



Multivariate Analysis of Orthogonal Range Searching and Graph Distances

Karl Bringmann¹ · Thore Husfeldt^{2,3} · Måns Magnusson⁴

Received: 27 November 2018 / Accepted: 13 January 2020 / Published online: 14 March 2020
© The Author(s) 2020

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 $O\left(n \cdot \binom{k + \lceil \log n \rceil}{k} \cdot 2^k \log n\right)$, where k is linear in 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 et al. (in: Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, 2016. <https://doi.org/10.1137/1.9781611974331.ch28>) 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 (Comput Geom 42:815–824, 2009. <https://doi.org/10.1016/j.comgeo.2009.02.001>) 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 Algorithms 1:60–74, 1980. [https://doi.org/10.1016/0196-6774\(80\)90005-X](https://doi.org/10.1016/0196-6774(80)90005-X)). We also investigate the parameterization by vertex cover number.

Keywords Diameter · Radius · Wiener index · Orthogonal range searching · Treewidth · Vertex cover number

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

T. Husfeldt: Swedish Research Council Grant VR-2016-03855 and Villum Foundation Grant 16582.

✉ Thore Husfeldt
thore@itu.dk

Extended author information available on the last page of the article

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 [17] have shown that an algorithm that can distinguish between diameter 2 and 3 in an undirected sparse graph in subquadratic time would refute the Orthogonal Vectors conjecture.

However, for very sparse graphs, the running time becomes linear even for weighted graphs. For instance, the diameter of a star can be computed by finding the two largest edge weights. The diameter of a tree can be computed in linear time $O(n)$ by a folklore result that traverses the graph twice. In other words, for graphs with vertex cover number 1 or treewidth 1, the running time is $O(n)$.

The present paper investigates how these structural parameters influence the complexity of computing several graph distance measures. These measures are the *eccentricity* of every vertex (its maximum distance to any other vertex), the *diameter* and *radius* of the graph (the maximum and minimum eccentricities), and the *Wiener index* (the sum of the distances between all pairs of vertices); precise definitions are in Sect. 4.1. Throughout this paper we will write

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

Theorem 1 *The eccentricities, diameter, radius, and Wiener index of a given undirected n -vertex graph G with nonnegative integer weights can be computed in time*

1. $O(n \cdot B(n, k) \cdot 2^k)$ with $k = \text{vc}(G)$, where $\text{vc}(G)$ is the vertex cover number of G ,
2. $O(n \cdot B(n, k) \cdot 2^k \log n)$ with $k = 5\text{tw}(G) + 4$, where $\text{tw}(G)$ is the treewidth of G .

For every $\epsilon > 0$, the bounds in both cases are

$$n^{1+\epsilon} \exp O(k).$$

Since $\text{tw}(G) \leq \text{vc}(G)$, the treewidth result is in some sense stronger. However, the vertex cover result is slightly faster, already contains the core algorithmic idea, and avoids many distracting technicalities.

Theorem 1 improves the dependency on the treewidth over the running time

$$n^{1+\epsilon} \exp O(\text{tw}(G) \log \text{tw}(G))$$

of Abboud, Vassilevska Williams, and Wang [1]. Previously, Cabello and Knauer [7] had shown 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 exponential dependency on k as well as the time required for computing a tree

decomposition. The bound in Theorem 1 is tight in the following sense. Abboud et al. [1] also showed that under the Strong Exponential Time Hypothesis of Impagliazzo, Paturi, and Zane [13], there can be no algorithm that computes the diameter with running time

$$n^{2-\delta} \exp o(k) \quad \text{for any } \delta > 0, \quad (1)$$

for $k = \text{vc}(G)$ and (therefore also) $k = \text{tw}(G)$. In fact, this holds under the potentially weaker Orthogonal Vectors conjecture, see [20] for an introduction to these arguments. Thus, under this assumption, the dependency on k in Theorem 1 cannot be significantly improved, even if the dependency on n is relaxed from just above linear to just below quadratic. This closes an open question raised in [1].

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

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 in computational geometry. But crucially, $B(n, d)$ is always $n^\epsilon \exp O(d)$ for any $\epsilon > 0$, while $\log^d n$ is not.

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 in d dimensions 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 observe in the appendix that for unweighted graphs, the vertex cover result can be improved without using the techniques advertised in the present paper.

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 $n \exp O(k)$, matching (1). We do not know of a similar simplification for treewidth; the bound in Theorem 1.2, and the full construction behind it, seem to be the best we can do even for unit lengths.

1.1 Related Work

Abboud et al. [1] show that given a graph and a tree decomposition of width k , various graph distances can be computed in time $O(k^2 n \log^{k-1} n)$. This bound is $n^{1+\epsilon} \exp O(k \log k)$ for any $\epsilon > 0$. It is known how to compute an approximate tree decomposition with $k = O(\text{tw}(G))$ from the input graph G in time $n \exp O(\text{tw}(G))$ [6], so from a given graph (without a tree decomposition) the algorithm from [1] works in time $n^{1+\epsilon} \exp O(\text{tw}(G) \log \text{tw}(G))$, extending the construction of Cabello and Knauer [7] to superconstant treewidth. According to [7], the idea of expressing graph distances as coordinates was first mentioned by Shi [18].

If the diameter in the input graph is constant, the diameter can be computed in time $n \exp O(\text{tw}(G))$ [12]. This is tight in both parameters in the sense that [1] rules out the running time (1) 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 [12] deteriorates as $n \exp O(\text{tw}(G) \log \Delta)$. 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 [15], Willard [19], and Lee and Wong [14], 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 [9] achieves query time $O(\log^{d-1} n)$. In general, queries that *report* the points Q within a given range, instead of (like in the present paper) computing sums or maxima, incur an additional $O(|Q|)$ term in the query time.

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 of this paper [12]. 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 [7] 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 [16] (see the discussion in Sect. 3.3), and using a recent algorithm for approximate tree decompositions [6].

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 [9] 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 Sect. 2.1.

Table 1 summaries the known bounds for computing the diameter. It remains open to find an algorithm for diameter with running time $n \exp O(\text{tw}(G))$ even for unweighted graphs, 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
$$B(n, d) = O(\log^d n). \tag{2}$$

For any $\epsilon > 0$,

$$B(n, d) = n^\epsilon \exp O(d), \tag{3}$$

$$\log^d n = n^\epsilon \exp \Omega(d \log d), \tag{4}$$

Table 1 Bounds on algorithms for computing the diameter of a graph

Running time	Comment	Where
$O(n)$	If $\text{tw}(G) = 1$	Folklore
$n^{1+\epsilon} \exp O(\text{tw}(G))$		Theorem 1
$n \exp O(\text{vc}(G))$	If G is unweighted †	Theorem 4
$n \exp O(\text{tw}(G))$	If G has constant diameter ‡	[12]
$n \exp o(\text{vc}(G))$	Impossible under SETH, even given † and ‡	[1]

Recall that $\text{tw}(G) \leq \text{vc}(G)$, which implies some more bounds not given here

$$\log^d n = n^\epsilon \exp O(d \log d). \tag{5}$$

The first expression shows that $B(n, d)$ is always at least as good a bound 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). Our proof in fact establishes the stronger bound $B(n, d) \leq n^\epsilon + \exp O(d)$. Expression (5) just shows that (4) is maximally pessimistic.

Proof Write $h = \lceil \log n \rceil$. To see (2), consider first the case where $d < h$. Using $\binom{a}{b} \leq a^b/b!$ we see that

$$\binom{d+h}{d} \leq \binom{2h}{d} \leq \frac{(2h)^d}{d!} = \frac{2^d}{d!} h^d = O(\log^d n). \tag{6}$$

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 $\alpha \geq e$, we have $h/\ln h \leq d/\ln d$, which implies $\exp(h \ln d) \leq \exp(d \ln h)$ as needed.

For (3), we let $\delta = d/h$ and consider two cases: $\delta = o(1)$ or not. First, from Stirling’s formula we know $\binom{a}{b} \leq \left(\frac{ea}{b}\right)^b$, so

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

Using that $\delta \mapsto 2\delta \log(e(1+\delta)\delta^{-1})$ is positive in the interval $(0, \frac{1}{2}]$ and tends to 0 for $\delta \rightarrow 0$, we obtain $\binom{d+h}{d} \leq n^\epsilon$ for any sufficiently small δ .

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

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

We turn to (4). Let $\epsilon > 0$ and consider any function g such that for all $n \geq 1$,

$$\log^d n \leq n^\epsilon g(d).$$

Then $\log g(d) \geq d \log \log n - \epsilon \log n$. In particular, for $n = 2^d$, we have $\log g(d) \geq d \log d - \epsilon d = \Omega(d \log d)$, so $g(d) = \exp \Omega(d \log d)$.

Finally for (5), we repeat the argument from [1]. If $d \leq \epsilon \log n / \log \log n$ then $\log^d n = 2^{d \log \log n} \leq n^\epsilon$. In particular, if $d = o(\log n / \log \log n)$ then

$\log^d n = n^{o(1)}$. Moreover, for $d \geq \log^{1/2} n$ we have $\log \log n \leq 2 \log d$ and thus $\log^d n = 2^{d \log \log n} \leq 4^{d \log d}$. □

These calculations also show the regimes in which these considerations are at all interesting. For $d = o(\log n / \log \log n)$ both functions are bounded by $n^{o(1)}$, and the multivariate perspective gives no insight. For $d \geq \log n$, 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 nonnegative weights instead [7].

3 Orthogonal Range Queries

3.1 Preliminaries

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

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 $f : P \rightarrow M$ be a function and define for each subset $Q \subseteq P$

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

with the understanding that $f(\emptyset)$ is the identity in M . See Fig. 1 for a small example.

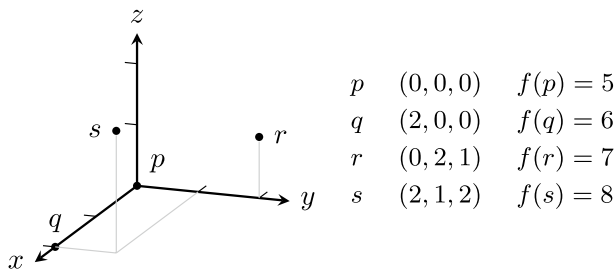


Fig. 1 Four points in three dimensions. With the monoid (\mathbb{Z}, \max) we have $f(\{p, r, s\}) = 8$

3.2 Range Trees

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

$$Q_L = \{q^{(1)}, \dots, q^{(\lceil r/2 \rceil)}\}, \quad Q_R = \{q^{(1+\lceil r/2 \rceil)}, \dots, q^{(r)}\}. \tag{7}$$

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

- $L[x]$, 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 $i \in \{1, \dots, d\}$ 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 $q = \text{med}_i Q$, $l[x] = \min_i(Q)$, $r[x] = \max_i(Q)$.
- C3** [Split Q .] Let Q_L and Q_R as given by (7), 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 $\{L, R, D\}$ 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 $P_{xL} = Q_L$, $P_{xR} = Q_R$ and $P_{xD} = Q$. The strings over $\{L, R, D\}$ 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 shows (part of) the range tree for the points in Fig. 1.

Lemma 6 *Let $n = |P|$. Algorithm C computes the d -dimensional range tree for P in time linear in $nd^2B(n, d)$.*

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 using $O(|Q|)$ many comparisons [5]. Each (lexicographic) comparison can take d steps. Thus, the running time for constructing $R_i(Q)$ is linear in $d|Q|$ 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 X denotes the set of nodes in the data structure, 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 . Set $h = \lceil \log n \rceil$. Every L or R corresponds to cutting the current points set in half, so if x contains more than h occurrences that are 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

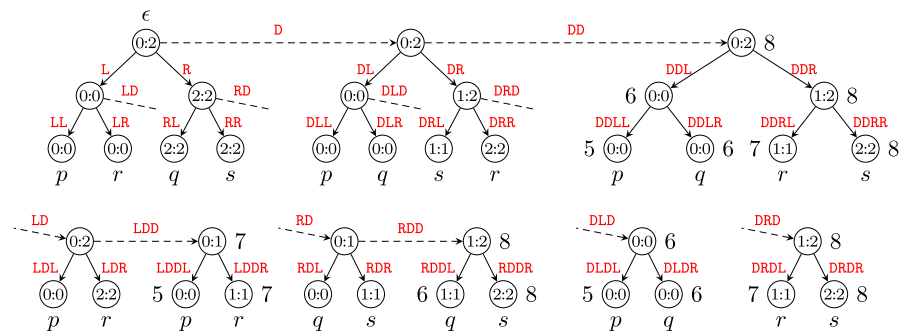


Fig. 2 Part of the range tree for the points from Fig. 1. The label of node x appears in red on the arrow pointing to x . Nodes contain $l[x]:r[x]$. The references $L[x]$ and $R[x]$ appear as children in a binary tree using usual drawing conventions. The reference $D[x]$ appears as a dashed arrow (possibly interrupted); the placement on the page follows no other logic than economy of layout and readability. References $D[x]$ from leaf nodes, such as $D[LL]$ 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, $P_{DR} = \{r, s\}$. However, note that these point sets are *not* explicitly stored in the data structure

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 $\binom{a+0}{0} + \dots + \binom{a+b}{b} = \binom{a+b+1}{b}$ repeatedly, we compute this number as

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

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

The bound follows from aggregating this contribution over all $p \in P$. In summary, the running time becomes

$$\sum_{x \in X} d|P_x| \leq d \sum_{p \in P} d \binom{d+h}{d} \leq nd^2 B(n, d).$$

□

The running time in the above lemma can be improved with some effort to $O(d^2 n \log n + dnB(n, d))$, but this would not affect our overall results.

Search.

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

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

Algorithm Q(Query). Given integer $i \in \{1, \dots, d\}$, 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 $j \in \{1, \dots, i-1\}$, this algorithm returns $\bigoplus \{f(q) : q \in Q \cap B\}$.

- 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 $l_j \leq p_j \leq r_j$ for all $j \in \{1, \dots, i-1\}$ for each $p \in P_x$. Then the query algorithm on input x and i returns $f(B \cap P_x)$.

Proof We use 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_{xD} is $i + 1$, and $D[x]$ store the $(i + 1)$ th range tree for P_{xD} . Thus, by induction the algorithm returns $f(P_{xD} \cap B)$, which equals $f(P_x \cap B)$ because $P_{xD} = P_x$.

Otherwise, by induction, $f_L = f(P_{xL} \cap B)$ and $f_R = f(P_{xR} \cap B)$. Since $P_{xL} \cup P_{xR} = P_x$, we have $f(P_x \cap B) = f((P_{xL} \cap B) \cup (P_{xR} \cap P)) = f_L \oplus f_R$. \square

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

$$2^d \binom{h + d}{d} \quad (d \geq 1, h \geq 0). \tag{8}$$

We will show by induction in d that (8) is the correct bound for every call to a d -dimensional range tree for a point set P_x , where $h = \lceil \log |P_x| \rceil$. In the base case, for $d = 1$, we need to show that the number of visited nodes is at most $2^1 \binom{h + 1}{1} = 2(h + 1)$ for any height h . But this is just the standard observation that interval searching amounts to traversing two root–leaf paths, each of which contains at most h non-root nodes, and that the height of balanced binary search tree on n values is at most $1 + \lceil \log n \rceil$.

We now carefully consider the case for $d \geq 2$. 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 (8). Step Q3 leads to a recursive call for a $(d - 1)$ -dimensional range tree over the same point set $P_{xD} = P_x$, and we verify for $h \geq 1$:

$$\begin{aligned} 1 + 2^{d-1} \binom{h + d - 1}{d - 1} &\leq 2^{d-1} \binom{h + d - 1}{d} + 2^{d-1} \binom{h + d - 1}{d - 1} \\ &= 2^{d-1} \binom{h + d}{d} < 2^d \binom{h + d}{d}. \end{aligned}$$

The case $h = 0$ (i.e., P_x is a singleton) is immediate. 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 Sect. 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_{yR} satisfies $l_i \leq \min_i(P_{yR}) \leq r_i$, 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

$$\begin{aligned}
 h + \sum_{i=0}^{h-1} 2^{d-1} \binom{d-1+i}{d-1} &\leq 2^{d-1} \sum_{i=0}^h \binom{d-1+i}{d-1} = 2^{d-1} \sum_{i=0}^h \binom{d-1+i}{i} \\
 &= 2^{d-1} \binom{d+h}{h},
 \end{aligned}$$

where the inequality uses the bound $h \leq 2^{d-1} \binom{d-1+h}{d-1}$, which is immediate for $d \geq 2$ and $h \geq 0$. The total number of nodes visited is at most twice that value, and therefore bounded by (8). □

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 [16], which also contains combinatorial arguments of how to interpret the binomial coefficients in the context of spatial data structures. A later paper of Chan [8] also takes the recurrences as a starting point, and observes asymptotically improved solution for the related question of dominance queries.

4 Graph Distances

4.1 Preliminaries

We consider an undirected graph G whose edges have nonnegative integer weights. The set of vertices is $V(G)$ and has size n . For a vertex subset U we write $G[U]$ for the induced subgraph. The neighbourhood $N(v)$ of v are the vertices that share an edge with v .

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 length of a shortest u, v -path, i.e., the minimum of $l(P)$ over all u, v -paths P . The *Wiener index* of G , denoted $wien(G)$ is $\sum_{u,v \in V(G)} d(u, v)$. The *eccentricity* of a vertex u , denoted $e(u)$ is given by $e(u) = \max\{d(u, v) : v \in V(G)\}$. The *diameter* of G , denoted $diam(G)$ is $\max\{e(u) : u \in V(G)\}$. The *radius* of G , denoted $rad(G)$ is $\min\{e(u) : u \in V(G)\}$.

4.2 Separated Eccentricities

We follow the construction of [7].

Given a 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,

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

In particular, $e(u) = e(u, V(G))$ and $e(u) = \max\{e(u, X), e(u, Y)\}$ if $X \cup Y = V(G)$.

A vertex subset Z *separates* X and Y if every x, y -path with $x \in X$ and $y \in Y$ and $x \neq y$ contains a vertex from Z .

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

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

See Fig. 3 for a small example.

Lemma 9 *If Z separates X and Y then $e(x, Y) = \max_{i=1}^k e_i(x, Y)$ for $x \in X$.*

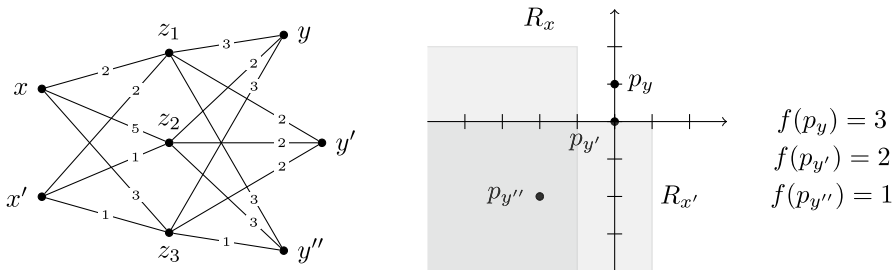


Fig. 3 Left: Example with $Z = \{z_1, z_2, z_3\}$ and $Y = Z \cup \{y, y', y''\}$. We have $e(x, Y) = 5$ (along xz_1y) and $e(x', Y) = 3$. For the case $i = 3$ we see $e_3(x, Y) = 4$ along xz_3y'' , because there are no shortest paths from x via z_3 to y or y' , and the one-edge path xz_3 itself is shorter. Similarly, $e_3(x', Y) = 3$ (along $x'z_3y'$). Middle: The corresponding points in \mathbb{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 $e_3(x', Y) = l(x'z_3y') = d(x', z_3) + f(p_{y'}) = 1 + 2 = 3$

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 $j \in \{1, \dots, k\}$ from the definition. \square

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

Lemma 10 *Let $i \in \{1, \dots, k\}$ and assume $\{z_1, \dots, z_k\}$ separates X and Y . We will define a set of points $\{p_y : y \in Y\}$ and a function f on this set as follows. Define for each $y \in Y$ the k -dimensional point*

$$p_y = \begin{pmatrix} d(z_i, y) - d(z_1, y) \\ \vdots \\ d(z_i, y) - d(z_k, y) \end{pmatrix} \quad \text{with } f(p_y) = d(z_i, y). \tag{9}$$

Define for each $x \in X$ the rectangle

$$R_x = I_1 \times \dots \times I_k, \quad \text{where } I_j = [-\infty, d(x, z_j) - d(x, z_i)]. \tag{10}$$

Then

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

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

$$d(x, z_i) + d(z_i, y) \leq d(x, z_j) + d(z_j, y), \quad \text{for all } j \in \{1, \dots, k\},$$

equivalently,

$$d(z_i, y) - d(z_j, y) \leq d(x, z_j) - d(x, z_i), \quad \text{for all } j \in \{1, \dots, k\}. \tag{11}$$

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 . \square

We note that the i th 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 the possible (but tedious) improvement that arises from this observation.

We have arrived at the following algorithm:

Algorithm S (Separated Eccentricities). *Given an undirected, connected graph G with nonnegative integer weights and vertex subsets X and Y such that $V(G) = X \cup Y$ and a separator Z of size k , this algorithm computes the eccentricity $e(x, Y)$ of every vertex $x \in X \cup Z$.*

- S1** [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 $z \in Z$.
- S2** [Build range trees.] For each $i \in \{1, \dots, k\}$, construct a k -dimensional range tree for the points $\{p_y : y \in Y\}$ given by (9) using the monoid (\mathbf{Z}, \max) .
- S3** [Query range trees.] For each $x \in X$ and for each $i \in \{1, \dots, k\}$ query the i th range tree for the rectangle R_x given by (10) and add $d(x, z_i)$. The result is $e_i(x, Y)$ by Lemma 10. Set $e(x, Y) = \max_{i=1}^k e_i(x, Y)$.

Algorithm S is correct by the observation in Step S3 (based on Lemma 10) and Lemma 9.

Lemma 11 *Algorithm S runs in time $O(km \log n + n2^k B(n, k))$.*

Proof The first term accounts for Step S1. Using Lemma 2, we see that Steps S2 and S3 take time $O(|Y|k^2 \cdot B(|Y|, k))$ and $O(|X|2^k \cdot B(|Y|, k))$ for each $i \in \{1, \dots, k\}$. Since $k^2 = O(2^k)$ and both $|Y|$ and $|X|$ are at most n , both expressions are asymptotically dominated by the second term. \square

4.3 Parameterization by Vertex Cover Number

Graphs with small vertex cover number allow for a particularly simple application of the construction from Sect. 4.2, because the same small separator (namely, the vertex cover itself) separates every vertex from the rest of the graph.

A *vertex cover* is a vertex subset C of $V(G)$ such that every edge in G has at least one endpoint in C . The smallest k for which a vertex cover of size k exists is the *vertex cover number* of a graph, denoted $\text{vc}(G)$. The number of edges in such a graph is at most $n \cdot \text{vc}(G)$.

Proof of Theorem 1.1, distances Set $X = V(G) - C$, $Y = V(G)$, and $Z = C$. Clearly, every vertex x is separated from all vertices by its neighbourhood $N(x)$. Since C is a vertex cover, $N(x) \subseteq C$ for all $x \notin C$. Thus, Z separates X and Y . Note that we have $e(x) = e(x, Y)$, so it suffices to run algorithm S to compute all eccentricities. The running time is immediate from Lemma 11 with Z of size $k \leq \text{vc}(G)$ and $m \leq kn$. From the eccentricities, the radius and diameter can be computed in linear time using their definition. \square

4.4 Parameterization by Treewidth

In the more general case, not all paths from x need to pass through the separator Z . Therefore we cannot determine $e(x)$ from $e(x, Y)$ alone, as we did in Sect. 4.3. Instead, since $e(x) = \max\{e(x, X), e(x, Y)\}$, it remains to compute $e(x, X)$. However, $e(x, X)$ is entirely determined by the subgraph $G[X]$ (once we add shortcuts inside Z), so we can handle this recursively. The necessary recursive decomposition is provided by a tree decomposition. We need the approximate treewidth construction

of Bodlaender et al. [6]. The analysis of the resulting recurrence for superconstant dimension follows from Abboud et al. [1].

We need a decomposition from [7]. Let $k + 1 < n$. A skew k -separator tree T of an n -vertex graph 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

$$\frac{n}{k + 1} \leq |L_t \cup Z_t| \leq \frac{nk}{k + 1}, \tag{12}$$

- 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 12 ([7, Lemma 3] with [6, Theorem 1]) *For a given n -vertex input graph G , a skew $(5\text{tw}(G) + 4)$ -separator tree can be computed in time $n \exp O(\text{tw}(G))$.*

We are ready for the algorithm.

Algorithm E (Eccentricities). *Given an undirected, connected graph G with non-negative 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 $n / \ln n < 3k(k + 1)$ find all distances using Dijkstra’s algorithm. Terminate.
- E2** [Find $e(x, Y)$] Compute $e(x, Y)$ for all $x \in X$ using algorithm S.
- 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** [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 $x \in X$. Set $e(x) = \max\{e(x, X), e(x, Y)\}$.
- E5** [Flip.] Repeat Steps E2–4 with the roles of X and Y exchanged.

Lemma 13 *The running time of Algorithm E is $O(n \cdot B(n, k) \cdot 2^k \log n)$.*

Proof Assume $n \geq 8$. Let $T(n, d)$ denote the running time of Algorithm E.

The graph G has treewidth $O(k)$, so it has $O(nk)$ edges. Step E1 consists of n executions of Dijkstra’s algorithm with n bounded by $O(k^2 \log k)$. This takes time $O(k^5 \log^2 k)$, which is bounded by $O(2^k)$. Step E3 was analysed in Lemma 11 and

takes time $O(n2^k B(n, k))$. Accounting for the recursive calls in Step E4 for both X and Y using $|Y| \leq n - |X| + k$, we arrive at the divide-and-conquer recurrence

$$T(n, k) = \begin{cases} O(k^5 \log^2 k), & \text{if } n / \ln n < 4k(k + 1); \\ n \cdot S(n, k) + T(|X|, k) + T(n - |X| + k, k), & \text{otherwise,} \end{cases}$$

for some non-decreasing function S satisfying $S(n, k) = O(2^k B(n, k))$. We would expect this recurrence to solve to roughly $S(n, k) \cdot n \log n$ if the partition were perfectly balanced and k were constant, but the dependence on k is not clear, so we give a careful analysis.

The lemma is implied by the bound

$$T(n, k) \leq 3(k + 1) \cdot S(n, k) \cdot n \ln n, \tag{13}$$

which we will show by strong induction in n for all k . Write $s = |X|$ and $r = n - s + k$. By induction, we can bound

$$\frac{T(s, k) + T(r, k)}{3(k + 1)} \leq S(s, k) \cdot s \ln s + S(r, k) \cdot r \ln r \leq S(n, k) \cdot (s \ln s + r \ln r). \tag{14}$$

From the bounds (12) on s we have $s \leq nk / (k + 1)$ and $r \leq n - n / (k + 1) + k = (nk / (k + 1)) + k$, so if we set

$$t = \frac{nk}{k + 1} + k,$$

then both $s \leq t$ and $r \leq t$. Thus, we can get rid of s and r in the last term of (14) as

$$s \ln s + r \ln r \leq s \ln t + r \ln t = s \ln t + (n - s + k) \ln t = n \ln t + k \ln t \leq n \ln t + k \ln n. \tag{15}$$

Step E1 ensures $k(k + 1) \leq n / (3 \ln n) \leq \frac{1}{3}n$, so we get

$$t = \frac{nk}{k + 1} + \frac{k(k + 1)}{k + 1} \leq n \cdot \frac{k + \frac{1}{3}}{k + 1}.$$

Using the bound $\ln y \leq y - 1$ for $y > 0$, we have

$$\ln t \leq \ln n + \ln \left(\frac{k + \frac{1}{3}}{k + 1} \right) \leq \ln n + \left(\frac{k + \frac{1}{3}}{k + 1} - 1 \right) = \ln n - \frac{2}{3(k + 1)}. \tag{16}$$

Using this in (15), and then going back to (14), we have

$$\frac{T(s, k) + T(r, k)}{3(k + 1) \cdot S(n, k)} \leq n \ln t + k \ln n \leq n \ln n - \frac{2n}{3(k + 1)} + k \ln n \leq n \ln n - \frac{n}{3(k + 1)},$$

where the last step uses $k \ln n \leq n / (3(k + 1))$, which is ensured by Step E1. Returning to the recurrence, we can now verify

$$T(n, k) = n \cdot S(n, k) + T(r, k) + T(s, k) \leq n \cdot S(n, k) + n \cdot S(n, k) \cdot (3(k + 1) \ln n - 1),$$

which simplifies to (13). □

We can now establish Theorem 1:

Proof of Thm. 1.2, distances To compute all eccentricities for a given graph, we find a k -skew separator for $k = 5\text{tw}(G) + 4$ using Lemma 12 in time $n \exp O(\text{tw}(G))$. We then run Algorithm E, using Lemma 13 to bound the running time. From the eccentricities, the radius and diameter can be computed in linear time using their definition. □

4.5 Extension to Wiener Index

Algorithm E can be modified to compute the Wiener index, as described in [7, Sec. 4], completing the proof of Theorem 1. Instead of repeating those arguments, we content ourselves here with pointing out the necessary modifications in our presentation.

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 $e(x, Y)$. Such a query can be handled with our data structure and a more careful choice of monoid, but another technical issue appears. While the distance maxima satisfied $e(x, Y) = \max\{e_1(x, Y), \dots, e_k(x, Y)\}$ according to Lemma 9, no similar expression holds for distance sums. This is simply because there can be shortest x, y -paths via two different z_i , and their contribution would lead to overcounting. The solution is to associate the distance $d(x, y)$ with exactly one $i \in \{1, \dots, k\}$, namely the smallest i for which a shortest x, y -path passes through z_i .

This leads to the following (somewhat laborious) construction. For $x \in X$, partition $Y = Y_1 \cup \dots \cup Y_k$ into disjoint sets such that $y \in Y_i$ if and only if (i) there is a shortest x, y -path through z_i and (ii) there is none through z_j for $j < i$. Then define $s_i(x, Y)$ as the sum of distances from x to Y_i :

$$s_i(x, Y) = \sum_{y \in Y_i} d(x, y_i).$$

We observe that $s_1(x, Y) + \dots + s_k(x, Y)$ is the sum of distances from x to all vertices in Y .

To compute $s_i(x, Y)$, we modify the construction from Lemma 10 slightly. The coordinates of p_y are as before. The rectangle R_x associated with x for $i \in \{1, \dots, k\}$ now becomes $[-\infty, r_1] \times \dots \times [-\infty, r_k]$ where

$$r_j = \begin{cases} d(x, z_j) - d(x, z_i) - 1, & j < i; \\ d(x, z_j) - d(x, z_i), & j \geq i. \end{cases}$$

Following the proof of Lemma 10, the ‘ -1 ’ above for $j < i$ ensures that $p_y \in R_x$ now also requires

$$d(x, z_i) + d(z_i, y) < d(x, z_j) + d(z_j, y) \quad (i < j).$$

In other words, x, y -paths through z_j for $j < i$ cannot be shortest paths, so we have avoided overcounting. The domain of the function f in (9) is changed to the monoid of positive integer tuples (a, b) with the operation $(a, b) \oplus (a', b') = (a + a', b + b')$ with identity element $(0, 0)$. The value associated with vertex p_y is changed to $f(p(y)) = (1, d(z_i, y))$.

It then holds that the query for R_x will return a tuple (N, S) where

$$N = |Y_i| \text{ and } S = \sum_{y \in Y_i} d(z_i, y) \quad (17)$$

so that $s_i(x, Y)$ can be computed as

$$s_i(x, Y) = N \cdot d(x, z_i) + S.$$

These changes suffice to establish the Wiener index part of Theorem 1.1, finishing the argument for vertex cover.

To extend these results to the recursive construction for treewidth from Sect. 4.4 now only requires some delicacy regarding how sums of distances cross the separator. This part is carefully argued in [7, Lemma 8], and there is no reason to repeat it here. With the these changes, Theorem 1.2 is established.

Acknowledgements Open access funding provided by Lund University. We thank Amir Abboud and Rasmus Pagh for useful discussions.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

Unweighted Graphs with Small Vertex Cover

Eccentricities and Wiener Index

In a graph with vertex cover C , all paths from $v \notin C$ have their second vertex in C . If the graph is unweighted, two vertices $v, w \notin C$ with $N(v) = N(w)$ have the same distances to the rest of the graph. Since $N(v) \subseteq C$ it suffices to consider all 2^k many subsets of C . The details are given in Algorithm U.

Algorithm U (Unweighted graph). *Given a connected, unweighted, undirected graph G and a vertex cover C , this algorithm computes the eccentricity of each vertex and the Wiener index.*

- U1** [Initialise.] Set $wien(G) = 0$. Insert each $v \notin C$ into a dictionary D indexed by $N(v)$.
- U2** [Distances from C .] For each $v \in C$, perform a breadth-first search from v in G , computing $d(v, u)$ for all $u \in V(G)$. Let $e(v) = \max_u d(v, u)$ and increase $wien(G)$ by $\frac{1}{2} \sum_u d(v, u)$.
- U3** [Distances from $V(G) - C$.] Choose any $v \in D$. Perform a breadth-first search from v in G , computing $d(v, u)$ for all $u \in V(G)$. For each $w \in D$ with $N(w) = N(v)$ (including v itself), let $e(w) = \max_u d(v, u)$, increase $wien(G)$ by $\frac{1}{2} \sum_u d(v, u)$, and remove w from D . Repeat step U3 until D is empty.

Theorem 14 *The eccentricities and Wiener index of an unweighted, undirected, connected n -vertex graph and vertex cover number k can be computed in time $O(nk2^k)$.*

Proof It is well known that a minimum vertex cover can be computed in the given time bound [11].

For the running time of Algorithm U, we already observed that for each $v \notin C$ the neighbourhood $N(v)$ is entirely contained in C . Thus, there are only 2^k different neighbourhoods used as an index to D and we can bound the number of BFS computations in Step U3 by 2^k . (Step U2 incurs another k such computations.) The number of edges in a graph with vertex cover number k is $O(nk)$. Assuming constant-time dictionary operations, the running time of the algorithm is therefore $O(nk2^k)$.

To see correctness, we need to argue that the distances computed for $w \in D$ in Step 4 are correct. First, to argue $d(v, z) = d(w, z)$ for all $z \notin \{v, w\}$ consider shortest paths $P = v v_2 \dots z$ and $Q = w w_2 \dots z$, possibly with $v_2 = z$ or $w_2 = z$. The suffix $P' = v_2 \dots z$ is itself a shortest path, of length $l(P) - 1$. Since $N(v) = N(w)$, the path wP' exists and is a shortest w, z -path as well, and therefore of length $l(Q)$. We conclude that $l(Q) = 1 + l(P') = 1 + l(P) - 1 = l(P)$.

It is *not* true that $d(v, z) = d(w, z)$ for $z \in \{v, w\}$. Instead, we have $d(v, w) = d(w, v)$ (both equal 2) and $d(v, v) = d(w, w)$ (both equal 0). Thus, the contributions from v and w to W are the same, and the sets $d(v, \cdot)$ and $d(w, \cdot)$ have the same maxima. \square

One way of implementing the neighbourhood-indexed dictionary within the required time bounds (and without using randomised hashing) is as trie of height k whose 01-labelled root-leaf paths describe (the binary representation of) subsets of C ; at the leaves we store vertices in a linked list. Insertion requires k operations and happens n times in U1. The iteration over all vertices in U3 can be performed by traversing all the lists at all the leaves, in constant time per vertex. This establishes the second part of Theorem 4.

Faster Eccentricities

Vertex cover number is an extremely well studied parameter, so it makes sense to look for further algorithmic improvements. The best current algorithm for *finding* a vertex cover runs in time $O(nk + 1.274^k)$ [10], so the bound in Theorem 14

is dominated by the distance computation. Thus it may make sense to look for distance computation algorithms with running times of the form $nk + g(k)$ rather than $m \cdot g(k)$. We now present such an algorithm for eccentricities. (It does not work for the Wiener index.)

First, we observe that if C is a vertex cover, then no path can contain consecutive vertices from $V(G) - C$. Thus, we can modify the graph by inserting length-2 shortcuts between nonadjacent vertices in C that share a neighbour without changing the pairwise distances in the graph. We can now run Dijkstra restricted to the subgraph $G[C \cup \{v\}]$, noting that the second layer of the shortest path tree is contained in $N(v)$, which is contained in C . Thus, the number of such computations that are different is bounded by 2^k , the number of neighbourhoods. The eccentricity $e(v)$ can be derived from the shortest path tree as follows. Let $E(v)$ denote the *eccentric* vertices from v in C , i.e., the vertices belonging to C at distance $\max_{u \in C} d(v, u)$. Note that $E(v)$ contains exactly the vertices at the deepest layer of the shortest path tree from v in $G[C \cup \{v\}]$. The only vertices u in G that can be farther away from v than $E(v)$ must have their entire neighbourhood $N(u)$ contained in $E(v)$. See Fig. 4.

The only confusion arises if the only such vertex is v itself. To handle these details we need to determine, for each cover subset $S \subseteq C$, if the number of u with $N(u) \subseteq S$ is 0, 1, or more. This can be solved by a fast zeta transform in time $2^k k$, see [4], or more directly as follows. For each $S \subseteq C$, let

$$h(S) = \begin{cases} \{w\}, & \text{if } N(w) \subseteq S \text{ for exactly one } w \notin C; \\ \emptyset, & \text{if there is no } w \notin C \text{ with } N(w) \subseteq S; \\ C, & \text{otherwise.} \end{cases}$$

(The third value is an arbitrary placeholder.) Then $h(S)$ can be computed for all $S \subseteq C$ in a bottom-up fashion.

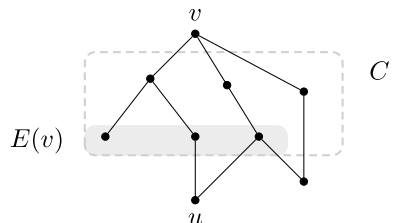
The details are given in the following algorithm.

Algorithm F (Faster Eccentricities Parameterized by Vertex Cover). *Given a connected, unweighted, undirected graph G and a vertex cover C , this algorithm computes the eccentricity of each vertex.*

F1 [Initialise.] Insert each $v \in V(G) - C$ into a dictionary D indexed by $N(v)$. Set $h(S) = \emptyset$ for all $S \subseteq C$.

F2 [Compute h .] For each $u \notin C$, set $h(N(u)) = \{u\}$ if $h(N(u)) = \emptyset$, otherwise $h(N(u)) = C$. For each nonempty subset $S \subseteq C$ in increasing order of size,

Fig. 4 The distance from v to u is 3 and $N(u) \subseteq E(v)$



compute $W = h(S) \cup \bigcup_{w \in S} h(S - \{w\})$. If $|W| > 1$ then set $h(S) = C$. Else set $h(S) = W$.

F3 [Shortcuts.] For each pair of covering vertices $u, v \in C$, if $uv \notin E(G)$ but u and v share a neighbour outside C , add the edge uv to $E(G)$ with length 2.

F4 [Eccentricities from C .] For each $v \in C$, compute shortest distances in $G[C]$ from v . Set $d = \max_{u \in C} d(v, u)$ and let $E(v)$ denote the vertices in C at distance d . Let

$$e(v) = \begin{cases} d + 1, & \text{if } h(E(v)) - \{v\} \neq \emptyset, \text{ [equivalently, } E(v) \supseteq N(w) \text{ for some } w \neq v \text{];} \\ d, & \text{otherwise.} \end{cases}$$

F5 [Eccentricities from $V(G) - C$.] For each $v \in D$, compute shortest distances in $G[C \cup \{v\}]$ from v . Set $d = \max_{w \in C} d(v, w)$ and let $E(v)$ denote the vertices in C at distance d . For each $u \in D$ (including v itself) with $N(u) = N(v)$ [and therefore $E(u) = E(v)$] let

$$e(u) = \begin{cases} d + 1, & \text{if } h(E(u)) - \{u\} \neq \emptyset, \text{ [equivalently, } E(u) \supseteq N(w) \text{ for some } w \neq u \text{];} \\ d, & \text{otherwise,} \end{cases}$$

and remove u from D .

Theorem 15 *The eccentricities of an unweighted, undirected, connected graph with n vertices and vertex cover number k can be computed in time $O(nk + 2^k k^2)$.*

Proof If G has only one or two vertices, we calculate all distances naïvely. Otherwise, we run Algorithm F .

Step F1 needs to visit every of the nk edges. The first part of Step F2 visits at most $n - k$ vertices and spends time $O(k)$ at each. There are 2^k subsets of C , bounding the running time of the second part of Step F2 to $O(2^k k)$. Step F3 can be performed in time $O(2^k k^2)$ (instead of the obvious $O(nk^2)$) by iterating over $w \in D$ and all pairs $u, v \in N(w)$. The shortest path computations in Steps F4 and F5 take time $O(k^2)$ each using Dijkstra’s algorithm, for a total of $O(2^k k^2)$. The dictionary contains at most n values, so the total time of Step F4 and F5 is $O(n + 2^k k^2)$, assuming that the dictionary allows iteration over the vertices $u \in D$ with $N(u) = N(v)$ with constant overhead.

To see correctness, assume that we already performed the shortcut operation in Step F3. We argue for correctness of Step F5, Step F4 is similar. Let $u \in V(G)$ and consider an eccentric vertex z and a shortest u, z -path P . We want to show that the algorithm sets $e(u)$ to the length of P .

First assume there exists such an eccentric vertex z belonging to C . Then $z \in E(u)$. Moreover, there can be no vertex $w \neq u$ with $N(w) \subseteq E(u)$, because otherwise w would have a shortest path from u through $E(u)$ and therefore be farther away than z . Thus, Step F5 picks the second branch and correctly sets $e(u)$ to $d = d(u, z)$.

Otherwise, none of the eccentric vertices z belong to C . There are a number of cases. If P is just the edge uz then every vertex in G has distance at most 1 to u . If G is a star then $C = N(z) = E(u) = \{u\}$ and $d = 0$. Moreover, some third vertex w has its neighbourhood (namely, $\{u\}$) completely contained in $E(u)$ so Step F5 correctly

sets $e(u) = d + 1 = 1$. If $P = uz$ and G is not a star then it must contain a triangle, so $|C| > 1$ (possibly containing u) and the vertices in $E(u)$ (which could be many) are all at distance 1. Moreover, there cannot exist $w \neq u$ with $N(w) \subseteq E(u)$ because then there would be a shortest u, w -path of length 2. Thus, Step F5 correctly sets $e(u)$ to $d = 1$.

The remaining case is when the u, z -path P with $z \notin C$ contains at least 3 vertices. Let w denote the penultimate vertex, so the P is of the form $u \cdots wz$. Since $z \notin C$, we have $w \in C$. The distance from u to w is therefore $d = \max_{w \in C} d(u, w)$. It remains to argue that the assignment to $e(u)$ in Step F5 picks the second branch. Consider the neighbourhood $N(z)$. Every neighbour x of z must belong to C . Moreover, its distance $d(u, x)$ is d , because if it were longer then w would not belong to $E(u)$, if it were shorter then there would be a shorter path $u \cdots xz$ than P . Therefore $x \in E(u)$. Thus, we have established that $N(z) \subseteq E(u)$, so we conclude that Step F5 correctly sets $e(u)$ to $d + 1$. \square

This establishes the first part of Theorem 4.

References

1. Abboud, A., Williams, V.V., Wang, J.R.: 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). <https://doi.org/10.1137/1.9781611974331.ch28>
2. Bentert, M., Nichterlein, A.: Parameterized complexity of diameter. *CoRR*, arXiv:1802.10048 (2018)
3. Bentley, J.L.: Multidimensional divide-and-conquer. *Commun. ACM* **23**(4), 214–229 (1980). <https://doi.org/10.1145/358841.358850>
4. Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion-exclusion. *SIAM J. Comput.* **39**(2), 546–563 (2009)
5. Blum, M., Floyd, R.W., Pratt, V.R., Rivest, R.L., Tarjan, R.E.: Time bounds for selection. *J. Comput. Syst. Sci.* **7**(4), 448–461 (1973). [https://doi.org/10.1016/S0022-0000\(73\)80033-9](https://doi.org/10.1016/S0022-0000(73)80033-9)
6. Bodlaender, H.L., Drange, P.G., Dregi, M.S., Fomin, F.V., Lokshtanov, D., Pilipeczuk, M.: An $O(c^k n)$ 5-approximation algorithm for treewidth. *SIAM J. Comput.* **45**(2), 317–378 (2016). <https://doi.org/10.1137/130947374>
7. Cabello, S., Knauer, C.: Algorithms for bounded treewidth with orthogonal range searching. *Comput. Geom.* **42**(9), 815–824 (2009). <https://doi.org/10.1016/j.comgeo.2009.02.001>
8. Chan, T.M.: All-pairs shortest paths with real weights in $O(n^3 / \log n)$ time. *Algorithmica* **50**(2), 236–243 (2008). <https://doi.org/10.1007/s00453-007-9062-1>
9. Chazelle, B.: Lower bounds for orthogonal range searching: I. The reporting case. *J. Assoc. Comput. Mach.* **37**(2), 200–212 (1990). <https://doi.org/10.1145/77600.77614>
10. Chen, J., Kanj, I.A., Jia, W.: Vertex cover: Further observations and further improvements. *J. Algorithms* **41**, 280–301 (2001). <https://doi.org/10.1006/jagm.2001.1186>
11. Downey, R.G., Fellows, M.R.: *Parameterized Complexity*. Springer, New York (1999)
12. Husfeldt, T.: 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), pp. 16:1–16:11, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2017). <https://doi.org/10.4230/LIPIcs.IPEC.2016.16> (2017)
13. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.* **63**(4), 512–530 (2001). <https://doi.org/10.1006/jcss.2001.1774>

14. Lee, D.-T., Wong, C.K.: Quintary trees: a file structure for multidimensional database systems. *ACM Trans. Database Syst.* **5**(3), 339–353 (1980). <https://doi.org/10.1145/320613.320618>
15. Lueker, G.S.: A data structure for orthogonal range queries. In: 19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16–18 October 1978, pp 28–34. IEEE Computer Society (1978). <https://doi.org/10.1109/SFCS.1978.1>
16. Monier, L.: Combinatorial solutions of multidimensional divide-and-conquer recurrences. *J. Algorithms* **1**(1), 60–74 (1980). [https://doi.org/10.1016/0196-6774\(80\)90005-X](https://doi.org/10.1016/0196-6774(80)90005-X)
17. Roditty, L., Williams, V.V.: 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, pp. 515–524 (2013). <https://doi.org/10.1145/2488608.2488673>
18. Shi, Q.: Efficient algorithms for network center/covering location optimization problems. PhD thesis, School of Computing Science, Simon Fraser University (2008)
19. Willard, D.E.: New data structures for orthogonal range queries. *SIAM J. Comput.* **14**(1), 232–253 (1985). <https://doi.org/10.1137/0214019>
20. Williams, V.V.: 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, pp. 17–29 (2015). <https://doi.org/10.4230/LIPIcs.IPEC.2015.17>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Affiliations

Karl Bringmann¹ · Thore Husfeldt^{2,3}  · Måns Magnusson⁴

Karl Bringmann
kbringma@mpi-inf.mpg.de

Måns Magnusson
mans.magnusson.888@student.lu.se

¹ Max-Planck-Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

² BARC, IT University of Copenhagen, Copenhagen, Denmark

³ Lund University, Lund, Sweden

⁴ Department of Computer Science, Lund University, Lund, Sweden