Multivariate Analysis of Orthogonal Range Searching and Graph Distances

Karl Bringmann
Max-Planck-Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
kbringma@mpi-inf.mpg.de

Thore Husfeldt[1]
BARC, IT University of Copenhagen, Denmark, and Lund University, Sweden
thore@itu.dk
https://orcid.org/0000-0001-9078-4512

Måns Magnusson
Department of Computer Science, Lund University, Sweden
mans.magnusson.888@student.lu.se

Abstract

We show that the eccentricities, diameter, radius, and Wiener index of an undirected n-vertex graph with nonnegative edge lengths can be computed in time $O(n \cdot \binom{k+\lceil \log n \rceil}{k} \cdot 2^k k^2 \log n)$, where k is the treewidth of the graph. For every $\epsilon > 0$, this bound is $n^{1+\epsilon} \exp O(k)$, which matches a hardness result of Abboud, Vassilevska Williams, and Wang (SODA 2015) and closes an open problem in the multivariate analysis of polynomial-time computation. To this end, we show that the analysis of an algorithm of Cabello and Knauer (Comp. Geom., 2009) in the regime of non-constant treewidth can be improved by revisiting the analysis of orthogonal range searching, improving bounds of the form $\log^d n$ to $\binom{d+\lceil \log n \rceil}{d}$, as originally observed by Monier (J. Alg. 1980).

We also investigate the parameterization by vertex cover number.

2012 ACM Subject Classification Theory of computation → Shortest paths, Theory of computation → Parameterized complexity and exact algorithms, Theory of computation → Computational geometry, Mathematics of computing → Paths and connectivity problems

Keywords and phrases Diameter, radius, Wiener index, orthogonal range searching, treewidth, vertex cover number

Digital Object Identifier 10.4230/LIPIcs.IPEC.2018.4

Acknowledgements We thank Amir Abboud and Rasmus Pagh for useful discussions.

1 Introduction

Pairwise distances in an undirected, unweighted graph can be computed by performing a graph exploration, such as breadth-first search, from every vertex. This straightforward procedure determines the diameter of a given graph with n vertices and m edges in time $O(nm)$. It is surprisingly difficult to improve upon this idea in general. In fact, Roditty and Vassilevska Williams [14] have shown that an algorithm that can distinguish between diameter 2 and 3 in an undirected sparse graph in subquadratic time refutes the Orthogonal Vectors conjecture.

[1] Swedish Research Council grant VR-2016-03855 and Villum Foundation grant 16582.

13th International Symposium on Parameterized and Exact Computation (IPEC 2018).
Editors: Christophe Paul and Michał Pilipczuk; Article No. 4; pp. 4:1–4:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

However, for very sparse graphs, the running time becomes linear. In particular, the diameter of a tree can be computed in linear time $O(n)$ by a folklore result that traverses the graph twice. In fact, an algorithm by Cabello and Knauer shows that for constant treewidth $k \geq 3$, the diameter (and other distance parameters) can be computed in time $O(n \log^{k-1} n)$, where the Landau symbol absorbs the dependency on k as well as the time required for computing a tree decomposition. The question raised in [1] is how the complexity of this problem grows with the treewidth of the graph. We show the following result:

▶ **Theorem 1.** *The eccentricities, diameter, radius, and Wiener index of a given undirected n-vertex graph G of treewidth $\operatorname{tw}(G)$ and nonnegative edge lengths can be computed in time linear in*

$$n \cdot \binom{k + \lceil \log n \rceil}{k} \cdot 2^k k^2 \log n \tag{1}$$

where $k = 5\operatorname{tw}(G) + 4$.

For every $\epsilon > 0$, the bound (1) is $n^{1+\epsilon} \exp O(\operatorname{tw}(G))$. This improves the dependency on the treewidth over the running time $n^{1+\epsilon} \exp O\big(\operatorname{tw}(G) \log \operatorname{tw}(G)\big)$ of Abboud, Vassilevska Williams, and Wang [1]. Our improvement is tight in the following sense. Abboud *et al.* [1] also showed that under the Strong Exponential Time Hypothesis of Impagliazzo, Paturi, and Zane [10], there can be no algorithm that computes the diameter with running time

$$n^{2-\delta} \exp o(\operatorname{tw}(G)) \qquad \text{for any } \delta > 0\,. \tag{2}$$

In fact, this holds under the potentially weaker Orthogonal Vectors conjecture, see [17] for an introduction to these arguments. Thus, under this assumption, the dependency on $\operatorname{tw}(G)$ in Theorem 1 cannot be significantly improved, even if the dependency on n is relaxed from just above linear to just below quadratic. Our analysis encompasses the Wiener index, an important structural graph parameter left unexplored by [1].

Perhaps surprisingly, the main insight needed to establish Theorem 1 has nothing to do with graph distances or treewidth. Instead, we make – or re-discover – the following observation about the running time of d-dimensional range trees:

▶ **Lemma 2** **([13]).** *A d-dimensional range tree over n points supporting orthogonal range queries for the aggregate value over a commutative monoid has query time $O(2^d \cdot B(n,d))$ and can be built in time $O(nd \cdot B(n,d))$, where*

$$B(n,d) = \binom{d + \lceil \log n \rceil}{d}\,.$$

This is a more careful statement than the standard textbook analysis, which gives the query time as $O(\log^d n)$ and the construction time as $O(n \log^d n)$. For many values of d, the asymptotic complexities of these bounds agree – in particular, this is true for constant d and for very large d, which are the main regimes of interest to computational geometers. But crucially, $B(n,d)$ is always $n^\epsilon \exp O(d)$ for any $\epsilon > 0$, while $\log^d n$ is not.

After Lemma 2 is realised, Theorem 1 follows via divide-and-conquer in decomposable graphs, closely following the idea of Cabello and Knauer [6] and augmented with known arguments [1, 5]. We choose to give a careful presentation of the entire construction, as some of the analysis is quite fragile.

Using known reductions, this implies that the following multivariate lower bound on orthogonal range searching is tight:

▶ **Theorem 3** (Implicit in [1]). *A data structure for the orthogonal range query problem for the monoid* $(\mathbf{Z}, \max)$ *with construction time* $n \cdot q'(n,d)$ *and query time* $q'(n,d)$*, where*

$$q'(n,d) = n^{1-\epsilon} \exp o(d)$$

for some $\epsilon > 0$*, refutes the Strong Exponential Time hypothesis.*

We also investigate the same problems parameterized by vertex cover number:

▶ **Theorem 4.** *The eccentricities, diameter, and radius of a given undirected, unweighted* n*-vertex graph* G *with vertex cover number* k *can be computed in time* $O(nk + 2^k k^2)$*. The Wiener index can be computed in time* $O(nk2^k)$*.*

Both of these bounds are $n \exp O(k)$. It follows from [1] that a lower bound of the form (2) holds for this parameter as well.

1.1 Related work

Abboud *et al.* [1] show that given a graph and an optimal tree decomposition, various graph distances can be computed in time $O(k^2 n \log^{k-1} n)$, where $k = \mathrm{tw}(G)$. This bound is $n^{1+\epsilon} \exp O(k \log k)$ for any $\epsilon > 0$. This subsumes the running time for finding an approximate tree decomposition with $k = O(\mathrm{tw}(G))$ from the input graph [5], which is $n \exp O(k)$. Their algorithm extends the construction of Cabello and Knauer [6] to superconstant treewidth. According to [6], the idea of expressing graph distances as coordinates was first mentioned by Shi [15].

If the diameter in the input graph is constant, the diameter can be computed in time $n \exp O(\mathrm{tw}(G))$ [9]. This is tight in both parameters in the sense that [1] rules out the running time (2) even for distinguishing diameter 2 from 3, and every algorithm needs to inspect $\Omega(n)$ vertices even for treewidth 1. For non-constant diameter Δ, the bound from [9] deteriorates as $n \exp O(\mathrm{tw}(G) \log \Delta)$. However, the construction cannot be used to compute the Wiener index.

The literature on algorithms for graph distance parameters such as diameter or Wiener index is very rich, and we refer to the introduction of [1] for an overview of results directly relating to the present work. A recent paper by Bentert and Nichterlein [2] gives a comprehensive overview of many other parameterisations.

Orthogonal range searching using a multidimensional range tree was first described by Bentley [3], Lueker [12], Willard [16], and Lee and Wong [11], who showed that this data structure supports query time $O(\log^d n)$ and construction time $O(n \log^{d-1} n)$. Several papers have improved this in various ways by factors logarithmic in n; for instance, Chazelle's construction [8] achieves query time $O(\log^{d-1} n)$.

1.2 Discussion

In hindsight, the present result is a somewhat undramatic resolution of an open problem that has been viewed as potentially fruitful by many people [1], including the second author [9]. In particular, the resolution has led neither to an exciting new technique for showing conditional lower bounds of the form $n^{2-\epsilon} \exp \omega(k)$, nor a clever new algorithm for graph diameter. Instead, our solution follows the ideas of Cabello and Knauer [6] for constant treewidth, much like in [1]. All that was needed was a better understanding of the asymptotics of bivariate functions, rediscovering a 40-year old analysis of spatial data structures [13] (see the discussion in Sec. 3.3), and using a recent algorithm for approximate tree decompositions [5].

Of course, we can derive some satisfaction from the presentation of asymptotically tight bounds for fundamental graph parameters under a well-studied parameterization. In particular, the surprisingly elegant reductions in [1] cannot be improved. However, as we show in the appendix, when we parameterize by vertex cover number instead of treewidth, we can establish even cleaner and tight bounds without much effort.

Instead, the conceptual value of the present work may be in applying the multivariate perspective on high-dimensional computational geometry, reviving an overlooked analysis for non-constant dimension. To see the difference in perspective, Chazelle's improvement [8] of d-dimensional range queries from $\log^d n$ to $\log^{d-1} n$ makes a lot of sense for small d, but from the multivariate point of view, both bounds are $n^\epsilon \exp \Omega(d \log d)$. The range of relationships between d and n where the multivariate perspective on range trees gives some new insight is when d is asymptotically just shy of $\log n$, see Sec. 2.1.

It remains open to find an algorithm for diameter with running time $n \exp O(\mathrm{tw}(G))$, or an argument that such an algorithm is unlikely to exist under standard hypotheses. This requires better understanding of the regime $d = o(\log n)$.

2 Preliminaries

2.1 Asymptotics

We summarise the asymptotic relationships between various functions appearing in the present paper:

▶ **Lemma 5.**

$$B(n,d) = O(\log^d n)\,. \tag{3}$$

For any $\epsilon > 0$*,*

$$B(n,d) = n^\epsilon \exp O(d)\,, \tag{4}$$

$$\log^d n = n^\epsilon \exp \Omega(d \log d)\,, \tag{5}$$

$$\log^d n = n^\epsilon \exp O(d \log d)\,. \tag{6}$$

The first expression shows that $B(n,d)$ is always at least as informative as $O(\log^d n)$. The next two expressions show that from the perspective of parameterised complexity, the two bounds differ asymptotically: $B(n,d)$ depends single-exponentially on d (no matter how small $\epsilon > 0$ is chosen), while $\log^d n$ does not (no matter how *large* ϵ is chosen). Expression (6) just shows that (5) is maximally pessimistic.

Proof. Write $h = \lceil \log n \rceil$. To see (3), consider first the case where $d < h$. Using $\binom{a}{b} \leq a^b/b!$ we see that

$$\binom{d+h}{d} \leq \binom{2h}{d} \leq \frac{(2h)^d}{d!} = \frac{2^d}{d!} h^d = O(\log^d n)\,. \tag{7}$$

Next, if $d \geq h$ then

$$\binom{d+h}{d} = \binom{d+h}{h} \leq \binom{2d}{h} = \frac{2^h}{h!} d^h \leq d^h\,,$$

provided $h \geq 4$. It remains to observe that $d^h \leq h^d = O(\log^d n)$. Indeed, since the function $\alpha \mapsto \alpha / \ln \alpha$ is increasing for $\alpha \geq \mathrm{e}$, we have $h/\ln h \leq d/\ln d$, which implies $\exp(h \ln d) \leq \exp(d \ln h)$ as needed.

For (4), we let $\delta = d/h$ and consider two cases. First, from Stirling's formula we know $\binom{a}{b} \le \left(\frac{ea}{b}\right)^b$, so

$$\binom{d+h}{d} = \binom{(1+\delta)h}{\delta h} \le \left(\frac{e(1+\delta)h}{\delta h}\right)^{\delta h} \le \left(\frac{e(1+\delta)}{\delta}\right)^{2\delta \log n} = n^{2\delta \log(e(1+\delta)\delta^{-1})}.$$

Using that $\delta \mapsto 2\delta \log(e(1+\delta)\delta^{-1})$ is a monotone increasing function in the interval $\left(0, \frac{1}{2}\right]$ that tends to 0 for $\delta \to 0$, we obtain $\binom{d+h}{d} \le n^{\epsilon}$ for any sufficiently small δ.

It remains to consider the case that $\delta \ge c$ for some positive constant c depending only on ϵ. In this case, we have

$$\binom{d+h}{d} \le \binom{(1+1/c)d}{d} < 2^{(1+1/c)d} = \exp O(d).$$

We turn to (5). Assume that there is a function g such that

$$\log^d n = n^c g(d).$$

Then choose $b > 1$ and consider d such that $d = b^{-1} \log n$. Then

$$g(d) \ge \frac{\log^d n}{n^c} = 2^{d \log\log n - c \log n} = 2^{d \log(bd) - cbd} = \exp \Omega(d \log d).$$

Finally for (6), we repeat the argument from [1]. If $d \le \epsilon \log n / \log\log n$ then $\log^d n = 2^{d \log\log n} \le n^{\epsilon}$. In particular, if $d = o(\log n / \log\log n)$ then $\log^d n = n^{o(1)}$. Moreover, for $d \ge \log^{1/2} n$ we have $\log\log n \le 2 \log d$ and thus $\log^d n = 2^{d \log\log n} \le 4^{d \log d}$. ◀

These calculations also show the regimes in which these considerations are at all interesting. For $d = o(\log n / \log\log n)$ then both functions are bounded by $n^{o(1)}$, and the multivariate perspective gives no insight. For $d \ge \log n$, both bounds exceed n, and we are better off running n BFSs for computing diameters, or passing through the entire point set for range searching.

2.2 Model of computation

We operate in the word RAM, assuming constant-time arithmetic operations on coordinates and edge lengths, as well as constant-time operations in the monoid supported by our range queries. For ease of presentation, edge lengths are assumed to be nonnegative integers; we could work with abstract nonnegative weights instead [6].

3 Orthogonal Range Queries

3.1 Preliminaries

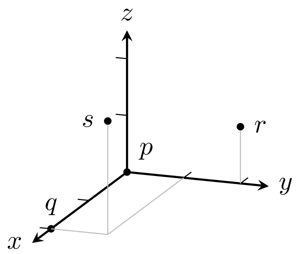
Let P be a set of d-dimensional points. We will view $p \in P$ as a vector $p = (p_1, \ldots, p_d)$.

A commutative *monoid* is a set M with an associative and commutative binary operator $\oplus$ with identity. The reader is invited to think of M as the integers with $-\infty$ as identity and $a \oplus b = \max\{a, b\}$.

Let $f \colon P \to M$ be a function and define for each subset $Q \subseteq P$

$$f(Q) = \bigoplus \{\, f(q) \colon q \in Q \}$$

with the understanding that $f(\emptyset)$ is the identity in M.



p	$(0,0,0)$	$f(p)=5$
q	$(2,0,0)$	$f(q)=6$
r	$(0,2,1)$	$f(r)=7$
s	$(2,1,2)$	$f(s)=8$

Figure 1 Four points in three dimensions. With the monoid $(\mathbf{Z}, \max)$ we have $f(\{p,r,s\}) = 8$.

3.2 Range Trees

Consider dimension $i \in \{1,\ldots,d\}$ and enumerate the points in Q as $q^{(1)},\ldots,q^{(r)}$ such that $q_i^{(j)} \leq q_i^{(j+1)}$, for instance by ordering after the ith coordinate and breaking ties lexicographically. Define $\mathrm{med}_i(Q)$ to be the *median* point $q^{(\lceil r/2\rceil)}$, and similarly $\min_i(Q) = q^{(1)}$ and $\max_i(Q) = q^{(r)}$. Set

$$Q_L = \{q^{(1)},\ldots,q^{(\lceil r/2\rceil)}\}, \qquad Q_R = \{q^{(1+\lceil r/2\rceil)},\ldots,q^{(r)}\}\,. \tag{8}$$

For $i \in \{1,\ldots,d\}$, the *range tree* $R_i(Q)$ for Q is a node x with the following attributes:

- $L[x]$, a reference to range tree $T_i(Q_L)$, called the *left child* of x. Only exists if $|Q| > 1$.
- $R[x]$, a reference to range tree $T_i(Q_R)$, called the *right child* of x. Only exists if $|Q| > 1$.
- $D[x]$, a reference to range tree $T_{i+1}(Q)$, called the *secondary*, *associate*, or *higher-dimensional* structure. Only exists for $i < d$.
- $l[x] = \min_i(Q)$.
- $r[x] = \max_i(Q)$.
- $f[x] = f(Q)$. Only exists for $i = d$.

Construction

Constructing a range tree for Q is a straightforward recursive procedure:

▶ **Algorithm C** (Construction). *Given integer $i \in \{1,\ldots,d\}$ and a list Q of points, this algorithm constructs the range tree $R_i(Q)$ with root x.*

C1 [Base case $Q = \{q\}$.] Recursively construct $D[x] = T_{i+1}(Q)$ if $i < d$, otherwise set $f[x] = f(q)$. Set $l[x] = r[x] = q_i$. Return x.

C2 [Find median.] Determine $q = \mathrm{med}_i\, Q$, $l[x] = \min_i(Q)$, $r[x] = \max_i(Q)$.

C3 [Split Q.] Let Q_L and Q_R as given by (8), note that both are nonempty.

C4 [Recurse.] Recursively construct $L[x] = R_i(Q_L)$ from Q_L. Recursively construct $R[x] = R_i(Q_R)$ from Q_R. If $i < d$ then recursively construct $D[x] = T_{i+1}(Q)$. If $i = d$ then set $f[x] = f[L[x]] \oplus f[R[x]]$.

The data structure can be viewed as a collection of binary trees whose nodes x represent various subsets P_x of the original point set P. In the interest of analysis, we now introduce a scheme for naming the individual nodes x, and thereby also the subsets P_x. Each node x is identified by a string of letters from $\{\mathrm{L},\mathrm{R},\mathrm{D}\}$ as follows. Associate with x a set of points, often called the *canonical subset* of x, as follows. For the empty string ϵ we set $P_\epsilon = P$. In general, if $Q = P_x$ then $P_{x\mathrm{L}} = Q_L$, $P_{x\mathrm{R}} = Q_R$ and $P_{x\mathrm{D}} = Q$. The strings over $\{\mathrm{L},\mathrm{R},\mathrm{D}\}$ can be understood as uniquely describing a path through in the data structure; for instance, L means 'go *left*, i.e., to the left subtree, the one stored at $L[x]$' and D means 'go to the next *dimension*, i.e., to the subtree stored at $D[x]$.' The name of a node now describes the unique path that reaches it.

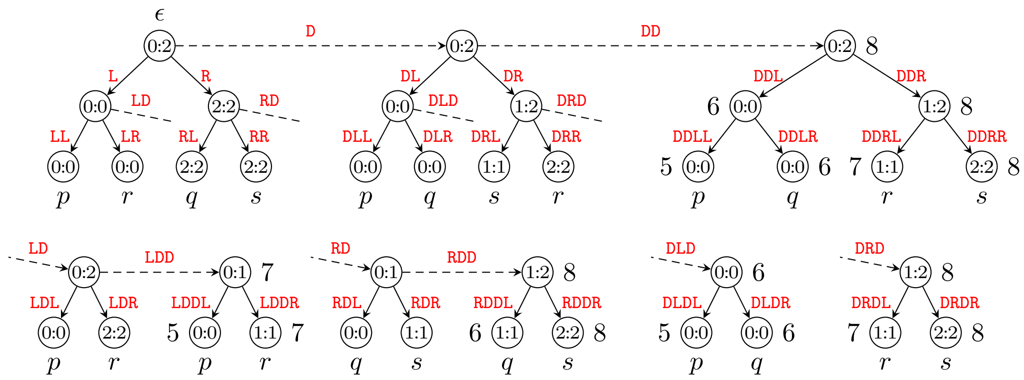


Figure 2 Part of the range tree for the points from Fig. 1. The label of node x appears in red on the arrow pointing to x. Nodes contain $l[x]$:$r[x]$. The references $L[x]$ and $R[x]$ appear as children in a binary tree using usual drawing conventions. The reference $D[x]$ appears as a dashed arrow (possibly interrupted); the placement on the page follows no other logic than economy of layout and readability. References $D[x]$ from leaf nodes, such as $D[\text{LL}]$ leading to node LLD, are not shown; this conceals 12 single-node trees. The '3rd-dimensional nodes,' whose names contain two Ds, show the values $f[x]$ next to the node. To ease comprehension, leaf nodes are decorated with their canonical subset, which is a singleton from $\{p, q, r, s\}$. The reader can infer the canonical subset for an internal node as the union of leaves of the subtree; for instance, $P_{\text{DR}} = \{r, s\}$. However, note that these point sets are *not* explicitly stored in the data structure.

▶ **Lemma 6.** *Let $n = |P|$. Algorithm C computes the d-dimensional range tree for P in time linear in $nd \cdot B(n, d)$.*

Proof. We run Algorithm C on input P and $i = 1$.

Disregarding the recursive calls, the running time of algorithm C on input i and Q is dominated by Steps C2 and C3, i.e., splitting Q into two sets of equal size. It is known that this task can be performed in time linear in $|Q|$ [4]. Thus, the running time for constructing $R_i(Q)$ is linear in $|Q|$ plus the time spent in recursive calls.

This means that we can bound the running time for constructing $T_1(P)$ by bounding the sizes of the sets P_x associated with every node x in the data structure. If for a moment X denotes the set of all these nodes then we want to bound

$$\sum_{x \in X} |P_x| = \sum_{x \in X} |\{\, p \in P : p \in P_x \,\}| = \sum_{p \in P} |\{\, x \in X : p \in P_x \,\}| \,.$$

Thus, we need to determine, for given $p \in P$, the number of subsets P_x in which p appears. By construction, there are fewer than d occurrences of D in x. Moreover, if x contains more than h occurrences of either L or R then P_x is empty. Thus, x has at most $h + d$ letters. For two different strings x and x' that agree on the positions of D, the sets P_x and $P_{x'}$ are disjoint, so p appears in at most one of them. We conclude that the number of sets P_x such that $p \in P_x$ is bounded by the number of ways to arrange fewer than d many Ds and at most h non-Ds. Using the identity $\binom{a+0}{0} + \cdots + \binom{a+b}{b} = \binom{a+b+1}{b}$ repeatedly, we compute

$$\sum_{i=0}^{d-1} \sum_{j=0}^{h} \binom{i+j}{j} = \sum_{i=0}^{d-1} \binom{i+h+1}{h} = \sum_{i=0}^{d-1} \binom{i+h+1}{i+1} =$$

$$(-1) + \sum_{i=0}^{d} \binom{i+h}{i} = \binom{h+d+1}{d} - 1 = \frac{h+d+1}{h+1} \binom{h+d}{d} - 1 \le d \binom{d+h}{d} \,.$$

The bound follows from aggregating this contribution over all $p \in P$. ◀

Search

In this section, we fix two sequences of integers $l_1, \ldots, l_d$ and $r_1, \ldots, r_d$ describing the *query box* B given by

$$B = [l_1, r_1] \times \cdots \times [l_d, r_d] .$$

▶ **Algorithm Q** (Query). *Given integer $i \in \{1, \ldots, d\}$, a query box B as above and a range tree $R_i(Q)$ with root x for a set of points Q such that every point $q \in Q$ satisfies $l_j \leq q_j \leq r_j$ for $j \in \{1, \ldots, i-1\}$, this algorithm returns $\bigoplus\{ f(q) \colon q \in Q \cap B \}$.*

Q1 [Empty?] If the data structure is empty, or $l_i > r[x]$, or $l[x] > r_i$, then return the identity in the underlying monoid M.

Q2 [Done?] If $i = d$ and $l_d \leq \min_d[x]$ and $\max_d[x] \leq r_d$ then return $f[x]$.

Q3 [Next dimension?] If $i < d$ and $l_i \leq l[x]$ and $r[x] \leq r_i$ then query the range tree at $D[x]$ for dimension $i + 1$. Return the resulting value.

Q4 [Split.] Query the range tree $L[x]$ for dimension i; the result is a value f_L. Query the range tree $R[x]$ for dimension i; the result is a value f_R. Return $f_L \oplus f_R$.

To prove correctness, we show that this algorithm is correct for each point set $Q = P_x$.

▶ **Lemma 7.** *Let $i = D(x) + 1$, where $D(x)$ is the number of* Ds *in x. Assume that P_x is such that $l_j \leq p_i \leq r_j$ for all $j \in \{1, \ldots, i-1\}$ for each $p \in P_x$. Then the query algorithm on input x and i returns $f(B \cap P_x)$.*

Proof. Backwards induction in $|x|$.

If $|x| = h + d$ then P_x is the empty set, in which case the algorithm correctly returns the identity in M.

If the algorithm executes Step Q2 then B is satisfied for all $q \in P_x$, in which case the algorithm correctly returns $f[x] = f(P_x)$.

If the algorithm executes Step Q3 then B satisfies the condition in the lemma for $i + 1$, and the number of Ds in $P_{x\mathrm{D}}$ is $i + 1$, and $D[x]$ store the $(i + 1)$th range tree for $P_{x\mathrm{D}}$. Thus, by induction the algorithm returns $f(P_{x\mathrm{D}} \cap B)$, which equals $f(P_x \cap B)$ because $P_{x\mathrm{D}} = P_x$.

Otherwise, by induction, $f_L = f(P_{x\mathrm{L}} \cap B)$ and $f_R = f(P_{x\mathrm{R}} \cap B)$. Since $P_{x\mathrm{L}} \cup P_{x\mathrm{R}} = P_x$, we have $f(P_x \cap B) = f((P_{x\mathrm{L}} \cap B) \cup (P_{x\mathrm{R}} \cap P)) = f_L \oplus f_R$. ◀

▶ **Lemma 8.** *If x is the root of the range tree for P then on input $i = 1$, x, and B, the query algorithm returns $f(P \cap B)$ in time linear in $2^d B(n, d)$.*

Proof. Correctness follows from the previous lemma.

For the running time, we first observe that the query algorithm does constant work in each visited node. Thus it suffices to bound the number of visited nodes as

$$2^d \binom{h+d}{d} \qquad (d \geq 1, h \geq 0) . \tag{9}$$

We will show by induction in d that (9) holds for every call to a d-dimensional range tree for a point set P_x, where $h = \lceil \log |P_x| \rceil$. The two easy cases are Q1 and Q2, which incur no additional nodes to be visited, so the number of visited nodes is 1, which is bounded by (9). Step Q3 leads to a recursive call for a $(d-1)$-dimensional range tree over the same point set $P_{x\mathrm{D}} = P_x$, and we verify

$$1 + 2^{d-1} \binom{h+d-1}{d-1} \leq 2^d \binom{h+d}{d} .$$

The interesting case is Step Q4. We need to follow two paths from x to the leaves of the binary tree of x. Consider the leaves l and r in the subtree rooted at x associated with the points $\min_i(P_x)$ and $\max_i(P_x)$ as defined in Sec. 3.2. We describe the situation of the path Y from l to x; the other case is symmetrical. At each internal node $y \in Y$, the algorithm chooses Step Q4 (because $l_i \geq l[y]$). There are two cases for what happens at yL and yR. If $l_i \leq \text{med}_i(P_y)$ then $P_{y\text{R}}$ satisfies $l_i \leq \min_i(P_{y\text{R}}) \leq r_i$, so the call to yR will choose Step Q3. By induction, this incurs $2^{d-1}\binom{d-1+i}{d-1}$ visits, where i is the height of y. In the other case, the call to yL will choose Step Q1, which incurs no extra visits. Thus, the number of nodes visited on the left path is at most

$$h + \sum_{i=0}^{h-1} 2^{d-1}\binom{d-1+i}{d-1},$$

and the total number of nodes visited is at most twice that:

$$2h + 2^d \sum_{i=0}^{h-1}\binom{d-1+i}{d-1} \leq 2^d \sum_{i=0}^{h}\binom{d-1+i}{d-1} = 2^d\binom{d+h}{d}.$$ ◀

3.3 Discussion

The textbook analysis of range trees, and similar d-dimensional spatial algorithms and data structures sets up a recurrence relation like

$$r(n,d) = 2r(n/2,d) + r(n,d-1),$$

for the construction and

$$r(n,d) = \max\{\, r(n/2,d), r(n,d-1)\,\},$$

for the query time. One then observes that $n\log^d n$ and $\log^d n$ are the solutions to these recurrences. This analysis goes back to Bentley's original paper [3].

Along the lines of the previous section, one can show that the functions $n \cdot B(n,d)$ and $B(n,d)$ solve these recurrences as well. A detailed derivation can be found in [13], which also contains combinatorial arguments of how to interpret the binomial coefficients in the context of spatial data structures. A later paper of Chan [7] also takes the recurrences as a starting point, and observes asymptotically improved solution for the related question of dominance queries.

4 Graph Distances

We present the algorithm for computing the diameter. The construction closely follows Cabello and Knauer [6], but uses the range tree bounds from Section 3. The analysis is extended to superconstant dimension as in Abboud *et al.* [1]. Using the approximate treewidth construction of Bodlaender *et al.* [5], we can pay more attention to the parameters of the recursive decomposition into small-size separators.

4.1 Preliminaries

We consider an undirected graph G with n vertices and m edges with nonnegative integer weights. The set of vertices is $V(G)$. For a vertex subset U we write $G[U]$ for the induced subgraph.

A path from u to v is called a u, v-path and denoted P. The *length* of a path, denoted $l(P)$, is the sum of its edge lengths.

The *distance* from vertex u to vertex v, denoted $d(u, v)$, is the minimum length of shortest u, v-path. The *Wiener index* of G, denoted $\operatorname{wien}(G)$ is $\sum_{u,v \in V(G)} d(u, v)$. The *eccentricity* of a vertex u, denoted $e(u)$ is given by $e(u) = \max\{\, d(u, v) \colon v \in V(G) \,\}$. The *diameter* of G, denoted $\operatorname{diam}(G)$ is $\max\{\, e(u) \colon u \in V(G) \,\}$. The *radius* of G, denoted $\operatorname{rad}(G)$ is $\min\{\, e(u) \colon u \in V(G) \,\}$.

4.2 Separation

A vertex subset Z separates X and Y if every x, y-path with $x \in X$ and $y \in Y$ contains a vertex from Z. A *skew k-separator tree* T of G is a binary tree such that each node t of T is associated with a vertex set $Z_t \subseteq V(G)$ such that

- $|Z_t| \leq k$,
- If L_t and R_t denote the vertices of G associated with the left and right subtrees of t, respectively, then Z_t separates L_t and R_t and

$$\frac{n}{k+1} \leq |L_t \cup Z_t| \leq \frac{nk}{k+1}\,, \tag{10}$$

- T remains a skew k-separator even if edges between vertices of Z_t are added.

It is known that such a tree can be found from a tree decomposition, and an approximate tree decomposition can be found in single-exponential time. We summarise these results in the following lemma:

▶ **Lemma 9** ([6, Lemma 3] with [5, Theorem 1])**.** *For a given n-vertex input graph G, a skew $(5\operatorname{tw}(G) + 4)$-separator tree can be computed in time $n \exp O(\operatorname{tw}(G))$.*

4.3 Algorithm

We follow the construction of [6].

Given graph G, let $\mathcal{S}_{x,w}$ denote the set of shortest x, w-paths. We refine the notion of eccentricity to a subset W of vertices. Formally,

$$e(x, W) = \max_{w \in W} \{\, l(P) \colon P \in \mathcal{S}_{x,w} \,\}\,.$$

We will consider a situation where $V(G) = X \cup Y$ with separator $Z = X \cap Y$. We can then compute $e(x)$ as $\max\{e(x, X), e(x, Y)\}$ for each $x \in X$. The first term is found recursively in $G[X]$; the interesting part is the computation of $e(x, Y)$.

Enumerate $Z = \{z_1, \ldots, z_k\}$. For $i \in \{1, \ldots, k\}$ define the ith eccentricity $e_i(x, Y)$ as the maximum distance from x to any vertex in Y 'via z_i.' Formally,

$$e_i(x, Y) = \max_{y \in Y} \{\, l(P) \colon P \in \mathcal{S}_{x,y}, z_i \in V(P) \,\}\,.$$

See Figure 3 for a small example.

▶ **Lemma 10.** *If Z separates X and Y then $e(x, Y) = \max_{i=1}^{k} e_i(x, Y)$ for $x \in X$.*

Proof. A shortest x, y-path with $y \in Y$ must contain a vertex from Z, say z_i. Thus, $e(x, Y) \leq e_i(x, Y)$. Conversely, $e(x, Y) \geq e_j(x, Y)$ for all $j \in \{1, \ldots, k\}$ from the definition. ◀

Now we can write the eccentricity *via z_i* as the distance *to z_i* plus a range query:

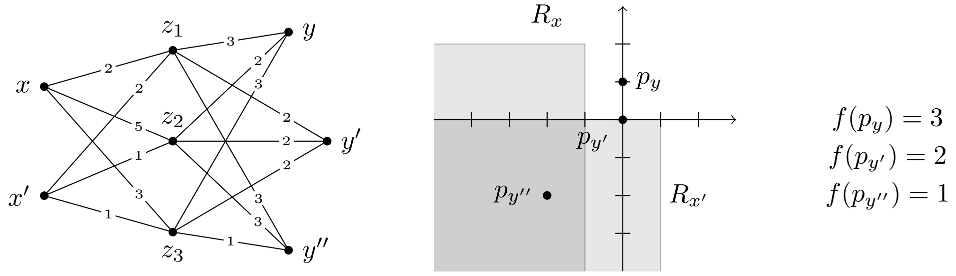


Figure 3 Left: Example with $Z = \{z_1, z_2, z_3\}$ and $Y = Z \cup \{y, y', y''\}$. We have $e(x, Y) = 5$ (along xz_1y) and $e(x', Y) = 3$. For the case $i = 3$ we see $e_3(x, Y) = 4$ along xz_3y'', because there are no shortest paths from x via z_3 to y or y', and the one-edge path xz_3 itself is shorter. Similarly, $e_3(x', Y) = 3$ (along $x'z_3y'$). Right: The corresponding points in $\mathbf{Z}^2$, only the first two coordinates are shown, and only for the points in $Y \setminus Z$. The points corresponding to y' and y'' both belong to the rectangle for x', certifying that there are shortest x', y'- and x', y''-paths through z_3. Right: Over $R_{x'}$, the point $p_{y'}$ maximises f. We have $e_3(x', Y) = l(x'z_3y') = d(x', z_3) + f(p_{y'}) = 1 + 2 = 3$.

▶ **Lemma 11.** *Let $i \in \{1, \ldots, k\}$ and assume $\{z_1, \ldots, z_k\}$ separates X and Y. Define for each $y \in Y$ the k-dimensional point*

$$p_y = \begin{pmatrix} d(z_i, y) - d(z_1, y) \\ \vdots \\ d(z_i, y) - d(z_k, y) \end{pmatrix} \quad \text{with } f(p_y) = d(z_i, y)\,. \tag{11}$$

Define for each $x \in X$ the rectangle

$$R_x = \times_{j=1}^{k} [-\infty, d(x, z_j) - d(x, z_i)]\,. \tag{12}$$

Then

$$e_i(x, Y) = d(x, z_i) + \max_{y\colon p_y \in R_x} f(p_y)\,.$$

Proof. Consider a shortest x, y-path P containing $z_i \in Z$. No other x, y-path is shorter than P, so in particular we have

$$d(x, z_i) + d(z_i, y) \le d(x, z_j) + d(z_j, y)\,, \qquad j \in \{1, \ldots, k\}\,,$$

equivalently,

$$d(z_i, y) - d(z_j, y) \le d(x, z_j) - d(x, z_i)\,, \qquad j \in \{1, \ldots, k\}\,. \tag{13}$$

which means $p_y \in R_x$. Moreover, if y is chosen so that P attains the eccentricity $e_i(x, Y)$ then $e_i(x, Y) = l(P) = d(x, z_i) + d(z_i, y)$ and p_y maximises $f(p_y) = d(z_i, y)$ over the points in R_x. ◀

One observes that the ith coordinate of p_y is always 0 and of R_y is always $[-\infty, 0]$, so the reduction is actually to a $(k-1)$-dimensional range query instance. However, we are mainly interested in the asymptotic dependency on k, so we avoid this possible (but tedious) improvement.

We are ready for the algorithm.

▶ **Algorithm E** (Eccentricities). *Given an undirected, connected graph G with nonnegative integer weights and a skew k-separator tree with root t, this algorithm computes the eccentricity $e(v)$ of every vertex $v \in V(G)$. We write $Z = Z_t$, $X = L_t \cup Z_t$, and $Y = R_t \cup Z_t$.*

E1 [Base case.] If $n/\ln n < 4k(k+1)$ find all distances using Dijkstra's algorithm. Terminate.
E2 [Distances from separator.] Compute $d(z, v)$ for each $z \in Z, v \in V(G)$ using k applications of Dijkstra's algorithm. Compute $e(z, Y) = \max_{y \in Y} d(z, y)$ for each $z \in Z$.
E3 [Add shortcuts.] For each pair $z, z' \in Z$, add the edge zz' to G, weighted by $d(z, z')$. Remove duplicate edges, retaining the shortest.
E4.1 [Start iterating over $\{z_1, \ldots, z_k\}$.] Let $i = 1$.
E4.2 [Build range tree for z_i.] Construct a k-dimensional range tree for the points $\{\, p_y : y \in Y \,\}$ given by (11) using the monoid $(\mathbf{Z}, \max)$.
E4.3 [Query range tree.] For each $x \in X$, query the rectangle R_x given by (12) and add $d(x, z_i)$. The result is $e_i(x, Y)$ by Lemma 11.
E4.4 [Next z_i.] If $i < k$ then increase i and go to E4.1.
E5 [Recurse on $G[X]$ and combine.] Recursively compute the distances in $G[X]$ using the left subtree of t as a skew k-separator tree. The result are eccentricities $e(x, X)$ for each $x \in X$. For each $x \in X$, set $e(x, Y) = \max_{i=1}^{k} e_i(x, Y)$ from Step E4.3, then set $e(x) = \max\{e(x, X), e(x, Y)\}$.
E6 [Flip.] Repeat Steps E4–5 with the roles of X and Y exchanged.

4.4 Running Time

▶ **Lemma 12.** *The running time of Algorithm E is $O(n \cdot B(n, k) \cdot 2^k k^2 \log n)$.*

We omit the proof. We can now establish Theorem 1 for diameter and radius.

Proof of Thm. 1, distances. To compute all eccentricities for a given graph we find a k-skew separator for $k = 5\,\mathrm{tw}(G)+4$ using Lemma 9 in time $n \exp O(\mathrm{tw}(G))$. We then run Algorithm E, using Lemma 12 to bound the running time. From the eccentricities, the radius and diameter can be computed in linear time using their definition. ◀

Algorithm E can be modified to compute the Wiener index, as described in [6, Sec. 4], completing the proof of Theorem 1. The main observation is that the sum of distances between all pair $u, v \in V(G)$ can be written as pairwise distances within X, within Y, and between X and Y, carefully subtracting contributions from these sums that were included twice. The orthogonal range queries for vertex $x \in X$ now need to report the sum of distances to every $y \in Y$, rather than just the value of the maximum distance $e(x; Y)$. To this end, we use the monoid of positive integer tuples (d, r) with the operation $(d, r) \oplus (d', r') = (d + d', r + r')$ with identity element $(0, 0)$. The value associated with vertex x in Step E4.2 is $f(p(y)) = (1, d(z_i, y))$. To avoid overcounting, the definition of R_x and $e_i(x, Y)$ have to be changed carefully.

References

1 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. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.

2 Matthias Bentert and André Nichterlein. Parameterized Complexity of Diameter. *CoRR*, abs/1802.10048, 2018. arXiv:1802.10048.

3 Jon Louis Bentley. Multidimensional Divide-and-Conquer. *Commun. ACM*, 23(4):214–229, 1980. doi:10.1145/358841.358850.

4 Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time Bounds for Selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973. doi:10.1016/S0022-0000(73)80033-9.

5 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michał Pilipczuk. An $O(c^k n)$ 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016. doi:10.1137/130947374.

6 Sergio Cabello and Christian Knauer. Algorithms for bounded treewidth with orthogonal range searching. *Comput. Geom.*, 42(9):815–824, 2009. doi:10.1016/j.comgeo.2009.02.001.

7 Timothy M. Chan. All-Pairs Shortest Paths with Real Weights in $O(n^3/\log n)$ Time. *Algorithmica*, 50(2):236–243, 2008. doi:10.1007/s00453-007-9062-1.

8 Bernard Chazelle. Lower Bounds for Orthogonal Range Searching: I. The Reporting Case. *J. Assoc. Comput. Mach.*, 37(2):200–212, 1990. doi:10.1145/77600.77614.

9 Thore Husfeldt. Computing Graph Distances Parameterized by Treewidth and Diameter. In *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:11, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2016.16.

10 Russel Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

11 Der-Tsai Lee and Chakkuen K. Wong. Quintary Trees: A File Structure for Multidimensional Database Systems. *ACM Trans. Database Syst.*, 5(3):339–353, 1980. doi:10.1145/320613.320618.

12 George S. Lueker. A Data Structure for Orthogonal Range Queries. In *19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16-18 October 1978*, pages 28–34. IEEE Computer Society, 1978. doi:10.1109/SFCS.1978.1.

13 Louis Monier. Combinatorial Solutions of Multidimensional Divide-and-Conquer Recurrences. *J. Algorithms*, 1(1):60–74, 1980. doi:10.1016/0196-6774(80)90005-X.

14 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 515–524, 2013. doi:10.1145/2488608.2488673.

15 Qiaosheng Shi. *Efficient algorithms for network center/covering location optimization problems.* PhD thesis, School of Computing Science, Simon Fraser University, 2008.

16 Dan E. Willard. New data structures for orthogonal range queries. *SIAM J. Comput.*, 14(1):232–253, 1985. doi:10.1137/0214019.

17 Virginia Vassilevska Williams. Hardness of Easy Problems: Basing Hardness on Popular Conjectures such as the Strong Exponential Time Hypothesis (Invited Talk). In *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, pages 17–29, 2015. doi:10.4230/LIPIcs.IPEC.2015.17.