Approximation Algorithms for Min-Distance Problems

Mina Dalirrooyfard, Virginia Vassilevska Williams, Nikhil Vyas,
Nicole Wein, Yinzhan Xu and Yuancheng Yu*

Abstract

We study fundamental graph parameters such as the Diameter and Radius in directed graphs, when distances are measured using a somewhat unorthodox but natural measure: the distance between u and v is the *minimum* of the shortest path distances from u to v and from v to u. The center node in a graph under this measure can for instance represent the optimal location for a hospital to ensure the fastest medical care for everyone, as one can either go to the hospital, or a doctor can be sent to help.

By computing All-Pairs Shortest Paths, all pairwise distances and thus the parameters we study can be computed exactly in $\tilde{O}(mn)$ time for directed graphs on n vertices, m edges and nonnegative edge weights. Furthermore, this time bound is tight under the Strong Exponential Time Hypothesis [Roditty-Vassilevska W. STOC 2013] so it is natural to study how well these parameters can be *approximated* in $O(mn^{1-\varepsilon})$ time for constant $\varepsilon > 0$. Abboud, Vassilevska Williams, and Wang [SODA 2016] gave a polynomial factor approximation for Diameter and Radius, as well as a constant factor approximation for both problems in the special case where the graph is a DAG. We greatly improve upon these bounds by providing the first constant factor approximations for Diameter, Radius and the related Eccentricities problem in general graphs. Additionally, we provide a hierarchy of algorithms for Diameter that gives a time/accuracy trade-off.

1 Introduction

The diameter, radius and eccentricities of a graph are fundamental parameters that have been extensively studied [13, 19, 12, 17, 3, 14, 11, 16, 5, 6, 25, 26, 9, 18, 23, 22, 10, 1, 7] (and many others). The eccentricity of a vertex v is the largest distance between v and any other vertex. The diameter is the maximum eccentricity of a vertex in the graph, thus measuring how far apart two nodes can be, and the radius is the minimum eccentricity, measuring the maximum distance to the most central node.

The distance between two vertices in an undirected graph is just the shortest path distance $d(\cdot, \cdot)$ between them. For directed graphs, however, this notion of distance d is no longer necessarily symmetric, and rather than being a distance *between* two nodes, it measures the distance in a given direction. Several related notions of pairwise distance that are symmetric have been studied. These include the roundtrip distance [15] which for two vertices u and v is just $d(u, v)+d(v, u)$, the max-distance [2] which is $\max\{d(u, v), d(v, u)\}$, and the min-distance [2] which is $\min\{d(u, v), d(v, u)\}$.

Each of these notions of distance has a particular application. For instance, one would have to pay the roundtrip distance when going to the store and back. On the other hand, if one needs medical assistance, one could either go to the hospital, or have a physician come to the home — the time to receive care is then measured by the min-distance. Another example of min-distance is in symmetric-key encryption: any pair of parties can create a shared private key by using only one-way communication.

*minad@mit.edu, virgi@mit.edu, nvyas@mit.edu, nwein@mit.edu, xyzhan@mit.edu, ycyu@mit.edu, MIT EECS and CSAIL

For each notion of distance, the diameter, radius and eccentricity parameters are well-defined. Given the shortest path distances $d(\cdot,\cdot)$ for all vertices, the parameters for each distance measure can be computed in $O(n^2)$ time in n vertex graphs. The fastest known algorithms for All-Pairs Shortest Paths (APSP) [24, 20, 21] give the fastest known algorithms to compute these parameters exactly, running in $n^3/\exp(\sqrt{\log n})$ time and $O(mn + n^2 \log\log n)$, respectively on m-edge, n-vertex graphs. Furthermore, under the Strong Exponential Hypothesis, there is no $O(m^{2-\varepsilon})$ time algorithm for Diameter in unweighted graphs (and thus also for any of these notions of Diameter and Eccentricities in directed graphs) [22]. For Radius, the same lower bound holds but under the "Hitting Set" conjecture [2].

As exact computation is expensive, it makes sense to resort to approximation algorithms. For the shortest path distance versions of Diameter, Eccentricities and Radius, there are several fast algorithms that achieve various small constant approximation ratios [22, 10, 8, 4]. For instance, for Diameter, a folklore linear time algorithm can achieve a 2-approximation, and an $\tilde{O}(m^{3/2})$ time[1] algorithm can achieve a $3/2$-approximation [22, 10].

Many of these algorithms [22, 10, 4] work for any distance measure that satisfies the triangle inequality. Thus they work for the shortest paths distance, max-distance and roundtrip distance. The min-distance however does not satisfy the triangle inequality: e.g. you might have edges (x, y) and (z, y), and thus the min-distance between x and y and between y and z are both 1, yet there may be no directed path between x and z in any direction, so that the min-distance between them may be ∞.

This issue makes it much more difficult to design fast approximation algorithms for Min-Diameter, Min-Radius and Min-Eccentricities (the parameters of interest under the min-distance). The only known non-trivial algorithms are by Abboud et al. [2]. For Min-Diameter [2] gives a near-linear time 2-approximation algorithm if the input is a directed acyclic graph. For general graphs, the only nontrivial fast approximation algorithm is an $\tilde{O}(mn^{1-\varepsilon})$ time n^{ε}-approximation algorithm for any constant $\varepsilon > 0$. (No constant factor approximation algorithm is known that runs significantly faster than just computing APSP.) For Min-Radius, [2] gives an $\tilde{O}(m\sqrt{n})$ time 3-approximation algorithm for directed acyclic graphs. For general graphs, they only achieve a very weak n-approximation in near-linear time that checks if the Min-Radius is finite. There are no known approximation algorithms for Min-Eccentricities faster than just computing APSP.

1.1 Our Results

The main goal of our paper is to obtain new fast, $O(mn^{1-\varepsilon})$ time for some constant $\varepsilon > 0$, algorithms for Min-Diameter, Min-Radius and Min-Eccentricities (thus beating the $\tilde{O}(mn)$ time of exact computation). We achieve this by developing powerful new techniques that can handle the complications that arise due to the fact that the min-distance does not satisfy the triangle inequality.

Our results are as follows. For Min-Diameter we achieve a hierarchy of algorithms trading off running time with approximation accuracy.

Theorem 1.1. *For any integer $0 < \ell \leq O(\log n)$, there is an $\tilde{O}(mn^{1/(\ell+1)})$ time randomized algorithm that, given a directed weighted graph G with edge weights non-negative and polynomial in n, can output an estimate $\tilde{D}$ such that $D/(4\ell - 1) \leq \tilde{D} \leq D$ with high probability, where D is the min-diameter of G.*

When we set $\ell = 1$, we obtain an $\tilde{O}(m\sqrt{n})$ time 3-approximation algorithm, and when we set $\ell = \lceil \log n \rceil$, we get an $\tilde{O}(m)$ time $O(\log n)$-approximation.

Our tradeoff achieves the first constant factor approximation algorithms for Min-Diameter in general graphs that run in $O(mn^{1-\varepsilon})$ time for constant $\varepsilon > 0$. Such a result was only known for directed acyclic graphs, whereas for general graphs the only known efficient algorithm could achieve an n^{ε}-approximation.

[1] We use $\tilde{O}$ notation to hide polylogarithmic factors

For Min-Radius, we also achieve the first constant factor approximation algorithm for general graphs running in $O(mn^{1-\varepsilon})$ time for some constant $\varepsilon > 0$. Such a result was only known for directed acyclic graphs, whereas for general graphs the only known efficient algorithm could only check if the Min-Radius is finite.

Theorem 1.2. *For any constant δ with $1 > \delta > 0$, there is an $\tilde{O}(m\sqrt{n}/\delta)$ time randomized algorithm, that given a directed weighted graph G with edge weights positive and polynomial in n, can output an estimate R' such that $R \leq R' \leq (3+\delta)R$ with high probability, where R is the min-radius of G.*

Finally, we obtain the first $O(mn^{1-\varepsilon})$ time (for constant $\varepsilon > 0$) constant factor approximation algorithms for the Min-Eccentricities of all vertices in a graph. For unweighted graphs we are able to obtain a close to 3 approximation in $\tilde{O}(m\sqrt{n})$ time. For weighted graphs, our approximation factor grows to 5, while the running time is the same. Previously, the only algorithm to approximate the Min-Eccentricities computed them exactly via an APSP computation.

Theorem 1.3. *For any constant δ with $1 > \delta > 0$, there is an $\tilde{O}(m\sqrt{n}/\delta)$ time randomized algorithm, that given a directed weighted graph $G = (V, E)$ with weights positive and polynomial in n, can output an estimate $\varepsilon'(s)$ for every vertex $s \in V$ such that $\varepsilon(s) \leq \varepsilon'(s) \leq (5+\delta)\varepsilon(s)$ with high probability, where $\varepsilon(s)$ is the min-eccentricity of vertex s in G.*

Theorem 1.4. *For any constant δ with $1 > \delta > 0$, there is an $\tilde{O}(m\sqrt{n}/\delta^2)$ time randomized algorithm, that given a directed unweighted graph $G = (V, E)$, can output an estimate $\varepsilon'(s)$ for every vertex $s \in V$ such that $\varepsilon(s) \leq \varepsilon'(s) \leq (3+\delta)\varepsilon(s)$ with high probability, where $\varepsilon(s)$ is the min-eccentricity of the vertex s in G.*

1.2 Our Techniques

To obtain our results, we develop powerful new techniques which we outline below.

Partial search graphs. The idea of partial search graphs is used in the algorithms of [2] for Min-Radius and Min-Diameter on DAGs. These algorithms use the following high-level framework: perform Dijkstra's algorithm from some vertices and then perform a *partial* Dijkstra's algorithm from *every* vertex. The partial search from a vertex v is with respect to a carefully defined partial search graph $G_v \subset G$. The crux of the analysis for the algorithms on DAGs is to argue that if the executions of Dijkstra's algorithm on the full graph did not find a good estimate for the desired quantity (either min-diameter or min-radius), then the partial search from some vertex v returns a good estimate of the min-eccentricity of v, which in turn is a good estimate for the desired quantity. In DAGs it is natural to define the partial search graphs G_v by considering a topological ordering of the vertices and letting each G_v be some interval containing v (though defining the exact intervals requires some work). For general graphs it is completely unclear how to even define such intervals since there is no natural notion of an ordering of the vertices, and thus figuring out what the G_v's should be is nontrivial. Our approach to overcoming this hurdle is to carefully define a DAG-like structure in general graphs. Such a structure may be of independent interest.

Defining a DAG-like structure in general graphs. It would be ideal to directly reduce the problem on general graphs to the problem on DAGs, however it is very unclear how to do this. Instead, we recognize that it suffices to define a *DAG-like* structure in general graphs. As a first step, we use the following idea. Suppose we have performed Dijkstra's algorithm from a vertex v. We let $S_v = \{u : d(u,v) < d(v,u)\}$ and we let $T_v = \{u : d(u,v) > d(v,u)\}$[2]. Then, we partially order the vertices so that the vertices in S_v

[2] u's with $d(u,v) = d(v,u)$ are added to either S_v or T_v as specified in the formal definition later

appear before v and those in T_v appear after v. We note that this partial ordering is "DAG-like" because it is consistent with the topological ordering of a DAG; that is, if we apply this partition into S_v and T_v to a DAG then there trivially exists a topological ordering such that every vertex in S_v appears before v and every vertex in T_v appears after v. After partitioning into S_v and T_v, we recursively partition each set to create a more precise partial ordering. Importantly, we show that by recursively sampling vertices randomly, we can guarantee that our partitioning is approximately balanced which is crucial for the runtime analysis. The obtained partial ordering is the starting point for all of our algorithms.

Min-Diameter: graph augmentation. The Min-Diameter algorithm on DAGs from [2] relies heavily on the following key property of DAGs. Consider a topological ordering and the graphs induced by the first and second halves of the ordering; which are defined with respect to the middle vertex in the ordering. For all pairs of vertices in the same half of the ordering, their min-distance in the graph induced by this half is the same as their min-distance in the full graph. As previously mentioned, if we sample a vertex v, we can make sure that S_v and T_v are approximately balanced, so that we can think of S_v and T_v as corresponding to the first and second half of a DAG topological ordering, respectively. However it is unclear how to obtain a property of S_v and T_v analogous to the above key property of DAGs. In particular, the min-distance between a pair of vertices in the graph induced by S_v could be wildly different from their min-distance in the full graph, since paths whose endpoints are in S_v can contain vertices outside of S_v. To overcome this hurdle, we *augment* the graph induced by S_v and the graph induced by T_v by carefully adding edges so that distances within these augmented graphs approximate distances in the original graph.

Min-Radius: refined DAG-like structure Our Min-Radius algorithm is much more delicate than our Min-Diameter algorithm due to the fact that for Min-Radius we care about small distances instead of large distances. In particular, the graph augmentation idea from our Min-Diameter algorithm does not help for Min-Radius because although the augmentations do not distort large distances much, they heavily distort small distances. Furthermore, the previously mentioned DAG-like structure for general graphs does not suffice for Min-Radius. However we use it as a starting point to define a more refined DAG-like partial ordering. Most of our algorithm is concerned with precisely arranging vertices in this partial ordering. Specifically, we structure the partial ordering to satisfy *roughly* the following property: for every pair of vertices u, v such that u appears before v in the partial ordering, $d(v,u)$ is large while $d(u,v)$ is small.

1.3 Notation

Given a graph $G = (V, E)$, $n = |V|$ and $m = |E|$. Graphs are directed and have non-negative weights polynomial in n unless otherwise specified. For any pair of vertices u and v, the *distance from u to v* $d(u,v)$ is the length of the shortest directed path from u to v. When the context is not clear, we write $d_G(u,v)$ to specify the graph G. The *min-distance* between a pair of vertices u and v is $d_{min}(u,v) = \min\{d(u,v), d(v,u)\}$. The *min-diameter* of a graph is $\max_{u,v\in V} d_{min}(u,v)$. The *min-radius* of a graph is $\min_{v\in V} \max_{u\in V} d_{min}(u,v)$. For any vertex v, the *min-eccentricity* of v is $\varepsilon(v) = \max_{u\in V} d_{min}(u,v)$. When the context is not clear, we say $\varepsilon_G(v)$ to specify the graph G. Note that we do not use the *min* subscript to denote the min-eccentricity of a vertex. For an algorithm with input size n we use *with high probability* to denote the probability $> 1 - 1/n^c$ for all constants c. We say some quantity is $poly(n)$ to mean it is $O(n^c)$ for some fixed constant c. We use $\tilde{O}$ notation to hide polylogarithmic factors.

1.4 Organization

In Section 2 we give an overview of all of our algorithms, in Section 3 we describe a graph partitioning procedure that begins all of our algorithms, in Section 4 we describe our Min-Diameter algorithms,

in Section 5 we describe our Min-Radius algorithm, and in Section 6 we describe our Min-Eccentricities algorithm.

2 Overview of Algorithms

We use the algorithms from [2] for Min-Diameter and Min-Radius on DAGs as inspiration. For each problem, we first outline the DAG algorithm and then provide intuition for how to apply these ideas to general graphs.

2.1 Min-Diameter

Algorithm for DAGs

We begin by outlining the $\tilde{O}(n+m)$ time 2-approximation algorithm for Min-Diameter on DAGs from [2]. Consider a topological ordering of the vertices and perform Dijkstra's algorithm from the middle vertex v. Then recurse on the graphs induced by the vertices in the first half (before v) and in the second half (after v). A key observation in the analysis is that if the true endpoints s^* and t^* of the min-diameter fall on opposite sides of v in the ordering, then the min-eccentricity $\varepsilon(v)$ of v is a 2-approximation for the min-diameter D. This is because if $\varepsilon(v) < D/2$ and s^* and t^* fall on opposite sides of v in the ordering, then $d(s^*, v) < D/2$ and $d(v, t^*) < D/2$ so $d(s^*, t^*) < D$, a contradiction. So, suppose (without loss of generality) that s^* and t^* both fall before v in the ordering. Since the graph is a DAG, every path between s^* and t^* only uses vertices before v in the ordering. Thus, the min-distance between s^* and t^* in the graph induced by the first half of the graph is still D.

Algorithm for general graphs

We now outline a precursor to our Min-Diameter algorithm for general graphs that mimics the algorithm for DAGs. This $\tilde{O}(n+m)$ time algorithm does not achieve a constant approximation factor, however it provides intuition for our constant-factor approximation algorithms. We begin by performing Dijkstra's algorithm from a vertex v and constructing S_v and T_v as defined in the previous section. Analogously to the DAG algorithm if the true min-diameter endpoints s^* and t^* fall into different sets S_v, T_v then the min-eccentricity $\varepsilon(v)$ is a 2-approximation. This is because if $\varepsilon(v) < D/2$, $s^* \in S_v$, and $t^* \in T_v$ then $d(s^*, v) < D/2$ and $d(v, t^*) < D/2$ so $d(s^*, t^*) < D$, a contradiction. However, unlike the DAG algorithm, we cannot simply recurse independently on the graphs induced by S_v and T_v since the shortest path between a pair of vertices in S_v may not be completely contained in S_v (and analogously for T_v).

To overcome this hurdle, before recursing we first augment the graphs induced by S_v and T_v by carefully adding edges so that distances within these augmented graphs approximate distances in the original graph. Specifically, for every vertex $u \in S_v$, we add the directed edge (u, v) with weight 0 and the directed edge (v, u) with weight $\max\{0, d(v,u) - \varepsilon(v)\}$. This choice of edges allows us to argue that the distances within the augmented graphs are approximations of the distances in G up to an additive error of $2\varepsilon(v)$. Then, by returning the maximum of $\varepsilon(v)$ and the min-diameter estimates from recursing on the augmented graphs, we get an approximation guarantee, which turns out to be a logarithmic factor. Intuitively, the approximation factor is not constant because the recursion causes the distance distortion to compound at each level of recursion.

To reduce the approximation factor to a constant, we would like to decrease the number of recursion levels. To achieve this, we initially partition the graph into more than just two parts S_v and T_v, by sampling more vertices. For our $\tilde{O}(m\sqrt{n})$ time 3-approximation, we perform a full Dijkstra's algorithm from $\tilde{O}(\sqrt{n})$ vertices to define an ordered partition of the vertices into $\tilde{O}(\sqrt{n})$ parts of $\tilde{O}(\sqrt{n})$ vertices each. Then we apply the above idea of adding weighted edges within each part, however we must refine the definition of the

graph augmentation to take into account *all* of the $\tilde{O}(\sqrt{n})$ vertices we initially perform Dijkstra's algorithm from, instead of just v. Finally we use brute force (without recursion) on each part in the partition by running an exact all-pairs shortest paths algorithm.

To achieve our time-accuracy trade-off algorithm, we carefully combine ideas from the logarithmic factor approximation and the 3-approximation algorithms. Specifically, we initially perform Dijkstra's algorithm from fewer than $\sqrt{n}$ vertices to define an ordered partition with larger parts than in the 3-approximation. Then we augment the graph induced by each part and carry out a constant number of recursion levels to further partition the graph before applying brute-force.

2.2 Min-Radius

Algorithm for DAGs

We begin by outlining the $\tilde{O}(m\sqrt{n})$ time 3-approximation algorithm for Min-Radius on DAGs from [2], which is very different from and more involved than the Min-Diameter algorithm on DAGs. We begin by considering a topological ordering of the vertices and performing Dijkstra's algorithm from a set W of $\tilde{O}(\sqrt{n})$ evenly spaced vertices including the first and last vertex. If a vertex $v \in W$ has min-eccentricity at most twice the true min-radius R then we have obtained a 2-approximation. (We do not know R in advance but we repeatedly run the algorithm with different values of R to perform a binary search on R.)

Otherwise, we will define intervals in the ordering such that the min-center c cannot be contained in any of these intervals. A key observation is that if there is a pair of vertices (u, v) such that u appears before v in the topological ordering and $d(u, v) > 2R$, then the min-center c cannot fall between u and v in the topological ordering. This is because if it did, then $d(u, c) \leq R$ and $d(c, v) \leq R$, so $d(u, v) \leq 2R$, a contradiction. We define the intervals that cannot contain c as follows: for all $v \in W$ we let a_v be the first vertex in the ordering such that $d(a_v, v) > 2R$ (if it exists, otherwise $a_v = v$) and define b_v to be the last vertex in the ordering such that $d(v, b_v) > 2R$ (if it exists, otherwise $b_v = v$). Then, the key observation implies that c cannot fall in the interval $[a_v, b_v]$ in the ordering. Now, we have a set of possibly overlapping intervals that cannot contain c. We take the union of these intervals to get a set of disjoint intervals that cannot contain c.

Every vertex u that does not appear in such an interval, falls between two consecutive intervals I_u and I'_u. We define the partial search graph of u to be the graph induced by the set of vertices in I_u or I'_u or between I_u and I'_u. After performing the partial searches, the algorithm returns 3 times the minimum min-radius of all partial search graph. Next we give the idea of the analysis, which demystifies the factor of 3 in the returned value.

We claim that if the min-eccentricity of a vertex with respect to its partial search graph is at most R, then its min-eccentricity with respect to the full graph is at most $3R$, and the min-eccentricity of the true min-center with respect to its partial search graph is at most R (because for any path in a DAG whose starting and ending points are in a certain interval, every vertex in the path is in that interval). Thus, assuming the claim, $3R$ is a 3-approximation for the min-radius. We now outline the proof of the claim. Let u be the min-center with the minimum min-radius R of all partial search graphs. Let $v \in W$ such that a_v is the first vertex (in the topological order) of I_u, then $v \in I_u$ and $d(v, u) \leq R$. Furthermore, by the definition of a_v, all vertices that appear before the beginning of the interval I_u have distance at most $2R$ to v, and thus distance at most $3R$ to u. A symmetric argument holds for vertices that appear after the end of the interval I'_u. Hence the min-eccentricity of u with respect to the full graph is at most $3R$.

This algorithm runs in time $O(m\sqrt{n})$ because the vertices of W are evenly spaced so there are no more than $\sqrt{n}$ vertices between each pair of consecutive intervals. This implies that in the partial searches, each edge is only scanned $O(\sqrt{n})$ times. (Furthermore, repeatedly running the algorithm to binary search for R

adds a logarithmic factor to the runtime.)

Algorithm for general graphs

We now give a high-level outline of our $\tilde{O}(m\sqrt{n})$ time 3-approximation algorithm for Min-Radius. This algorithm is much more delicate than our Min-Diameter algorithm, hence more of the details are deferred to the full description. We begin by running Dijkstra's algorithm from a set W of $\tilde{O}(\sqrt{n})$ randomly sampled vertices to recursively partition the vertices into S_v and T_v as outlined in Section 1.2. This defines an initial DAG-like structure, however our analysis requires constructing a much more refined DAG-like structure.

Perhaps counter-intuitively, it makes sense to place vertices that are *far* from each other in the graph *close* to each other in the DAG-like structure. The reason for this is illuminated by the Min-Radius algorithm on DAGs, in which we find pairs of vertices u, v that are far from each other and apply the key observation that the min-center cannot be between u and v in the topological ordering. Intuitively, it is as if we collapse the interval between u and v in the DAG since we do not have to search within this interval for the min-center. An analogous key observation is true for general graphs: if there is a pair of vertices (u, v) with $d_{min}(u, v) > 2R$, then either $c \in S_u \cap S_v$ or $c \in T_u \cap T_v$. This is because if $c \in T_u \cap S_v$, then $d(u, c) \leq R$ and $d(c, v) \leq R$ so $d(u, v) \leq 2R$, a contradiction; the last case $c \in S_u \cap T_v$ is symmetric. In our algorithm for general graphs, we ensure that far vertices are near each other in the DAG-like structure by doing the following: we let the *far graph* G_{far} be an undirected graph on V with an edge between $u \in W$ and $v \in V$ if $d_{min}(u, v) > 2R$. All vertices in W that are in the same connected component in G_{far} will be grouped in the DAG-like structure. We let F_i be the set of vertices in W that are in the i^{th} connected component of G_{far}.

To construct the DAG-like structure, we show that precisely chosen groups of F_is can be merged to create *supercomponents*, which constitute a DAG-like structure in the following sense: there is an ordering of supercomponents such that for every pair of vertices $u, v \in W$ where the supercomponent containing u appears before that containing v, $d(u, v)$ is small and $d(v, u)$ is large. Specifically, we define the *close graph* H whose vertex set is the set of F_is. We add a directed edge between a pair of vertices in H if there exists a short path (length $\leq 5R$) between the corresponding F_is. Then we merge all F_is that appear in the same strongly connected component of H into a supercomponent. This contraction of strongly connected components of H results in a DAG, which defines the ordering of the supercomponents.

Now that we have arranged the vertices in W into a DAG-like structure, we would like to fit every vertex in the graph into this structure. Based on the precise way that we have defined the supercomponents, we can use an intricate argument to show *roughly* the following property: for every vertex v there exists an i such that for every vertex $u \in W$ in the first i supercomponents, $d(u, v)$ is small and for every vertex $u \in W$ in the remaining supercomponents, $d(v, u)$ is small.

After fitting every vertex into the refined DAG-like ordering, we can define each partial search graph to be an interval in the ordering that is large enough to contain several supercomponents. In the algorithm for DAGs, there were two important properties of the partial search graphs: (1) the min-eccentricity of the true min-center with respect to its partial search graph is at most R, and (2) if the min-eccentricity of a vertex with respect to its partial search graph is at most R then its min-eccentricity with respect to the full graph is at most $3R$. We show that due to the precise structure of the supercomponents, refinements of properties (1) and (2) are also true for general graphs.

Intuitively, property (1) is roughly true because for every pair of vertices $u, v \in W$ such that u's supercomponent appears before v's in the ordering, $d(v, u) > 5R$, since otherwise this pair of supercomponents would be in the same strongly connected component of H and would have been merged into a single supercomponent. This implies that paths of length at most R to or from the min-center cannot stray beyond its partial search graph. Intuitively, property (2) is roughly true because for every pair of vertices $u, v \in W$

such that u's supercomponents appears before v's in the ordering, $d(u,v) \leq 2R$ because otherwise, u and v would be in the same component of G_{far} and thus be in the same supercomponent. Thus, like the argument for DAGs, for all u, all vertices that appear before u's partial search graph G_u have distance at most $2R$ to each supercomponent in G_u, and thus distance at most $3R$ to u. A symmetric argument holds for vertices after u in the ordering.

2.3 Min-Eccentricities

Our Min-Eccentricities algorithm is a modification of our Min-Radius algorithm. In our Min-Radius algorithm, we identify a vertex whose min-eccentricity is at most about $3R$, where R is the true min-radius. In our Min-Eccentricities algorithm, we show that with some extra bookkeeping, the algorithm can identify *all* vertices with min-eccentricity at most about 5ρ for any ρ. We run the algorithm repeatedly, increasing ρ by a factor of $(1+\delta)$ at each execution until we have estimated the min-eccentricity of every vertex.

The major modification of the Min-Radius algorithm here is that if one of the vertices that we run Dijkstra from has min-eccentricity at most 3ρ, we cannot stop running the algorithm, as we can in the Min-Radius algorithm. Instead, we use this vertex as a tool to find vertices with min-eccentricity at most 5ρ.

3 Preliminary Graph Partitioning

In this section we describe a graph partitioning procedure we use as a first step in our Min-Diameter, Min-Radius, and Min-Eccentricities algorithms. The goal of this partitioning is to define a DAG-like structure in general directed graphs.

Definition 3.1. *Assign each vertex a unique ID from $[n]$. For each vertex v, let $S_v = \{u \in V : d(u,v) < d(v,u) \vee [d(u,v) = d(v,u) \wedge ID(u) < ID(v)]\}$. Let $T_v = V \setminus (S_v \cup \{v\})$.*

The runtime of our algorithms relies on whether the partition into S_v and T_v is *balanced*. Using the observation that if $u \in S_v$, then $v \in T_u$, the following lemma shows that for most vertices, the partition is indeed approximately balanced.

Lemma 3.1. *For any graph on n vertices there are more than $\frac{n}{2}$ vertices v such that $\frac{|S_v|}{8} \leq |T_v| \leq 8|S_v|$.*

More generally, for any $U \subseteq V$, there are more than $\frac{|U|}{2}$ vertices $v \in U$ such that $\frac{|S_v \cap U|}{8} \leq |T_v \cap U| \leq 8|S_v \cap U|$.

Proof. Since the first statement is a special case of the second statement with $U = V$, we prove the more general statement. Let $|U| = k$. Let M be a $k \times k$ matrix indexed by the vertices in U where $M_{u,v} = -1$ if $u \in S_v \cap U$, $M_{u,v} = 1$ if $u \in T_v \cap U$, and $M_{u,u} = 0$ for $u \in U$. Note that M is skew-symmetric, i.e., $M_{u,v} = -M_{v,u}$ for all u, v. For any $A, B \subseteq U$, let M_B be the $k \times |B|$ submatrix consisting of the columns indexed by B, and let $M_{A,B}$ the $|A| \times |B|$ submatrix of M_B consisting of its rows indexed by A.

Suppose for contradiction there is a set $C \subset U$ of $\frac{k}{4}$ vertices v such that $|T_v \cap U| > 8|S_v \cap U|$. Then M_C contains at least $\frac{8}{9}k \cdot \frac{k}{4} = \frac{2}{9}k^2$ ones.

The $\frac{k}{4} \times \frac{k}{4}$ submatrix $M_{C,C}$ is also skew-symmetric, so at most half of its entries are ones, i.e., $M_{C,C}$ contains at most $\frac{k^2}{32}$ ones. Letting $\bar{C} = U \setminus C$, we see that $M_{\bar{C},C}$ has $\frac{3}{4}k \times \frac{k}{4} = \frac{3}{16}k^2$ entries, and hence at most $\frac{3}{16}k^2$ ones. In total, M_C contains at most $\frac{7}{32}k^2 < \frac{2}{9}k^2$ ones, contradiction.

Therefore the number of vertices $v \in U$ such that $|T_v \cap U| > 8|S_v \cap U|$ is less than $\frac{k}{4}$, and symmetrically the number of vertices $v \in U$ such that $|T_v \cap U| < \frac{|S_v \cap U|}{8}$ is less than $\frac{k}{4}$. Hence more than half of the vertices $v \in U$ have that $\frac{|S_v \cap U|}{8} < |T_v \cap U| < 8|S_v \cap U|$. □

Next, we describe how we use Lemma 3.1 to recursively construct a balanced partition of the vertices into a given number of of sets.

Lemma 3.2. *Given a graph G with n vertices and a constant $c > 0$, in $\tilde{O}(mn^{1-c})$ time we can partition V into disjoint sets $W, V_1, V_2, \ldots, V_{q+1}$, where $q = |W| = n^{1-c}$, such that with high probability:*

1. *for all i, $|V_i| = \Theta(\frac{n}{q})$;*
2. *for all $i \neq j$, there exists a vertex $w \in W$ such that either $V_i \subseteq S_w$,$V_j \subseteq T_w$, or $V_i \subseteq T_w$,$V_j \subseteq S_w$;*
3. *for all $U \subseteq W$, let $V_U = \left(\bigcap_{w \in U} S_w\right) \bigcap \left(\bigcap_{w \in W \setminus U} T_w\right)$, then $V_U \subseteq V_i$ for some $i \in [q+1]$.*

Proof. We begin with $W = \emptyset$ and we will iteratively populate W with vertices. We let $\mathcal{V}_0 = \{V\}$ and for all $i \in [q]$ when we add the i^{th} vertex to W, we will construct $\mathcal{V}_i$ from $\mathcal{V}_{i-1}$ by partitioning the largest set in $\mathcal{V}_{i-1}$ into two parts. After adding q vertices to W we will have constructed $\mathcal{V}_q = \{V_1 \ldots V_{q+1}\}$.

For all $i \in [q]$, let A_i, B_i be the largest and smallest sets in $\mathcal{V}_i$, respectively.

We describe how to construct W and $\mathcal{V}_q$ inductively. Suppose $|W| = r - 1$ and we have constructed $\mathcal{V}_{r-1}$. By Lemma 3.1, if we randomly sample $O(\log^2 n)$ vertices from A_{r-1}, with probability at least $1 - 2^{-\log^2 n} = 1 - n^{-\log n}$ we will sample a vertex w_r such that $A_S = A_{r-1} \cap S_{w_r}$ and $A_T = A_{r-1} \cap T_{w_r}$ differ by a factor of at most 8. We add w_r to W and let $\mathcal{V}_r = \mathcal{V}_{r-1} \cup \{A_S, A_T\} \setminus \{A_{r-1}\}$.

By union bound over the $q = n^{1-c}$ partitionings, with probability at least $1 - n^{1-c-\log n}$, every partitioning produces two sets that differ in size by a factor of at most 8.

We prove property 1 by induction on $|W| = r$. Specifically, we will show that for all $r \in [q]$, $|A_r| \leq 9|B_r|$. This implies that $|A_q| = O(|B_q|)$, and property 1 follows. Lemma 3.1 implies that $|A_1| \leq 9|B_1|$. Assume inductively that $|A_{r-1}| \leq 9|B_{r-1}|$. Since no subset grows in size, $|A_r| \leq |A_{r-1}|$ and $|B_r| \leq |B_{r-1}|$. If $|B_r| = |B_{r-1}|$, then $|A_r| \leq |A_{r-1}| \leq 9|B_{r-1}| = 9|B_r|$. Otherwise, $|B_r| < |B_{r-1}|$, which implies that B_r is one of the two sets obtained by partitioning A_{r-1}. In this case $|A_{r-1}| \leq 9|B_r|$ by Lemma 3.1. Hence $|A_r| \leq |A_{r-1}| \leq 9|B_r|$, completing the induction.

Property 2 follows from the partitioning procedure: for any $i \neq j$, if for all $w \in W$, $V_i, V_j \subseteq S_w$ or $V_i, V_j \subseteq T_w$ then $V_i \cup V_j$ would never have been partitioned.

Property 3 also follows from the partitioning procedure: observe that for all $w \in W$ and all $U \subseteq W$, $V_U \subseteq S_w$ or $V_U \subseteq T_w$, so V_U is never partitioned and thus $V_U \subseteq V_i$ for some $i \in [q+1]$.

Since we sample $n^{1-c}\log^2 n$ vertices and for all v finding S_v, T_v takes $O(m)$ time, the runtime is $\tilde{O}(mn^{1-c})$.

□

4 Min-Diameter Algorithm

Throughout this section, let D be the min-diameter, and let s^*, t^* the endpoints of the min-diameter. In this section we prove the time/accuracy trade-off theorem for Min-Diameter.

Theorem 4.1. *For any integer $0 < \ell \leq O(\log n)$, there is an $\tilde{O}(mn^{1/(\ell+1)})$ time randomized algorithm that, given a directed weighted graph G with edge weights non-negative and polynomial in n, can output an estimate $\tilde{D}$ such that $D/(4\ell - 1) \leq \tilde{D} \leq D$ with high probability, where D is the min-diameter of G.*

We first prove a special case of Theorem 4.1 where $\ell = 1$.

4.1 An $\tilde{O}(m\sqrt{n})$ time 3-approximation

Theorem 4.2. (Theorem 4.1 *with $\ell = 1$) There is an $\tilde{O}(m\sqrt{n})$ time randomized algorithm, that given a directed weighted graph $G = (V, E)$ with edge weights non-negative and polynomial in n, can output an estimate $\tilde{D}$ such that $D/3 \leq \tilde{D} \leq D$ with high probability, where D is the min-diameter of G.*

4.1.1 Algorithm Description

Applying Lemma 3.2 with $q = \sqrt{n}$ we obtain a partition of the vertices into $W, V_1, V_2, \ldots, V_{\sqrt{n}+1}$.

We perform Dijkstra's algorithm from every vertex in W and define $D' = \max_{w \in W} \varepsilon(w)$. We will later show that D' is a good approximation of the Min-Diameter when s^* and t^* are not in the same vertex set V_i.

For every $i \in [\sqrt{n}+1]$, define $W_i^S = \{w \in W : V_i \subseteq S_w\}$, and $W_i^T = \{w \in W : V_i \subseteq T_w\}$. Then, for every i, we construct two graphs G_i^S and G_i^T. The first graph G_i^S contains all vertices of V_i and an additional node w_i^S. It has the following edges:

1. For every directed edge $(u, v) \in E$ such that $u, v \in V_i$, add this edge to G_i^S.
2. Add a directed edge from w_i^S to every $v \in V_i$, with weight $\max\left\{\min_{w \in W_i^S} d(w, v) - D', 0\right\}$, and a directed edge from every $v \in V_i$ to w_i^S with weight 0.

The second graph G_i^T is symmetric to G_i^S. It contains all vertices in V_i and an additional node w_i^T. It has the following edges:

1. For every directed edge $(u, v) \in E$ such that $u, v \in V_i$, add this edge to G_i^T.
2. Add a directed edge from every $v \in V_i$ to w_i^T, with weight $\max\left\{\min_{w \in W_i^T} d(v, w) - D', 0\right\}$, and add a directed edge from w_i^T to every $v \in V_i$ with weight 0.

For all i, we run an exact all-pairs shortest paths algorithm on G_i^S and G_i^T. This allows us to compute for all i and all $u, v \in V_i$ the quantity $\min\{d_{G_i^S}(u, v), d_{G_i^T}(u, v)\}$, which we denote by $d_i'(u, v)$.

We choose the larger between D' and $\max_{i \in [\sqrt{n}+1], u, v \in V_i} \min\{d_i'(u, v), d_i'(v, u)\}$ as our final estimate for the min-diameter.

4.1.2 Analysis

The following lemma will be used to show that D' is a good estimate for the min-diameter if s^* and t^* happen to fall into different sets V_i

Lemma 4.1. *For all vertices v, if either $s^* \in S_v$, $t^* \in T_v$, or $t^* \in S_v$, $s^* \in T_v$, then $\varepsilon(v) \geq D/2$.*

Proof. We only consider the case when $s^* \in S_v$ and $t^* \in T_v$ as the other case is symmetric. By way of contradiction, assume that $\varepsilon(v) < D/2$, then we have $d_{\min}(s^*, v) < D/2$ and $d_{\min}(t^*, v) < D/2$. Since $s^* \in S_v$, $d(s^*, v) = d_{\min}(s^*, v) < D/2$; similarly, since $t^* \in T_v$, $d(v, t^*) = d_{min}(t^*, v) < D/2$. Therefore, by the triangle inequality, $d(s^*, t^*) < D$, a contradiction. □

The next two lemmas are used for the case where s^* and t^* fall into the same set V_i.

Lemma 4.2. *For every i, and every pair of vertices $u, v \in V_i$, $d_i'(u, v) \leq d(u, v)$; that is, $\min\{d_{G_i^S}(u, v), d_{G_i^T}(u, v)\} \leq d(u, v)$.*

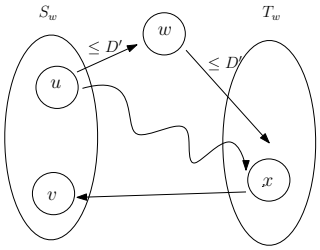

Figure 1: The case where $u, v \in S_w$ and the shortest path from u to v contains a node $x \in T_w \cup \{w\}$.

Proof. Take any shortest path in the original graph G from u to v. If this path does not leave V_i, then this path also exists in G_i^S and G_i^T, and thus the inequality is true.

It remains to prove for the case when the shortest u, v path in the original graph leaves V_i. Let $x \notin V_i$ be any vertex on a shortest u, v path. By Lemma 3.2, property 2, there exists $w \in W$ such that $x \in S_w \cup \{w\}$ and $V_i \subseteq T_w$, or $x \in T_w \cup \{w\}$ and $V_i \subseteq S_w$. We first assume $x \in T_w \cup \{w\}$ and $V_i \subseteq S_w$ as shown in Figure 1, and the other case is symmetric.

Since x is on the shortest path from u to v, we have $d(u, v) \geq d(x, v)$. Also, we have $d(w, x) \leq D'$, by definition of D'. Therefore,

$$\begin{aligned} d(u, v) &\geq d(x, v) \\ &\geq d(x, v) + \big(d(w, x) - D'\big) \\ &\geq d(w, v) - D' \end{aligned} \tag{1}$$

Now consider the path $u \to w_i^S \to v$ in G_i^S. The first part $u \to w_i^S$ costs 0, because there is an edge from u to w_i^S with weight 0; the second part $w_i^S \to v$ costs at most $\max\{0, d(w, v) - D'\}$. If $d(w, v) < D'$, then $d'_i(u, v) \leq d_{G_i^S}(u, v) = 0 \leq d(u, v)$; otherwise, $d'_i(u, v) \leq d_{G_i^S}(u, v) \leq d(w, v) - D' \leq d(u, v)$, where the last step is Equation 1.

When $x \in S_w \cup \{w\}$, and $V_i \subseteq T_w$, we have a symmetric argument: $d(u, v) \geq d(u, x) \geq d(u, x) + (d(x, w) - D') \geq d(u, w) - D'$. Consider the path $u \to w_i^T \to v$ in G_i^T. The second part $w_i^T \to v$ costs 0, because there is an edge from w_i^T to v with weight 0; the first part $u \to w_i^T$ costs at most $\max\{0, d(u, w) - D'\}$. If $d(u, w) < D'$, then $d'_i(u, v) \leq d_{G_i^T}(u, v) = 0 \leq d(u, v)$; otherwise, $d'_i(u, v) \leq d_{G_i^T}(u, v) \leq d(u, w) - D' \leq d(u, v)$. □

Lemma 4.3. *For every i, and every pair of vertices $u, v \in V_i$, $d'_i(u, v) \geq d(u, v) - 2D'$; that is, $d_{G_i^S}(u, v) \geq d(u, v) - 2D'$ and $d_{G_i^T}(u, v) \geq d(u, v) - 2D'$.*

Proof. We only provide full proof for $d_{G_i^S}(u, v) \geq d(u, v) - 2D'$. The inequality for G_i^T can be proved by a symmetrical argument. If the shortest path from u to v in G_i^S does not contain w_i^S, then this path also exists in the original graph G, and thus the inequality is true.

Otherwise, the shortest path from u to v in G_i^S contains w_i^S, as shown in Figure 2. All edges on the shortest path from w_i^S to v exist in the original graph G except for the first edge from w_i^S to some node x, since a shortest path cannot use the vertex w_i^S more than once. That is, $d_{G_i^S}(x, v) = d(x, v)$.

By the definition of w_i^S and the edges incident to it, there exists a $w \in W_i^S$ such that $d(w, x) \leq d_{G_i^S}(w_i^S, x) + D'$. Thus, we have

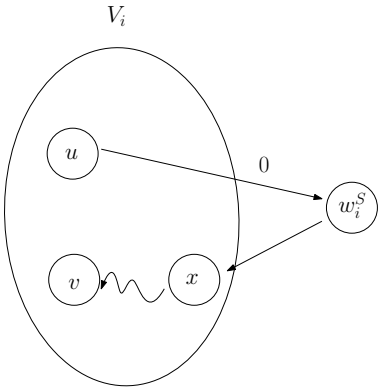

Figure 2: A shortest u, v path in G_i^S that contains w_i^S. The path goes from u, directly to w_i^S using a weight 0 edge, then directly to a vertex x, and finally reaches v.

$$
\begin{aligned}
d_{G_i^S}(u,v) &= d_{G_i^S}(u,w_i^S) + d_{G_i^S}(w_i^S,x) + d_{G_i^S}(x,v) \\
&= d_{G_i^S}(w_i^S,x) + d_{G_i^S}(x,v) && \text{since } d_{G_i^S}(u,w_i^S) = 0 \text{ by construction} \\
&= d_{G_i^S}(w_i^S,x) + d(x,v) && \text{from argument above} \\
&\geq d(w,x) - D' + d(x,v) && \text{by the definition of } w \\
&\geq d(w,v) - D' && \text{by the triangle inequality} \\
&\geq \big(d(w,v) - D'\big) + \big(d(u,w) - D'\big) && \text{since } d(u,w) \leq D' \text{ by definition} \\
&\geq d(u,v) - 2D' && \text{by the triangle inequality}
\end{aligned}
$$

□

We are now ready to prove our approximation ratio guarantee: $D/3 \leq \tilde{D} \leq D$. Clearly $D' \leq D$ because D' is the min-eccentricity of a vertex. By Lemma 4.2 $\max_{i,u\in V_i,v\in V_i} \min\{d'_i(u,v), d'_i(v,u)\} \leq \max_{i,u\in V_i,v\in V_i} d_{min}(u,v) \leq D$. Therefore, we never over estimate the Min-Diameter.

If $s^* \in W$ or $t^* \in W$, then since we run Dijkstra from all vertices in W we have $D' = D$. So assuming that $s^*, t^* \notin W$, we have two cases.

Case 1: s^* and t^* are not in the same vertex set V_i. By Lemma 3.2, property 2, there exists $w \in W$ such that one of s^* and t^* is in S_w and the other is in T_w, so by Lemma 4.1, $\varepsilon(w) \geq D/2$. Since $D' \geq \varepsilon(w)$, we have $D' \geq D/2$.

Case 2: s^* and t^* are in the same vertex set V_i for some i. By Lemma 4.3, $\min\left(d'_i(s^*,t^*), d'_i(t^*,s^*)\right) \geq d_{min}(s^*,t^*) - 2D' = D - 2D'$. Since $\max\{D-2D', D'\} \geq D/3$, we get a 3-approximation.

Runtime analysis It takes $\tilde{O}(m\sqrt{n})$ time to perform the partitioning from Lemma 3.2 and to perform Dijkstra's algorithm from all $w \in W$ since $|W| = O(\sqrt{n})$.

For all i, the number of vertices in G_i^S is $|V_i| + 1 = O(\sqrt{n})$ with high probability by property 1 of Lemma 3.2 and the number of edges is $m_i + O(\sqrt{n})$ where m_i is the number of edges in the graph induced by V_i. Hence we can run an all-pairs shortest paths algorithm on G_i^S in time $\tilde{O}((m_i + \sqrt{n})\sqrt{n})$. Summing over all i gives us $\tilde{O}(m\sqrt{n})$. The same analysis also works for G_T^i.

4.2 Time/accuracy trade-off algorithm

4.2.1 Algorithm Description

We begin by briefly outlining the differences between our trade-off algorithm and our $O(m\sqrt{n})$ time algorithm. For our trade-off algorithm, instead of applying Lemma 3.2 to sample $q = \sqrt{n}$ vertices, we will apply Lemma 3.2 with a smaller value of q to save time. This results in a smaller set W and larger sets V_i. In our $O(m\sqrt{n})$ time algorithm, we had time to apply brute force (i.e. run all-pairs shortest paths) on the graphs G_i^S and G_i^T, however in our trade-off algorithm we do not. Instead, we apply recursion. Simply constructing G_i^S and G_i^T and recursing on both of them does not suffice because each recursive call only returns the min-diameter, whereas we require knowing all distances. To overcome this issue, instead of constructing G_i^S and G_i^T separately, we construct a graph G_i that combines these two graphs. Then, we show that it suffices to recurse on G_i to compute only its min-diameter rather than all distances.

The algorithm is as follows. We apply Lemma 3.2 with $q = O(n^{1/(\ell+1)})$ to partition the vertices into $W, V_1, V_2, \ldots, V_{q+1}$. We perform Dijkstra's algorithm from every vertex in W and define $D' = \max_{w \in W} \varepsilon(w)$. For every $i \in [\sqrt{n}+1]$, we define $W_i^S = \{w \in W : V_i \subseteq S_w\}$, and $W_i^T = \{w \in W : V_i \subseteq T_w\}$. For every $i \in [q+1]$, we construct the graph G_i as follows. The vertex set of G_i is all vertices V_i and two additional vertices w_i^S and w_i^T. It contains the following edges:

1. For every directed edge $(u, v) \in E$ such that $u, v \in V_i$, add this edge to G_i.
2. Add a directed edge from w_i^S to every $v \in V_i$, with weight $\max\{\min_{w \in W_i^S} d(w, v) - D', 0\}$, and add a directed edge from every v to w_i^S with weight 0.
3. Add a directed edge from every $v \in V_i$ to w_i^T, with weight $\max\{\min_{w \in W_i^T} d(v, w) - D', 0\}$, and add a directed edge from w_i^T to every $v \in V_i$ with weight 0.

For all i, we recursively compute a $(4\ell - 5)$-approximation for the Min-Diameter of G_i by calling the algorithm for $\ell - 1$. We use the $\ell = 1$ algorithm from the previous section as the base case.

We choose the larger between D' and the maximum approximated Min-Diameter over all G_i as our final estimate.

4.2.2 Analysis

Before proving the main theorem for Min-Diameter, we need to prove two lemmas for G_i, which are analogous to Lemma 4.2 and Lemma 4.3.

Lemma 4.4. *For every i, and every pair of vertices $u, v \in V_i$, $d(u, v) \geq d_{G_i}(u, v)$.*

Proof. Since $G_i^S \subseteq G_i$ and $G_i^T \subseteq G_i$, we have $d_{G_i}(u, v) \leq d_{G_i^S}(u, v)$ and $d_{G_i}(u, v) \leq d_{G_i^T}(u, v)$. Then by Lemma 4.2, we have $d(u, v) \geq \min\{d_{G_i^S}(u, v), d_{G_i^T}(u, v)\} \geq d_{G_i}(u, v)$. □

Lemma 4.5. *For every i, and every pair of vertices $u, v \in V_i$, $d_{G_i}(u, v) \geq d(u, v) - 4D'$.*

Proof. Consider the shortest path from u to v in G_i. If this path does not contain both w_i^S and w_i^T, then this path exists in G_i^S or G_i^T, and thus we can directly apply Lemma 4.3 to get $d_{G_i}(u, v) \geq d_{G_i^S}(u, v) \geq d(u, v) - 2D'$, or $d_{G_i}(u, v) \geq d_{G_i^T}(u, v) \geq d(u, v) - 2D'$.

Otherwise, the shortest path from u to v contain both w_i^S and w_i^T. Such path can only be one of the following two forms:

- $u \to w_i^S \to x \to w_i^T \to v$ for some vertex $x \in V_i$. The first half $u \to w_i^S \to x$ is contained in G_i^S, so we can apply Lemma 4.3 to get $d_{G_i}(u,x) = d_{G_i^S}(u,x) \geq d(u,x) - 2D'$; similarly, the second half $x \to w_i^T \to v$ is contained in G_i^T so $d_{G_i}(x,v) \geq d(x,v) - 2D'$. In total, $d_{G_i}(u,v) = d_{G_i}(u,x) + d_{G_i}(x,v) \geq (d(u,x) - 2D') + (d(x,v) - 2D') \geq d(u,v) - 4D'$.
- $u \to w_i^T \to x \to w_i^S \to v$ for some vertex $x \in V_i$. We can similarly split this path to two halves, and apply the same analysis as the previous case to get $d_{G_i}(u,v) \geq d(u,v) - 4D'$.

□

We are now ready to prove our approximation ratio guarantee: $D/(4\ell-1) \leq \tilde{D} \leq D$. We prove the result inductively. When $\ell = 1$, it is exactly Theorem 4.2. Now assume it is true for $\ell - 1$, and we will prove it for ℓ.

Clearly $D' \leq D$ because D' is the min-eccentricity of a vertex. By induction, the $(4\ell-5)$-approximation for the min-diameter of G_i never exceeds the true min-diameter of G_i. Then by Lemma 4.4, the min-diameter of G_i does not exceed the min-diameter of G. Therefore, we never over estimate the min-diameter.

If $s^* \in W$ or $t^* \in W$, then since we run Dijkstra from all vertices in W we have $D' = D$. So assuming that $s^*, t^* \notin W$, we have two cases.

Case 1: s^* and t^* are not in the same vertex set V_i. By Lemma 3.2, property 2, there exists $w \in W$ such that one of s^* and t^* is in S_w and the other is in T_w, so by Lemma 4.1, $\varepsilon(w) \geq D/2$. Since $D' \geq \varepsilon(w)$, we have $D' \geq D/2$.

Case 2: s^* and t^* are in the same vertex set V_i for some i. If $D' \geq D/(4\ell-1)$, D' is already a good approximation. So assume $D' < D/(4\ell-1)$. By Lemma 4.5, $\min\{d_{G_i}(s^*,t^*), d_{G_i}(t^*,s^*)\} \geq d_{min}(s^*,t^*) - 4D' = D - 4D'$. Since we calculate a $(4\ell-5)$-approximation of G_i's min diameter, our estimate is at least

$$(D - 4D')/(4\ell-5) \geq (D - 4(D/(4\ell-1)))/(4\ell-5) = D/(4\ell-1)$$

Runtime analysis It takes $\tilde{O}(mn^{1/(\ell+1)})$ time to perform the partitioning from Lemma 3.2 and to perform Dijkstra's algorithm from all $w \in W$ since $|W| = O(n^{1/(\ell+1)})$. For all i, the number of vertices in G_i is $|V_i| + 2 = O(n^{\ell/(\ell+1)})$ with high probability by Lemma 3.2, property 1, and the number of edges is $m_i + O(n^{\ell/(\ell+1)})$ where m_i is the number of edges in the graph induced by V_i. By induction, it takes $\tilde{O}\left((m_i + n^{\ell/(\ell+1)})\left(n^{\ell/(\ell+1)}\right)^{1/\ell}\right)$ time to compute a $(4\ell-5)$-approximation of Min-Diameter of G_i for each i. Summing over all i gives us $\tilde{O}(mn^{1/(\ell+1)})$.

Note that we apply Lemma 3.2 at most $poly(n)$ times in the recursion and this the only randomization so the whole algorithm works with high probability.

5 Min-Radius Algorithm

Theorem 5.1. *For any constant δ with $1 > \delta > 0$, there is an $\tilde{O}(m\sqrt{n}/\delta)$ time randomized algorithm that, given a directed weighted graph $G = (V, E)$ with weights positive and polynomial in n, can output an estimate R' such that $R \leq R' \leq (3+\delta)R$ with high probability, where R is the min-radius of the G.*

Proof. We fix a value r and our algorithm either certifies that $R > r$ or $R \leq 3r$. Then by a binary search argument we get a $(3+\delta)$-approximation as follows. Let $\delta' = \delta/3$. Starting from $r = 1$, we run the algorithm and increase r for each run. If the output of the algorithm is that $R \leq 3r$, then we stop. Otherwise (if $R > r$), we run the algorithm with the new value $r_{new} = (1+\delta')r$. This contributes a multiplicative

factor of $\log_{1+\delta'} R = \tilde{O}(1/\delta)$ to the total runtime. Suppose that for some value of r we have $R \leq 3r$. So from the previous run of the algorithm, we know that $R > r/(1+\delta')$. Letting $R' = 3r$, we have $R \leq 3r = R' < 3(1+\delta')R = (3+\delta)R$, which means that R' is a $(3+\delta)$-approximation. Now we present the algorithm.

Algorithm Step 1: Preliminaries

Let c be the min-center (which is unknown). First we remove all the edges with weight more than r, because if $R \leq r$, this removal does not change the min-radius. Then we sample a set W of $\sqrt{n}$ vertices according to Lemma 3.2. For every vertex $v \in W$, we run Dijkstra's algorithm from and to v to obtain the min-distance between v and all other vertices. If there exists a vertex $v \in W$ with $\varepsilon(v) \leq 3r$, we have certified that $R \leq 3r$ so we are done.

Algorithm Step 2: Constructing the "far graph"

Now we can assume that for each $v \in W$, $\varepsilon(v) > 3r$. We say that a pair of vertices is *far* if their min-distance is more than $2r$, and let the *far graph* G_{far} be an undirected unweighted graph on V defined as follows: for each $u \in W$ and $v \in V$, (u, v) is an undirected edge if u and v are far. We partition W based on the connected components of G_{far}. Specifically, for all i define Z_i to be the i^{th} connected component of G_{far} which contains at least one vertex in W. Let $F_i = W \cap Z_i$, note that F_i is non-empty.

Analysis Step 2

Remember that we defined $S_U = \bigcap_{v \in U} S_v$ and $T_U = \bigcap_{v \in U} T_v$.

By constructing G_{far}, we prune the set of candidate min-centers, as specified in the following lemma.

Lemma 5.1. *If $R \leq r$, then for any F_i either $c \in S_{F_i}$ or $c \in T_{F_i}$.*

Proof. First note that we have $S_{F_i} \cup T_{F_i} \neq V \setminus F_i$. We know that $c \notin F_i$ as $F_i \subseteq W$. By way of contradiction, assume that there are two vertices $u, v \in F_i$ such that $c \in S_u \cap T_v$. Consider a path in G_{far} from u to v. There must be a pair of adjacent vertices (u', v') on the path such that $c \in S_{u'} \cap T_{v'}$. Then, by definition, u' and v' are far (with respect to the original graph G). Since $c \in S_{u'} \cap T_{v'}$, we have $d(v', c) \leq r$ and $d(c, u') \leq r$, so by the triangle inequality $d(v', u') \leq 2r$. Thus u' and v' are not far, a contradiction. □

Algorithm Step 3: Defining a DAG-like structure

a) Constructing the "close graph" The purpose of constructing the close graph is that it allows us to either perform Dijkstra's algorithm from some additional vertices and obtain a good estimate (see step b), or "merge" some connected components of the far graph to further prune the set of vertices that could be the min-center (see step c). The *close graph* G_{close} is an unweighted directed graph with one vertex f_i for each F_i. For all i and j, let (f_i, f_j) be an edge in G_{close} if for some $u \in F_i$ and some $v \in F_j$, $d(u, v) \leq 5r$.

b) Additional Dijkstra We now perform Dijkstra's algorithm from some additional vertices, which are carefully chosen so that either we find a vertex with small min-eccentricity and are done in this step, or we can define a DAG-like structure in the graph (step c). We compute the strongly connected components (SCCs) of G_{close}. For each SCC $Q = (V_Q, E_Q)$, find $E'_Q \subseteq E_Q$ with $|E'_Q| \leq 2|V_Q|$ such that $Q' = (V_Q, E'_Q)$ is strongly connected; it is simple to show that such an E'_Q exists and we include the proof in the appendix for completeness (Lemma A.2). Let $E' = \cup_Q E'_Q$. Note that every edge $e \in E'$ corresponds to a path P_e of length at most $5r$ in the original graph G. For each $e \in E'$, find an ordered set V_e of at most 9 vertices on P_e that divide P_e into subpaths of length at most r; it is simple to show that such a V_e exists and we include the proof in the appendix for completeness (Lemma A.1). We run Dijkstra's algorithm from every vertex in V_e and if we find a vertex v with $\varepsilon(v) \leq 3r$ then we are done.

c) Constructing the DAG of "supercomponents" Let H be the DAG created by contracting every strongly connected component of G_{close} into a single vertex. That is, there is an edge from u to v in H if the strongly connected component v is reachable from the strongly connected component u. Let k be the number of vertices in H; we number the vertices in H from 1 to k according to a topological ordering. For each $j \in [k]$, we merge the set of F_i's represented by vertex j in H into a *supercomponent* W_j. Formally, if we define F_u to be the connected component of G_{far} that contains u, a vertex $u \in W$ is in supercomponent W_j if f_u is in the strongly connected component of H represented by vertex j.

d) Fitting the remaining vertices into the DAG structure In the previous step, we defined a DAG-like structure on the vertices in W. Now we place the rest of the vertices into this structure. We partition the rest of the vertices based on whether they could potentially be the min-center. We define the vertex sets C and B next and in the analysis we prove that $c \in C$ (among other properties of C and B). We will use the following notation: for any distance $d > 0$, let $S_v^d = \{u \in S_v : d(u, v) \leq d\}$ and let $T_v^d = \{u \in T_v : d(v, u) \leq d\}$. Remember that for any set U of vertices, we defined $S_U^d = \bigcap_{v \in U} S_v^d$, and $T_U^d = \bigcap_{v \in U} T_v^d$.

- For $i = 1, \ldots, k+1$, let $v \in C_i$ if for all $j < i$, $v \in T_{W_j}^{2r}$ and for all $j \geq i$, $v \in S_{W_j}^{2r}$. Let $C = \cup_{i=1}^{k+1} C_i$.
- For $i = 2, \ldots, k+1$, let $v \in B_i$ if $v \notin C$ and i is the largest integer for which $v \in T_{W_{i-1}}^{2r}$. Let $v \in B_1$ if there is no such i and $v \notin C$. Let $B = \cup_{i=1}^{k+1} B_i$.

Analysis Step 3

Figure 3 shows a summary of the structure of the graph which we will describe in the following observations and lemmas.

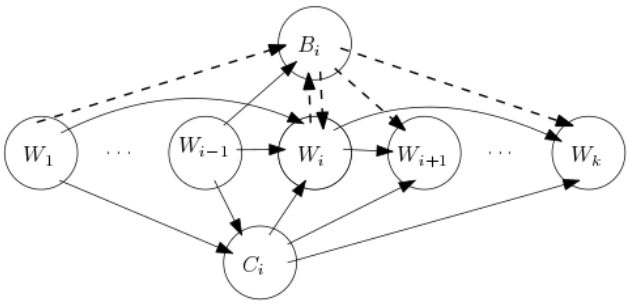

Figure 3: The graph structure for the sets W_i, B_i and C_i. Solid lines are paths of length at most $2r$ between any member of the outgoing set to any member of the incoming set. Dashed lines are paths of length at most $2r$ which might not exist between all pairs, which is expressed more accurately in Lemma 5.4.

We first observe two important properties of supercomponents:

Observation 5.1. *For every pair of vertices $v_i \in W_i$ and $v_j \in W_j$ with $i < j$, $d(v_j, v_i) > 5r$.*

This is true because if $d(v_j, v_i) \leq 5r$, then there is an edge from f_j to f_i in G_{close}, so there is an edge from j to i in H. Since $i < j$, this contradicts the topological ordering of H.

Observation 5.2. *For every pair of vertices $v_i \in W_i$ and $v_j \in W_j$ with $i < j$, $v_i \in S_{W_j}^{2r}$ and $v_j \in T_{W_i}^{2r}$.*

This is true because v_i and v_j are in different F_k's since W_i and W_j are collections of disjoint sets of F_i's. So v_i and v_j are not far i.e. $d_{min}(v_i, v_j) \leq 2r$ and by Observation 5.1 we know that $d(v_j, v_i) > 5r > 2r$, so it must be that $d(v_i, v_j) \leq 2r$. Since this is true for all vertices $v_j \in W_j$, we have $v_i \in S^{2r}_{W_j}$. Similarly, $v_j \in T^{2r}_{W_i}$.

We now prove a refinement of Lemma 5.1 where we consider supercomponents instead of far graph components. This further prunes the vertices that could potentially be the min-center.

Lemma 5.2. *If $R \leq r$, then for each $i = 1, \ldots, k$, either $c \in S_{W_i}$ or $c \in T_{W_i}$.*

Proof. Fix i and suppose by way of contradiction that there are nodes $u, v \in W_i$ such that $c \in S_u \cap T_v$. By Lemma 5.1, u and v must be in different F_i's say F_u and F_v.

Recall that by the definition of a supercomponent, f_u and f_v are in the same strongly connected component of G_{close}. So there is a path P from f_u to f_v in G_{close} such that all of its edges are in E'. By Lemma 5.1 Since $c \in S_u \cap T_v$, we have that $c \in S_{F_u} \cap T_{F_v}$. So there are two consecutive nodes f_j and $f_{j'}$ on P (in that order) such that $c \in S_{F_j} \cap T_{F_{j'}}$.

Recall that each edge $e \in E'$ corresponds to a path P_e of length at most $5r$ in the original graph. Let e be the edge $(f_j, f_{j'})$ and consider P_e and V_e, where V_e is the set of vertices that divides P_e into subpaths of length at most r. Since the endpoints of P_e are in F_j and $F_{j'}$ respectively, there exists a pair of vertices u', v' consecutive in V_e (in that order) such that $c \in S_{u'} \cap T_{v'}$. We note that $d(u', v') \leq r$.

Now we claim that $\varepsilon(v') \leq 3r$. This is because $d(v', c) \leq r$ and $d(c, v') \leq d(c, u') + d(u', v') \leq 2r$. Consider an arbitrary vertex $w \in V$. Either $d(c, w) \leq R$ or $d(w, c) \leq R$. If $d(c, w) \leq R$ then $d(v', w) \leq d(v', c) + d(c, w) \leq 2r$. If $d(w, c) \leq R$, then $d(w, v') \leq d(w, c) + d(c, v') \leq 3r$. In this case, the algorithm would have stopped after step 3b.

□

We now prove that $c \in C$, which further prunes the vertices that could potentially be the min-center.

Lemma 5.3. *If $R \leq r$, then $c \in C$.*

Proof. By Lemma 5.2, either $c \in S_{W_i}$ or $c \in T_{W_i}$. Since c is the min-center and $R \leq r$, if $c \in S_{W_i}$ then $c \in S^{2r}_{W_i}$, and similarly if $c \in T_{W_i}$ then $c \in T^{2r}_{W_i}$. We claim that for each $i < j$, $S^{2r}_{W_i} \cap T^{2r}_{W_j} = \emptyset$, which completes the proof. Suppose otherwise and let i and j be such that $i < j$ and there is a vertex $v \in S^{2r}_{W_i} \cap T^{2r}_{W_j}$. Then for every vertex $v_i \in W_i$ and $v_j \in W_j$, $d(v_j, v) \leq 2r$ and $d(v, v_i) \leq 2r$, so $d(v_j, v_i) \leq 4r$. This contradicts Observation 5.1. □

Now we prove that the vertices in B fit into the DAG structure in a similar but weaker sense than the vertices in C:

Lemma 5.4. *Consider a node $v \in B_i$. Then for all $z \geq i$ except for at most two values, we have $v \in S^{2r}_{W_z}$. And for all $z \leq i$ except for at most two values, we have $v \in T^{2r}_{W_z}$.*

Proof. We first observe that there is at most one j such that v is far from some vertex in W_j. This is because if v were far from two vertices u, w in different supercomponents, then G_{far} would contain the edges (u, v) and (w, v) making u and w in the same connected component of G_{far}, and thus in the same supercomponent. We fix j and consider two cases:

Case 1: Suppose by way of contradiction that for some node $w \in W_z$ for some $z < i$, $z \neq j$, we have $v \in S^{2r}_w$. We know that $z < i - 1$, since by definition of B_i, we have $v \in T^{2r}_{W_{i-1}}$. Let $w' \in W_{i-1}$ be an arbitrary node, then $d(w', w) \leq d(w', v) + d(v, w) \leq 2r + 2r < 5r$, a contradiction to Observation 5.1.

Case 2: Now suppose that for some node $w \in W_z$ for some $z > i$, $z \neq j$, we have $v \in T_w^{2r}$. We will show that $j = i$ and $z = i + 1$; that is, for all $z' \geq i + 2$, we have that $v \in S_{W_{z'}}^{2r}$. If there is some node $w' \in W_i$ such that $v \in S_{w'}^{2r}$, then $d(w, w') \leq d(w, v) + d(v, w') \leq 2r + 2r < 5r$, a contradiction to Observation 5.1. Assume that there is no such w' i.e. $d(v, w') > 2r$ for all $w' \in W_i$. Then for every node $w' \in W_i$, either v and w' are far or $d(w', v) \leq 2r$. If for all $w' \in W_i$, $d(w', v) \leq 2r$, then $v \in T_{W_i}^{2r}$, which cannot happen since by the definition of B_i, i is the biggest integer that $v \in T_{W_{i-1}}^{2r}$. Thus, v is far from some vertex in W_i so we have that $j = i$. If $z > i + 1$, then by definition of B_i there is some vertex $u \in W_{i+1}$ such that $v \in S_u^{2r}$. So $d(w, u) \leq d(w, v) + d(v, u) \leq 2r + 2r < 5r$, a contradiction to Observation 5.1. So it must be that $z = i + 1$. So for all $z' \geq i + 2$, we have that $v \in S_{W_{z'}}^{2r}$. □

We have observed stronger properties than Lemma 5.4 for vertices $v \in W_i$ (Observation 5.2) and $v \in C_i$ (by definition), so we have the following corollary.

Corollary 5.1. *Lemma 5.4 is true for all $v \in B_i \cup C_i \cup W_i$. Moreover, for such v's, we have $v \in T_{W_{i-1}}^{2r}$.*

Algorithm Step 4: Partial search

From each of the potential min-centers, we will run Dijkstra's algorithm on a small subgraph of G. For each $i = 1, \ldots, k + 1$, let G_i be the subgraph of G induced by $W_{i-6} \cup \ldots \cup W_{i+3} \cup B_{i-6} \cup \ldots \cup B_{i+3} \cup C_{i-5} \cup \ldots \cup C_{i+3}$. Define $\bar{C}_i$ to be the set of nodes $v \in C_i$ such that v is within min-distance r from all vertices in W (we know this set of nodes because we have already run Dijkstra's algorithm from and to every vertex in W). For every vertex v in $\bar{C}_i$, run Dijkstra's algorithm from v with respect to the graph G_i. If v is within min-distance r from all nodes in $U_i = C_i \cup B_{i-2} \cup B_{i-1} \cup B_i$, we will show that $R \leq 3r$. If there is no such v, we will show that $r < R$.

Analysis Step 4

The following two claims prove that our algorithm either certifies that $R > r$ or $R \leq 3r$.

Claim 1. *For some i, if $c \in \bar{C}_i$ and $R \leq r$, then for all $u \in U_i$, the min-distance between c and u with respect to G_i is at most r.*

Claim 2. *If a vertex $v \in \bar{C}_i$ is within min-distance r from all vertices in U_i with respect to the graph G_i, then $R \leq \varepsilon(v) \leq 3r$.*

Proof of Claim 1. We will prove something slightly stronger: for all i any shortest path in G between two nodes $u, u' \in U_i$ that has length at most r is completely contained in G_i.

By way of contradiction, suppose that the shortest path P from u to u' is not completely in G_i. Define V_r and V_l to be sets of nodes on the right and left of G_i respectively, i.e. $V_r = W_{i+4} \cup \ldots \cup W_k \cup B_{i+4} \cup \ldots \cup B_{k+1} \cup C_{i+4} \cup \ldots \cup C_{k+1}$ and $V_l = W_1 \cup \ldots \cup W_{i-7} \cup B_1 \cup \ldots \cup B_{i-7} \cup C_1 \cup \ldots \cup C_{i-6}$.

First suppose that P contains some node $v_r \in V_r$. There is some $j > i+3$ such that $v_r \in B_j \cup C_j \cup W_j$. So by Corollary 5.1, $v_r \in T_{W_{j-1}}^{2r}$. Furthermore, Corollary 5.1 implies that there is some $j' \in \{i, i+1, i+2\}$ such that $u' \in S_{W_{j'}}^{2r}$. Pick $w_{j'} \in W_{j'}$ and $w_{j-1} \in W_{j-1}$. We have that $d(w_{j-1}, w_{j'}) \leq d(w_{j-1}, v_r) + d(v_r, u') + d(u', w_{j'}) \leq 2r + r + 2r = 5r$. Since $j - 1 > j'$, this contradicts Observation 5.1. This case is shown in Figure 4.

Now suppose that P contains some node v_l in V_l. The argument in this case is symmetric to the previous case. Since $u \in U_i$, there is some $j \in \{i - 2, i - 1, i\}$ such that $u \in B_j \cup C_j$, and hence by Corollary 5.1, $u \in T_{W_{j-1}}^{2r}$. Furthermore, Corollary 5.1 implies that there is at least one value $j' \in \{i - 5, i - 4, i - 3\}$ such that $v_l \in S_{W_{j'}}^{2r}$. Pick $w_{j'} \in W_{j'}$ and $w_{j-1} \in W_{j-1}$. We have that $d(w_{j-1}, w_{j'}) \leq d(w_{j-1}, u) + d(u, v_l) + d(v_l, w_{j'}) \leq 2r + r + 2r = 5r$. Since $j' < j - 1$, this contradicts Observation 5.1. □

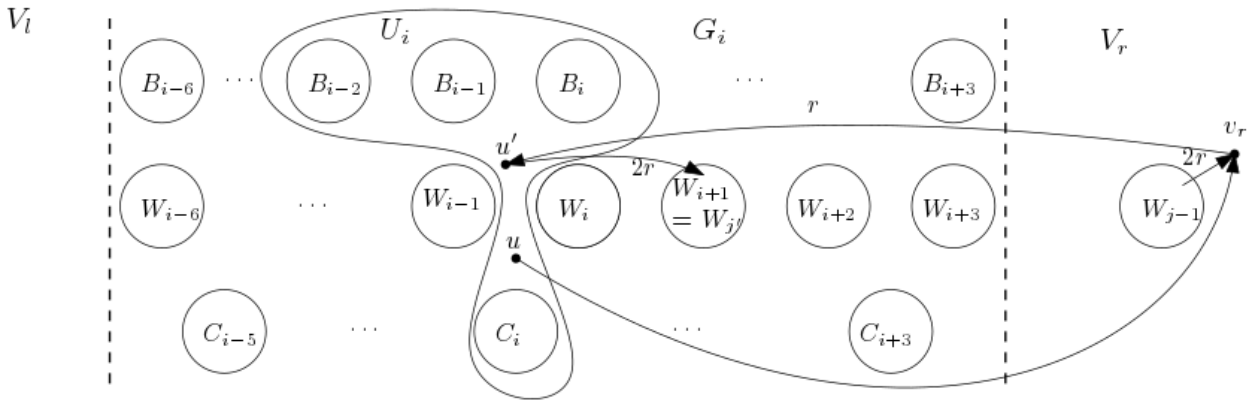

Figure 4: First case in Claim 1 where the path P from u to u' passes through some vertex $v_r \in V_r$. In this figure $j' = i + 1$. The upper bound on the weight of each part of the path from W_{j-1} to $W_{j'}$ is written on the edges.

Proof of Claim 2. We show that for any node $u \in V$ we have $d_{min}(u, v) \leq 3r$. We have 3 cases:
Case 1: $u \in W$: From the definition of $\bar{C}_i$, we know that v has min-distance at most r to all vertices in W.
Case 2: $u \in C_j$ for some $j = 1, \ldots, k+1$. If $j = i$, then $u \in U_i$ so we know that $d_{min}(u, v) \leq r$. If $j > i$, then pick some vertex $w_i \in W_i$. By the definition of C_i and C_j we know that $d(v, u) \leq d(v, w_i) + d(w_i, u) \leq r + 2r = 3r$. Similarly, if $j \leq i - 1$, pick some vertex $w_{i-1} \in W_{i-1}$. Then $d(u, v) \leq d(u, w_{i-1}) + d(w_{i-1}, v) \leq 2r + r = 3r$.
Case 3: $u \in B_j$ for some $j = 1, \ldots, k+1$. If $j \in \{i-2, i-1, i\}$, then since $v \in \bar{C}_i$ we know that $d_{min}(u, v) \leq r$. So first suppose that $j \leq i - 3$. Then by Lemma 5.4, there is at least one $j' \in \{j+1, \ldots, i-1\}$, such that $u \in S^{2r}_{W_{j'}}$. Pick some node $w_{j'} \in W_{j'}$. So by the definition of C_i we have that $d(u, v) \leq d(u, w_{j'}) + d(w_{j'}, v) \leq 2r + r = 3r$. Now suppose that $j \geq i + 1$. Then by definition of B_j we know that $u \in T^{2r}_{W_{j-1}}$. Pick some vertex $w_{j-1} \in W_{j-1}$. Since $j - 1 \geq i$ and by the definition of C_i, we have that $d(v, u) \leq d(v, w_{j-1}) + d(w_{j-1}, u) \leq r + 2r = 3r$.

□

Runtime analysis

We analyze the running time of each step.
Step 1, preliminaries: $\tilde{O}(m\sqrt{n})$. This is because each Dijkstra in Step 1 takes $\tilde{O}(m)$ time and $|W| = \sqrt{n}$.
Step 2, constructing the "far graph": $\tilde{O}(n\sqrt{n})$. Each edge in the far graph has at least one endpoint in W, and so the construction of the far graph takes $O(n\sqrt{n})$ time. Note that the existence of each edge in the far graph was determined in Step 1.
Step 3, defining a DAG-like structure:

a, constructing the "close graph": $\tilde{O}(n\sqrt{n})$. This is because the connected components of the far graph can be determined in $O(n\sqrt{n})$ time since it has that many edges. The number of components containing a node in W are not more than $|W|$, and so the close graph which is on at most $|W|$ nodes can be constructed in time $O(|W|^2) = O(n)$.

b, additional Dijkstra: $\tilde{O}(m\sqrt{n})$. This is because by Lemma A.2, the number of vertices we run Dijkstra from in SCC Q of the close graph is at most $9|E_Q| \leq 18|V_Q|$. Since the number of vertices in close graph is at most $|W|$, we run Dijkstra from at most $18|W| = \tilde{O}(\sqrt{n})$ vertices. Also running the algorithm of Lemma A.2 takes $O(|E_Q|) = O(|V_Q|^2)$ for each SCC Q, which takes $O(|W|^2) = \tilde{O}(n)$ time in total.

c, constructing the DAG of "supercomponents": $\tilde{O}(n)$. This is because H has at most $|W|$ vertices, so obtaining the DAG ordering of H takes at most $|W|^2 = \tilde{O}(n)$ time.

d, fitting the remaining vertices into the DAG structure: $\tilde{O}(n\sqrt{n})$. For each vertex in V, it takes $O(|W|)$ time to see which set it belongs to, since it only depends on its distances to and from the vertices in W.

Step 4, partial search: $\tilde{O}(m\sqrt{n})$. The Dijkstras ran in G_i take $\tilde{O}(m_i|C_i|)$ time, where m_i is the number of edges with at least one endpoint in G_i. By Lemma 3.2, property 3, with high probability $|C_i| = O(\sqrt{n})$. We know that $\bar{C}_i \subseteq C_i$ and so the running time of this step is $O(\sqrt{n}\sum_{i=1}^{k+1} m_i)$. Now since each node is in at most 10 G_is, we have that each edge is also in at most 20 G_is, and hence $\sum_{i=1}^{k+1} m_i \leq 20m$.

So overall the algorithm runs in $\tilde{O}(m\sqrt{n})$ time.

□

6 Min-Eccentricities Algorithm

The min-eccentricities algorithm is similar to the min-radius algorithm. Below we will describe the modifications.

Theorem 6.1. *For any constant δ with $1 > \delta > 0$, there is an $\tilde{O}(m\sqrt{n}/\delta)$ time randomized algorithm, that given a directed weighted graph $G = (V, E)$ with weights positive and polynomial in n, can output an estimate $\varepsilon'(s)$ for every vertex $s \in V$ such that $\varepsilon(s) \leq \varepsilon'(s) \leq (5+\delta)\varepsilon(s)$ with high probability, where $\varepsilon(s)$ is the min-eccentricity of the vertex s in G.*

Proof. We fix a value ρ and our algorithm certifies for each $s \in V$ that either $\varepsilon(s) > \rho$ or $\varepsilon(s) \leq 5\rho$ with high probability. Starting from $\rho = 1$, we will run the algorithm and increase ρ for each run. We will call the vertices for which we have certified $\varepsilon(s) \leq 5\rho$ for earlier values of ρ as *marked*. Let $\delta' = \delta/5$. Starting from $\rho = 1$, we run the algorithm. If the output of the algorithm is that $\varepsilon(s) \leq 5\rho$ and s was unmarked, then we will mark s and set $\varepsilon'(s) = 5\rho$. Then, we run the algorithm with the new value $\rho_{new} = (1+\delta')\rho$. Since $\varepsilon(s) \leq poly(n)$ for all $s \in V$, this contributes a multiplicative factor of $\log_{1+\delta'} n = \tilde{O}(1/\delta)$ to the total runtime. Suppose that for some value of ρ and for some vertex s we have $\varepsilon(s) \leq 5\rho$ and s was unmarked. From the previous run of the algorithm, we know that $\varepsilon(s) > \rho/(1+\delta')$. Then for $\varepsilon'(s) = 5\rho$, we have $\varepsilon'(s) \geq \varepsilon(s)$ and $\varepsilon'(s) \leq 5(1+\delta')\varepsilon(s) = (5+\delta)\varepsilon(s)$, which means that $\varepsilon'(s)$ is a $(5+\delta)$-approximation of $\varepsilon(s)$. After running the whole algorithm for this value of ρ we will also mark all such vertices s. Now we present the algorithm.

Throughout the algorithm ρ behaves analogously to r in the min-radius algorithm. Whenever we say that a certain part of the algorithm is the same we mean that it is same after replacing r by ρ. Note that any vertex s with $\varepsilon(s) = \rho$ satisfies the property that its min-distance to any vertex is at most ρ. This is analogous to the center vertex c in the Min-Radius algorithm using $r = \rho$.

Algorithm Step 1: Preliminaries

First we remove all the edges with weight more than ρ, because if for a vertex s with $\varepsilon(s) \leq \rho$, this removal does not change the min-eccentricity of s. Then we sample a set W of $\sqrt{n}$ vertices according to Lemma 3.2. For every vertex $v \in W$, we run Dijkstra's algorithm from and to v to obtain the min-distance between v and all other vertices. This means we know $\varepsilon(v)$ for all $v \in W$ and in particular we know if $\varepsilon(v) > \rho$ or $\varepsilon(v) \leq 3\rho$. We use the vertices in W with min-eccentricity less than 3ρ to detect vertices with min-eccentricity less than 5ρ in the graph.

Algorithm Step 2: Constructing the "far graph"

The far graph and the F_i's are defined the same way as in the min-radius algorithm.

Analysis Step 2

The purpose of constructing G_{far} is to prune the set of vertices that could potentially have low min-eccentricity. Next we state a modified Lemma 5.1.

Lemma 6.1 (Modification of Lemma 5.1)**.** *If for a vertex $s \in V \setminus W$, $\varepsilon(s) \leq \rho$, then for any F_i, either $s \in S_{F_i}$ or $s \in T_{F_i}$.*

Proof. For $s \in F_i$ note that $F_i \subseteq W$ and hence we know $\varepsilon(s)$ and have certified either $\varepsilon(s) > \rho$ or $\varepsilon(s) \leq 3\rho$. For the other vertices the proof is analogous to that of Lemma 5.1 □

Algorithm Step 3: Defining a DAG-like structure

a) Constructing the "close graph" The purpose of constructing the close graph is that it allows us to perform Dijkstra's algorithm from some additional vertices and certify some vertices as having min-eccentricities $\leq 5\rho$. Then we "merge" some connected components of the far graph to further prune the set of vertices that could be having small min-eccentricities (see step c). G_{close} is defined as in the min-radius algorithm.

b) Additional Dijkstra This step of the algorithm diverges from the min-radius algorithm at the end, and hence we state it in full detail. Similar to the min-radius algorithm, we perform Dijkstra's algorithm from some additional vertices, which are chosen so that we detect more vertices with low min-eccentricity and at the end define a DAG-like structure (step c). Recall that we compute the strongly connected components (SCCs) of G_{close}. For each SCC $Q = (V_Q, E_Q)$, find $E'_Q \subseteq E_Q$ with $|E'_Q| \leq 2|V_Q|$ such that $Q' = (V_Q, E'_Q)$ is strongly connected (where the existence of E'_Q is shown in Lemma A.2). Let $E' = \cup_Q E'_Q$. Recall that every edge $e \in E'$ corresponds to a path P_e of length at most 5ρ in the original graph G. For each $e \in E'$, find an ordered set V_e of at most 9 vertices on P_e that divide P_e into sections of length at most ρ (see Lemma A.1). For each $e \in E'$, we run Dijkstra's algorithm from and to every vertex in V_e. This means we know $\varepsilon(v)$ for all $v \in V_e$; and in particular we know whether $\varepsilon(v) > \rho$ or $\varepsilon(v) \leq 3\rho$. Now here is the new part of the algorithm in this step: For every consecutive pair of vertices (a, b) in V_e over all e with $\varepsilon(a), \varepsilon(b) \leq 3\rho$ we certify for all $s \in S_b^\rho \cap T_a^\rho$ that $\varepsilon(s) \leq 5\rho$.

c) Constructing the DAG of "supercomponents" The graphs H, W_i's and the "supercomponents" are defined as in the min-radius algorithm.

d) Fitting the remaining vertices into the DAG structure In the previous step, we defined a DAG-like structure on the vertices of W. Now we place the rest of the vertices into this structure. We partition the rest of the vertices based on whether they haven't been certified to have eccentricity $\leq 5\rho$ *and* could potentially have small eccentricity. Vertex sets C and B are defined as in the min-radius algorithm. In the analysis we prove that all vertices which haven't been certified to have eccentricity $\leq 5\rho$ *and* could potentially have small eccentricity must be in C, among other properties of C and B.

Analysis Step 3

First note that one major difference of this algorithm and the min-radius algorithm is in part b; in the min-radius algorithm we stop whenever we find a good approximate center among the vertices in V_es, but here we can only upper bound the eccentricity of some vertices by 5ρ if we find vertices with eccentricity $\leq 3\rho$ among V_es.

We first show that if for some vertex s and for some consecutive pair of vertices (a, b) in V_e such that $\varepsilon(a), \varepsilon(b) \leq 3\rho$ and $s \in S_b^\rho \cap T_a^\rho$, then $\varepsilon(s) \leq 5\rho$. This is derived by Lemma 6.2 which we state bellow, by the following substitution: let $c = s$, $\gamma_1 = \rho, \gamma_2 = 2\rho$ and $\gamma_3 = 3\rho$.

Lemma 6.2. *Consider vertices b, c such that $d(b,c) \leq \gamma_1$, $d(c,b) \leq \gamma_2$ and $\varepsilon(b) \leq \gamma_3$ then $\varepsilon(c) \leq \gamma_3 + \max(\gamma_1, \gamma_2)$.*

Proof. Consider a vertex v, as $\varepsilon(b) \leq \gamma_3$ either $d(v,b) \leq \gamma_3$ or $d(b,v) \leq \gamma_3$. If $d(v,b) \leq \gamma_3$ then $d(v,c) \leq d(v,b) + d(b,c) \leq \gamma_3 + \gamma_1$. Otherwise $d(b,v) \leq \gamma_3$ then $d(c,v) \leq d(c,b) + d(b,v) \leq \gamma_3 + \gamma_2$. In both cases $\varepsilon(c) \leq \gamma_3 + \max(\gamma_1, \gamma_2)$. □

Now we observe an important property of supercomponents with an analogous proof to that of Observation 5.1.

Observation 6.1 (Modification of Observation 5.1)**.** *For every pair of vertices in $v_i \in W_i$ and $v_j \in W_j$ with $i < j$, $d(v_j, v_i) > 5\rho$.*

We now prove a modification of Lemma 5.2. This further prunes the vertices that could potentially have small eccentricity.

Lemma 6.3 (Modification of Lemma 5.2)**.** *If for a vertex $s \in V$, $\varepsilon(s) \leq \rho$ and we haven't yet certified $\varepsilon(s) \leq 5\rho$ then for each $i = 1, \ldots, k$, either $s \in S_{W_i}$ or $s \in T_{W_i}$.*

Proof. Fix i and suppose by way of contradiction that there are nodes $u, v \in W_i$ such that $s \in S_u \cap T_v$ and $\varepsilon(s) \leq \rho$. By Lemma 6.1, u and v must be in different F_i's say F_u and F_v.

Recall that by the definition of a supercomponent, f_u and f_v are in the same strongly connected component of G_{close}. So there is a path P from f_u to f_v in G_{close} such that all of its edges are in E'. By Lemma 6.1 since $s \in S_u \cap T_v$, we have that $s \in S_{F_u} \cap T_{F_v}$. So there are two consecutive nodes f_j and $f_{j'}$ on P (in that order) such that $s \in S_{F_j} \cap T_{F_{j'}}$.

Recall that an edge $e \in E'$ corresponds to a path P_e of length at most 5ρ in the original graph. Let e be the edge $(f_j, f_{j'})$ and consider P_e and V_e. Since the endpoints of P_e are in F_j and $F_{j'}$ respectively, there exists a pair of vertices u', v' consecutive in V_e (in that order) such that $s \in S_{u'} \cap T_{v'}$. We note that $d(u', v') \leq \rho$.

Recall that we assumed that $\varepsilon(s) \leq \rho$. Note as well that $d(v', s) \leq \rho$ and $d(s, v') \leq d(s, u') + d(u', v') \leq 2\rho$. Then, using Lemma 6.2 with $b = s, \gamma_3 = \rho, c = v', \gamma_1 = 2\rho, \gamma_2 = \rho$, we get that $\varepsilon(v') \leq \rho + \max\{2\rho, \rho\} = 3\rho$. A symmetric argument holds for u', giving $\varepsilon(u'), \varepsilon(v') \leq 3\rho$. In this case, the algorithm would have already marked s in step 3b as it is in the intersection of $S^{\rho}_{u'} \cup T^{\rho}_{v'}$. □

We now prove that for vertices s which have small min-eccentricity and have not been certified as such, $s \in C$. The proof is analogous to that of Lemma 5.3.

Lemma 6.4 (Modification of Lemma 5.3)**.** *If for a vertex $s \in V$, $\varepsilon(s) \leq \rho$ and we haven't yet certified $\varepsilon(s) \leq 5\rho$ then $s \in C$.*

Now we prove that the vertices in B fit into the DAG structure in a similar but weaker sense than the vertices in C. The proofs are analogous to those of Lemma 5.4 and Corollary 5.1.

Lemma 6.5 (Modification of Lemma 5.4)**.** *Consider a node $v \in B_i$. Then for all $z \leq i$ except for at most two values, we have $v \in T^{2\rho}_{W_z}$. And for all $z \geq i$ except for at most two values, we have $v \in S^{2\rho}_{W_z}$.*

Corollary 6.1 (Modification of Corollary 5.1)**.** *Lemma 6.5 is true for all $v \in B_i \cup C_i \cup W_i$. Moreover for all $v \in B_i \cup C_i \cup W_i$, we have $v \in T^{2\rho}_{W_{i-1}}$.*

Algorithm Step 4: Partial search

From each of the potential vertices with small min-eccentricity in C, we will run Dijkstra's algorithm on a small subgraph of G. G_i and U_i are defined as in the min-radius algorithm. Define $\bar{C}_i$ to be the set of nodes $v \in C_i$ such that v is within min-distance ρ from all vertices in W (we know this set of nodes because we have already run Dijkstra's algorithm from and to every vertex in W). From each node $v \in \bar{C}_i$ run Dijkstra's algorithm from and to v with respect to the graph G_i. If v is within min-distance ρ from all nodes in U_i, we will show that this certifies that $\varepsilon(s) \leq 3\rho$ and otherwise $\varepsilon(s) > \rho$.

Analysis Step 4

The following two claims prove that our algorithm for vertices $s \in C$ either certifies that $\varepsilon(s) > \rho$ or $\varepsilon(s) \leq 3\rho$. The proofs are analogous to those of Claim 1 and Claim 2.

Claim 3 (Modification of Claim 1)**.** *If $s \in C_i$ and $\varepsilon(s) \leq \rho$, then for all $u \in U_i$, the min-distance between c and u with respect to G_i is at most ρ.*

Claim 4 (Modification of Claim 2)**.** *If a vertex s is within min-distance ρ from all vertices in U_i in G_i, then $\varepsilon(s) \leq 3\rho$.*

For all the vertices for which we haven't certified either $\varepsilon(s) \leq 3\rho$ or $\varepsilon(s) \leq 5\rho$ we know that $\varepsilon(s) > \rho$ and can certify that.

□

The runtime is $\tilde{O}(m\sqrt{n})$ with analogous runtime analysis to that of the min-radius algorithm.

6.1 $(3+\delta)$-approximation for unweighted graphs

In this part we show that given an unweighted graph, by a slight modification of the min-eccentricity algorithm in Theorem 6.1, we are able to improve the approximation factor of the min-eccentricity problem to match that of the min-radius problem, namely we present a $(3+\delta)$-approximation algorithm for every $\delta > 0$.

Theorem 6.2. *For any constant δ with $1 > \delta > 0$, there is an $\tilde{O}(m\sqrt{n}/\delta^2)$ time randomized algorithm, that given a directed unweighted graph $G = (V, E)$, can output an estimate $\varepsilon'(s)$ for every vertex $s \in V$ such that $\varepsilon(s) \leq \varepsilon'(s) \leq (3+\delta)\varepsilon(s)$ with high probability, where $\varepsilon(s)$ is the min-eccentricity of the vertex s in G.*

Proof. There are only two parts of the algorithm in Theorem 6.1 that change:

(1) Letting $\delta' = \delta/5$, in each run of the algorithm, for each vertex s, we certify that either $\varepsilon(s) > \rho$ or $\varepsilon(s) \leq (3+\delta')\rho$ (instead of $\varepsilon(s) \leq 5\rho$). The subsequent changes follow naturally: We start from $\rho = 1$ and we run the algorithm and increase ρ by a factor of $(1+\delta')$. We call the vertices for which we have certified $\varepsilon(s) \leq (3+\delta')\rho$ for earlier values of ρ as *marked*, and if for an unmarked vertex s the output of the algorithm is $\varepsilon(s) \leq (3+\delta')\rho$, then we let $\varepsilon'(s) = (3+\delta')\rho$. If for some value of ρ and for some vertex s we have $\varepsilon(s) \leq (3+\delta')\rho$ and s was unmarked, then from the previous run of the algorithm, we know that $\varepsilon(s) > \rho/(1+\delta')$. So for $\varepsilon'(s) = (3+\delta')\rho$, we have $\varepsilon'(s) \geq \varepsilon(s)$ and $\varepsilon'(s) \leq (3+\delta')(1+\delta')\varepsilon(s) = (3+\delta)\varepsilon(s)$.

(2) In step 3, part b of the algorithm (Additional Dijkstra), recall that each edge $e \in E'$ is a path of length at most 5ρ in G. Now instead of dividing each e into at most 9 subpaths of length at most ρ, we divide it into subpaths of length at most $\delta'\rho/2 \geq 1$ using at most $20/\delta' - 1 = O(1/\delta')$ vertices which we call V_e. The rest of this step follows naturally: We run Dijkstra from and to each $v \in V_e$, so we know that whether $\varepsilon(v) > \rho$ or $\varepsilon(v) < (2+\delta'/2)\rho$. For every consecutive pair of vertices (a, b) in V_e over all e with

$\varepsilon(a), \varepsilon(b) \leq (2+\delta'/2)\rho$ we certify for all $s \in S_b^\rho \cap T_a^\rho$ that $\varepsilon(s) \leq (3+\delta')\rho$. This is indeed true by Lemma 6.2 (in the statement of the lemma, let $c = s$, $\gamma_1 = \delta\rho/2, \gamma_2 = (1+\delta/2)\rho$ and $\gamma_3 = (2+\delta'/2)\rho$).

First note that by this change the number of vertices that we do Dijkstra from/to in step 3(b) of the algorithm is now $O(|W|/\delta') = \tilde{O}(\sqrt{n}/\delta') = \tilde{O}(\sqrt{n}/\delta)$ (see runtime analysis of step 3(b) in Theorem 5.1). The runtime of the other steps are not changed, so the overall runtime of the algorithm is $\tilde{O}(m\sqrt{n}/\delta^2)$.

The main issue in the min-eccentricity algorithm that didn't allow us to get a $(3+\delta')$ approximation is that we could have potentially big weighted edges, and that didn't let us divide 5ρ-length paths into smaller parts. The analysis of this part is due to Lemma 6.3, which is modified as in Lemma 6.6.

□

Lemma 6.6 (Modification of Lemma 6.3)**.** *If for a vertex $s \in V$, $\varepsilon(s) \leq \rho$ and we haven't yet certified $\varepsilon(s) \leq (3+\delta')\rho$ then for each $i = 1, \ldots, k$, either $s \in S_{W_i}$ or $s \in T_{W_i}$.*

Proof. The proof is similar to that of Lemma 6.3, with a change at the end of the argument because of our finer division of paths. Fix i and suppose by way of contradiction that there are nodes $u, v \in W_i$ such that $s \in S_u \cap T_v$ and $\varepsilon(s) \leq \rho$. Similar to Lemma 6.3, we can assume that there are two vertices u', v' that we have done Dijkstra from such that $s \in S_{u'} \cap T_{v'}$ and $d(u', v') \leq \rho\delta'/2$.

Now we claim that $\varepsilon(v') \leq (2+\delta'/2)\rho$. Note that $d(v', s) \leq \rho$ and $d(s, v') \leq d(s, u') + d(u', v') \leq (1+\delta'/2)\rho$. Consider an arbitrary vertex $w \in V$. Either $d(s, w) \leq \rho$ or $d(w, s) \leq \rho$. If $d(s, w) \leq \rho$ then $d(v', w) \leq d(v', s) + d(s, w) \leq 2\rho$. If $d(w, s) \leq \rho$, then $d(w, v') \leq d(w, s) + d(s, v') \leq (2+\delta'/2)\rho$. A symmetric argument holds for u'. In this case, the algorithm would have already marked s in step 3b as it is in the intersection of $S_{u'}^\rho \cap T_{v'}^\rho$. □

Acknowledgements

The authors would like to thank the members of the MIT course 6.S078 open problem sessions, especially Thuy-Duong Vuong, Robin Hui, and Ali Vakilian. These sessions were organized by Erik Demaine, Ryan Williams, and Virginia Vassilevska Williams, and used the collaboration software Coauthor, created by Erik Demaine.

References

[1] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1681–1697, 2015.

[2] Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 377–391, 2016.

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

[4] Arturs Backurs, Liam Roditty, Gilad Segal, Virginia Vassilevska Williams, and Nicole Wein. Towards tight approximation bounds for graph diameter and eccentricities. In *Proceedings of the 50th Annual*

ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018, pages 267–280, 2018.

[5] B. Ben-Moshe, B. K. Bhattacharya, Q. Shi, and A. Tamir. Efficient algorithms for center problems in cactus networks. *Theoretical Computer Science*, 378(3):237 – 252, 2007.

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

[7] Michele Borassi, Pierluigi Crescenzi, Michel Habib, Walter A. Kosters, Andrea Marino, and Frank W. Takes. Fast diameter and radius bfs-based computation in (weakly connected) real-world graphs: With an application to the six degrees of separation games. *Theoretical Computer Science*, 2015. accepted.

[8] Massimo Cairo, Roberto Grossi, and Romeo Rizzi. New bounds for approximating extremal distances in undirected graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 363–376, 2016.

[9] T. M. Chan. All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time. *ACM Transactions on Algorithms*, 8(4):34, 2012.

[10] Shiri Chechik, Daniel H. Larkin, Liam Roditty, Grant Schoenebeck, Robert Endre Tarjan, and Virginia Vassilevska Williams. Better approximation algorithms for the graph diameter. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1041–1052, 2014.

[11] V. Chepoi, F. Dragan, and Y. Vaxès. Center and diameter problems in plane triangulations and quadrangulations. In *Proc. SODA*, pages 346–355, 2002.

[12] V. Chepoi and F. F. Dragan. A linear-time algorithm for finding a central vertex of a chordal graph. In *ESA*, pages 159–170, 1994.

[13] F. R. K. Chung. Diameters of graphs: Old problems and new results. *Congr. Numer.*, 60:295–317, 1987.

[14] D.G. Corneil, F.F. Dragan, M. Habib, and C. Paul. Diameter determination on restricted graph families. *Discr. Appl. Math.*, 113:143 – 166, 2001.

[15] L. Cowen and C. Wagner. Compact roundtrip routing for digraphs. In *SODA*, pages 885–886, 1999.

[16] D. Dvir and G. Handler. The absolute center of a network. *Networks*, 43:109 – 118, 2004.

[17] D. Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms and Applications*, 3(3):1–27, 1999.

[18] Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. Networks cannot compute their diameter in sublinear time. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1150–1162. SIAM, 2012.

[19] S.L. Hakimi. Optimum location of switching centers and absolute centers and medians of a graph. *Oper. Res.*, 12:450 – 459, 1964.

[20] Seth Pettie. A faster all-pairs shortest path algorithm for real-weighted sparse graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 85–97. Springer, 2002.

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

[22] Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, STOC '13, pages 515–524, New York, NY, USA, 2013. ACM.

[23] O. Weimann and R. Yuster. Approximating the diameter of planar graphs in near linear time. In *Proc. ICALP*, 2013.

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

[25] C. Wulff-Nilsen. Wiener index, diameter, and stretch factor of a weighted planar graph in subquadratic time. *Technical report, University of Copenhagen*, 2008.

[26] Raphael Yuster. Computing the diameter polynomially faster than apsp. *arXiv preprint arXiv:1011.6181*, 2010.

A Appendix

Lemma A.1. *Given a weighted graph G and a path P in G from v to u of length at most zr for some integers z and r, one can find in $O(|P|)$ time vertices $v_1, \ldots, v_{z'}$ such that $z' \leq 2z - 1$ and they divide P into subpaths of length at most r if there are no edges of weight more than r on the path. Equivalently, $|P_{v_i v_{i+1}}| \leq r$, for $i = 0, \ldots, z$, where $v_0 = v, v_{z'+1} = u$ and $P_{v_i v_{i+1}}$ is the part of the path P between v_i and v_{i+1}.*

Proof. Start from $v_0 = v$ and go through the path until the last vertex w such that $d(v, w) \leq r$ but $d(v, w') > r$ where w' is the node right after w on the path. Note that since there are no edges of weight more than r, such w exists. Let $v_1 = w$. Starting from v_1, we can do the same and find all vertices $v_2, \ldots, v_{z'}$. It is remained to prove that $z' < 2z$. By the definition of v_1, we know that $d(v_0, v_2) > r$. Similarly, we can argue that $d(v_i, v_{i+2}) > r$ for all $i = 0, \ldots, z' - 1$. So $d(v_0, v_{2i}) > ir$. Since $|P| \leq zr$, we have $z' \leq 2z - 1$. We went through the vertices of P once, so the running time is linear in terms of the length of the path. □

Lemma A.2. *There is an algorithm that given a strongly connected graph $H = (V, E)$, outputs in $O(|E|)$ time a subset $E' \subseteq E$ of size at most $2(|V| - 1)$ such that $H' = (V, E')$ is strongly connected.*

Proof. For any vertex v do a BFS to and from v and denote by E' the union of edges in the two computed BFS trees. $H' = (V, E')$ is strongly connected as for every ordered pair of vertices (a, b) we can go from a to b by following the path $a \to v \to b$. It is clear that since E' is the union of two trees, $|E'| \leq 2(|V| - 1)$. □