A Survey on Approximate APSP Algorithms

LRYP

January 20, 2026

Abstract

This survey provides a comprehensive overview of recent progress on approximate all-pairs shortest paths (APSP), emphasizing the algorithmic frameworks that underlie current bounds for both weighted and unweighted graphs. We categorize and analyze recent breakthroughs in $(2, 0)$-approximations and additive $+k$-approximations, highlighting the interplay between degree thresholds and path lengths. Furthermore, we discuss the latest improvements utilizing $(\min, +)$-matrix multiplication. Finally, we present experimental results benchmarking purely combinatorial additive approximation algorithms against standard BFS.

Contents

1 Introduction **2**

2 Preliminaries **3**
- 2.1 Notation . . . 3
- 2.2 Basic Framework for Unweighted Graphs . . . 3
- 2.3 Basic Framework for Weighted Graphs . . . 4

3 Weighted 2-Approximation **4**

4 Unweighted 2-Approximation **6**

5 Unweighted Additive Approximation **7**
- 5.1 Combinatorical Algorithms . . . 7
- 5.2 Faster Algorithms with FMM . . . 8
- 5.3 Newest Improvement . . . 9

6 Unweighted Approximation for Certain Distances **10**
- 6.1 Additive Approximation for Short Paths . . . 10
- 6.2 2-Approximation for Long Paths . . . 11

7 Numerical Values of Time Complexity Exponents for Algorithms Utilizing FMM **12**

8 Experiments **12**

9 References **13**

1 Introduction

The All-Pairs Shortest Paths (APSP) problem is a fundamental problem in graph theory:

Problem 1 (APSP)**.** *Given a graph $G = (V, E)$, find the shortest path distance $d(u, v)$ between every pair of vertices u, v.*

APSP is intimately connected to matrix multiplication. For standard APSP variants, finding algorithms faster than their corresponding matrix multiplication bounds remains a major open problem:

- It is conjectured that neither APSP on directed weighted graphs with n vertices nor $(\min, +)$-matrix multiplication for $n \times n$ matrices can be computed in $O(n^{3-\epsilon})$ time.
- APSP on undirected unweighted graphs can be computed in $\tilde{O}(n^{\omega})$ time, where ω denotes the exponent of standard $n \times n$ matrix multiplication [Sei95].
- APSP on directed unweighted graphs can be computed in $\tilde{O}(n^{\mu})$ time, where μ is the common value when $x + \omega(1, x, 1) = 3 - x$ [Zwi02].

Consequently, approximate APSP has garnered significant research interest:

Problem 2 ((α, β)-approximate APSP)**.** *Given a graph $G = (V, E)$, estimate the shortest path distance $\delta(u, v)$ for every pair u, v, such that the estimate satisfies $d(u, v) \leq \delta(u, v) \leq \alpha \cdot d(u, v) + \beta$.*

Dor et al. [DHZ96] demonstrated that computing a $(2-\epsilon)$-approximation for undirected graphs, or any approximation for directed graphs, cannot be faster than boolean matrix multiplication (BMM). $+1$-approximation for undirected unweighted graph is also as hard as BMM [ACIM96]. Current research primarily focuses on the following directions:

- For undirected unweighted graphs:
  - Purely additive approximations ($+2$ and $+2k$).
  - Near-additive approximations $(1 + \epsilon, \beta)$.
  - 2-multiplicative approximations.
  - $(2, 1)$-approximations.
  - Partial case: additive approximation for short paths, or 2-approximation for long paths.
- For undirected weighted graphs:
  - 2-multiplicative approximations.
  - $(2 + \epsilon)$-multiplicative approximations.
  - $(2, \max W)$ and $(2 + \epsilon, \max W)$-approximations.
  - $7/3$ and 3-multiplicative approximations.
- Spanners, emulators, and distance oracles for both unweighted and weighted approximations.

This survey primarily focuses on the fundamental algorithmic frameworks shared across these methods. Specifically, we review the state-of-the-art algorithms and current upper bounds for weighted and unweighted 2-approximations, unweighted $+2k$-approximations, and both additive and multiplicative approximations regarding paths of specific lengths in unweighted graphs.

2 Preliminaries

2.1 Notation

In the subsequent sections, we employ standard notation which is summarized in the table below without further definition:

Notation	Meaning
k-approximation	$(k, 0)$-approximation
$+k$-approximation	$(1, k)$-approximation
$d(u, v)$	Distance from u to v
$\delta(u, v)$	Approximated distance from u to v
$\pi(u, v)$	Sequence of vertices on the shortest path from u to v
$\deg(u)$	Degree of vertex u
n	$\lvert V\rvert$ (Number of vertices)
m	$\lvert E\rvert$ (Number of edges)
$\omega(\alpha, \beta, \gamma)$	Exponent of multiplying an $n^\alpha \times n^\beta$ matrix by an $n^\beta \times n^\gamma$ matrix
ϵ	An arbitrarily small positive real number

In the complexity analysis that follows, we assume $m = \Omega(n)$. We adopt the convention that the O notation suppresses polylogarithmic factors (equivalent to the standard $\tilde{\mathrm{O}}$ notation). This choice is motivated by the fact that complexities involving Fast Matrix Multiplication (FMM) often contain intricate factors. Under this convention, the time complexity for both Breadth-First Search (BFS) and Dijkstra's algorithm is stated as $\mathrm{O}(m)$.

Several algorithms discussed in this survey employ case analysis based on the structural characteristics of the shortest path $\pi(u, v)$. It is important to note that the actual algorithms do not need to explicitly identify which specific case a path belongs to; rather, they simply compute the minimum value over all possible cases.

2.2 Basic Framework for Unweighted Graphs

Aingworth et al. [ACIM96] designed an $\mathrm{O}(n^{5/2})$ +2-approximation algorithm for unweighted graphs. This work introduced a foundational framework for unweighted graph approximation, which can be modified and combined to derive improved algorithms. The algorithm splits the problem of computing $\delta(u, v)$ into two cases:

1. The shortest path $\pi(u, v)$ passes through at least one vertex of degree $\geq n^x$. The strategy involves constructing a hitting set D for $V_1 = \{u \mid \deg(u) \geq n^x\}$, such that every $u \in V_1$ is adjacent to at least one vertex in D. After computing distances for pairs in $D \times V$ using BFS, $\delta(u, v)$ is updated via $\min_{t \in D}\{d(u, t) + d(t, v)\}$.

2. All vertices on $\pi(u, v)$ have degree $< n^x$. In this case, BFS only needs to run on the restricted edge set $E_1 = \{(u, v) \in E \mid u \notin V_1 \vee v \notin V_1\}$, which is sparse.

We refer to the structure in the first case as "the dense method" throughout the rest of this survey.

The hitting set can be constructed via random sampling or a greedy procedure; details are omitted.

Theorem 1. *Both randomized and deterministic algorithms exist to compute a hitting set of size* $O(n^{1-x} \log n)$ *for vertices of degree* $\geq n^x$.

The dense method can be applied hierarchically to speed up the BFS phase. If the maximum degree of vertices on $\pi(u, v)$ lies between a and b, one can run BFS from the hitting set of $\{u \mid \deg(u) \geq a\}$ on the edges $\{(u, v) \in E \mid \deg(u) \leq b \vee \deg(v) \leq b\}$, which takes $O(n^2 b/a)$ time. By partitioning the degree interval into $O(\log n)$ segments $[n^x, 2n^x)$, $[2n^x, 4n^x)$, $\cdots$, the BFS phase then requires only $O(n^2)$ time. Consequently, the bottleneck shifts to enumerating vertices in the hitting set to update δ, a step that can be optimized using FMM techniques.

2.3 Basic Framework for Weighted Graphs

Similar to the unweighted case, the high-level strategy of utilizing "intermediate vertices" to approximate distances is also employed for weighted graphs, though the detailed structures differ. The general framework is as follows:

Sample a set of vertices D of size $O(n^{1-x})$. For each $u \in V$, identify its nearest vertex in D, denoted as $p(u)$. Define the *bunch* of u, denoted $B(u)$, as the set containing u, $p(u)$, and all vertices strictly closer to u than $p(u)$. Additionally, define the *cluster* of v as $C(v) = \{u \mid v \in B(u)\}$. Finally, run Dijkstra's algorithm from all vertices in D.

Dor et al. [DHZ96] demonstrated that the quantity $\min\{d(u, p(u)) + d(p(u), v), d(u, p(v)) + d(p(v), v)\}$ yields a 3-approximation of $d(u, v)$, achieving a time complexity of $O(n^{\frac{7}{3}})$.

Beyond distance computation, constructing the set D is less straightforward than in unweighted graphs. However, as this is not the primary focus of this survey, we state the result directly.

Theorem 2 ([TZ05])**.** *There exists an algorithm that constructs a set* D *of size* $O(n^{1-x})$ *in time* $O(mn^x)$ *(simultaneously computing* $d(u, v)$ *for all* $v \in B(u)$*). W.h.p. the size of every bunch and cluster is bounded by* $O(n^x)$.

3 Weighted 2-Approximation

Dory et al. [DFK+24] presented an $O(n^{\frac{1}{2}}m+n^2)$ algorithm for weighted 2-approximation, which is essentially a simplified exposition of the algorithm originally devised by Baswana and Kavitha [BK10].

The fundamental method for weighted graphs achieves a 2-approximation under specific conditions:

Theorem 3. *If there exists an intermediate vertex on the shortest path* $\pi(u, v)$ *that lies outside both* $B(u)$ *and* $B(v)$*, i.e. if* $\exists x \in \pi(u, v) \setminus (B(u) \cup B(v))$*, then:*

$$\min\{d(u, p(u)) + d(p(u), v), d(u, p(v)) + d(p(v), v)\} \leq 2d(u, v).$$

Proof. Assume WLOG $d(u, x) \leq d(x, v)$; then $d(u, x) \leq d(u, v)/2$. Thus,

$$d(u, p(u)) + d(p(u), v) \leq d(u, p(u)) + d(p(u), u) + d(u, v) \leq 2d(u, x) + d(u, v) \leq 2d(u, v). \qquad \square$$

However, if $\pi(u, v)$ is entirely contained within $B(u) \cup B(v)$, this approximation does not guarantee a ratio of 2. To address this, the hierarchical optimization strategy used in the unweighted case is adapted for this method as follows:

Sample a hierarchy of sets $V = D_0 \supseteq D_1 \supseteq \cdots \supseteq D_k$. At each successive level, the set size approximately halves, while the corresponding bunch size doubles. Let $p_i(u)$ denote the vertex in D_i nearest to u, and let $B_i(u)$ and $C_i(u)$ denote the bunch and cluster of u at level i, respectively.

Consider a pair u, v such that level i is the last level where $\pi(u, v)$ contains intermediate vertices. The bottleneck becomes computing $d(p_i(u), v)$, which costs $\mathrm{O}(m|D_i|)$ time. This cost must be reduced, otherwise the entire optimization renders ineffective.

To reduce the time of Dijkstra's algorithm starting from $p_i(u)$, we reduce the number of edges. Observe that the 2-approximation guarantee (under the assumption $d(u, x) \le d(x, v)$) relies on the inequality $d(p_i(u), v) \le d(p_i(u), u) + d(u, v)$. Therefore, it is not necessary to compute the exact value of $d(p_i(u), v)$.

Theorem 4. *[DFK+24] Let*

$$\delta(p_i(u), x) = \min_{u' \in B_{i+1}(u)} \{d(p_i(u), u) + d(u, u') + w(u', x)\}$$

and

$$E_{i+1} = \{(v, x) \mid w(v, x) \le d(v, p_{i+1}(v))\}.$$

Running Dijkstra from $p_i(u)$ on the edges $(p_i(u) \times V) \cup E_{i+1}$ with the corresponding weights, guarantees that $d(p_i(u), v) \le d(p_i(u), u) + d(u, v)$ for any v such that $\pi(u, v) \subseteq B_{i+1}(u) \cup B_{i+1}(v)$.

Proof. If $v \in B_{i+1}(u)$, the claim is trivial. Otherwise, let $u' \in B_{i+1}(u)$ be the last vertex on $\pi(u, v)$ inside the bunch, and let x be its successor. Since

$$d(p_i(u), u) + d(u, v) = d(p_i(u), u) + d(u, u') + w(u', x) + d(x, v) = \delta(p_i(u), x) + d(x, v),$$

we only need to prove that the path $\pi(x, v)$ consists entirely of edges in E_{i+1}. Assume $\pi(x, v) = [v_k = x, v_{k-1}, \cdots, v_1, v_0 = v]$. Since $x \in B_{i+1}(v)$, simple induction and the triangle inequality show that $x \in B_{i+1}(v_j)$ for all j; consequently, $v_{j+1} \in B_{i+1}(v_j)$, implying the edges satisfy the weight condition. □

From the proof that $\pi(x, v) \subseteq B_{i+1}(v)$, we observe that the computation of $\delta(p_i(u), x)$ is only necessary when $B_{i+1}(u)$ and $B_{i+1}(v)$ partition the path $\pi(u, v)$.

The time complexity analysis consists of two parts:

1. Computing $\delta(p_i(u), x)$ by enumerating u' and x. The time required for level i is

$$\sum_{u'} \deg(u')|C_{i+1}(u')| = \mathrm{O}\left(m \max_{u'} |C_{i+1}(u')|\right).$$

2. Running Dijkstra from D_i, taking $|D_i| \cdot (n + |E_{i+1}|)$ time. To analyze $|E_{i+1}|$, we first assume D_{i+1} is uniformly sampled. For a vertex u, consider its neighbors $v_1, \cdots, v_{\deg(u)}$ sorted by edge weight in ascending order. Each neighbor belongs to D_{i+1} with probability $|D_{i+1}|/n$. Thus, the expected number of edges added to E_{i+1} is

$$\sum_{j=1}^{\deg(u)} \left(1 - \frac{|D_{i+1}|}{n}\right)^j \le \frac{n}{|D_{i+1}|}.$$

The size of $C_{i+1}(u')$ is bounded by the guarantee of Theorem 2, while the assumption in the second part is realized by a slight adjustment: uniformly sample a hierarchy $V = S_0 \supseteq S_1 \supseteq \cdots \supseteq S_k$ such that $|S_{i+1}| = |S_i|/2$, and then merge D_i with S_i.

The only scenario not handled by the above procedure is when $B_k(u) \cup B_k(v)$ fails to cover $\pi(u, v)$. In this case, simply running Dijkstra from D_k on the original graph and performing the standard update suffices. In conclusion, the overall time complexity is

$$\sum_{i=0}^{k-1} \left(m \cdot \frac{n}{|D_{i+1}|} + \frac{n^2}{|D_{i+1}|} |D_i| \right) + m|D_k|.$$

If $|D_k| = \mathrm{O}(n^{1-x})$, the time complexity becomes $\mathrm{O}(mn^x + n^2 + mn^{1-x})$, which is balanced to $\mathrm{O}(n^{\frac{1}{2}}m + n^2)$ when $x = 1/2$.

The above algorithm can be further optimized using fast matrix multiplication (FMM), albeit at the cost of relaxing the approximation ratio to $2 + \epsilon$:

Theorem 5 ([EN22]). *$(1+\epsilon)$-approximate MSSP from n^r sources can be computed in $\mathrm{O}(m^{1+o(1)} + n^{\omega(1,r,1)}\epsilon^{-\mathrm{O}(1)} \log W)$ time by a randomized algorithm.*

By substituting the Dijkstra computations in the final step of the aforementioned algorithm with this approximation scheme, the time complexity can be balanced to $\mathrm{O}(n^{2.214}\epsilon^{-\mathrm{O}(1)} \log W)$ for dense graphs (where $m = \Theta(n^2)$).

4 Unweighted 2-Approximation

Determining whether 2-approximation algorithms can be faster than +2-approximation algorithms in unweighted graphs has been a long-standing open problem. Significant progress has been made in recent years, driven by FMM techniques. Dory et al. [DFK+24] and Saha and Ye [SY23] independently and nearly simultaneously proposed similar 2-approximation algorithms running in $\mathrm{O}(n^{2.032})$ time.

These algorithms leverage existing additive approximation techniques to achieve a 2-approximation for long paths.

Theorem 6 ([DHZ96]). *For any even integer $k \geq 4$, there exists an algorithm that computes a $+k$-approximation in $\mathrm{O}(\min\{n^{2-\frac{2}{k+2}}m^{\frac{2}{k+2}}, n^{2+\frac{2}{3k-2}}\})$ time.*

Additive approximation algorithms, including the one mentioned above, will be discussed in detail in the subsequent section.

Observing that a $+k$-approximation yields a 2-approximation whenever the shortest path distance is at least k, a $+\log n$-approximation can be computed in $\mathrm{O}(n^2)$ time. Motivated by this, the algorithm classifies pairs based on both vertex degree and path length:

1. The shortest path $\pi(u, v)$ passes through at least one vertex of degree $\geq n^x$.
   - 1.1. $d(u, v) \geq \log n$. Execute the $+\log n$-approximation algorithm, which takes $\mathrm{O}(n^2)$ time.
   - 1.2. $d(u, v) < \log n$. Using the hierarchical optimization for the dense method, the computational bottleneck becomes evaluating $\delta(u, v) = \min_{t \in D}\{d(u, t) + d(t, v)\}$. Since bounded $(\min, +)$-matrix multiplication can be computed in $\mathrm{O}(Mn^\omega)$ time (where M is the bound on values), this step requires $\mathrm{O}(n^{\omega(1,1-x,1)})$ time.

2. The shortest path $\pi(u, v)$ consists entirely of vertices with degree $< n^x$. In this case, run the weighted graph 2-approximation algorithm (discussed in the previous section), which takes $O(n^{\frac{1}{2}}m + n^2)$ time.

The overall time complexity is

$$O\left(n^{\omega(1,1-x,1)} + n^{\frac{3}{2}+x}\right).$$

Balancing the two terms yields a time complexity of $O(n^{2.032})$. Whether a 2-approximation can be computed in $O(n^2)$ time remains an open problem.

5 Unweighted Additive Approximation

5.1 Combinatorical Algorithms

Dor et al. [DHZ96] proposed a graph construction that enables the computation of a $+2$-approximation on sparse graphs faster than the dense method discussed earlier. This construction consequently yields the result presented in Theorem 6.

Theorem 7. *Let D be a hitting set for all vertices of degree $\geq a$. Let E^* be the set of edges connecting every vertex of degree $\geq a$ to one of its neighbors in D, and let $E' = \{(u, v) \in E \mid \deg(u) < a \vee \deg(v) < a\}$. If the maximum degree of the vertices on the shortest path $\pi(u, v)$ is between a and b, then running Dijkstra from u on the edge set $(\{u\} \times D) \cup E^* \cup E'$ yields a $+2$-approximation for $d(u, v)$, where the weight of edges in $\{u\} \times D$ is the actual distance in the original graph.*

Proof. Let x be the last vertex on $\pi(u, v)$ with degree $\geq a$. By definition, x is adjacent to some $y \in D$. The constructed edge set contains the edge (u, y) (via the direct connection $\{u\} \times D$), the edge (y, x) (via E^*), and the path $\pi(x, v)$ (since all vertices on $\pi(x, v)$ have degree $< a$, the edges belong to E'). Thus:

$$\delta(u, v) \leq d(u, y) + 1 + d(x, v) \leq d(u, x) + 1 + 1 + d(x, v) = d(u, v) + 2. \qquad \square$$

In the following discussion, we refer to this approach as "the sparse method". Similar to the dense method, this approach can be applied hierarchically; however, stacking it k times yields a $+2k$-approximation.

The $O(n^{7/3})$ time algorithm for $+2$-approximation is obtained by stacking one level of the dense method over one level of the sparse method. Here, we directly introduce the general algorithm.

To achieve a time complexity of $O(n^{2-\frac{2}{k+2}}m^{\frac{2}{k+2}})$ for even $k \geq 2$, consider degree thresholds $s_1 > \cdots > s_{\frac{k}{2}+1} = 1$, where $s_i = (m/n)^{1-\frac{i}{k/2+1}}$. Let $V_i = \{u \mid \deg(u) \geq s_i\}$ and $E_i = \{(u, v) \in E \mid \deg(u) < s_{i-1} \vee \deg(v) < s_{i-1}\}$ (with $E_1 = E$). Let D_i be a hitting set for V_i, and let E_i^* denote the corresponding hitting edges. For $i = 1, \cdots, k/2 + 1$, run Dijkstra starting from every $x \in D_i$ on the edge set $(\{x\} \times D_{i-1}) \cup E_{i-1}^* \cup E_i$.

By enforcing $D_{\frac{k}{2}+1} = V$, the procedure yields the final result upon completion. A simple induction shows that $\delta(x, v)$ for $x \in D_i$ provides a $+2(i-1)$-approximation. We omit the full time complexity analysis.

To achieve $O(n^{2+\frac{2}{3k-2}})$ time for even $k \geq 4$, we require a carefully designed hybrid of the dense and sparse methods. Set degree thresholds $s_0 > s_1 > s_2 > \cdots > s_{\frac{k}{2}+1}$, and define V_i, E_i, D_i, E_i^* accordingly. The algorithm proceeds as follows:

Level 0. Process vertices with degree $\geq s_0$ using the hierarchical dense method. Specifically, for every j, run Dijkstra from the hitting set of vertices with degree $\geq 2^j s_0$ on edges $\{(u,v) \in E \mid \deg(u) < 2^{j+1}s_0 \vee \deg(v) < 2^{j+1}s_0\}$. This prepares them for +2-approximation queries in later steps.

Level 1. Run Dijkstra from D_1 on E_1.

Level 2. Run Dijkstra from each $u \in D_2$ on the edges $(\{u\} \times D_1) \cup E_1^* \cup E_2$. Subsequently, for every $v \in D_3$, update $\delta(u,v)$ using $\min_{t \in D_0}\{d(u,t) + d(t,v)\}$. Here, D_0 represents the union of the hitting sets from all sub-layers of level 0.

Level ≥ 3. Run Dijkstra from each $u \in D_i$ on $(\{u\} \times D_{i-1}) \cup E_{i-1}^* \cup E_i$, for $i = 3, \cdots, k/2+1$.

Theorem 8. *The algorithm described above computes a $+k$-approximation.*

Proof. It suffices to prove that for $u \in D_2$ and $v \in D_3$, $\delta(u,v)$ is a +2-approximation; the remainder of the proof follows the logic of the previous algorithm. We consider three cases:

1. $\pi(u,v)$ contains a vertex of degree $\geq s_0$. In this case, $\min_{t \in D_0}\{d(u,t) + d(t,v)\}$ yields a +2-approximation.
2. The maximum degree of vertices on $\pi(u,v)$ lies between s_1 and s_0. The proof is identical to that of Theorem 7.
3. All vertices on $\pi(u,v)$ have degree $< s_1$. In this case, $\pi(u,v)$ is entirely covered by E_2. □

The time complexity is given by:

$$O\left(\frac{n^3}{s_0 s_2 s_3} + \sum_{i=1}^{k/2} \frac{n^2 s_i}{s_{i+1}}\right).$$

By choosing $s_i = s_0^{1-\frac{i}{k/2+1}}$ and $s_0 = n^{\frac{k+2}{3k-2}}$, we achieve the minimum complexity of $O(n^{2+\frac{2}{3k-2}})$.

5.2 Faster Algorithms with FMM

Although no subcubic algorithm currently exists for exact $(\min,+)$-matrix multiplication, improved results have been achieved for specific matrix classes and approximation scenarios:

Theorem 9 ([Dü23], generalized by [SY23])**.** *The $(\min,+)$-product of two integer matrices of size $n^\alpha \times n^\beta$ and $n^\beta \times n^\gamma$, where the second matrix is row-monotone and its entries are bounded by $O(n^\mu)$, can be computed in $O(n^{\frac{\alpha+\beta+\mu+\omega(\alpha,\beta,\gamma)}{2}})$ time.*

Theorem 10 ([Zwi02])**.** *A $(1+\epsilon)$-approximate $(\min,+)$-product of two matrices of size $n^\alpha \times n^\beta$ and $n^\beta \times n^\gamma$ with integer entries bounded by M can be computed in $O(n^{\omega(\alpha,\beta,\gamma)}\epsilon^{-1}\log W)$ time.*

The result of Theorem 9 can be leveraged to optimize the computation of $\min_{t \in D_0}\{d(u,t) + d(t,v)\}$. This is achieved by reordering all vertices according to their DFS order (entry times). By adding the entry timestamp in_v to $d(t,v)$, the bounded-difference matrix is transformed into a row-monotone matrix with values in the range of $O(n)$.

For $+2$-approximation, applying Theorem 9 to the $O(n^{7/3})$ approach proposed by Dor et al. [DHZ96] yields an $O(n^{2.259})$ algorithm. The exponent corresponds to the common value of both sides of the following equation when equality holds:

$$\frac{3 - 2x + \omega(1, 1-2x, 1)}{2} = 2 + x.$$

Using Theorem 10, we obtain a $(1+\epsilon, 2)$-approximation in $O(n^{2.152})$ time, where the exponent corresponds to the equation:

$$\omega(1, 1-2x, 1) = 2 + x.$$

Similarly, this approach extends to $+k$-approximation (for even $k \geq 4$). Let $s_0 = n^x$. Using Theorem 9, we obtain a $+k$-approximation whose time complexity exponent corresponds to the following equation:

$$\frac{1}{2}\left(3 - \frac{k}{k/2+1}x + \omega\left(1 - \frac{k/2-1}{k/2+1}x, 1-x, 1 - \frac{k/2-2}{k/2+1}x\right)\right) = 2 + \frac{x}{k/2+1}.$$

The complexity exponent for $(1+\epsilon, k)$-approximation is given by the equation:

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

5.3 Newest Improvement

Jin et al. [JKSW25] recently achieved improved time bounds for additive approximations by modifying the structure of the dense method. The high-level idea is to decompose the graph into several bounded-diameter components. This allows the dense method to utilize matrix multiplication with constant-bounded entries, rather than the more expensive bounded-difference matrix multiplication. The algorithm relies primarily on the following two theorems (proofs omitted):

Theorem 11. *[JKSW25] Given a graph G and a parameter d, one can deterministically partition the vertices into $V = R \cup \bigcup_i H_i$ in linear time such that for every i, $d \leq |H_i| \leq 2d$ and $\max_{u,v \in H_i} d(u,v) \leq 4$, while for every $u \in R$, $\deg(u) < d$.*

Theorem 12. *[JKSW25] The $(\min, +)$-product of two integer matrices of size $n^\alpha \times n^\beta$ and $n^\beta \times n^\gamma$, where every column of the second matrix has a value range bounded by M, can be computed in $O(Mn^{\omega(\alpha,\beta,\gamma)})$ time.*

For all pairs (u,v) where the maximum degree on the shortest path $\pi(u,v)$ lies between s and $2s$, a $+2$-approximation can be computed as follows: First, obtain a hitting set D for vertices of degree $\geq s$, and run Dijkstra from D on the restricted edge set $\{(u,v) \in E \mid \deg(u) < 2s \vee \deg(v) < 2s\}$. Next, using an appropriate parameter d, apply Theorem 11 to construct a partition $V = R \cup \bigcup_i H_i$. For $u \in V$ and $v \in H_i$, $\delta(u,v)$ can be computed using Theorem 12 on matrices of size $n \times |D|$ and $|D| \times |H_i|$, observing that the range of each column in the second matrix is bounded by the diameter of H_i, which is constant.

To compute $\delta(u,v)$ for $u \in V$ and $v \in R$, we run Dijkstra on the edges $(u \times \bigcup_i H_i) \cup \{(u',v') \mid u' \in R\}$. By analyzing the structure of $\pi(u,v)$, it can be shown that this procedure yields $\delta(u,v) \leq d(u,v)+2$.

Assuming $s = n^x$ and $d = n^y$, the time complexity of the above procedure is:

$$\mathrm{O}\left(n^{1-y+\omega(1,1-x,y)} + n^{2+y}\right).$$

By taking the faster of this algorithm and the $\mathrm{O}(n^{3/2}m^{1/2})$ algorithm from [DHZ96] (which performs better on sparse graphs), and optimizing over s, the overall complexity becomes:

$$\mathrm{O}\left(\max_{x\in[0,1]} \min\left\{n^{2+\frac{x}{2}}, \min_{y\in[0,x]}\left\{n^{1-y+\omega(1,1-x,y)} + n^{2+y}\right\}\right\}\right).$$

This balances to $\mathrm{O}(n^{2.255})$. By employing a more sophisticated matrix multiplication design that calculates approximate distances for $u,v \in \bigcup_i H_i$ first and then incorporates vertices in R, the complexity can be further improved to $\mathrm{O}(n^{2.226})$.

By stacking this approach with the sparse method, the algorithm extends to $+k$-approximation for even $k \geq 4$. Following the procedure above, we obtain a $+2$-approximation for every pair $u \in D', v \in V$ where the maximal degree on $\pi(u,v)$ is between s and $2s$ (where D' is a hitting set for vertices of degree $\geq s' = n^{x'}$). This step requires $n^{1-y+\omega(1-x',1-x,y)}$ time. Combined with $k/2-1$ levels of the sparse method, which take $\mathrm{O}(n^{2+\frac{x'}{k/2-1}})$ time, we achieve a $+k$-approximation. The overall time complexity is:

$$\mathrm{O}\left(\max_{x\in[0,1]} \min_{x'\in[0,x]} \min\left\{n^{2+\frac{x}{k/2+1}}, \min_{y\in[0,x]}\left\{n^{1-y+\omega(1-x',1-x,y)} + n^{2+y}\right\} + n^{2+\frac{x'}{k/2-1}}\right\}\right).$$

Setting $x' = (k/2-1)x/(k/2+1)$ and $y = (k/2)x/(k/2+1)$ to balance the terms, the final exponent is determined by:

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

6 Unweighted Approximation for Certain Distances

6.1 Additive Approximation for Short Paths

Roditty [Rod23] was the first to demonstrate that a 2-approximation can be computed faster than a $+2$-approximation. He presented a combinatorial algorithm that utilizes a faster subroutine for computing a $+2$-approximation for distances $d(u,v) \leq 3$.

Theorem 13 ([Rod23])**.** *A $+2$-approximation for pairs (u,v) with distance $d(u,v) \leq 3$ can be computed in $\mathrm{O}(\min\{n^{\frac{5}{3}}m^{\frac{1}{3}}, n^{\frac{9}{4}}\})$ time.*

Proof. Set degree thresholds $s_1 > s_2 > s_3 > s_4 = 1$, and define the sets V_i, E_i, D_i, E_i^* accordingly (as in the previous section). For each i, run BFS from D_i on the edge set E_i to obtain distances $\delta(\cdot,\cdot)$. We focus on the case where $d(u,v) = 3$ (the case $d(u,v) \leq 2$ is simpler). Let the shortest path be $\pi(u,v) = [u,u_2,v_2,v]$. We distinguish cases based on the structure of $\pi(u,v)$:

1. If $\pi(u,v)$ contains a vertex from V_1, update $d(u,v)$ using $\min_{t\in D_1}\{d(u,t)+d(t,v)\}$.

2. Otherwise, if $u \in V_2$ or $v \in V_2$: Assume WLOG $u \in V_2$ and is adjacent to $t \in D_2$. Then $1 + \delta(t, v) = 1 + d(t, v)$ provides a $+2$-approximation.

3. Otherwise, if $u \in V_4$ and $v \in V_4$: Assume WLOG $\deg(v_2) \geq \deg(u_2)$, and let v_2 be adjacent to a hitting set vertex t in the corresponding level. Run Dijkstra on the edge set $(\{u\} \times V) \cup E^* \cup E_4$. Then,
$$\delta(u, v) \leq \delta(u, t) + 1 + 1 \leq d(u, v_2) + 3 = d(u, v) + 2.$$

4. Otherwise, we have $u \in V_3$ or $v \in V_3$, and both $u, v \in V_3 \cup V_4$. Assume WLOG $u \in V_3$ and is adjacent to $t \in D_3$. In this case, compute BFS from u in $G[V \setminus V_1]$ up to distance 2. Update $\delta(t, x)$ using $\delta(u, x) + 1$ for every x such that $\delta(u, x) \leq 2$. Then, run Dijkstra on $(\{t\} \times V) \cup E^* \cup E_3$ for every $t \in D_3$ and finally update $\delta(u, v)$ with the corresponding $\delta(t, v) + 1$. Thus,
$$\delta(u, v) \leq \delta(t, v) + 1 \leq \delta(t, v_2) + 2 \leq \delta(u, v_2) + 3 = d(u, v_2) + 3 = d(u, v) + 2. \qquad \square$$

The algorithm described above requires time
$$\mathrm{O}\left(\frac{n^3}{s_1} + \sum_{i=1}^{4} \frac{n^2 s_i}{s_{i-1}} + n s_1 s_2\right).$$
By choosing different assignments for s_i, we can achieve a complexity of either $\mathrm{O}(n^{\frac{9}{4}})$ or $\mathrm{O}(n^{\frac{5}{3}} m^{\frac{1}{3}})$.

Through a more complex case analysis, and by utilizing the trick of running BFS up to 2 layers from u, [Rod23] demonstrated that for even $k \geq 4$, a $+k$-approximation for pairs (u, v) with distance $\leq k + 2$ can be computed in $\mathrm{O}(n^{2 - \frac{2}{k+4}} m^{\frac{2}{k+4}})$ time.

Since a $+k$-approximation for pairs with distance $\geq k$ naturally serves as a 2-approximation for such pairs, combining this with the algorithm above yields an $\mathrm{O}(\min\{n^{\frac{5}{3}} m^{\frac{1}{3}}, n^{\frac{9}{4}}\})$ time 2-approximation algorithm. Generalizing this, we obtain an $\mathrm{O}(\min\{n^{2 - \frac{2}{k+4}} m^{\frac{2}{k+4}}, n^{2 + \frac{2}{3k-2}}\})$ time 2-approximation algorithm for pairs of distance $\geq k$.

6.2 2-Approximation for Long Paths

Saha and Ye [SY23] modified the $+k$-approximation algorithm introduced in Section 5.1 to create a hybrid of $+k$-approximation and 2-approximation, effectively resulting in a 2-approximation for pairs of distance $\geq k$.

Recall the algorithm with time complexity $\mathrm{O}(n^{2 + \frac{2}{3k-2}})$. At level 2, remove the step that runs Dijkstra from $u \in D_2$ on $(\{u\} \times D_1) \cup E_1^* \cup E_2$. Instead, add the following step: run the weighted graph 2-approximation algorithm on E_1. With this modification, this $(k/2 + 1)$-level algorithm computes a $+k$-approximation for all pairs where $\pi(u, v)$ contains a vertex of degree $\geq s_0$. The weighted graph 2-approximation handles the case where all vertices on $\pi(u, v)$ have degree $< s_0$.

After the modification, the time complexity becomes
$$\mathrm{O}\left(n^{\frac{3}{2}} s_0 + \frac{n^3}{s_0 s_2 s_3} + \sum_{i=2}^{k/2} \frac{n^2 s_i}{s_{i+1}}\right),$$
which can be balanced to $\mathrm{O}(n^{2 + \frac{1}{2k-2}})$.

FMM optimization can also be applied. Let $s_0 = n^x$, $s_2 = n^y$, and $s_i = s_2^{1-\frac{i-2}{k/2-1}}$. The complexity is given by:

$$\mathrm{O}\left(n^{\frac{3}{2}+x} + n^{\omega(1-y,1-x,1-\frac{k/2-2}{k/2-1}y)} + n^{2+\frac{y}{k/2-1}}\right).$$

Balancing the terms yields a complexity exponent equal to the value where the following equation holds:

$$\omega\left(1-\left(\frac{k}{2}-1\right)\left(x-\frac{1}{2}\right), 1-x, 1-\left(\frac{k}{2}-2\right)\left(x-\frac{1}{2}\right)\right) = \frac{3}{2}+x.$$

7 Numerical Values of Time Complexity Exponents for Algorithms Utilizing FMM

The following table presents a comparison of the time complexity exponents for the algorithms for dense graphs ($m = \Theta(n^2)$) utilizing FMM discussed in the preceding sections. The numerical solutions were obtained using the Complexity Term Balancer [Bra].

k		**2**	**4**	**6**	**8**
$+k$**-Approx**	[DHZ96]	2.3334	2.2000	2.1250	2.0910
	[Dü23]	2.2590			
	[SY23]		2.1550	2.1030	2.0773
	[JKSW25]	2.2255	2.1462	2.1026	
$(1+\epsilon, k)$**-Approx**	[DFK+24]	2.1514	2.1186	2.0978	2.0835
	[SY23]		2.0929	2.0578	2.0421
2**-Approx for** $d \geq k$	[SY23]	2.0311	2.0190	2.0105	2.0073

8 Experiments

We implemented the $+k$-approximation algorithms proposed by Dor et al. [DHZ96] as described in Section 5.1 using C++. The implementation includes the $\mathrm{O}(n^{2-\frac{2}{k+2}}m^{\frac{2}{k+2}})$ algorithm for sparse graphs and the $\mathrm{O}(n^{2+\frac{2}{3k-2}})$ algorithm for dense graphs (incorporating the special case of $\mathrm{O}(n^{7/3})$ for +2-approximation). We benchmarked these algorithms against a brute-force BFS baseline.

Two implementation details warrant mention. First, we replaced Dijkstra's algorithm with a multi-source BFS. This optimization is valid because only the edges incident to the source vertices may have weights other than 1. Second, for the construction of the hitting set, we employed a deterministic greedy strategy: at each step, we select the vertex that covers the maximum number of remaining vertices with degrees exceeding the threshold. While it is theoretically straightforward to prove that the size of the resulting set remains $\mathrm{O}(n \log n/s)$, in practice, the size is significantly smaller than this bound.

We generated two categories of graphs, both with $n = 5000$ vertices. The code was compiled with `-O2` optimization and run on Intel® Core™ i7 processor 14650HX with Turbo disabled. We report the average execution time over three trials, excluding I/O operations. The experimental results are presented below:

Dataset 1: Each edge exists with a probability of 1/3, resulting in $m \approx 4.16 \times 10^6$ edges. Baseline BFS time: 32.79s.

Algorithm	+2	+4	+6	+8
Sparse	1.30s	1.39s	1.39s	1.41s
Dense	1.45s	1.37s	1.41s	1.44s

Dataset 2: Vertex degrees are first sampled from a bounded power law distribution with $\alpha = 0.5$. Edges are then randomly connected while respecting these degree constraints, resulting in $m \approx 2.12 \times 10^6$ edges. Baseline BFS time: 17.78s.

Algorithm	+2	+4	+6	+8
Sparse	1.17s	1.11s	1.10s	1.12s
Dense	0.62s	0.59s	0.61s	0.62s

For Dataset 1, we observed that almost no vertices had degrees below the maximum threshold; consequently, the hierarchical layering was effectively inactive, leading to insignificant variations in the experimental results. For Dataset 2, it is notable that as the approximation ratio increases, the execution time first decreases and then increases. This behavior occurs because, although the theoretical complexity exponent decreases, the constant factors in the implementation increase. Additionally, the sparse algorithm performed slower than the dense algorithm in this setting.

9 References

[ACIM96] Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani, *Fast estimation of diameter and shortest paths (without matrix multiplication)*, Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM/SIAM, 1996, pp. 547–553.

[BK10] Surender Baswana and Telikepalli Kavitha, *Faster algorithms for all-pairs approximate shortest paths in undirected graphs*, SIAM Journal on Computing **39** (2010), no. 7, 2865–2896.

[Bra] Jan van den Brand, *Complexity term balancer*, `jvdbrand.com/complexity/`, Tool to balance complexity terms depending on fast matrix multiplication.

[DFK+24] Michal Dory, Sebastian Forster, Yael Kirkpatrick, Yasamin Nazari, Virginia Vassilevska Williams, and Tijn de Vos, *Fast 2-approximate all-pairs shortest paths*, Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024 (David P. Woodruff, ed.), SIAM, 2024, pp. 4728–4757.

[DHZ96] Dorit Dor, Shay Halperin, and Uri Zwick, *All pairs almost shortest paths*, 37th Annual Symposium on Foundations of Computer Science, FOCS 1996, Burlington, Vermont, USA, 14-16 October, 1996, IEEE Computer Society, 1996, pp. 452–461.

[Dü23] Anita Dürr, *Improved bounds for rectangular monotone min-plus product and applications*, Information Processing Letters **181** (2023), 106358.

[EN22] Michael Elkin and Ofer Neiman, *Centralized, Parallel, and Distributed Multi-Source Shortest Paths via Hopsets and Rectangular Matrix Multiplication*, 39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022) (Dagstuhl, Germany) (Petra Berenbrink and Benjamin Monmege, eds.), Leibniz International Proceedings in Informatics (LIPIcs), vol. 219, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, pp. 27:1–27:22.

[JKSW25] Ce Jin, Yael Kirkpatrick, Michał Stawarz, and Virginia Vassilevska Williams, *Improved additive approximation algorithms for apsp*, 2025.

[Rod23] Liam Roditty, *New algorithms for all pairs approximate shortest paths*, Proceedings of the 55th Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC 2023, Association for Computing Machinery, 2023, p. 309–320.

[Sei95] R. Seidel, *On the all-pairs-shortest-path problem in unweighted undirected graphs*, Journal of Computer and System Sciences **51** (1995), no. 3, 400–403.

[SY23] Barna Saha and Christopher Ye, *Faster approximate all pairs shortest paths*, 2023.

[TZ05] Mikkel Thorup and Uri Zwick, *Approximate distance oracles*, J. ACM **52** (2005), no. 1, 1–24.

[Zwi02] Uri Zwick, *All pairs shortest paths using bridging sets and rectangular matrix multiplication*, J. ACM **49** (2002), no. 3, 289–317.