Computing the Exact Radius of Large Graphs

Stefan Funke
Universität Stuttgart, Germany

Claudius Proissl
Universität Stuttgart, Germany

Sabine Storandt
Universität Konstanz, Germany

Abstract

The radius of a graph is an important structural parameter which plays a key role in social network analysis and related applications. It measures the minimum shortest path distance that is required to reach all nodes in the graph from a single node. A node from which all other nodes are within a distance equal to the radius is called a center of the graph. In a graph with n nodes and m edges, the center and the radius can be determined in $\tilde{\mathcal{O}}(nm)$ by computing shortest path distances between all pairs of nodes. Fine-grained complexity results suggest that asymptotically faster algorithms are unlikely to exist. In this paper, we describe a novel randomized algorithm for exact radius computation in weighted digraphs with an expected running time in $\tilde{\mathcal{O}}(d^3 m)$ where d is the so-called combinatorial dimension. Our methodology is inspired by Clarkson's algorithm for LP-type problems. The value of d denotes the size of a basis, which is a smallest subset of nodes which enforce the same radius as the whole node set. While we show that there exist graphs with $d \in \Theta(n)$, our empirical analysis reveals that even large real-world graphs have small combinatorial dimension. This allows us to compute the radius in near-linear time on such instances. The significantly improved scalability can be clearly observed in our experimental evaluation on a diverse set of benchmark graphs.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Radius, Graph Center, LP-type, Combinatorial Dimension

Digital Object Identifier 10.4230/LIPIcs.SEA.2025.17

1 Introduction

A center of a graph is a node that minimizes the maximum shortest path distance to all other nodes, and the graph radius is the distance from the center to the node furthest from it. The radius is a fundamental graph parameter with numerous applications in theory and practice, including fine-grained complexity reductions [3], parameterized bounds for combinatorial problems [25], and network analysis [26, 13, 18]. The center itself is relevant, for example, in facility location problems [17, 20] or rumor spreading analysis [23, 15].

Formally, given a strongly connected directed graph $G(V, E)$, $|V| = n$, $|E| = m$ with non-negative edge weights $w : E \to \mathbb{R}^+$, the *eccentricity* of a node $v \in V$ is defined as $\epsilon(v) := \max_{u \in V} c(v, u)$ where $c(.,.)$ denotes the shortest path cost in G. The *radius* of the graph is defined as $r := \min_{v \in V} \epsilon(v)$ and a node v with $\epsilon(v) = r$ is called a *center* of G.

The eccentricity of a node v can be determined by a one-to-all shortest path computation from v in G. Therefore, the graph radius can be computed by first solving the all-pair-shortest-path problem (APSP) and then determining the node with the minimum eccentricity. APSP can be solved in $\mathcal{O}(n^3)$ by the Floyd-Warshall algorithm or in $O(n^2 \log n + nm)$ using n runs of Dijkstra's algorithm. The natural question arises as whether the radius can be computed significantly faster with an algorithm that is not based on APSP. But recent fine-grained complexity results leave little hope for such a breakthrough. It was proven in [1] that computing the radius is equivalent under subcubic reductions to APSP. Thus, a subcubic algorithm for radius computation would imply a subcubic algorithm for APSP, which is conjectured to not exist. In [3], a conditional lower bound for radius computation was shown which implies that an algorithm with a running time in $\mathcal{O}(n^a m^b)$ for any constants a, b with $a + b < 2$ is also unlikely to exist. Thus, the Dijkstra based algorithm appears to be optimal up to logarithmic factors even in sparse graphs.

These negative results apply to general input graphs. But, of course, improvements might still be possible for special graph classes. In [2], it was proven that there is a so called fixed parameter subquadratic algorithm with a running time in $2^{\mathcal{O}(t \log t)} n^{1+o(1)}$ on graphs with treewidth t. Clearly, the practical relevance of this algorithm is restricted to input graphs where the treewidth is a small constant and it also has not yet been implemented. In this paper, we devise a randomized algorithm for exact radius computation with an expected running time in $\mathcal{O}(d^3 m \log^2 n)$ where the parameter d denotes the so called combinatorial dimension for which we always have $d = O(n)$. Our algorithm is based on formulating radius computation as an LP-type problem and tailoring existing methods for solving such problems to our use case. In our experimental evaluation, we show that many graph instances indeed exhibit a small combinatorial dimension, and thus our newly developed algorithm computes the radius in these graphs with a near-linear running time.

1.1 Related Work

For some restricted graph classes, subcubic or even subquadratic algorithms for radius computation are known. This includes trees [12], maximal outerplanar graphs [11], interval graphs [17], and distance-hereditary graphs [10]. For unweighted graphs, APSP can be solved within the same time as integer matrix multiplication, which currently is possible in $\tilde{\mathcal{O}}(n^\omega)$ with $\omega \leq 2.371552$ [27].

To achieve even remotely practical running times on large graphs, one typically needs to resort to approximation algorithms. For undirected and unweighted graphs, an almost 3/2-approximation was shown to be possible in time $\tilde{\mathcal{O}}(m\sqrt{n} + n^2)$ [4]. The running time was improved to $\tilde{\mathcal{O}}(m\sqrt{n})$ in [21]. In [2], it was argued that a subquadratic $(3/2 - \delta)$-approximation algorithm for radius in undirected graphs for some $\delta > 0$ is unlikely to exist, as this would disprove prevailing hardness conjectures in $\mathcal{P}$. This result applies to weighted and unweighted graphs. In undirected graphs, a simple 2-approximation for the radius can be obtained in $\mathcal{O}(n \log n + m)$ time by picking any node and computing its eccentricity with a single run of Dijkstra's algorithm. In directed graphs, there is a $\mathcal{O}(m\sqrt{n} \log^2 n)$ Monte Carlo algorithm that provides an approximation factor of 2 [21]. In [13], a scalable algorithm for radius estimation in unweighted graphs was proposed and implemented in HADOOP. They strive to compute so called effective eccentricities, which are defined as the 90th-percentile of the shortest path distances from the node in question. Still, they resort to approximation algorithms to achieve fast running times on clusters with 90 machines.

For directed but unweighted graphs, an exact algorithm was described and empirically evaluated in [5]. It computes both the radius and the diameter of a graph simultaneously by maintaining and updating lower and upper bounds for these parameters until their tightness is proven. Updates happen via one-to-all shortest path computations using BFS. While the theoretical running time of this algorithm is in $\mathcal{O}(nm)$, the experiments reveal that oftentimes a very small number of BFS runs suffices to obtain the final result. Borassi et al in [6] develop an axiomatic framework under which many algorithms (among others also [5]) were proven to work efficiently if certain properties (power law degree distribution and some additional conditions) are met, which was shown to be true for many power law random graphs with high probability but also empirically for many unweighted real-world graphs.

1.2 Contribution

In this paper, we provide a new perspective on radius computation in graphs by showing that it can be interpreted as an LP-type problem. We prove several structural properties that are necessary for this interpretation to work.

Multiple algorithms have been proposed in the literature to solve LP-type problems [22, 8, 16]. Their running time depends on the input size as well as the so called combinatorial dimension d. We adapt Clarkson's algorithm [8] to our use case and provide a detailed analysis in which we show a running time for radius computation in $\tilde{\mathcal{O}}(d^3 m)$. Notably, our algorithm works for weighted and directed graphs. However, while in many geometrical LP-type problems the combinatorial dimension is known a priori, the same is not true for the graph radius problem. We thus have to extend our algorithm to deal with unknown d. We show that this is possible without increasing the asymptotic running time of the algorithm. While we provide an example graph instance with $d \in \Theta(n)$, our experimental evaluation shows that the combinatorial dimension of real-world graphs is very small. In fact, among the tested graphs (including large social networks, web graphs, and road networks), we never encountered a d value larger than 16. Accordingly, the practical running time of our algorithm is almost linear in the size of the graph. This allows us to compute the exact radius for graphs with billions of edges, which were deemed intractable before. For the *com-friendster* network[1] with 656 million nodes and 1.8 billion edges, solving APSP would take more than 250 years on our hardware, while our novel algorithm returns the solution in less than 4 hours.

2 Graph Radius Computation as LP-Type Problem

LP-type problems were first defined by Sharir and Welzl [24]. They are a generalization of linear programs that retain many of their combinatorial properties but are not restricted to linear constraints. For an LP-type problem, there needs to be a finite set of constraints S and a measure function f that assigns a value to each subset of S. The function f needs to satisfy the following two properties:

- *Monotonicity.* For $A \subseteq B \subseteq S : f(A) \leq f(B)$. This implies that adding more constraints does not decrease the function value.
- *Locality.* Let $A \subseteq B \subseteq S$ and $f(A) = f(B)$: If for $s \in S - B$ we have $f(B \cup \{s\}) > f(B)$ then $f(A \cup \{s\}) > f(A)$.

A *basis* of an LP-type problem is a set $B \subseteq S$ such that $f(B) = f(S)$ and $\forall B' \subset B$ we have $f(B') < f(B)$. So a basis is a (set-)minimal subset of constraints that defines the same function value as the entire set, and removing any constraint from it changes that value. The *combinatorial dimension* d of an LP-type problem is defined as the maximum cardinality of a basis. Algorithms for solving LP-type problems aim at quickly identifying such a basis. These algorithms work particularly well in case the basis is much smaller than the size of S, which means that S contains many redundant constraints. For example, for $d \in O(1)$, a linear program with n constraints can be solved in expected $\mathcal{O}(n)$ time [24].

We now show that the computation of the radius of a given digraph $G(V, E)$ with edge weights is an LP-type problem with a minimization measure function. For ease of exposition, we assume from now on that the shortest path costs $c(v, w)$ between node pairs $v, w \in V$ are all pairwise distinct. This can be ensured by adding a small random number to each edge

[1] `https://snap.stanford.edu/data/com-Friendster.html`

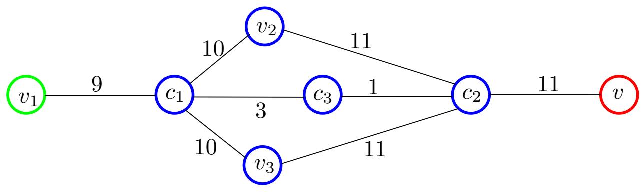

Figure 1 Consider the sets $A = \{v_2, v_3, c_1, c_2, c_3\}$ and $B = A \cup \{v_1\}$. We have $r_A = r_B = 10$ with center c_1. Considering v as well we see that $c(c_1, v) = 15 > 10$ and get $r_{A\cup\{v\}} = 11$ with center c_2, but $r_{B\cup\{v\}} = 12$ with center c_3.

weight. Under this assumption the center of the graph is uniquely defined. Our measure function is the radius $r := \min_{v\in V} \epsilon(v)$ where $\epsilon(v) := \max_{u\in V} c(v, u)$ denotes the eccentricity of the node v. Thus, each node poses a constraint for the radius value. For a node subset $F \subseteq V$ we define the eccentricity as well as the radius with respect to F as follows:

$$\epsilon_F(v) = \max_{u\in F} c(v, u)$$

$$r_F = \min_{v\in V} \epsilon_F(v)$$

This means that we only care about distances to nodes in F and not to all nodes in V when defining the radius of a node subset. We now show that r_F fulfills the requirements of a measure function for an LP-type problem.

▶ **Lemma 1.** *r_F is both monotone and local.*

Proof. Monotonicity holds as for $A \subseteq B \subseteq S$ we clearly have $\epsilon_A(v) \leq \epsilon_B(v)$ and thus $r_A \leq r_B$. For locality, let $A \subseteq B \subseteq V$ be node sets and $v \in V \setminus B$. We claim that if $r_A = r_B$ and $r_{B\cup\{v\}} > r_B$ we also have $r_{A\cup\{v\}} > r_A$. From $r_A = r_B$, it follows that r_A and r_B are realized by the same unique (center) vertex v' in V. If $r_{B\cup\{v\}} > r_B$, this means that $c(v', v) > r_B = r_A$. The lemma follows. ◀

The above lemma states that if adding a vertex v to B increases the radius, adding the same vertex to $A \subset B$ also increases the radius. Note that the lemma does not claim that $r_{A\cup\{v\}} = r_{B\cup\{v\}}$. An example where this is indeed not the case is provided in Figure 1.

The basis B_W for a set of vertices $W \subseteq V$ is an inclusion-minimal subset $B \subseteq W$ with $r_B = r_W$. The combinatorial dimension of our problem is defined as $d := \max_{W\subseteq V} |B_W|$. In the next section, we present an algorithm that uses this formulation of radius computation as an LP-type problem to solve the problem much faster than the naive algorithm in case the combinatorial dimension d is small.

3 Adapting Clarkson's Algorithm for Radius Computation

Our algorithm is heavily inspired by Clarkson's second algorithm for LP-type problems [8], which is an iterative randomized algorithm. It starts with a small random subset of constraints R (where the notion of small depends on the combinatorial dimension d) and computes an optimal solution for R. Then, it checks whether any constraints are violated for this solution. If that is not the case, the current solution is obviously optimal. Otherwise, it follows that one or more violators need to be included in R to get a correct solution. To make this more likely to happen in the next round, weights/multipliers are assigned to all constraints. Initially those are uniformly set to one. But whenever a constraint is identified to be a violator, its weight is doubled. In the next round, a new sample R is chosen. The probability of a constraint to be sampled is proportional to its weight. The algorithm repeats the sampling and reweighing process until the optimal solution that satisfies all constraints is identified. This happens in the iteration in which R is a superset of a basis B for the problem.

We now tailor this algorithm to our problem of radius computation. There are two main ingredients needed for the Clarkson algorithm to work: An efficient method that computes the optimal solution for a given sample R and an algorithm that identifies the set of violators or certifies the global optimality of the solution. The radius r_R and the center c_R for a given node subset R can be obtained as follows: We allocate an array of size n which is supposed to store for each node $v \in V$ the maximum distance to the nodes in R. Initially, all array entries are set to zero. By running Dijkstra's algorithm once from each node $w \in R$ on the reverse edges and updating the maximum distance array of a node v whenever a computed shortest path distance $c(v, w)$ exceeds its current value, we can compute the eccentricities ϵ_R in $\mathcal{O}(|R|(n \log n + m))$ time and deduce the radius and the center from these values in linear time. To compute the set of violators, we need to find all nodes that further away from c_R than r_R. This can easily be checked with a single run of Dijkstra's algorithm from c_R. If the shortest path distance from c_R to all nodes $v \in V$ is within r_R, we know that $c_R = c_V$ and $R \supset B$. Otherwise, we can gather all nodes with a shortest path distance greater than r_R from c_R and apply the above described reweighing procedure to these violators.

For now, we assume the combinatorial dimension d to be known. With the careful choice of the sample size $|R|$ being set to $6d^2$ we will show that after an expected number of $O(d \log n)$ iterations, the basis B must be part of the sample R. As we need $|R| + 1$ runs of Dijkstra's algorithm per iteration as discussed above, the expected total number of Dijkstra computation is in $O(|R| d \log n)$. Thus, the total running time for graph radius computation is in $\mathcal{O}(d^3 \log n(n \log n + m))$. Algorithm 1 shows the respective pseudo-code. In the algorithm, we maintain for each node v of the node set V a multiplicity (initially $\mu_v = 1$) to be able to increase the probability of a node being picked in a subsequent iteration by doubling its multiplicity. The weight $\mu(X)$ of the violator set X counts each violator $v \in V$ according to its multiplicity.

Algorithm 1 Clarkson's Algorithm for Graph Radius.

```
   Input: Graph $G(V, E)$
   Output: Center $c_V$ and radius $r_V$
 1 if $|V| \leq 6d^2$ then
 2     Compute $c_V, r_V$ naively
 3     return $(c_V, r_V)$
 4 $s \leftarrow 6d^2$
 5 repeat
 6     Choose a random subset $R \subseteq V$ of size $s$
 7     Compute the center $c_R$ and radius $r_R$
 8     $X \leftarrow \{v \in V \mid d(c_R, v) > r_R\}$
 9     if $\mu(X) \leq \frac{1}{3d}\mu(V)$ then
10         foreach $v \in X$ do
11             $\mu_v \leftarrow 2\mu_v$
12 until $X = \emptyset$
13 return $(c_R, r_R)$
```

Quite obviously, the algorithm produces an optimal solution if it terminates as it has computed the optimum center c_R and radius r_R for a subset $R \subset V$ and all other nodes are within distance r_R from c_R.

The goal is now to analyze the expected running time of the algorithm in more detail. The following analysis follows closely the one in [8]. We call an iteration *successful* if the weight of the violator set X is not too high, that is, $\mu(X) \leq \frac{1}{3d}n$. To upper bound the number iterations needed until we have such a successful iteration, we first investigate the expected weight of the violator set.

▶ **Lemma 2.** $E(\mu(X)) \leq \frac{1}{6d}\mu(V)$.

Proof. In this proof we consider a multiset $\widetilde{V}$ which has each $v \in V$ appear μ_v times in $\widetilde{V}$, $\widetilde{n} = |\widetilde{V}|$. Look at the following bipartite graph (A, B), where there is a node $a \in A$ for each s-sized subset R_a of $\widetilde{V}$ and a node $b \in B$ for each $(s+1)$-sized subset R_b in $\widetilde{V}$. There is an edge (a, b) if and only if, $R_b = R_a \cup \{v\}$ and $d(c_{R_a}, v) > r_{R_a}$. Note that the degree of a node $a \in A$ corresponds to weight of the violator set for sample set R_a and we are interested in the average degree of a node in A as this is the expected weight of the violator set. The degree of a node $b \in B$ is always at most d (the combinatorial dimension of the problem), though, as for at most d nodes, it is true that ignoring that node allows for a smaller radius. Therefore, the expected weight of the violator set X can be bounded as follows:

$$E(\mu(X)) \leq \frac{\binom{\widetilde{n}}{s+1} \cdot d}{\binom{\widetilde{n}}{s}} = d\frac{\widetilde{n} - s}{s+1}$$

Choosing $s = 6d^2$ yields the claim. ◀

This lemma together with Markov's inequality imply that the expected number of iterations until we have a successful iteration is at most 2. Next, we investigate the relationship between the number of successful iterations and the size of a basis B.

▶ **Lemma 3.** *Let k be some positive integer. After kd successful iterations we have*

$$2^k \leq \mu(B) < ne^{k/3}$$

for a basis B of V.

Proof. Every successful iteration increases the weight of V by at most $\frac{1}{3d}\mu(V)$, therefore we get

$$\mu(B) \leq \mu(V) \leq n\left(1 + \frac{1}{3d}\right)^{kd} < ne^{k/3}$$

For the lower bound we observe that in each round, at least one of the violators is a constraint in B. That means there is a constraint that has been doubled at least k times and hence $\mu(B) \geq 2^k$. ◀

Using this lemma we can now bound the number of successful iterations until we have $B \subseteq R$ and hence have computed the correct solution.

▶ **Theorem 4.** *The graph radius r_V and the graph center c_V can be computed with expected at most $36d^3 \ln n$ calls to Dikstra's algorithm.*

Proof. For kd iterations with $k = 3\ln n$ we get $2^k = n^{3/\log e} > n^2 = ne^{k/3}$, that is, the lower bound in the Lemma 3 overtakes the upper bound. Thus, the number of iterations is at most $2 \cdot 3d\ln n = 6d\ln n$ iterations in expectation. With a sample size of $6d^2$, each iteration makes roughly $6d^2$ calls to Dijkstra's algorithm. Therefore, the expected total number of calls is upper bounded by $36d^3 \ln n$. ◀

Accordingly, we get a running time in $\mathcal{O}(d^3 n \log^2 n + d^3 m \log n)$ or ignoring log terms in $\tilde{\mathcal{O}}(d^3 m)$. For small d, this is a huge improvement over the APSP baseline which runs in $\tilde{\mathcal{O}}(mn)$. But so far we assumed that the combinatorial dimension d is known and used it as an explicit parameter in our algorithm. Unfortunately, we do not have access to this value in practice. In the next section, we discuss how to overcome this issue without increasing the asymptotic running time of the algorithm.

4 Combinatorial Dimension of the Graph Center Problem

Clarkson's algorithm requires knowledge of the combinatorial dimension d of the problem to choose a reasonable size for the sample R. For many LP-type problems, the dimension can be bounded a priori. This is true, for example, for the minimum enclosing circle problem where given a set of points in the plane, the goal is to find the smallest circle that contains all points. The combinatorial dimension is known to be at most 3 for this problem [16].

Unfortunately, we will prove next that no such bound exists for the graph radius problem by constructing a graph with combinatorial dimension $d \in \Theta(n)$.

▶ **Theorem 5.** *There exist graphs for which the combinatorial dimension of the radius computation problem is $d = n/2$.*

Proof. We assume $|V| = n$ to be even. The first $n/2$ graph nodes, $v_1, \dots, v_{n/2}$ form a clique in which all edge costs are equal to 1. Additionally, each clique node $v_i, i = 1, \dots, n/2$ has an edge of cost 9 to each node $v_j, j = n/2 + 1, \dots, n$ except for $j = i + n/2$. Figure 2 illustrates the graph structure for $n = 4$. The eccentricity of the non-clique nodes is 19, equal to their shortest path distance to any other non-clique node. The eccentricity of the clique nodes is 10 induced by the shortest path from v_i to $v_{i+n/2}$. Thus, we have $r_V = 10$. We now claim that the non-clique nodes $v_j, j = n/2 + 1 \dots, n$ form a basis B for V. Clearly, $r_B = r_V$ holds. It remains to show that B is inclusion-minimal, that is, for all $v \in B$ we have that $B' := B \setminus \{v\}$ is not a basis as $r_{B'} < r_B = r_V$. To see that, let v be any node $v_j, j = n/2 + 1 \dots, n$. If we remove v_j from B, we know that the eccentricity of node v_i with $i = j - n/2$, which is connected to every node in $B \setminus \{v_j\}$ with a direct edge of cost 9, drops to 9. Thus, we have $r_{B'} = 9 < 10 = r_B$. We conclude that B is indeed inclusion-minimal and thus a proper basis for V. Therefore, the combinatorial dimension of the graph is $d = |B| = n/2$. ◀

Furthermore, we observe that the size of a basis is not monotone with respect to the number of constraints. So it might happen that for $F \subset V$ the size of a basis for F exceeds the size of a basis for V, see Figure 3 for an example. This is why we defined the combinatorial dimension of the problem as the maximum basis size over all possible subsets of V.

4.1 Dealing with unknown Combinatorial Dimension

These observations seem to be detrimental to the applicability of Clarkson's algorithm to our problem. However, we use the following idea to overcome these issues: We start with the assumption that d is a small constant and increase its value if we could not find a solution

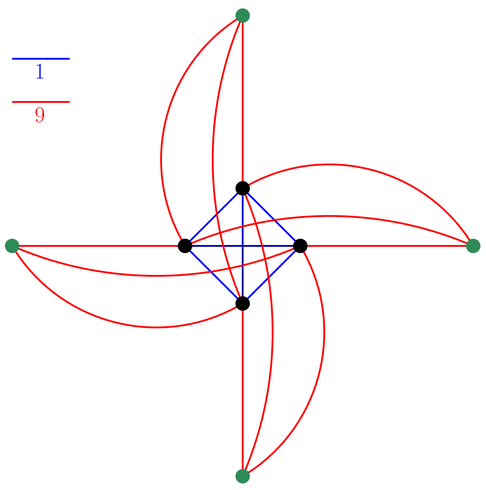

Figure 2 Example graph with $d = n/2$. The basis B is formed by the green nodes.

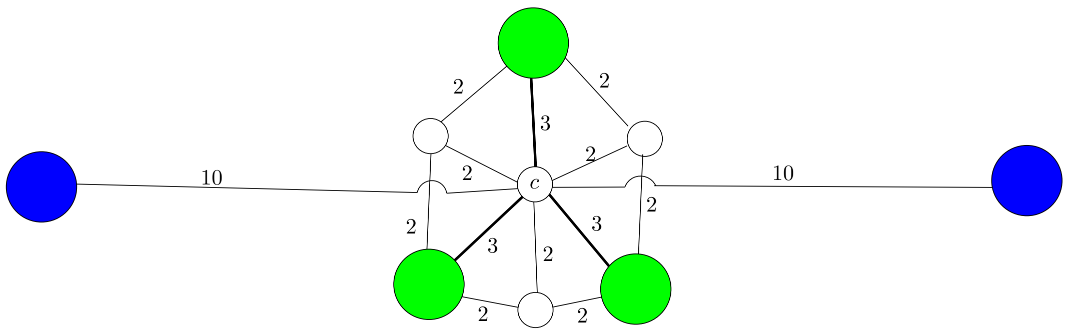

Figure 3 The radius for the green nodes is 3 and all of them are part of the basis (none can be dropped without changing the radius). For the green *and* the blue nodes we have a radius of 10 (with the same center c, but only the two blue nodes form the basis).

within a reasonable number of iterations. The increase should not be too small since then the running time for the unsuccessful attempts dominates the overall running time. At the same time, the increase should also not be too large, as overestimating d leads to large sample sizes and thus long running times per iteration.

More concretely we start with an estimated value of e.g. $d = 2$ in our implementation and run the algorithm for at most $24d \ln n$ iterations. If a solution is returned, we are done. Otherwise, we might have simply been unlucky or our estimation of d is too small for the algorithm to succeed. Thus, we update $d := d \cdot \sqrt[3]{2}$ and repeat the algorithm with this new estimation. We refer to this algorithm as round-based algorithm.

We will show that the expected number of calls to Dijkstra's algorithm remains almost the same as when the algorithm is executed with known d. The argument is similar, e.g., to the one in Chan's output-sensitive algorithm for convex hull computation in the plane where the number of extreme points is not known, but is approached by *squaring*, see [7]. In our case, the analysis is slightly more involved as our base algorithm is randomized.

In the following, let d^* be the true combinatorial dimension of the problem instance. We now prove that as soon as our estimated value d exceeds the true combinatorial dimension, the probability that the algorithm does not return a solution after the specified number of iterations is small.

▶ **Lemma 6.** *For $d \geq d^*$, the probability that the algorithm does not return a solution within $24d \ln n$ iterations is at most $1/4$.*

Proof. If $d > d^*$, the expected number of iterations is at most $6d \ln n$ based on our analysis provided in the proof of Theorem 4 and the fact that the expected number of iterations is not negatively affected if d is larger than the true combinatorial dimension. Using Markov's inequality, the probability of exceeding this expected number by more than a factor of 4 is upper bounded by $1/4$. The lemma follows. ◀

With the help of this lemma, we now prove our main theorem.

▶ **Theorem 7.** *The round-based algorithm computes c_V, r_V with an expected number of $O((d^*)^3 \log n)$ calls to Dijkstra's algorithm without knowledge of d^*.*

Proof. We first bound the cost for the attempts where $d < d^*$. We let each of these attempts run for at most $24d \ln n$ iterations, issuing at most $24d^3 \ln n$ calls to Dijkstras algorithm. Since we multiply d by $\sqrt[3]{2}$ after each unsuccessful attempt, the upper bound for the cost of the previous round is always half of that of the current round. Thus, the sum of upper bounds of all rounds prior to using $d \geq d^*$ for the first time is at most the upper bound for the cost of that round. As for this round, we have $d \leq d^* \sqrt[3]{2}$, the cost for all prior rounds is upper bounded by $24((d^* \sqrt[3]{2})^3 \ln n = 48(d^*)^3 \ln n$.

For the attempts with $d \geq d^*$ we know by virtue of Lemma 6 that the probability that a single round fails is at most $1/4$. Hence the probability that we have to execute round number i with $d \geq d^*$ is at most $1/4^i$. Accordingly, the expected number of Dijkstra calls issued in the ith round with $d > d^*$ can be upper bounded as follows:

$$\frac{1}{4^i} 24(d^*(\sqrt[3]{2})^i)^3 \ln n = \frac{2^i (d^*)^3 24 \ln n}{4^i} = \frac{24(d^*)^3 \ln n}{2^i}$$

Therefore the expected total number of iterations is bounded by

$$48(d^*)^3 \ln n + \sum_{i=0}^{\infty} \frac{24(d^*)^3 \ln n}{2^i} = 96(d^*)^3 \ln n.$$

Thus, the expected number of calls to Dijkstra's algorithm is in $\mathcal{O}((d^*)^3 \log n)$. ◀

Accordingly, the number of calls stays asymptotically the same as for the case in which d^* is known. Factoring in the cost of $O(n \log n + m)$ per run of Dijkstra's algorithm, we get the following total running time for radius and center computation in graphs.

▶ **Corollary 8.** *For a strongly connected weighted digraph with combinatorial dimension d, we can compute the graph radius in expected $O((n \log n + m)d^3 \log n) = O(d^3 n \log^2 n + d^3 m \log n)$ without explicit knowledge of d.*

5 Experimental Results

Our algorithm is implemented in C++ (g++ 11.4.0) and executed on a Ryzen 7950x 16-core machine with 192GB of RAM running Ubuntu Linux 22.04.

We picked several benchmark graphs from the Stanford Large Network Dataset Collection (SNAP), [14]. These were all unweighted graphs ranging from few thousand nodes up to more than half a billion nodes and almost 2 billion edges. Additionally we used as weighted graphs the road networks of Germany and Europe extracted from the OpenStreetMap project data [19]. See Table 1, columns 1–5 for the characteristics of our data sets. For each graph we ran our algorithm on the largest strongly connected component.

The considered SNAP data includes collaboration/social network/interaction data (*caAstroPh, email-enron, soc-livejournal1, com-friendster*), two web graphs (*web-BerkStan, web-Google*), a co-purchase-graph (*amazon0505*) and a peer-to-peer graph (*p2p-gnutella*, which we interpreted as an undirected graph since the largest strongly connected component was too small). We picked within each category the largest instance(s).

5.1 Implementation Details

In the following, we describe some of our design chocies and implementation details that impact the practical running time of our algorithm.

5.1.1 Multithreading

Our implementation uses straightforward multithreading via OpenMP [9]. In particular, the computation of the optimum solution for the subset R is distributed among the available threads. This naturally suggests choosing the initial size s of the sampled subset to match the number of available CPU threads (in our case 32, which corresponds to $d \approx 2.3$). Then, after an unsuccessful attempt (after $24d \ln n$ iterations) we simply increase s by a factor of $2^{\frac{2}{3}}$, though we never had to in our practical experiments.

5.1.2 Multiset representation

Crucial operation of the algorithm is the random sampling from a weighted vertex set. If the nodes are $v_0, \ldots, v_{n-1}$ with multiplicities $m_0, \ldots, m_{n-1}$, we first compute prefix sums $P_i = \sum_{j=0}^{i-1} m_i$ for $i = 0, \ldots, n$. Then we repeatedly choose a random number $x \in [0, P_n]$ and add node v_i to the subset R if $P_{i-1} \leq x < P_i$.

5.1.3 Basis Computation

Our algorithm terminates once the optimum solution for R yields a center with radius which also contains all other nodes of the graph. R could also be viewed as a certificate for the optimality of the computed center and radius. R is not necessarily a basis, though, as it

Table 1 Considered benchmark data sets. For all graphs, $|V|$ and $|E|$ denote the number of nodes/edges for the largest (strongly) connected component on which we run our algorithm. *time* denotes the running time of our algorithm for a sample size of 32, *radius* the resulting graph radius and *size* the size of a minimal basis for the final solution. For comparison, the last column is the estimated running time of the naive approach (single core).

name	dir ected	wei ght	|V|	|E|	time	radius	basis size	naive estimate
ca-AstroPh	no	no	17,903	394,003	194ms	8	5	34s
email-enron	yes	no	33,696	361,622	241ms	7	2	100s
p2p-gnutella31	no	no	62,561	147,878	649ms	7	5	7m
amazon0505	yes	no	390,304	3,255,816	4,582ms	20	2	>10h
web-BerkStan	yes	no	334,857	4,523,232	3,096ms	107	2	>5h
web-Google	yes	no	434,818	3,419,124	4,902ms	23	2	>10h
roadnet-CA	no	no	2.0M	5.5M	75s	494	8	>177h
soc-livejournal1	yes	no	3.8M	65.8M	164s	10	2	>100d
com-friendster	no	no	656M	1,806M	11,553s	19	2	>4000y
Germany	yes	yes	24.9M	50.3M	812s	1,829,848	4	>4y
Europe	yes	yes	65.8M	122.7M	1,957s	9,889,376	3	>30y

might not be set minimal. We guarantee set minimality in a trivial fashion by computing the optimum iteratively omitting one $v \in R$. If a dropped node does not affect the optimum solution, it can be dropped for the following iterations. Finally, the remaining nodes form a basis of the optimum solution. The cost of this basis computation is essentially computing $|R|$ optimum solutions for subsets of size at most $|R| - 1$.

5.2 Radius computation times

In the right half of Table 1 we see the running times of our randomized algorithm when started with a sample size of 32 (as our CPU has that many threads) – this corresponds to an estimated combinatorial dimension of ≈ 2.3. It is noteworthy that in no case it was necessary to increase this estimation, even though the final basis size was often larger (up to 8) – all computations completed within less than 70 iterations. For example, for the social network graph *soc-livejournal1*, our algorithm finished after only 21 iterations (2 of which were 'unsuccessful' in that the violator set was too large), taking 164 seconds. The computed graph center had distance at most 10 to all other nodes of the graph and was determined by only 2 nodes. A single Dijkstra on that graph takes around 2.3 seconds, so a naive computation of the graph center via $|V|$ Dijkstra calls would take more than 100 days on a *single-core machine*. On our 16-core CPU it would still take several days.

For the largest graph, the social network graph *com-friendster* with more than half a billion nodes, naive computation would take thousands of years single-core, our algorithm completes within less than 4 hours.

5.3 Empirically Estimating the Combinatorial Dimension

In the previous experiment we only reported on the basis size of the final solution. For the algorithm to succeed quickly, though, it is also important that for subsets of V the combinatorial dimension is small. And we have seen that the combinatorial dimension is not necessarily monotone when considering subsets $F \subseteq V$, so we estimated the combinatorial

Table 2 Experimentally determined basis sizes (combinatorial dimension) for different vertex subsets; averaged over 100 trials.

	Basis size							
	$\lvert R\rvert = 8$		$\lvert R\rvert = 32$		$\lvert R\rvert = 256$		$\lvert R\rvert = 2048$	
Network	max	Avg	max	Avg	max	Avg	max	Avg
ca-AstroPh	4	2.4	5	2.8	5	2.4	5	2.3
p2p-gnutella31	6	3.9	11	5.3	11	6.5	16	6.3
web-Google	5	2.6	6	3.0	6	2.5	4	2.3
Germany	3	2.7	5	3.4	5	3.8	5	4.0

Table 3 Radii, basis sizes and number of iteration unperturbed vs perturbed.

	unperturbed			perturbed		
Network	radius	basis size	iterations	radius	basis size	iterations
ca-Astro-Ph	8	5	17	8.00...	3	16
p2p-gnutella31	7	5	21	7.00...	11	39
web-Google	23	2	16	23.00...	3	17
roadnet-CA	494	8	62	494.00...	8	74

dimension not only for the complete node set, but also for random node sets of varying sizes. In Table 2 we see the results for the *ca-AstroPh*, *p2p-gnutella31*, *web-Google*, and the *Germany* graph. It is quite striking that within 100 trials for each subset size we never encountered a basis size larger than 16, in particular for subset sizes of $s = 32$ which was the one used for all results of Table 1 (it was never necessary to double s), explaining the extremely good performance of our algorithm. But also for smaller and larger subset sizes we did not encounter large bases. We are still looking for real-world graphs with large combinatorial dimension, even though it is easy to synthetically create graphs with arbitrarily large combinatorial dimension as we have seen in Theorem 5.

5.4 Distinctness of Shortest Path Distances

For our theoretical analysis we assumed that shortest path costs $c(v, w)$ are all pairwise distinct for all $v, w \in V$. Most of the graphs in the previous section do not fulfill this assumption, in particular for all unweighted graphs, there are obviously pairs of nodes with equal shortest path costs. Yet, as already mentioned, if Algorithm 1 terminates, it produces an optimal solution, even if shortest path cost distinctness is not fulfilled. All experiments in the previous Sections were in fact run on the unaltered problem instances. We did not expect the behavior to change a lot in practice when introducing a perturbation of the edge weights, yet in Table 3 we show the results when adding to each edge cost a very small random number. Apart from *p2pgnutella31*, the other instances did only differ slightly in terms of basis size and number of iterations required. Due to the perturbation we had to switch to floating-point arithmetic, though, which incurred several problems like unreliable violation tests etc. We strongly recommend to perform all computations with an exact number type.

6 Conclusions

We presented an exact algorithm for computing the center of a graph (directed or undirected, weighted or unweighted) which runs in near-linear time if the *combinatorial dimension* of the graph is a constant. Our algorithm is essentially an adaptation of an algorithm of

Clarkson [8] originally invented for low-dimensional linear programming. In our context, the combinatorial dimension describes the cardinality of a minimal subset of vertices that have the same center and radius than the original graph. While we could show that there exist graphs with combinatorial dimension $\Theta(n)$, many real-world graphs that model, e.g., social networks, communication patterns, collaborations, road networks etc. empirically exhibit very low combinatorial dimension. Due to that somewhat surprising finding, we can compute the exact graph center even for graphs with billions of edges in few hours where the standard approach would take several hundred years.

In future work, it would be interesting to explore whether efficiency of the algorithm can also be proven for non-distinct shortest paths. Furthermore, while random subset selection is an integral part of the algorithm to prove the respective expected running time bounds, greedy selection strategies were shown to perform well in practice [5] and in theory [6] in the unweighted case. Analyzing those strategies directly or designing new deterministic strategies with performance guarantees in the weighted case could reveal new insights into the problem structure.

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*, pages 1681–1697. SIAM, 2014.

2 Amir Abboud, Virginia Vassilevska Williams, and Joshua 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*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.CH28.

3 Udit Agarwal and Vijaya Ramachandran. Fine-grained complexity for sparse graphs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 239–252, 2018. doi:10.1145/3188745.3188888.

4 Piotr Berman and Shiva Prasad Kasiviswanathan. Faster approximation of distances in graphs. In *Workshop on Algorithms and Data Structures*, pages 541–552. Springer, 2007. doi:10.1007/978-3-540-73951-7_47.

5 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*, 586:59–80, 2015. doi:10.1016/J.TCS.2015.02.033.

6 Michele Borassi, Pierluigi Crescenzi, and Luca Trevisan. An axiomatic and an average-case analysis of algorithms and heuristics for metric properties of graphs. In *SODA*, pages 920–939. SIAM, 2017.

7 Timothy M Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16(4):361–368, 1996. doi:10.1007/BF02712873.

8 Kenneth L. Clarkson. Las vegas algorithms for linear and integer programming when the dimension is small. *J. ACM*, 42(2):488–499, 1995. doi:10.1145/201019.201036.

9 Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

10 Feodor F Dragan. Dominating cliques in distance-hereditary graphs. In *Scandinavian Workshop on Algorithm Theory*, pages 370–381. Springer, 1994. doi:10.1007/3-540-58218-5_34.

11 Arthur M Farley and Andrzej Proskurowski. Computation of the center and diameter of outerplanar graphs. *Discrete Applied Mathematics*, 2(3):185–191, 1980. doi:10.1016/0166-218X(80)90039-6.

12 Gabriel Y Handler. Minimax location of a facility in an undirected tree graph. *Transportation Science*, 7(3):287–293, 1973.

13 U Kang, Charalampos E Tsourakakis, Ana Paula Appel, Christos Faloutsos, and Jure Leskovec. Hadi: Mining radii of large graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(2):1–24, 2011. doi:10.1145/1921632.1921634.

14 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

15 Wuqiong Luo, Wee Peng Tay, Mei Leng, and Maria Katrina Guevara. On the universality of the jordan center for estimating the rumor source in a social network. In *2015 IEEE International Conference on Digital Signal Processing (DSP)*, pages 760–764. IEEE, 2015. doi:10.1109/ICDSP.2015.7251978.

16 Jiří Matoušek, Micha Sharir, and Emo Welzl. A subexponential bound for linear programming. In *Proc. 8th Annual Symposium on Computational Geometry*, pages 1–8, 1992.

17 Stephan Olariu. A simple linear-time algorithm for computing the center of an interval graph. *International Journal of Computer Mathematics*, 34(3-4):121–128, 1990. doi:10.1080/00207169008803870.

18 Marcia Oliveira and Joao Gama. An overview of social network analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):99–115, 2012. doi:10.1002/WIDM.1048.

19 The OpenStreetMap project. *OpenStreetMap*, 2024. URL: https://www.openstreetmap.org/.

20 Frédéric Protin. A new algorithm for graph center computation and graph partitioning according to the distance to the center. *arXiv preprint arXiv:1910.02248*, 2019. arXiv:1910.02248.

21 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proc. 45th annual ACM Symposium on Theory of Computing*, pages 515–524, 2013. doi:10.1145/2488608.2488673.

22 Raimund Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry*, 6:423–434, 1991. doi:10.1007/BF02574699.

23 Devavrat Shah and Tauhid Zaman. Rumors in a network: Who's the culprit? *IEEE Transactions on information theory*, 57(8):5163–5181, 2011. doi:10.1109/TIT.2011.2158885.

24 Micha Sharir and Emo Welzl. A combinatorial bound for linear programming and related problems. In *STACS 92: 9th Annual Symposium on Theoretical Aspects of Computer Science Cachan, France, February 13–15, 1992 Proceedings 9*, pages 567–579. Springer, 1992.

25 Sabine Storandt. Bounds and algorithms for geodetic hulls. In *Conference on Algorithms and Discrete Applied Mathematics*, pages 181–194. Springer, 2022. doi:10.1007/978-3-030-95018-7_15.

26 Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442, 1998.

27 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3792–3835. SIAM, 2024. doi:10.1137/1.9781611977912.134.