Faster Approximation of Distances in Graphs

Piotr Berman Shiva Prasad Kasiviswanathan
Department of Computer Science and Engineering
Pennsylvania State University
e-mail: {berman,kasivisw}@cse.psu.edu

Abstract

Let $G = (V, E)$ be an weighted undirected graph on n vertices and m edges, and let d_G be its shortest path metric. We present two simple deterministic algorithms for approximating all-pairs shortest paths in G. Our first algorithm runs in $\tilde{O}(n^2)$ time, and for any $u, v \in V$ reports distance no greater than $2d_G(u, v)+h(u, v)$. Here, $h(u, v)$ is the largest edge weight on a shortest path between u and v. The previous algorithm, due to Baswana and Kavitha that achieved the same result was randomized. Our second algorithm for the all-pairs shortest path problem uses Boolean matrix multiplications and for any $u, v \in V$ reports distance no greater than $(1+\epsilon)d_G(u, v)+2h(u, v)$. The currently best known algorithm for Boolean matrix multiplication yields an $O(n^{2.24+o(1)}\epsilon^{-3}\log(n\epsilon^{-1}))$ time bound for this algorithm. The previously best known result of Elkin with a similar multiplicative factor had a much bigger additive error term.

We also consider approximating the diameter and the radius of a graph. For the problem of estimating the radius, we present an almost 3/2-approximation algorithm which runs in $\tilde{O}(m\sqrt{n} + n^2)$ time. Aingworth, Chekuri, Indyk, and Motwani used a similar approach and obtained analogous results for diameter approximation. Additionally, we show that if the graph has a small separator decomposition a 3/2-approximation of both the diameter and the radius can be obtained more efficiently.

1 Introduction

Consider the all-pairs shortest path (henceforth, referred to as APSP) problem. Given a graph[§] $G = (V, E)$ on $|V| = n$ and $|E| = m$ edges, the goal is to compute the distances between all pairs of vertices. The currently best known algorithm for the APSP problem with real non-negative weights has $O(mn + n^2 \log\log n)$ running time [24]. The currently best known upper bound on the worst case complexity is $O(n^3 \log^3\log n/\log^2 n)$ due to a very recent paper by Chan [7]. For the simpler case of unweighted graphs, using fast matrix multiplication, Galil and Margalit [17, 18], and Seidel [25] have obtained algorithms that run in $\tilde{O}(n^\omega)$[¶] time, where ω denotes the exponent of (square) matrix multiplication algorithm used. The currently best known matrix multiplication algorithm of Coppersmith and Winograd [11] results in $\omega < 2.376$.

The existing lower bounds on the complexity of the APSP is the trivial $\Omega(n^2)$. The disparity between upper and lower bounds have led a recently growing interest in designing efficient algorithms for all-pairs approximate shortest paths. Let d_G be the shortest path metric induced by the connected graph G on its vertices. For a path P in G, let $l(P)$ denote the length of the P and let $h(P)$ denote the largest edge length (weight of the heaviest edge) in P. Let $Path(u, v)$ be the set of all paths from u to v in G. We say that an algorithm is a (a, b)-*approximation* of the APSP problem if for any pair of vertices $(u, v) \in V \times V$, the estimate $\delta(u, v)$ produced by the algorithm satisfies:

$$d_G(u, v) \le \delta(u, v) \le \min_{P \in Path(u,v)} \{a \cdot l(P) + b \cdot h(P)\}.$$

[§] Throughout the paper graphs are undirected unless mentioned otherwise.

[¶] The notation $\tilde{O}(f) \equiv O(f \operatorname{poly}\log f)$.

The multiplicative term a is referred to as the *stretch factor* and $b \cdot h(P)$ denotes the *additive error*. Note that for unweighted graphs the additive error is just b.

Over the last decade many algorithms have been designed for this problem that achieve sub-cubic time and/or sub-quadratic space. Here we state a few of the relevant results. For a more comprehensive overview of results refer to the survey by Zwick [28].

For unweighted graphs, Aingworth *et al.* [1] used an ingenious method to obtain an $\tilde{O}(n^{5/2})$ time $(1,2)$-approximation algorithm. Dor *et al.* designed an algorithm which for every even $t > 2$ runs in $\tilde{O}(\min\{n^{2-\frac{2}{t+2}} m^{\frac{2}{t+2}}, n^{2+\frac{2}{3t-2}}\})$ time and is a $(1,t)$-approximation. With multiplicative factors, Baswana *et al.* [3] provided a $(2,1)$-approximation algorithm that takes an expected $\tilde{O}(m^{2/3} n + n^2)$ time. For a $(2,3)$-approximation, they could improve the running time to expected $\tilde{O}(n^2)$ time. Recently, for arbitrarily small $\zeta, \rho, \epsilon > 0$, Elkin [14] designed a $(1+\epsilon, \beta(\zeta,\rho,\epsilon))$-approximation algorithm that runs in $O(mn^\rho + n^{2+\zeta})$ time. The constant β depends on ζ as $(1/\zeta)^{\log 1/\zeta}$, depends inverse exponentially on ρ, and depends inverse polynomially on ϵ.

For weighted graphs, Cohen and Zwick [9] building on the work of Dor *et al.* [12] provided fast algorithms with stretch factor 2, 7/3, and 3. In a recent improvement, Baswana and Kavitha [4] provided faster algorithm for the same stretch factors. They present algorithms that run in expected $\tilde{O}(\sqrt{m}n^{3/2})$ time and expected $\tilde{O}(n^{7/3})$ time for stretch factors of 2 and 7/3 respectively. Furthermore, they designed an expected $\tilde{O}(n^2)$ time $(2,1)$-approximation algorithm. Also on weighted graphs, Elkin [14] presented an $O(mn^\rho + n^{2+\zeta})$ time algorithm that for any $u, v \in V$ reports distance bounded by $(1+\epsilon)d_G(u,v) + W \cdot \beta(\zeta,\rho,\epsilon)$. Here, W is the ratio between the heaviest and lightest edge in the graph.

Diameter and radius are two important parameters of a graph. The eccentricity of a vertex is defined as the maximum distance between the vertex and any other vertex. The maximum eccentricity is the graph diameter and the minimum eccentricity is the graph radius. Both diameter and radius can be found by solving the APSP problem. Recently, Chan [6] has shown that diameter of unweighted directed graphs can be obtained in expected $O(mn \log^2 \log n / \log n + n^2 \log n / \log\log n)$ time. For some simpler classes of graphs like trees, outer-planar graphs, interval graphs, and distance-hereditary graphs fast algorithms are known for finding the diameter [13, 16, 23]. For general graphs however it is not clear whether these parameters can be obtained faster than obtaining the whole distance matrix.

On the approximation front, it is easy to estimate both the diameter and radius within a ratio 2 by performing a single-source shortest path from any vertex in the graph. No better result was known until Aingworth *et al.* [1] designed a 3/2-approximation algorithm for the diameter running in $\tilde{O}(m\sqrt{n} + n^2)$ time. Also recently, Boitmanis *et al.* [5] gave algorithms for approximating the diameter and the radius in $\tilde{O}(m\sqrt{n})$ time. The results are produced within an additive error of $O(\sqrt{n})$.

The situation seems no better even if we restrict our attention to the family of separable graphs (i.e., graphs with a small sized vertex separator). The family of separable graphs contains planar graphs, graphs with no fixed minor, k-overlap graphs, bounded tree-width graphs (for some of these results refer [2, 21, 22]). Even for the generally well-studied planar graphs the only known result seems to be that Eppstein [15], who has shown that if the planar graph has a constant bound on diameter, then the exact diameter can be found in linear time.

1.1 Our Contributions

We design two algorithms for the problem of APSP for weighted undirected graphs. Our first algorithm runs in $\tilde{O}(n^2)$ time and is a $(2,1)$-approximation. Earlier, Thorup and Zwick [27] have shown than for any $t < 3$, a data structure that answers t-approximate distance query in constant time must occupy $\Theta(n^2)$ space. This automatically implies a lower bound of $\Omega(n^2)$ on the

Approximation Ratio	Time	Notes
$(2,1)$	expected $\tilde{O}(n^2)$	Baswana and Kavitha [4]
stretch $= 1+\epsilon$, additive error $= W \cdot \beta(\zeta,\rho,\epsilon)$	$O(mn^\rho + n^{2+\zeta})$	Elkin [14]
$(2,1)$	$\tilde{O}(n^2)$	**this paper**
$(1+\epsilon,2)$	$O(n^{2.24+o(1)}\epsilon^{-3}\log(n\epsilon^{-1}))$	**this paper**

Figure 1: State-of-the-art for approximating APSP for weighted undirected graphs.

Problem	Graph Class	Factor	Time	Notes
Diameter	weighted, directed	$3/2$	$\tilde{O}(m\sqrt{n}+n^2)$	Aingworth *et al.* [1]
Radius	unweighted, undirected	$\approx 3/2$	$\tilde{O}(m\sqrt{n}+n^2)$	**this paper**
Diameter	weighted, directed, separable	$3/2$	$\tilde{O}(n^{1+2\mu}+n^{3\mu})$	**this paper**
Radius	weighted, undirected, separable	$\approx 3/2$	$\tilde{O}(n^{1+2\mu}+n^{3\mu})$	**this paper**

Figure 2: State-of-the-art (with small multiplicative factors) for the diameter, radius approximation.

space, therefore on time complexity of any $(2,0)$-approximation algorithm. Compared to the $(2,1)$-approximation algorithm of Baswana and Kavitha [4], our algorithm has the advantage of being simpler and deterministic (albeit at a cost of logarithmic factor in the running time).

We extend this result by showing that by relying on fast Boolean matrix multiplication a better approximation could be achieved at the expense of a small increase in the running time. More specifically, for any $\epsilon > 0$ we provide a $(1+\epsilon,2)$-approximation algorithm. Using the currently best known matrix multiplication algorithms yields an $O(n^{2.24+o(1)}\epsilon^{-3}\log(n\epsilon^{-1}))$ time bound for this algorithm. Interestingly, the running time of the algorithm doesn't depend on the actual weights in the graph. Moreover, since it is already known that distinguishing between distances 2 and 4 in unweighted graphs is as hard as Boolean matrix multiplication [12], we can't hope to obtain a similar running time for (say) a $(1+\epsilon, 2-3\epsilon)$-approximation algorithm without improving the current Boolean matrix multiplication bounds. As discussed earlier, Elkin [14] has another type of two-parameter approximation, as well as time/quality trade-off. However, it appears that his algorithm is faster only when the approximation quality is inferior. This is because his additive error term is,

$$W \cdot \left(\frac{8c_0}{\epsilon\zeta(\rho-\zeta/2)}\right)^{\lceil \log_{1-(\rho-\zeta/2)}\zeta/2\rceil+1} \lceil \log_{1-(\rho-\zeta/2)}\zeta/2\rceil^{\lceil \log_{1-(\rho-\zeta/2)}\zeta/2\rceil},$$

for a constant c_0.

We then turn our attention toward approximating the diameter and the radius of a graph. We first show that a variant of the algorithm proposed by Aingworth *et al.* [1] for approximating the diameter can be used for approximating the radius of unweighted undirected graphs. The algorithm gives an almost $3/2$-approximation of the radius in $\tilde{O}(m\sqrt{n}+n^2)$ time.

We improve these results for the class of weighted separable graphs. We show if every subgraph of size k has a k^μ-separator, then a $3/2$-approximation of the diameter can be achieved in $\tilde{O}(n^{1+\mu}+n^{3\mu})$ time. This result also extends to the case of directed graphs. We also present an algorithm that achieves almost $3/2$-approximation of the radius in $\tilde{O}(n^{1+\mu}+n^{3\mu})$ time. As a consequence of these results, the fact that planar graphs have $O(\sqrt{n})$ separator [21], and the fact that single-source shortest path on planar graphs can be done in $O(n)$ time [19], we obtain $O(n^{3/2})$ time algorithms for $3/2$-approximation of both the diameter and the radius of positive weighted planar graphs.

The Fig. 1 and Fig. 2 summarizes our results and puts them into context.

2 Preliminaries

For a weighted connected graph $G = (V, E)$ we use the following notation. We use $w_G : E \to \mathbb{R}^+$ to denote the weight function. For a set of vertices $U \subseteq V$, we define $d_G(u, U)$ as $\min_{v \in U} d_G(u, v)$ and $c_G(u, U)$ to be a (any, if more than one) vertex $w \in U$ with $d_G(u, w) = d_G(u, U)$. Similarly we define $f_G(u, U)$ to be a vertex $w \in U$ with $d_G(u, w) = \max_{v \in U} d_G(u, v)$.

The ℓ-neighborhood of U in G ($N_{G,\ell}(U)$) is a set of vertices in G that are at a distance at most ℓ from any vertex in U, i.e.,

$$N_{G,\ell}(U) = \{v \in V \mid \exists u \in U \text{ such that } d_G(u, v) \leq \ell\}.$$

We use $N_G(U)$ to denote the 1-neighborhood of U in G. For solving the diameter and radius problems, we also define

$$\mathtt{min\_ecc}(U, G) = \min_{u \in U} d_G(u, f_G(u, V)),$$
$$\mathtt{max\_ecc}(U, G) = \max_{u \in U} d_G(u, f_G(u, V)).$$

For a graph G, the center of the graph $\mathtt{cen}(G)$ is a vertex of the graph with eccentricity equal to the graph radius. We use $\mathtt{rad}(G)$ to denote the radius of G, and $\mathtt{dia}(G)$ to denote the diameter of G. Note that $\mathtt{max\_ecc}(V, G) = \mathtt{dia}(G)$ and $\mathtt{min\_ecc}(V, G) = \mathtt{rad}(G)$.

In shortest path algorithms, we use a symmetric $n \times n$ distance matrix $\{\delta(u, v)\}_{u,v}$ to hold the currently best upper bound on distance between all pairs of vertices in G. We use *dijkstra*$((V, F), \delta, u)$ to denote an invocation of Dijkstra's single-source shortest path from vertex u on the graph (V, F). Every invocation of the algorithm updates the row and column entries of u in the distance matrix δ, provided the distance found during this run is smaller than previous estimates. Initially $\delta(u, v) = 1$, if $(u, v) \in E$ and ∞ otherwise. We omit the distance matrix argument from *dijkstra*() when not required. We use *bfs*$((V, F), u)$ to denote an invocation of breadth-first search from u on the graph (V, F).

A subset of vertices $S \subseteq V$ of a graph (V, E) is a λ-separator ($\lambda < 1$) if the largest connected component in $V \setminus S$ has at most $\lambda|V|$ vertices. A $[\lambda, \mu]$-separator decomposition of G is a recursive decomposition of G using separators, where subgraphs of size k have λ-separators of size $O(k^\mu)$ for $\mu \in (0, 1)$. Studied in this framework, the planar separator theorem due to Lipton and Tarajn [21] is a $[2/3, 1/2]$-separator decomposition. Henceforth, we call a graph *separable* if it admits a $[\lambda, \mu]$-separator decomposition.

Given an optimization problem $\mathcal{P}$, for any instance $I \in \mathcal{P}$, and for any feasible solution $S(I)$, the approximation ratio of $S(I)$ with respect to I is defined as $\max\{\frac{S(I)}{Opt(I)}, \frac{Opt(I)}{S(I)}\}$. Here, $Opt(I)$ denotes an optimal solution of instance I.

2.1 Estimating Distances using Dominating Sets

A set of vertices D is said to dominate a set of vertices U if every vertex U has a neighbor in D. The use of dominating sets for solving shortest path problems was first employed by Aingworth *et al.* [1]. The idea is based on the simple observation that there is a small set of vertices that dominates all the high degree vertices of a graph. Therefore, paths going to high degree vertices can be efficiently approximated by taking a small detour through the dominating set. Cohen and Zwick [9] extended this result to the weighted case. For an input s, they have shown that a dominating set of size $O((n \log n)/s)$ can be constructed such that if $u \in V$ has degree at least s in G, then there is an edge $(u, v) \in E$ with $v \in D$, and (u, v) is one of the s lightest edges incident on u. We use $\mathtt{rank}_u(u, v)$ and $\mathtt{rank}_v(u, v)$ to denote the index of (u, v) in the sorted adjacency list of u and v respectively. The following observation based on greedy approximation algorithm for the set cover problem is central to our results.

Function preprocess$(G = (V, E), k)$
for $i \leftarrow 0$ to k do $s_i \leftarrow n/2^i$
for $i \leftarrow 1$ to k do $E_i \leftarrow \{(u, v) \in E \mid \mathrm{rank}_u(u, v) < s_{i-1}$ or $\mathrm{rank}_v(u, v) < s_{i-1}\}$
for $i \leftarrow 1$ to k do $D_i \leftarrow \mathrm{dom}(G, s_i)$
for $i \leftarrow 1$ to k do
$\forall u \in D_i$ call *dijkstra*$((V, E_i \cup E_{\mid u}), \delta, u)$

Figure 3: Preprocessing function for approximating APSP.

Algorithm apasp$_{(\mathbf{2},\mathbf{1})}(G, \delta)$
call **preprocess**$(G, \lceil \log n \rceil)$
for every $u, v \in V$ do
for $i \leftarrow 1$ to $\lceil \log n \rceil$ do
$\delta(u, v) \leftarrow \min\{\delta(u, v), \delta(u, c_G(u, D_i)) + \delta(c_G(u, D_i), v), \delta(v, c_G(v, D_i)) + \delta(c_G(v, D_i), u)\}$

Figure 4: $(2, 1)$-approximation algorithm for the APSP problem.

Lemma 1. *(Cohen and Zwick [9]) Let $G = (V, E)$ be an weighted undirected graph with n vertices and m edges. Let $1 \leq s \leq n$. A dominating set D of size $O((n \log n)/s)$ that dominate all vertices of degree at least s in the graph can be found in $O(m + n)$ time. Furthermore, if $u \in V$ is of degree at least s in G then there is an edge (u, v) with $v \in D$ such that $\mathrm{rank}_u(u, v) \leq s$.*

We use an algorithm (details omitted) based on Lemma 1, called $\mathrm{dom}(G, s)$. The algorithm receives $G = (V, E)$ and a degree threshold s as inputs, and outputs a set of vertices $D \subseteq V$ satisfying the properties of Lemma 1.

3 Approximation Algorithms for the APSP Problem

The idea behind the preprocessing step (function **preprocess**, Fig. 3) is to split the vertices into classes based on degree. The i^{th}-class contains vertices having degrees between $n/2^i$ to $n/2^{i+1}$. We use Lemma 1 to find a dominating set D_i for the vertices of the i^{th}-class. For a vertex u, $E_{\mid u}$ represents the set of all edges incident on u in G. The final step involves invoking Dijkstra from the vertices in D_i.

The dominating set D_i has a size at most $\min\{(n \log n)/s_i, n\}$ and the graph on which we run Dijkstra from the vertices in D_i has $O(ns_{i-1})$ edges. Therefore, the total time for the preprocessing step is $\tilde{O}(n^2)$.

3.1 $(2, 1)$-approximation Algorithm

In this subsection we present a simple algorithm that achieves a (2,1)-approximation (Fig. 4). The algorithm **apasp**$_{(\mathbf{2},\mathbf{1})}$ uses the distance matrix from the function **preprocess** to estimate the distances between every pair of vertices. In the final stages when the dominating set grows to linear size, the function **preprocess** just finds all-pairs shortest path in a graph with linear number of edges.

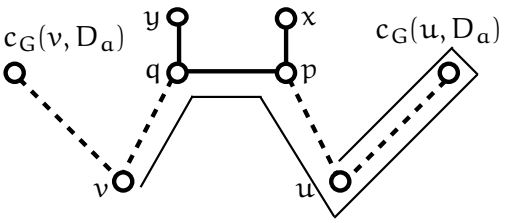
Theorem 1. *The algorithm* **apasp**$_{(\mathbf{2},\mathbf{1})}$ *runs in $O(n^2 \log^2 n)$ time, where n is the number of vertices in the input graph $G = (V, E)$, and is a $(2, 1)$-approximation to* APSP.

Proof. Consider any path P between the vertices u and v in G. If P is entirely contained in $E_{\lceil \log n \rceil}$, we get the actual distance of P from *dijkstra*$((V, E_{\lceil \log n \rceil}), \delta, u)$.

Otherwise, let $e = (p, q)$ be the first edge of P that is present in E_a but not in E_{a+1}. Let $x, y \in D_a$ be the vertices that dominate w and w' respectively. By construction we know that $w_G(x, p) \leq w_G(p, q)$ and $w_G(y, q) \leq w_G(p, q)$. Also, $\delta(u, c_G(u, D_a)) \leq d_G(u, x)$ and $\delta(v, c_G(v, D_a)) \leq d_G(v, y)$.

This is because the exact distance between u and $c_G(u, D_a)$ is preserved in $(V, E_a \cup E(c_G(u, D_a)))$. Similarly the exact distance between v and $c_G(v, D_a)$ is preserved in $(V, E_a \cup E(c_G(v, D_a)))$. Now the sum $d_G(u, x) + d_G(v, y) \leq l(P) + w_G(p, q)$. This implies that $\delta(u, c_G(u, D_a)) + \delta(v, c_G(v, D_a)) \leq l(P) + w_G(p, q)$. Now assume w.l.o.g. that $\delta(u, c_G(u, D_a)) \leq \delta(v, c_G(v, D_a))$ (otherwise, we just switch u and v). Therefore,

$$2\delta(u, c_G(u, D_a)) \leq l(P) + w_G(p, q).$$



Bold solid lines: actual graph edges.
Dotted lines: actual graph path.
Normal lines: represents path used
for proving the approximation ratio.

Finally, our estimate

$$\begin{aligned}
\delta(u, c_G(u, D_a)) + \delta(v, c_G(u, D_a)) &\leq \delta(u, G(u, D_a)) + \delta(u, G(u, D_a)) + l(P) \\
&\leq 2\delta(u, G(u, D_a)) + l(P) \\
&\leq 2 \cdot l(P) + w_G(p, q) \\
&\leq 2 \cdot l(P) + h(P).
\end{aligned}$$

Since similar inequality holds for any path between u and v, the estimate $\delta(u, v)$ is not more than $\min_{P \in Path(u,v)}\{2 \cdot l(P) + h(P)\}$. The complexity of the algorithm is dominated by the preprocessing step and from the previous discussion can be bounded by $O(n^2 \log^2 n)$. ❑

3.2 $(1 + \epsilon, 2)$-approximation Algorithm

We now describe a simple algorithm that uses fast algorithms for rectangular matrix multiplication of Boolean matrices to obtain a $(1 + \epsilon, 2)$-approximation. Let W be largest-edge weight in the graph G, after the edge weights are scaled so that the smallest non-zero edge weight in 1, i.e, ratio of heaviest to lightest edge is W. For the sake of simplicity, we will first describe a $(1 + \epsilon, 2)$-approximation algorithm with a running time of $O(n^{2.24+o(1)}\epsilon^{-2} \log(nW\epsilon^{-1}))$. We will later use this algorithm as a sub-routine in the main algorithm.

Preliminary Algorithm: Let j be an integer with $0 \leq j \leq \lceil \log_{1+\epsilon} nW \rceil$. Now with a dominating set D_i, define Boolean matrices of dimensions $n \times |D_i|$ as

$$B_{i,j}[u, v] = 1 \text{ iff } (1 + \epsilon)^j \leq \delta(u, v) < (1 + \epsilon)^{j+1} \text{ for } u \in V \text{ and } v \in D_i.$$

We can ignore all empty matrices, i.e., which don't have at least a 1. For a matrix M, its transpose is denoted by M^T.

Theorem 2. *If Boolean matrix multiplication of $n \times \ell$ by $\ell \times n$ matrices can be performed in $n^{2-\alpha\beta+o(1)}\ell^\beta$ time for constants α, β, then for any $\epsilon > 0$, the algorithm* $\mathbf{apasp}_{(1+\epsilon,2)}$ *runs in $\tilde{O}(n^{(2+3\beta-\alpha\beta)/(1+\beta)+o(1)} \log^2_{1+\epsilon}(nW))$ time, where n is the number of vertices in the input graph $G = (V, E)$, and is a $(1 + \epsilon, 2)$-approximation to* APSP.

Proof. Consider any path P between the vertices u and v. If P is entirely contained in $\hat{E}$, then we get the actual distance. Otherwise, let $e = (p, q)$ be the first edge of P that is present in E_a but not in E_{a+1} $(a \leq p)$. Let $x \in D_a$ be the vertex that dominates p. By construction we know that $w_G(x, p) \leq w_G(p, q)$.

When we invoke Dijkstra's algorithm from x, $\delta(u, x) \leq d_G(u, p) + w_G(p, q)$ and $\delta(x, v) \leq d_G(p, v) + w_G(p, q)$. Let $\epsilon = 3\epsilon'$. Consider the matrices $B_{a,r}$ such that $(1 + \epsilon')^r \leq \delta(u, x) <$

Algorithm apasp$_{(1+\epsilon,2)}(G,\delta)$

$p \leftarrow \lceil((1+\alpha\beta)\log n)/(1+\beta)\rceil$
call **preprocess**(G,p)
construct matrices $B_{i,j}$ for integers i,j with $i \in [1,p]$ and $j \in [0, \lceil\log_{1+\epsilon} nW\rceil]$
for $i \leftarrow 1$ to p do
  for $j \leftarrow 0$ to $\lceil\log_{1+\epsilon} nW\rceil$ do
    for $k \leftarrow j+1$ to $\lceil\log_{1+\epsilon} nW\rceil$ do
      let $A_{i,j,k} \leftarrow$ Boolean matrix product of $B_{i,j}$ and $B_{i,k}^T$
      for every $u,v \in V$ do

$$\delta(u,v) \leftarrow \begin{cases} \min\{\lfloor(1+\epsilon)^{j+1} + (1+\epsilon)^{k+1}\rfloor, \delta(u,v)\} & \text{if } A_{i,j,k}[u,v] = 1, \\ \delta(u,v) & \text{otherwise} \end{cases}$$

$\hat{E} \leftarrow \{(u,v) \in E \mid rank_u(u,v) < 2^p \text{ or } rank_v(u,v) < 2^p\}$
for every $u \in V$ call *dijkstra*$((V,\hat{E}),\delta,u)$

Figure 5: $(1+\epsilon,2)$-approximation algorithm for the APSP problem.

$(1+\epsilon')^{r+1}$ and $B_{a,s}$ such that $(1+\epsilon')^s \le \delta(x,v) < (1+\epsilon')^{s+1}$. The product matrix $A_{a,r,s}$ has 1 in the entry u,v (i.e., $A_{a,r,s}[u,v] = 1$). Therefore,

$$\begin{aligned} \delta(u,v) &\le d_G(u,x) + d_G(x,v) \\ &\le (1+\epsilon')\delta(u,x) + (1+\epsilon')\delta(x,v) \\ &\le (1+\epsilon')(d_G(u,p) + w_G(p,q) + d_G(p,v) + w_G(p,q)) \\ &\le (1+\epsilon')(l(P) + 2w_G(p,q)) \\ &\le (1+\epsilon)\cdot l(P) + 2\cdot h(P). \end{aligned}$$

Also since such an inequality holds for any path between u and v, the estimate $\delta(u,v)$ is not more than $\min_{P\in Path(u,v)}\{(1+\epsilon)\cdot l(P) + 2\cdot h(P)\}$. The time for performing a matrix multiplication can be bounded as $n^{2-\alpha\beta+o(1)}(2^p \log n)^\beta$. Therefore, the time needed for doing all the matrix multiplications is $\tilde{O}(n^{(2+3\beta-\alpha\beta)/(1+\beta)+o(1)}\log^2_{1+\epsilon}(nW))$. In the last step of the algorithm we perform Dijkstra from all vertices on a graph with at most $n/2^{p-1}$ edges. Therefore, performing all these Dijkstra also takes $\tilde{O}(n^{(2+3\beta-\alpha\beta)/(1+\beta)+o(1)})$ time. ❑

Remark: The dependence on ϵ in the running time can be reduced to $\epsilon^{-1}\ln\epsilon^{-1}\log_{1+\epsilon}(nW)$. The trick is to perform for every index $r \in \{1,2,\ldots,\lceil\log_{1+\epsilon}(nW)\rceil\}$ a "Boolean OR" of matrices $B_{i,s}$ $(s \le r - \epsilon^{-1}\ln\epsilon^{-1})$. Then we replace many matrix multiplications of form $B_{i,r} \times B_{i,s}$ by a single matrix multiplication of $B_{i,r}$ with the matrix constructed from the OR operation. The approximation ratio remains practically unchanged.

Let $\omega(1,x,1)$ be the infimum over all exponents ω' for which $n \times n^x$ by $n^x \times n$ Boolean matrices can be multiplied in $O(n^{\omega'})$ time. Let $\omega = \omega(1,1,1)$. The currently best known algorithm for rectangular matrix multiplication from Coppersmith [10] and Huang and Pan [20] provide

$$\omega(1,x,1) \le \begin{cases} 2 & \text{if } 0 \le x < \alpha, \\ 2 + \beta(x-\alpha) & \text{otherwise.} \end{cases}$$

This implies that $n \times \ell$ by $\ell \times n$ Boolean matrices can be multiplied in $n^{2-\alpha\beta+o(1)}\ell^\beta$ time. The constant α is defined as the supremum over all constants x for which $\omega(1,x,1) = 2$. Currently, $\alpha > 0.294$, $\beta = \frac{\omega-2}{1-\alpha}$, $\omega < 2.376$. We immediately get the following corollary from the discussion above.

Corollary 1. *There exists an implementation of the algorithm* $\mathbf{apasp}_{(1+\epsilon,2)}$ *that runs in* $O(n^{2.24+o(1)} \epsilon^{-2} \log(nW\epsilon^{-1}))$ *time.*

Main Algorithm: We now remove the dependence of W from the running time of the algorithm. Let $\textsc{Apsp}(\Lambda)$ be a (auxiliary) problem in which Apsp the ratio of the heaviest to lightest edge is bounded by Λ. For an instance of $\textsc{Apsp}(\Lambda)$ with n vertices (using the algorithm $\mathbf{apasp}_{(1+\epsilon,2)}$) we can compute a $(1+\epsilon, 2)$-approximation in $O(n^2\phi(n, \epsilon, \Lambda))$ time, where $\phi(n, \epsilon, \Lambda) = n^{0.24+o(1)}\epsilon^{-2} \log(n\Lambda\epsilon^{-1})$. We will use the fact that ϕ is a function growing in n. We now describe an $O(n^2\phi(n, \epsilon, \Lambda)\epsilon^{-1} \ln n)$ time algorithm for computing $((1+\epsilon)^2, 2(1+\epsilon))$-approximation of Apsp.

In our method, given an input graph G with n vertices, we produce a set of instances of $\textsc{Apsp}(\epsilon^{-1}n)$, say $G_1, \ldots, G_d$ with numbers of vertices $n_1, \ldots, n_d$ such that $n_i \leq n$, and $S_G = \sum_{i=1}^{d} n_i^2 \leq n^2\epsilon^{-1} \ln n$. The time needed to produce these instances and to combine their results will be $O(S_G)$, and the time needed to approximately solve these instances will be $O(\sum_i n_i^2\phi(n, \epsilon, \Lambda)) \leq S_G \cdot \phi(n, \epsilon, \Lambda)$.

We define instances of $\textsc{Apsp}(\epsilon^{-1}n)$ for distances in the range $(1+\epsilon)^k$ to $(1+\epsilon)^{k+1}$ (k integer) as follows. We assume w.l.o.g. that the minimum edge weight in G is 1, hence we consider only $k \geq 0$.

① remove edges with cost larger than $(1+\epsilon)^{k+1}$;

② make a separate instance for each connected component;

③ coalesce vertices that are connected by edges shorter than $((1+\epsilon)^k\epsilon)/n$ into super-vertices. If a vertex u is not coalesced with any other vertex, we view $\{u\}$ as a super-vertex of this instance;

④ eliminate instances with one vertex only.

Estimating S_G*:* We first decompose S_G into the sum of contribution of super-vertices: (a) in an instance G_l with n_l super-vertices, each super-vertex contributes n_l to n_l^2, (b) if super-vertex $\mathbf{u}$ in G_l is a set of g_l vertices, we decompose the contribution of $\mathbf{u}$ into g_l equal parts, n_l/g_l for each vertex in $\mathbf{u}$.

We say a vertex u is contained in an instance if there exists a super-vertex of the instance containing u. Now among the instances made for the distance range $[(1+\epsilon)^i, (1+\epsilon)^{i+1}]$, we use $G_i(u)$ to denote the instance containing vertex u (even though we may create many instances for a distance range, only one of them will contain u). We use $g_i(u)$ to denote the number of elements of the super-vertex of $G_i(u)$ that contains u. We denote the number of super-vertices of $G_i(u)$ as $n_i(u)$. The contribution of u to S_G at $G_i(u)$ is $\kappa_i(u) = n_i(u)/g_i(u)$. We want to show that the sum of all $\kappa_i(u)$'s is bounded by $n\epsilon^{-1} \ln n$. Note that for some values of i the instance $G_i(u)$ is not created, e.g., because of ④.

Let $N = \lceil \epsilon^{-1} \ln n \rceil$. The desired inequality holds if for every $j < N$ and every $u \in V$ we have the sum of all $\kappa_{j+iN}(u)$'s is bounded by n. This will bound the sum of all contributions to $n \times n \times N$.

Let $\mathbf{u}'$ be the super-vertex of u in the instance $G_{i+N}(u)$. Consider the instance $G_i(u)$. The key observation is that the union of the set of all super-vertices in the instance $G_i(u)$ is $\mathbf{u}'$. Therefore, $g_{i+N}(u) \geq n_i(u)$ and thus $n_i(u)/g_i(u) \leq g_{i+N}(u)/g_i(u)$. Note that $g_{i+N}(u) \geq g_i(u)$, and if $g_{i+N}(u) = g_i(u)$ then the instance $G_{i+N}(u)$ has one vertex, so it is not created, and thus there is no contribution to S_G.

Therefore, there exists an increasing sub-sequence $1 \leq \bar{g}_1 \leq \bar{g}_2 \ldots \leq \bar{g}_t \leq n$ of $g_i(u)$'s such that the sum of contributions of u for the distance ranges of the form $[(1+\epsilon)^{j+iN}, (1+\epsilon)^{j+iN+1}]$ is at most $\bar{g}_2/\bar{g}_1 + \bar{g}_3/\bar{g}_4 + \ldots \bar{g}_t/\bar{g}_{t-1}$. We can find the largest possible sum of this form as a function

of n, say $F(n)$. By induction we show, $F(n) = n$. By considering every possible $\hat{g}$ for $\bar{g}_{t-1}$, we have $F(n) = \max_{\hat{g}}\{n/\hat{g} + F(\hat{g})\}$. It is easy to see that for $\hat{g} > 1$ we have $n/\hat{g} < n - \hat{g}$, so the sum is maximal if it consists of one term only, $n/1$.

Construction of the instances: The construction uses disjoint sets data-structure. We start from the smallest distance, $1 = (1+\epsilon)^0$, the super-vertices are singleton sets and vertex sets for instances are connected components of edges of length 1. We maintain a disjoint sets data structure for super-vertices and another one for sets of super-vertices for instances. We also maintain an array of adjacency lists of super-vertices. When we advance from the distance range $[(1+\epsilon)^{i-1}, (1+\epsilon)^i]$ to $[(1+\epsilon)^i, (1+\epsilon)^{i+1}]$, for each edge (u, v) of length $(1+\epsilon)^{i-N}$ we merge the super-vertices of u and v, we also insert edges of length $(1+\epsilon)^{i+1}$.

We represent every super-vertex $\mathbf{u}$ with its selected element, say $\mathtt{boss}(\mathbf{u})$. Each adjacency list represents a number of original edges, say $m(\mathbf{u})$. When we merge $\mathbf{u}$ and $\mathbf{v}$, we may assume that $m(\mathbf{u}) \le m(\mathbf{v})$ and $\mathtt{boss}(\mathbf{v})$ will become the selected element of the union. For every w on the list of $\mathtt{boss}(\mathbf{u})$ we make two operations: in the adjacency list of w replace $\mathtt{boss}(\mathbf{u})$ with $\mathtt{boss}(\mathbf{v})$, and insert w to the adjacency list of $\mathtt{boss}(\mathbf{v})$. With a suitable data structure, each operation takes $O(\log n)$ steps, and when an edge is subjected to a pair of operations it becomes a member of an adjacency list with at least twice larger $m(\mathbf{u})$, hence the total work spend on updating the adjacency lists is $O(n^2 \log^2 n)$.

Given the array of adjacency lists and the list of elements (super-vertices), we can construct an instance with n_i super-vertices in $O(n_i^2)$ time. Therefore, the total time for constructing all the instances is $O(S_G)$.

Combining the results: If we create an instance for distance $[(1+\epsilon)^l, (1+\epsilon)^{l+1}]$, and in that instance we compute, for some u, v a distance approximation larger than $(1+\epsilon+2)(1+\epsilon)^{l+1}$, then we know that the true distance between u and v is above $(1+\epsilon)^{l+1}$, and thus it will be properly estimated in another instance. Similar reasoning applies if the computed distance is smaller than $(1+\epsilon)^l$. Therefore, when we scan the array of results for such an instance, we perform updates only for pairs of super-vertices that have computed distances in the range $(1+\epsilon)^l$ to $(3+\epsilon)(1+\epsilon)^{l+1}$. Given such a pair of super-vertices, say $\mathbf{u}$,$\mathbf{v}$ with computed distance $L' \in [(1+\epsilon)^l, (3+\epsilon)(1+\epsilon)^{l+1}]$, for each $u \in \mathbf{u}$ and $v \in \mathbf{v}$ we update (unless a smaller estimate was already present) the distance from u to v with $L' + (1+\epsilon)^l\epsilon$. The term $(1+\epsilon)^l\epsilon$ is needed to correct for the possible effect of collapsing edges of length less than $((1+\epsilon)^l\epsilon)/n$.

As a result, the time needed to combine the result is the time needed to read the result matrices, which equals to S_G, plus the number of updates in the matrix of final results, and we perform at most $\epsilon^{-1}\ln((1+\epsilon)(3+\epsilon))$ updates for each entry.

Since the above discussion holds for any $\epsilon > 0$, we obtain the following result.

Theorem 3. *Let G be a weighted undirected graph on n vertices. For any $\epsilon > 0$, there exists an $O(n^{2.24+o(1)}\epsilon^{-3}\log(n\epsilon^{-1}))$ time algorithm that is a $(1+\epsilon, 2)$-approximation to* APSP.

4 Approximating the Radius of Graphs

Aingworth *et al.* [1] presented a 3/2-approximation algorithm for estimating the diameter of weighted directed graphs. In this section we extend their algorithm and show that it can be used to obtain an almost 3/2-approximation of the radius of unweighted undirected graphs. The algorithm is presented in Fig. 6. We assume that $\mathtt{rad}(G) \ge 2$, the case of $\mathtt{rad}(G) = 1$ can be easily handled separately.

A s-partial breadth-first search is obtained by performing the breadth-first search from a vertex to the point where exactly s vertices (excluding the starting vertex) have been visited. A s-partial breadth-first from u on graph (V, F) is denoted by s-*bfs*$((V, F), u)$. Let $\mathtt{PBFS}(u)$ denote the set of vertices which are visited by an invocation of $\sqrt{n\log n}$-*bfs*(G, u).

Algorithm $\mathbf{rad_{3/2}}(G,\delta)$
for every $u \in V$ call $\sqrt{n \log n}$-*bfs*(G,u)
$w \leftarrow$ be the vertex having the maximum depth partial breadth-first search tree
for every $u \in \mathsf{PBFS}(w)$ call *bfs*(G,u)
compute a new graph $\widehat{G}$ from G by adding all edges of form (u,v)
where either $u \in \mathsf{PBFS}(v)$ or $v \in \mathsf{PBFS}(u)$
$D \leftarrow \mathtt{dom}(\widehat{G}, \sqrt{n \log n})$
for every $u \in D$ call *bfs*(G,u)
output $\min\{\mathtt{min\_ecc}(D,G), \mathtt{min\_ecc}(\mathsf{PBFS}(w),G)\}$

Figure 6: Almost 3/2-approximation of the radius.

The size of the dominating set D constructed in he algorithm $\mathbf{rad_{3/2}}$ is $O(\sqrt{n \log n})$ (follows from Lemma 1). We borrow the following simple result about the time needed to perform all the partial breadth-first searches.

Lemma 2. *(Aingworth* et al. *[1]) Let G be a graph with n vertices. The $\sqrt{n \log n}$-partial breadth-first searches from all vertices in G can be performed in $O(n^2 \log n)$ time.*

The following theorem completes the analysis of the algorithm. The main thrust of the proof is to condition on the distance between w (vertex with biggest partial breadth-first search depth) and $\mathsf{cen}(G)$. It is worth noting in the proof that the radius is approximated within a factor 3/2 in all but one case.

Theorem 4. *The algorithm $\mathbf{rad_{3/2}}$ runs in $O(m\sqrt{n} \log n + n^2 \log n)$, where n is the number of vertices and m is the number of edges in the input graph $G = (V,E)$, and gives an estimate of the radius that is at most $\lceil \frac{3}{2}\mathsf{rad}(G) \rceil$.*

Proof. Let $\mathsf{rad}(G) = r$. Let $k+1$ be the depth obtained by performing the partial breath-first search from w. We now condition the proof based on the distance $d_G(w, \mathsf{cen}(G))$.

Case 1: If $\mathbf{d_G(w, cen(G)) \geq 2k+2}$. This implies that $r/2 \geq k+1$. In this case we use the breadth-first searches done from the vertices in the dominating set D. If $\mathsf{cen}(G) \in D$, then the estimate equals the actual radius. Otherwise, since D is a dominating set there is a vertex $a \in D$ such that $(\mathsf{cen}(G), a)$ is an edge in $\widehat{G}$. If $(\mathsf{cen}(G), a)$ is also an edge in G. Then again the output is off by an additive error of at most 1. Now if $(\mathsf{cen}(G), a) \notin E$, then the existence of the edge implies that either $a \in \mathsf{PBFS}(\mathsf{cen}(G))$ or $\mathsf{cen}(G) \in \mathsf{PBFS}(a)$. In either case $d_G(\mathsf{cen}(G), a) \leq k+1$. Therefore, from *bfs*$(G,a)$, we get an estimate which is less than $\frac{3}{2}r$.

Case 2: If $\mathbf{d_G(w, cen(G)) \leq 2k}$. This implies that $r/2 \geq k$. In this case we consider a shortest path from $\mathsf{cen}(G)$ to w. This shortest path passes through a vertex $b \in N_{G,k}(w)$. Since we invoke *bfs*(G,b), the estimate is less than $\frac{3}{2}r$.

Case 3: If $\mathbf{d_G(w, cen(G)) = 2k+1}$. In this case we have $r/2 \geq k+1/2$. Now we again consider a shortest path from $\mathsf{cen}(G)$ to w. If the shortest path passes through a vertex $c \in N_{G,k+1}(w)$, then as we invoke *bfs*(G,c) we get an estimate which is less than $\frac{3}{2}r$. Otherwise, there definitely exists a vertex z in the shortest path such that $z \in N_{G,k}(w)$. Since we invoke *bfs*(G,z), the estimate is less than $r + k + 1 \leq \frac{3}{2}r + 1/2 \leq \lceil \frac{3}{2}r \rceil$.

The time complexity is dominated by the time for running breadth-first searches. The time for running the breadth-first searches from the vertices in D is $O(m\sqrt{n} \log n + n^2 \log n)$. The same time is needed for running breadth-first searches from all vertices in $\mathsf{PBFS}(w)$. Using these along with the result of Lemma 2 provides the claimed time bounds. ❑

5 Approximating Diameter and Radius of Separable Graphs

In this section we present algorithms for faster estimation of diameter and radius of graphs having a $[\lambda, \mu]$-separator decomposition, where the decomposition is either provided as part of the input or is quickly obtainable. For most of the well-known separable graphs, the latter condition holds true.

We start by proving a general statement about the maximum number of edges that a separable graph can have. Earlier known results had a weaker upper bound of $O(n + n^{2\mu})$ on the number of edges (see for example Cohen [8]).

Lemma 3. *Let G be a graph with n vertices and a $[\lambda, \mu]$-separator decomposition. Then number of edges in G is $O(n)$.*

Proof. The number of edges $E(n)$ in G satisfies the recurrence:

$$E(n) \leq \max_{\lambda}(E(\lambda n + Cn^{\mu}) + E((1-\lambda)n + Cn^{\mu})),$$

where C is a given constant from the size of the separator. We can show by induction that $E(n) \leq cn - dn^{\mu}$ for parameters $n \geq n_0$, c, and d selected as follows:
(a) choose n_0 such that $n - (2Cn^{\mu})/(\lambda^{\mu} + (1-\lambda)^{\mu} - 1) \geq 1$ for all $n \geq n_0$,
(b) choose $c = \binom{n_1}{2}$, where $n_1 = \max\{n_0/\lambda n_0/(1-\lambda)\}$, and
(c) choose d such that $2Cc/(\lambda^{\mu} + (1-\lambda)^{\mu} - 1) = d$.
For the base case we notice that the claim is true for $n_0 \leq n \leq n_1$. In the inductive step

$$\begin{aligned} E(n) &\leq \max_{\lambda}\{c\lambda n + cCn^{\mu} - d(\lambda n + Cn^{\mu})^{\mu} + c(1-\lambda)n + cCn^{\mu} - d((1-\lambda)n + Cn^{\mu})^{\mu}\} \\ &\leq cn + 2cCn^{\mu} - d(\lambda n)^{\mu} - d((1-\lambda)n)^{\mu} \\ &= cn + (2Cc - d(\lambda^{\mu} + (1-\lambda)^{\mu}))n^{\mu} \end{aligned}$$

Now $cn + (2Cc - d(\lambda^{\mu} + (1-\lambda)^{\mu}))n^{\mu} \leq cn - dn^{\mu}$ if

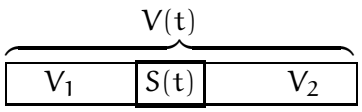
$$2Cc - d(\lambda^{\mu} + (1-\lambda)^{\mu}) \leq -d \Leftrightarrow 2Cc \leq d(\lambda^{\mu} + (1-\lambda)^{\mu} - 1)$$

Let $D = \lambda^{\mu} + (1-\lambda)^{\mu} - 1$. To satisfy the above condition (as noted above) we set d to $2Cc/D$. Therefore,

$$E(n) \leq c(n - \frac{2C}{D}n^{\mu}).$$

Finally, for the base case we chose n_0 such that $n - 2Cn^{\mu}/D \geq 1$ for all $n \geq n_0$ and $c = \max_{n_0 \leq n \leq n_1} \binom{n}{2} = \binom{n_1}{2}$. ❑

We use a rooted binary tree T_G to represent a separator decomposition of G (as in [8]). To avoid ambiguities, we refer to the vertices of a graph as *vertices* and vertices of a separator tree as *nodes*. Let $\mathrm{root}(T_G)$ be the root node of T_G.



Each node $t \in T_G$ is labeled by two subsets of vertices $V(t) \subseteq V$ and $S(t) \subseteq V(t)$. Let $G(t) = (V(t), E(t))$ denote the subgraph induced by $V(t)$. Then $S(t)$ is the separator in $G(t)$. Then $V(\mathrm{root}(T_G)) = V$ and $S(\mathrm{root}(T_G))$ is a separator in G. For any $t \in T_G$, the labels of its children t_1, t_2 are defined as follows: Let $V_1 \subset V(t)$ and $V_2 \subset V(t)$ be the components separated by $S(t)$ in $G(t)$. Then $V(t_1) = V_1 \cup (S(t) \cap N_G(V_1))$, $V(t_2) = V_2 \cup (S(t) \cap N_G(V_2))$.

We associate *boundary vertices*, $B(t)$ with each node t. The boundary of the $\mathrm{root}(T_G)$ is $\emptyset$. The boundary of every other node t is defined as $B(t) = S(p(t)) \cup B(p(t)) \cap V(t)$, where $p(t)$ is the parent of t in T_G. We now describe the preprocessing stage for the algorithms.

The algorithm **sep-preprocess** (Fig. 7) does Dijkstra from the vertices in $S(t)$ on a weighted graph $H(t)$. We now show that the graph $H(t)$ preserves the shortest distance between every pair of vertices from $V(t)$.

Function sep-preprocess(G, t)
construct a weighted graph $H(t) = (V(t), E(t) \cup B(t) \times B(t))$, where for $(u, v) \in B(t) \times B(t)$, $w_{H(t)}(u, v) = d_G(u, v)$ and for $e \in E(t)$, $w_{H(t)}(e) = w_G(e)$
for every $u \in S(t)$ call *dijkstra*$(H(t), u)$
create a related graph $\widehat{H}(t)$ (from $H(t)$): merge all vertices of $S(t)$ into a single vertex ϑ, remove and keep all edges in $H(t)$, including parallel edges
call *dijkstra*$(\widehat{H}(t), \vartheta)$ to determine $f_{H(t)}(\vartheta, V(t) \setminus S(t))$

Figure 7: Preprocessing function for approximating the diameter, radius of separable graphs.

Algorithm sep-dia$_{3/2}$ (G, T_G) (G is a separable graph)
for every $t \in T_G$ do
call **sep-preprocess**(G, t)
$max_1(t) \leftarrow max\_ecc(S(t), H(t))$
$max_2(t) \leftarrow max\_ecc(f_{H(t)}(\vartheta, V(t) \setminus S(t)), H(t))$
$max(t) \leftarrow \max\{max_1(t), \max_2(t)\}$
output $\max\{max(t) \mid t \in T_G\}$

Figure 8: 3/2-approximation of the diameter of separable graphs.

Lemma 4. *For any* $t \in T_G$ *and* $u, v \in V(t)$, $d_{H(t)}(u, v) = d_G(u, v)$.

Proof. Proof by induction. If $t = root(T_G)$, the claim is true. By the inductive hypothesis $d_{H(p(t))}(u, v) = d_G(u, v)$. Let $P = \{u, y_1, y_2, \ldots, v\}$ denote the shortest path from u to v in $H(p(t))$. If $P \subseteq V(t)$ then the shortest path is also present in $H(t)$. Otherwise, let i be smallest index such that $y_i \in S(p(t))$ and let j be the largest index such that $y_j \in S(p(t))$. Both y_i and y_j are in $B(t)$ and have an edge connecting them with weight $w_{H(t)}(y_i, y_j) = d_G(y_i, y_j)$. Thus distances are preserved in $H(t)$. ❑

5.1 3/2-approximation of the Diameter

We present an algorithm for weighted undirected graphs. The extension to the directed case is described later. Note that a simple consequence of Lemma 3 is that the diameter approximation algorithm of Aingworth *et al.* [1] runs in $\tilde{O}(n^2)$ time on separable graphs.

We assume that graph is strongly connected. The algorithm **sep-dia$_{3/2}$** (Fig. 8) operates on all nodes in the separator decomposition tree (T_G). For every $t \in T_G$, the function **sep-preprocess** is invoked. One can inductively see that weights for constructing the graph $H(t)$ is available. For $t = root(T_G)$ it is true. Now consider an edge (v_1, v_2) in $H(t)$. If either of v_1 or v_2 is in $B(p(t))$, inductively we know that weights are available. Otherwise, both v_1 and v_2 are in $S(p(t))$ and the weights are again available (refer Lemma 4).

In our analysis we assume that $dia(G) \geq 3$. The cases of $dia(G) = 1$ or $dia(G) = 2$ can be handled separately in a straightforward manner. We also assume that $dia(G)$ is a multiple of 3. Otherwise, the proof goes by replacing $dia(G)/3$ by $\lfloor dia(G)/3 \rfloor$.

Theorem 5. *Let* G *be a weighted undirected separable graph. The algorithm* **sep-dia$_{3/2}$** *runs in* $O(n^{1+\mu} \log n + n^{3\mu} \log n)$ *time, where* n *is the number of vertices in* G*, and gives an estimate of the diameter which is at least* $\frac{2}{3} dia(G)$.

Proof. Let a and b be the (any) two vertices whose distance defines the diameter, i.e., $d_G(a, b) = dia(G)$. Let $\triangle = dia(G)$. Let t be the node in T_G with $a \in V(t_1)$ and $b \in V(t_2)$, where t_1, t_2 are the children of t in the separator decomposition tree. If either $a \in S(t)$ or $b \in S(t)$ the estimate for diameter equals $\triangle$ (from Lemma 4). We now consider two cases:

Algorithm sep-rad$_{3/2}$ (G, T_G) (G is a separable graph)
$t \leftarrow \mathsf{root}(T_G)$
$\mathcal{S} \leftarrow \emptyset$
while $V(t) \neq \emptyset$
call **sep-preprocess**(G, t)
$\mathcal{S} \leftarrow \mathcal{S} \cup S(t)$
choose i such that $f_{H(t)}(\vartheta, V(t) \setminus S(t)) \in V(t_i)$
$t \leftarrow t_i$
for every $u \in \mathcal{S}$ call *dijkstra*(G, u)
output $\mathtt{min\_ecc}(\mathcal{S}, G)$

Figure 9: Almost 3/2-approximation of the radius of separable graphs.

Case 1: $\forall \mathbf{u} \in \mathbf{V(t_1)} \cup \mathbf{V(t_2)}$, $\exists \mathbf{v} \in \mathbf{S(t)}$ such that $\mathbf{d_G(u, v)} < \triangle/\mathbf{3}$.
Let w be the vertex in $S(t)$, such that $d_G(a, w) \leq \triangle/3$. Then definitely $d_G(w, b) \geq 2\triangle/3$. This implies that we get 3/2-approximation of the diameter as we do Dijkstra from w.
Case 2: There exists a vertex in $\mathbf{w} \in \mathbf{V(t_1)} \cup \mathbf{V(t_2)}$ with $\mathbf{d_G(w, S(t))} \geq \triangle/\mathbf{3}$.
This implies that $z = f_{H(t)}(s, V(t))$ is also at least $\triangle/3$ from all the vertices in $S(t)$. Assume w.l.o.g. that $z \in V(t_2)$. If the farthest vertex from z is at a distance $2\triangle/3$, this distance is our estimate and we are done. Otherwise, let $c \in S(t)$ be a vertex through which the shortest path from z to a passes. Then $d_G(z, a) = d_G(z, c) + d_G(c, a) \leq \triangle$. Since $d_G(z, a) \leq 2\triangle/3$ and $d_G(z, c) \geq \triangle/3$. This implies $d_G(c, a) \leq \triangle/3$. Since $d_G(a, b) = \triangle$, implying $d_G(b, c) \geq 2\triangle/3$. Again we get a 3/2-approximation when we do Dijkstra from c.

To analyze the running time, note that for all $t \in T_G$, $|B(t)| = O(n^\mu)$, therefore at every node t we introduce at most $O(n^{2\mu})$ edges. The total running time is dominated by the cost of running Dijkstra. Using Lemma 3 for the number of edges in G, gives the required bounds. ❑

Extension to Directed Graphs: Let G be a directed graph, we will denote by $\overleftarrow{G}$ the graph obtained from G by reversing the directions of all edges in G. A separator decomposition relies only on the unweighted undirected skeleton of G. For a node $t \in T_G$, the collapsed vertex ϑ in the function **sep-preprocess** is also defined over the undirected skeleton of $H(t)$. The complete graph between boundary vertices however is directed (with edges in both directions present).

For every node $t \in T_G$, we invoke the function **sep-preprocess** on both $G(t)$ and $\overleftarrow{G}(t)$. Now $\mathtt{max}_1(t)$ is defined as the maximum of $\mathtt{max\_ecc}(S(t), H(t))$ and $\mathtt{max\_ecc}(S(t), \overleftarrow{H}(t))$. Also $\mathtt{max}_2(t)$ is defined to be $\mathtt{max\_ecc}(f_{\overleftarrow{H(t)}}(\vartheta, V(t)), H(t))$. The analysis follows as in Theorem 5. For the analysis we look at whether for every $u \in V(t_1) \cup V(t_2)$ there exists a $v \in S(t)$ such that $d_{\overleftarrow{G}}(u, v) \leq \mathtt{dia}(G)/3$ or not.

We summarize the result in the following theorem.

Theorem 6. *Let G be a weighted directed separable graph. The algorithm* **sep-dia$_{3/2}$** *runs in $O(n^{1+\mu} \log n + n^{3\mu} \log n)$ time, where n is the number of vertices in G, and gives an estimate of the diameter which is at least $\frac{2}{3}\mathtt{dia}(G)$.*

5.2 3/2-approximation of the Radius

The algorithm **sep-rad$_{3/2}$** (Fig. 9) follows one path down the separator decomposition tree. For every node t in the path, the function **sep-preprocess** is invoked. As with **sep-dia$_{3/2}$**, we can inductively show that the weights needed for construction of the graphs $H(t)$ are available.

Theorem 7. *Let* G *be a weighted undirected separable graph. The algorithm* **sep-rad$_{3/2}$** *runs in* $O(n^{1+\mu}\log n + n^{3\mu}\log n)$ *time, where* n *is the number of vertices in* G*, and gives an estimate of the radius which is at most* $\lceil \frac{3}{2}\mathtt{rad}(G)\rceil$.

Proof. Let the radius $\mathtt{rad}(G)$ be equal to r. If $\mathtt{cen}(G) \in \mathcal{S}$ then the estimate equals r. Otherwise, let t be a node in T_G such that $\mathtt{cen}(G) \in V(t)$, but $\mathtt{cen}(G) \notin V(t')$ where $t = p(t')$. We now consider two cases:

Case 1: $\forall \mathbf{u} \in \mathbf{V(t_1)} \cup \mathbf{V(t_2)}$, $\exists \mathbf{v} \in \mathbf{S(t)}$ **such that** $\mathbf{d_G(u,v)} < \lfloor \mathbf{r/2} \rfloor$.
This implies that there exists a vertex $w \in S(t) \cap \mathcal{S}$ such that $d_G(\mathtt{cen}(G), w) \leq r/2$. Therefore, when we perform Dijkstra from w on G we get a 3/2-approximation of the radius.

Case 2: There exists a vertex in $\mathbf{w \in V(t_1) \cup V(t_2)}$ **with** $\mathbf{d_G(w, S(t)) \geq \lfloor r/2 \rfloor}$.
This implies that $z = f_{H(t)}(s, V(t))$ is also at least at a distance $\lfloor r/2 \rfloor$ from every vertex in $S(t)$. By construction we know that $z \in V(t')$. Consider the shortest path from z to $\mathtt{cen}(G)$ in t. From the previous discussion we know that there exists $c \in S(t)$ such that the shortest path between z and $\mathtt{cen}(G)$ passes through c. Since $d_G(z, c) \geq \lfloor r/2 \rfloor$, $d_G(\mathtt{cen}(G), c) \leq \lceil r/2 \rceil$. Now as $c \in \mathcal{S}$, we get an estimate of $\lceil \frac{3}{2}r \rceil$ due to *dijkstra*(G, c).

The analysis of the running time follows as in Theorem 5. Note that $|\mathcal{S}| = O(n^{\mu})$. Therefore, the Dijkstra's algorithm (on G) from all the vertices in $\mathcal{S}$ can be performed in $O(n^{1+\mu}\log n)$ time. ❑

6 Concluding Remarks

The running time of the algorithm **apasp$_{(1+\epsilon,2)}$** automatically improves if the current bounds on α or ω improves. The upper bound currently is only of theoretical interest, and for actual implementation one may wish to use other more practical schemes (like Strassen's multiplication [26]). The scheme described for removing the dependence of edge weights from the running time could be potentially used with other algorithms also.

Our algorithms for the radius approximation works only for undirected case (also unweighted for general graphs). The problem stems from the fact that even for strongly connected graphs $\mathtt{rad}(G)$ needn't be equal to $\mathtt{rad}(\overleftarrow{G})$, where as $\mathtt{dia}(G) = \mathtt{dia}(\overleftarrow{G})$. It would be quite interesting to overcome this problem. Also it would be interesting to design better/faster algorithms for approximating the diameter, radius of planar graphs.

Acknowledgement

The authors would like to thank Martin Fürer for many stimulating discussions on the algorithms presented here. We would also like to thank Surender Baswana for pointing us to references [4] and [14], and Timothy Chan for providing us a preliminary copy of [7].

References

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

[2] Alon, N., Seymour, P., and Thomas, R. A separator theorem for graphs with an excluded minor and its applications. In *STOC '90* (1990), pp. 293–299.

[3] Baswana, S., Goyal, V., and Sen, S. All-pairs nearly 2-approximate shortest-paths in O(n^2 polylog n) time. In *STACS '05* (2005), vol. 3404, Springer, pp. 666–679.

[4] Baswana, S., and Kavitha, T. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *FOCS '06* (2006), IEEE, pp. 591–602.

[5] Boitmanis, K., Freivalds, K., Ledins, P., and Opmanis, R. Fast and simple approximation of the diameter and radius of a graph. In *WEA '06* (2006), vol. 4007, Springer, pp. 98–108.

[6] Chan, T. M. All-pairs shortest paths for unweighted undirected graphs in o(mn) time. In *SODA '06* (2006), ACM, pp. 514–523.

[7] Chan, T. M. More algorithms for all-pairs shortest paths in weighted graphs. In *STOC '07 (To appear)* (2007), ACM.

[8] Cohen, E. Efficient parallel shortest-paths in digraphs with a separator decomposition. *Journal of Algorithms 21*, 2 (1996), 331–357.

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

[10] Coppersmith, D. Rectangular matrix multiplication revisited. *Journal of Complexity 13*, 1 (1997), 42–49.

[11] Coppersmith, D., and Winograd, S. Matrix multiplication via arithmetical progressions. *Journal of Symbolic Computation 9* (1990), 251–280.

[12] Dor, D., Halperin, S., and Zwick, U. All-pairs almost shortest paths. *SIAM Journal on Computing 29*, 5 (2000), 1740–1759.

[13] Dragan, F. F., Nicolai, F., and Brandstädt, A. LexBFS-orderings and power of graphs. In *WG '96* (1996), vol. 1197, Springer, pp. 166–180.

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

[15] Eppstein, D. Subgraph isomorphism in planar graphs and related problems. *Journal of Graph Algorithms and Applications 3*, 3 (1999).

[16] Farley, A. M., and Proskurowski, A. Computation of the center and diameter of outerplanar graphs. *Discrete Applied Mathematics 2* (1980), 185–191.

[17] Galil, Z., and Margalit, O. All pairs shortest distances for graphs with small integer length edges. *Information and Computation 134*, 2 (1997), 103–139.

[18] Galil, Z., and Margalit, O. All pairs shortest paths for graphs with small integer length edges. *Journal of Computer and System Sciences 54*, 2 (1997), 243–254.

[19] Henzinger, M. R., Klein, P. N., Rao, S., and Subramanian, S. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences 55*, 1 (1997), 3–23.

[20] Huang, X., and Pan, V. Y. Fast rectangular matrix multiplication and applications. *Journal of Complexity 14*, 2 (1998), 257–299.

[21] Lipton, R. J., and Tarjan, R. E. A separator theorem for planar graphs. *SIAM Journal of Applied Mathematics 36* (1979), 177–189.

[22] Miller, G., Teng, S. H., and Vavasis, S. A unified geometric approach to graph separators. In *FOCS '91* (1991), IEEE, pp. 538–547.

[23] Olariu, S. A simple linear-time algorithm for computing the center of an interval graph. *International Journal of Computer Mathematics 34* (1990).

[24] Pettie, S. A new approach to all-pairs shortest paths on real-weighted graphs. *Theoretical Computer Science 312*, 1 (2004), 47–74.

[25] Seidel, R. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences 51* (1995).

[26] Strassen, V. Gaussian elimination is not optimal. *Numerische Mathematik 14*, 3 (1969), 354–356.

[27] Thorup, M., and Zwick, U. Approximate distance oracles. *Journal of ACM 52*, 1 (2005), 1–24.

[28] Zwick, U. Exact and approximate distances in graphs - A survey. In *ESA '01* (2001), vol. 2161, Springer, pp. 33–48.

[29] Zwick, U. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM 49*, 3 (2002), 289–317.