pairs shortest paths in undirected graphs with integer weights*, in 40th Annual Symposium on Foundations of Computer Science, FOCS 1999, New York, NY, USA, October 17-18, 1999, IEEE Computer Society, 1999, pp. 605–615, https://doi.org/10.1109/SFFCS.1999.814635.

[16] V. Vassilevska Williams, *On some fine-grained questions in algorithms and complexity*, in Proceedings of the international congress of mathematicians: Rio de janeiro 2018, World Scientific, 2018, pp. 3447–3487.

[17] R. R. Williams, *Faster all-pairs shortest paths via circuit complexity*, SIAM J. Comput., 47 (2018), pp. 1965–1985, https://doi.org/10.1137/15M1024524.

[18] U. Zwick, *All pairs shortest paths using bridging sets and rectangular matrix multiplication*, J. ACM, 49 (2002), pp. 289–317, https://doi.org/10.1145/567112.567114.