Computing Graph Distances Parameterized by Treewidth and Diameter*

Thore Husfeldt

Lund University and IT University of Copenhagen, Lund, Sweden
thore.husfeldt@cs.lth.se

Abstract

We show that the eccentricity of every vertex in an undirected graph on n vertices can be computed in time $n \exp O(k \log d)$, where k is the treewidth of the graph and d is the diameter. This means that the diameter and the radius of the graph can be computed in the same time. In particular, if the diameter is constant, it can be determined in time $n \exp O(k)$. This result matches a recent hardness result by Abboud, Vassilevska Williams, and Wang [SODA 2016] that shows that under the Strong Exponential Time Hypothesis of Impagliazzo, Paturi, and Zane [J. Comp. Syst. Sc., 2001], for any $\epsilon > 0$, no algorithm with running time $n^{2-\epsilon} \exp o(k)$ can distinguish between graphs with diameter 2 and 3.

Our algorithm is elementary and self-contained.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Graph algorithms, diameter, treewidth, Strong Exponential Time Hypothesis

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.16

1 Introduction

In an undirected graph, the *eccentricity* of a vertex is its largest distance to another vertex. The graph's *diameter*, denoted $\operatorname{diam} G$, is the largest eccentricity of any of its vertices. The graph's *radius*, denote $\operatorname{rad} G$, is the smallest eccentricity of any its vertices. The *treewidth* is a well-studied sparseness measure of graphs. These are fundamental parameters that permeate both the design of graph algorithms and the analysis of networks in many scientific domains.

We show the following result:

▶ **Theorem 1.** *Given an undirected n-vertex graph G with integer weights. If G has diameter $\operatorname{diam} G$ and treewidth k, then we can compute the eccentricity of every vertex, and compute $\operatorname{diam} G$ and $\operatorname{rad} G$, in time $n \exp O(k \log \operatorname{diam} G)$.*

For constant $\operatorname{diam} G$ this matches a recent lower bound by Abboud, Vassilevska Williams, and Wang [1] under the Strong Exponential Time Hypothesis of Impagliazzo, Paturi, and Zane [6]. In particular, it settles the complexity of the very simple question of deciding if a given undirected, unweighted graph has diameter 2 or 3.

1.1 Related work

It is easy to see that the diameter of an unweighted graph can be computed in time $O(nm)$ by computing the eccentricity of each node using breadth first search. For sparse graphs with

* This work was done at the Simons Institute for the Theory of Computation at UC Berkeley, partially supported by Grant 2012–4730 from the Swedish Research Council.

$m = O(n)$ this running time becomes $O(n^2)$, a bound that seems difficult to improve. Recent work by Roditty and Vassilevska Williams [8] has provided an illuminating explanation for this phenomenon: An algorithm for computing the diameter in time $O(m^{2-\epsilon})$ would violate the Strong Exponential Time Hypothesis.

However, it is also clear that for some *very* sparse graphs, this bound can be broken. In particularly, the diameter of a tree can be computed in linear time as follows. From any node find a remotest node u using breadth first search. Then find a remotest node v from u using breadth first search again. Elementary arguments show that $\operatorname{diam} G = \operatorname{dist}(u, v)$.

A useful parameter for studying this phenomenon is *treewidth.* The treewidth of a graph is a well-studied measure of its sparsity, properly defined in Section 2.4. In the extremes, every tree has treewidth 1 and the n-clique has treewidth $n - 1$.

In this framework, we can ask how the complexity of computing the diameter deteriorates with treewidth. For example, can the problem be solved in time $O(n \log n)$ on graphs of logarithmic treewidth? Very surprisingly, Abboud, Vassilevska Williams, and Wang [1] showed that not only can the complexity of the parameter be disentangled from the number of vertices in the sense of parameterised complexity, but this dependency is at least *exponential*: They show that under the Strong Exponential Time Hypothesis, it takes time

$$n^{2-\epsilon} \exp \Omega(k) \qquad \text{for any } \epsilon > 0 \tag{1}$$

to compute the diameter of an n-vertex graph with treewidth k. The same bound holds for computing the radius, but under a stronger hypothesis called the Hitting Set Conjecture. See [1] for a thorough presentation and discussion of these results and their underlying hypotheses and a rich overview of related work.

This result is surprising because the innocuous diameter problem exhibits a similar dependency on treewidth as several NP-hard problems. For instance, under the same Strong Exponential Time Hypothesis, it is known that the NP-hard Independent Set problem cannot be solved in time $(2 - \epsilon)^k \operatorname{poly} n$ for any $\epsilon > 0$ [7]. Of course, the lower bound (1) does not imply that the diameter problem is an exponential-time problem. This is because, unlike for Independent Set, the exponential dependency does not persist throughout all dependencies of k on n. In this case, it becomes vacuous for $k = \Omega(\log n)$, where the quadratic-time algorithm takes over.

The lower bound holds even for very restricted diameter problems, such as deciding if the diameter is 2 or 3, and (consequently) for approximating the solution. The same authors provide an algorithm for computing the diameter with running time

$$O(k^2 n \log^k n) = n^{1+o(1)} \exp O(k \log k)\,. \tag{2}$$

That algorithm follows a method introduced by Cabello and Knauer [3] for computing the Wiener index, based on a reduction to a k-dimensional orthogonal range query problem and its solution by Willard [9]. The authors explain how to extend this idea to provide algorithms for radius and eccentricities within the same time bound.

Closing the gap between (1) and (2) is viewed as a very interesting open problem [1].

Our contribution is to offer yet another parameter to this analysis, by introducing a dependency on the diameter to the running time. In particular, we match the lower bound of Abboud *et al.* for the regime of constant diameters: If $\operatorname{diam} G = O(1)$ then the complexity of these problems is (under various hypotheses) $n \exp \Theta(k)$.

Planar graphs of constant diameter have constant treewidth, so for that class of graphs the diameter can be found in linear time, which is a known result due to Eppstein [5].

Our algorithm is elementary, in particular compared to data structure results leveraged to establish (2). Instead of following Cabello and Knauer, we demonstrate that the necessary information can be built by traversing the tree decomposition a number of times in different directions. Once the dependency of the problem on the diameter has been realised, the construction is unsurprising. However, the argument is quite fragile and ultimately relies on a careful (but entirely combinatorial) analysis of shortest paths.

2 Algorithm

2.1 Preliminaries

A *walk* is an alternating sequence of vertices and edges, say $v_0, e_1, \ldots, e_l, v_r$, where $e_i = v_{i-1}v_i$ for $1 \leq i \leq l$. A walk with endvertices u and v is called a u,v-*walk*. A walk with no repeated vertices is a *path*. We denote a u,v-path by symbols like uPv, making the endvertices explicit for readability. The vertices on uPv except for u and v are called *internal*. The *length* of a walk uPv is the number of edges (with repetitions) and denoted $l(uPv)$. If x is a vertex on the u,v-path uPv then we write uPx for the u,x-subpath of uPv, and xPv for the x,v-subpath of uPv. Two paths uPv and vQw can be concatenated into the walk $uPvQw$ with the obvious interpretation.

Let $\operatorname{dist}(u,v)$ denote the *distance* between u and v in a connected graph G, which is the length of the shortest u,v-path; with the understanding that $\operatorname{dist}(u,u) = 0$. The *eccentricity* of vertex u, denoted $e(u)$, is $\max\{\operatorname{dist}(u,v) \mid v \in V(G)\}$. The *diameter* of G, denoted $\operatorname{diam} G$, is $\max\{e(u) \mid u \in V(G)\}$. The *radius* of G, denoted $\operatorname{rad} G$, is $\min\{e(u) \mid u \in V(G)\}$.

The treewidth of a graph is the width of an optimal *tree decomposition*, an auxiliary structure that maintains the structure of G in a sparse fashion. (See Section 2.4 for a full definition.) Our algorithm requires a tree decomposition as input, and we note that this is provided within the time bounds of our own algorithm by a recent result by Bodlaender *et al.* [2] that we can state as follows:

▶ **Theorem 2** (Bodlaender et al.). *Given a graph G with n vertices and treewidth k, a nice tree decomposition of G of width $O(k)$ and with $O(nk)$ nodes can be computed in time $n \exp O(k)$.*

We also need the fact that given such a tree decomposition, we can compute pairwise distances quickly for all vertices in the same piece. For the purposes of exposition, we abstract this to a more general result due to Chaudhuri and Zariolagis [4] that falls slightly short of the ambitions on Theorem 1, by a factor $\log n$. We show in Section 3 how to replace this result with an explicit and self-contained construction with a better bound so as to establish the bound in Theorem 1.

▶ **Theorem 3** (Chaudhuri and Zariolagis). *Let G denote an n-vertex graph given with a tree decomposition of width k. In time $O(k^3 n \log n)$ we can compute a data structure such that for any $u, v \in V(G)$ we can compute $\operatorname{dist}(u,v)$ in time $O(k^3)$.*

Chaudhuri and Zariolagis [4] present various trade-offs between construction and query time that need not interest us here; the important part is the subquadratic dependency on n in the construction time and polynomial dependency on the treewidth, so that the time to compute the distance between a pair of vertices becomes negligible. We note that in itself, the above structure brings us no closer to our goal: To compute the diameter, we still need to evaluate $\operatorname{dist}(u,v)$ for all pairs of vertices, in time $O(n^2k^3)$.

2.2 Distance profiles

Let R denote a fixed integer; think of R as the *range* of distances we want to keep track of, ideally $\operatorname{diam} G \leq R$.

We denote by π a partition (U, V, W) of $V(G)$, where $V = \{v_1, \dots, v_r\}$ is a separator in G separating U and W.

(We will arrive at these partitions as induced by neighbouring nodes in a tree decomposition. We need the next two lemmas when we traverse the decomposition in both directions, which is why we avoid the terms 'above' and 'below' in favour of 'left' and 'right'. However, the reader is encouraged to think of V as a piece in the tree decomposition, with U denoting the vertices 'below' it, and W those 'above.' We give a more general framework here in order to avoid the proliferation of ultimately similar arguments and tedious case analyses.)

We will pay special attention to paths whose internal vertices belong to U. In particular, for $u \in U \cup V$ and $v \in V$ a u, v-path is *U-internal* if all its internal vertices belong to U. (As a possible source of confusion, the one-edge path uv is trivially U-internal. In fact, both u and v may belong to V, so a U-internal path might completely avoid U.) We then let $\operatorname{dist}_\pi(u, v) \in \{0, \dots, R\} \cup \{\infty\}$ denote the length of the shortest U-internal u, v-path of length at most R, or ∞ if no such path exists.

For each vertex $u \in U$ define the *distance profile* $p_\pi(u)$ as the vector of its distances to V:

$$p_\pi(u) = (\operatorname{dist}_\pi(u, v_1), \dots, \operatorname{dist}_\pi(u, v_r)) .$$

We let D_π denote the set of all such distance profiles,

$$D_\pi = \bigcup_{u \in U} p_\pi(u) .$$

This set contains the information that we will maintain while traversing the tree decomposition of the input graph; a vector of distances $(d_1, \dots, d_r)$ is contained in D_π exactly if there exists a vertex $u \in U$ with that distance profile, but we forget the identity of that vertex, and how many there are. In particular, $|D_\pi| \leq (R + 2)^r$, whereas U itself may be much larger.

2.3 Maintaining distance profiles over neighbouring cuts

Consider two partitions π and π' that differ only in a single vertex.

We consider two different ways in which π and π' can differ by moving a single vertex v_r 'one part to the left' in the partition. Thus, either the vertex v_r is moved from W to V, 'introducing' it to V; or the vertex v_r is moved from V to U, 'forgetting' it from V.

The next two lemmas handle each case separately.

▶ **Lemma 4** (Introduce). *Let $\pi' = (U, V - \{v_r\}, W \cup \{v_r\})$ be a partition with $V = \{v_1, \dots, v_r\}$ and consider the partition π given by (U, V, W).*

1. $D_\pi = D_{\pi'} \times \{\infty\}$.
2. *For $v_i, v_j \in V$ with $1 \leq i < j \leq r$, we have*

$$\operatorname{dist}_\pi(v_i, v_j) = \begin{cases} \operatorname{dist}_{\pi'}(v_i, v_j) , & \text{if } j < r ; \\ l(v_i v_r) , & \text{if } j = r \text{ and } v_i v_r \in E ; \\ \infty , & \text{otherwise} . \end{cases}$$

Proof. Consider the partition π.

Since $V - \{v_r\}$ is a separator, there are no edges from v_r to any vertex in U. In particular, no path that includes any vertex in U can include v_r.

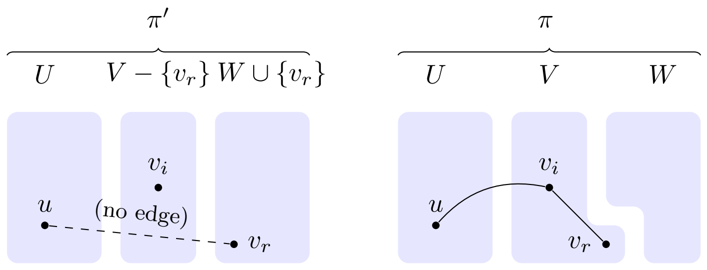

Figure 1 Vertex v_r is introduced to V.

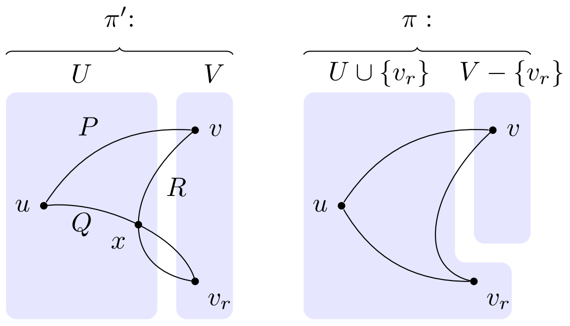

Figure 2 Vertex v_r is forgotten from V. The part W is not shown.

For the first part, consider the distance vector $(\mathrm{dist}_\pi(u,v_1),\ldots,\mathrm{dist}_\pi(u,v_r))\in D_\pi$. The last coordinate must be ∞ because there is no U-internal path from $u\in U$ to v_r. Moreover, there is no path from u to any other $v_i\in V$ passing through v_r either, so the remaining distances are unchanged.

For the second part, consider a pair of vertices $v_i,v_j\in V$. If $j\neq r$ then their U-internal distance does not change. If $v_j=v_r$ then the only possible U-internal path is the edge v_iv_r if it exists. ◀

The other case is more interesting:

▶ **Lemma 5** (Forget). *Let $\pi'=(U,V,W)$ be a partition with $V=\{1,\ldots,v_r\}$ and consider the partition π given by $(U\cup\{v_r\},V-\{v_r\},W)$.*

1. *D_π consists of the vector $(\mathrm{dist}_{\pi'}(v_1,v_r),\ldots,\mathrm{dist}_{\pi'}(v_{r-1},v_r))$ and for each $(d'_1,\ldots,d'_r)\in D_{\pi'}$ the vectors $(d_1,\ldots,d_{r-1})$ given by*

$$d_j=\min\{d'_j,d'_r+\mathrm{dist}_{\pi'}(v_j,v_r)\}\,. \tag{3}$$

2. *For each $v_i,v_j\in V$ with $1\le i<j<r$, we have*

$$\mathrm{dist}_\pi(v_i,v_j)=\min\{\mathrm{dist}_{\pi'}(v_i,v_j),\mathrm{dist}_{\pi'}(v_i,v_r)+\mathrm{dist}_{\pi'}(v_j,v_r)\}\,.$$

Proof. Let $u\in U\cup V$ and $v\in V-\{v_r\}$. Let uSv be a shortest $U\cup\{v_r\}$-internal path. Let uPv, uQv_r, and v_rRv be shortest U-internal paths, see Fig. 2.

We will show that

$$l(uSv)=\min\{l(uPv),l(uQv_r)+l(v_rRv)\}\,. \tag{4}$$

There are two cases. If the path uSv does not use v_r then it is U-internal. In particular, it has same length as the shortest U-internal path uPv. Let x be the earliest vertex on uQv_r

also appearing on v_rRv, possibly $x = v_r$. If $x \neq v_r$ then $uQxRv$ is a U-internal u, v-path, so that

$$l(uSv) = l(uPv) \leq l(uQxRv) \leq l(uQv_rRv) = l(uQv_r) + l(v_rRv)\,,$$

establishing (4) in this case. If $x = v_r$ then uQv_rRv is a path, and therefore no shorter than uSv. Thus,

$$l(uQv_r) + l(v_rRv) = l(uQv_rRv) \geq l(uSv) = l(uPv).$$

establishing (4) in this case.

If the path uSv does contain v_r then it decomposes into uSv_r and v_rSv, both of which are shortest U-internal paths. Thus we can write

$$l(uPv) \geq l(uSv) = l(uSv_rSv) = l(uSv_r) + l(v_rSv) = l(uQv_r) + l(v_rRv)\,,$$

where the first inequality merely observes that uPv is a shortest path in a smaller set of internal vertices than uSv. We have established (4).

To establish the lemma, set $v = v_j$. Provided all lengths are bounded by R, we can give (4) as

$$\mathrm{dist}_\pi(u, v_i) = \min\{\mathrm{dist}_{\pi'}(u, v_i), \mathrm{dist}_{\pi'}(u, v_r) + \mathrm{dist}_{\pi'}(v_r, v_i)\}\,. \tag{5}$$

For $u \in U \cup \{v_r\}$, write $p_\pi(u) = (d_1, \ldots, d_{r-1})$. If $u = v_r$ then the distances are simply given by $d_i = \mathrm{dist}_{\pi'}(v_i, v_r)$, because no U-internal v_i, v_r-path can use v_r as an internal node. For every other $u \in U$ let $p_{\pi'}(u) = (d'_1, \ldots, d'_r)$. Then (5) gives the first part of the lemma with $d_i = \mathrm{dist}_\pi(u, v_i)$ and $d'_i = \mathrm{dist}_{\pi'}(u, v_i)$. Finally, if $u \in V$ with $u = v_i$ for some $i \in \{1, \ldots, v_{r-1}\}$ then (5) gives the second part of the lemma.

It remains to verify that (5) holds also if some of the lengths in (4) exceed R. If $l(uSv) > R$ then $\mathrm{dist}_\pi(u, v_i) = \infty$. We already observed that $l(uSv)$ is at most $l(uPv)$ (the length of a shortest u, v-path internal in a subset of vertices) and also at most $l(uQv_rRv)$ (the length of a u, v-walk internal in the same set vertices). Thus, both values on the right hand side of (5) are also ∞. Conversely, if $l(uSv) \leq R$ then $l(uPv) \leq R$, in which case $\mathrm{dist}_{\pi'}(u, v_i) = l(uPv)$, or $l(uQv_r) + l(v_rRr) \leq R$, in which case both $\mathrm{dist}_{\pi'}(u, v_r) = l(uQv_r)$ and $\mathrm{dist}_{\pi'}(v_r, v_i) = l(v_rRv_i)$, because lengths are nonnegative. In any case, the minimum operation will pick the correct value. ◀

2.4 Tree decompositions

We consider the standard notion of a (nice) tree decomposition. To fix notation, consider a rooted, binary tree T and associate with each node $t \in T$ a set V_t of vertices, called a *piece*. Such a tree T is *nice* if it satisfies the following conditions:

1. if t is a leaf or the root then V_t is a singleton,
2. if t has a single child t' then there exists a vertex $v \in V$ such that either $v \notin V_{t'}$ and $V_t = V_{t'} \cup \{v\}$, in which case we say that t *introduces* v, or $v \in V_tt'$ and $V_t = V_{t'} - \{v\}$, in which case we say that t *forgets* v.
3. if t has two children t' and t'' then $V_t = V_{t'} = V_{t''}$.

The tree T forms a *tree decomposition of* G if the following conditions hold:

1. $V(G) = \bigcup_{t \in T} V_t$.
2. for each $uv \in E(G)$ there exists $t \in T$ such that $u, v \in V_t$.
3. for each $u \in V(G)$, the set of nodes $t \in T$ such that $u \in V_t$ are connected.

The *width* of a tree decomposition is $\max_{t\in T}|V_t|-1$. The *treewidth* of a graph is the minimum width of any tree decomposition of G.

For each node t, we define a tripartition $\pi(t)$ as the disjoint partition (U, V, W) of $V(G)$ given as follows:

1. V is the piece V_t associated with t in the tree decomposition.
2. Intuitively, U are the vertices 'below' t. Formally, let T' denote the successors of t in T. Then

$$U = \{\, V_{t'} \mid t' \in T' \,\} - V_t\,.$$

3. The remaining vertices belong to W, so $W = V(G) - (V_t \cup U)$.

We think of W as the vertices 'above' the node t, but remember that W includes vertices that are associated with siblings of t, so 'above' is a slightly misleading term.

We are finally ready to define our distance measures. For $u \in U \cup V$ and $v \in V$ consider $\operatorname{dist}_{\pi(t)}(u, v)$. Intuitively, this is the 'below'-internal distance in the sense that U contains the vertices 'below' the current node in the tree decomposition. The corresponding set of distance vectors is $D_{\pi(t)}$.

Symmetrically, from $\pi(t) = (U, V, W)$ we define the 'reverted' partition $\rho(t)$ as (W, V, U). Then, for $w \in W \cup V$ and $v \in V$ we consider $\operatorname{dist}_{\rho(t)}(w, v)$. Intuitively, this is the 'above'-internal distance in the sense that W are the vertices 'above' the current node in the tree decomposition. The corresponding set of distance profiles is $D_{\rho(t)}$.

We proceed to establish that the two sets $D_{\pi(t)}$ and $D_{\rho(t)}$ can be computed for each node $t \in T$ of the tree decomposition. The 'below'-values are established bottom-up, after which the 'above'-values are established top-down.

▶ **Algorithm D** (Distance profiles). *Given a graph G, its tree decomposition T, and an integer R such that $R \geq \operatorname{diam} G$, this algorithm computes the sets $D_{\pi(t)}$ and $D_{\rho(t)}$ for each $t \in T$.*

The algorithm works by traversing the tree decomposition twice, also computing for each $t \in T$ and each pair of vertices $u, v \in V_t$, the distances $\operatorname{dist}_{\pi(t)}(u, v)$ and $\operatorname{dist}_{\rho(t)}(u, v)$.

D1 – Traverse T bottom-up. For each leaf t of T, set $D_{\pi(t)} = \varnothing$. Traverse T bottom-up using Steps D2–D4. Then go to Step D5.

D2 – Introduce. If node t with child t' introduces vertex v_r to $V_{t'} = \{v_1, \ldots, v_{r-1}\}$ then compute $\operatorname{dist}_{\pi(t)}(d_i, d_j)$ for $i \leq i < j \leq r$ and $D_{\pi(t)}$ from $\operatorname{dist}_{\pi(t')}(d_i, d_j)$ and $D_{\pi(t')}$ using Lemma 4 with $\pi = \pi(t)$ and $\pi' = \pi(t')$.

D3 – Forget. If node t with child t' forgets vertex v_r from $V_{t'} = \{v_1, \ldots, v_r\}$ then compute $\operatorname{dist}_{\pi(t)}(d_i, d_j)$ for $1 \leq i < j < r$ and $D_{\pi(t)}$ from $\operatorname{dist}_{\pi(t')}(v_i, v_j)$ and $D_{\pi(t')}$ using Lemma 5 with $\pi = \pi(t)$ and $\pi' = \pi(t')$.

D4 – Join. If t joins t' and t'', with $V_t = V_{t'} = V_{t''} = \{1, \ldots, v_r\}$ then set

$$D_{\pi(t)} = D_{\pi(t')} \cup D_{\pi(t'')}\,,$$

and for each $u, v \in V_t$ set

$$\operatorname{dist}_{\pi(t)}(u, v) = \min\{\operatorname{dist}_{\pi(t')}(u, v), \operatorname{dist}_{\pi(t'')}(u, v)\}\,.$$

D5 – Traverse T top-down. At the root t of T, set $D_{\rho(t)} = \varnothing$. Traverse T top-down using Steps D6–D8. Then return.

D6 – Child of introduce. If t is the child of a node t' introducing v_r to $V_t = \{v_1, \ldots, v_{r-1}\}$ then compute $\operatorname{dist}_{\rho(t)}(v_i, v_j)$ for $1 \leq i < j < r$ and $D_{\rho(t)}$ from $\operatorname{dist}_{\rho(t')}(v_i, v_j)$ and $D_{\rho(t')}$ using Lemma 5 with $\pi = \rho(t)$ and $\pi' = \rho(t')$.

D7 – Child of forget. If t is the child of a node t' forgetting v_r from $V_t = \{v_1, \dots, v_r\}$ then compute $\mathrm{dist}_{\rho(t)}(v_i, v_j)$ for $1 \le i < i \le r$ and $D_{\rho(t)}$ from $\mathrm{dist}_{\rho(t')}(v_i, v_j)$ and $D_{\rho(t')}$ using Lemma 4 with $\pi = \rho(t)$ and $\pi' = \rho(t')$.

D8 – Child of join. If t is the child of a join node t' and the sibling of t'' then set

$$D_{\rho(t)} = D_{\rho(t')} \cup D_{\pi(t'')}\,,$$

and for each $u, v \in V_t$, set

$$\mathrm{dist}_{\rho(t)}(u, v) = \min\{\mathrm{dist}_{\rho(t')}(u, v), \mathrm{dist}_{\pi(t'')}(u, v)\}\,.$$

Note the asymmetry in Steps D4 and D8. On the way up, we join information from ‘below’ the child nodes; on the way down we join information from ‘above’ the parent node and ‘below’ the other sibling. The correctness of the simple minimum operation in those two steps crucially rests on the fact that $\mathrm{dist}_{\pi''}$ records only the distances that are internal to the first part U of π'' (and similarly for π' or ρ'). In particular, no new paths internal to the first parts of π or ρ (both of which contain many more vertices than U) are introduced at these join nodes.

We can now compute the eccentricity of each vertex from the following lemma:

▶ **Lemma 6** (Eccentricities). *Let v be a vertex in G with $e(v) \le r$. For any piece $V_t = \{v_1, \dots, v_r\}$ such that $v \in V_t$, we have*

$$e(v) = \max_{(d_1,\dots,d_r)} \min_{1 \le i \le r} d_i + \mathrm{dist}(v, v_i)\,, \tag{6}$$

where the maximum is taken over all distance profiles $(d_1, \dots, d_r) \in D_{\pi(t)} \cup D_{\rho(t)}$.

Proof. Consider a shortest path uPv. Assume $u \in U$, write π for $\pi(t)$ and let $(d_1, \dots, d_r) = p_\pi(u)$. Let v_i denote the first vertex on uPv that belongs to V_t, possibly $v_i = u$. Because uPv_i is a shortest paths and it is U-internal we have $l(uPv_i) = d_i$. Because v_iPv is a shortest path we have $l(v_iPv) = \mathrm{dist}(v_i, v)$. Thus, $l(uPv) = l(uPv_iPv) = d_i + \mathrm{dist}(v, v_i)$. To see that uPv attains the minimum over all i on the right and side of the expression, assume for a moment that there exits j such that $d_j + \mathrm{dist}(v, v_j) < d_i + \mathrm{dist}(v, v_i)$. Choose a shortest U-avoiding path uQv_j of length d_j and a shortest path v_jRv of length $\mathrm{dist}(v, v_j)$. Then the walk uQv_jRv would be shorter than the shortest path uPv, which is absurd. Thus,

$$\begin{aligned} l(uPv) &= \min_{1 \le i \le r} d_i + \mathrm{dist}(v, v_i) \\ &= \min_{1 \le i \le r} \left[p_{\pi(u)}\right]_i + \mathrm{dist}(v, v_i)\,, \end{aligned}$$

where $\left[p_{\pi(u)}\right]_i$ is the ith coordinate of the vector $p_{\pi(u)}$. Maximising over all $u \in U$ we arrive at

$$\begin{aligned} \max_{u \in U} l(uPv) &= \max_{u \in U} \min_{1 \le i \le r} \left[p_\pi(u)\right]_i + \mathrm{dist}(v, v_i) \\ &= \max_{p_\pi(u) \in D_\pi} \min_{1 \le i \le r} \left[p_\pi(u)\right]_i + \mathrm{dist}(v, v_i) \\ &= \max_{(d_1,\dots,d_r) \in D_\pi} \min_{1 \le i \le r} \{d_i + \mathrm{dist}(v, v_i)\}\,, \end{aligned}$$

by definition of D_π. Repeating this argument to W and the corresponding distance profile vectors indexed by ρ we see that the length of the longest shortest path uPv is indeed expressed by (6). ◀

Proof of Theorem 1, vertex-superlinear. Assume without loss of generality that G is connected.

First assume that we know a bound R with $R \geq \operatorname{diam} G$ but $R = O(\operatorname{diam} G)$. We compute a tree decomposition T of width $O(k)$ using Theorem 2. We then run Algorithm D on input G and T. From the resulting D_π and D_ρ, we use Theorem 3 and Lemma 6 to compute all eccentricities. From these values we can easily compute $\operatorname{diam} G = \max_v e(v)$, $\operatorname{rad}(G) = \min_v e(v)$, and list the vertices on the perimeter and in the center.

It remains to analyse the running time. Algorithm D performs two passes through T, which as $O(kn)$ nodes. At each node, the computation is dominated by the applications of Lemmas 4 and 5. This entails processing D_π and $D_{\pi'}$, both of which are bounded by $(R+2)^{O(k)}$. The total running time of algorithm D therefore bounded by $|T|(R+2)^{O(k)} = n \exp O(k \log \operatorname{diam} G)$. When we apply Lemma 6, we need to compute the pairwise distances $\operatorname{dist}(v, v_i)$ for $v, v_i \in V_t$ for each tree node $t \in T$. This entails $|T|\binom{O(k)}{2}$ computations, each requiring time $O(|T|k^6)$ after $O(k^3 n \log n)$ preprocessing according to Theorem 3. Since we have $|T| = O(kn)$, this step takes time $n^{1+o(1)} \operatorname{poly}(k)$.

If we do not know R, we can search for it iteratively $R = 2, 3, \ldots$, until no infinite eccentricities appear. This increases the running time by a factor of at most $\operatorname{diam} G$, which is absorbed in our time bounds. ◀

3 Vertex-linear time

We finish the proof of Theorem 1 with the desired time bound by showing how the distances between vertices within the same piece can be computed in time linear in n. The idea is to construct a graph whose edge lengths model those paths that have all their internal nodes outside of the current piece. A standard all-pairs shortest paths computation among those vertices then suffices.

▶ **Algorithm L** (Linear time distances). *Given a graph and a tree decomposition T, this algorithm computes* $\operatorname{dist}(u, v)$ *for each pair of vertices u, v belonging to V_t.*

L1 – Internal distances. Compute $\operatorname{dist}_{\pi(t)}$ and $\operatorname{dist}_{\rho(t)}$ as in algorithm D, with $R = n$.
L2 – Traverse T top-down. Process T top-down. At each node t perform Steps L3 and L4.
L3 – Construct H. Let H be the complete graph on vertex set V_t. If t has a single child then set $l(u,v) = \min\{\operatorname{dist}_{\pi(t)}(u,v), \operatorname{dist}_{\rho(t)}(u,v)\}$. If t has two children t' and t'' then set $l(u,v) = \min\{\operatorname{dist}_{\pi(t')}(u,v), \operatorname{dist}_{\pi(t'')}(u,v), \operatorname{dist}_{\rho(t)}(u,v)\}$.
L4 – Find distances between all pairs in H. Run the Bellman–Ford algorithm to compute the distances $\operatorname{dist}_H(u,v)$ in H between each pair of vertices $u, v \in V_t$. Let $\operatorname{dist}(u,v) = \operatorname{dist}_H(u,v)$.

▶ **Theorem 7.** *Algorithm L is correct. If G has n vertices and T has width k then the algorithm runs in time $O(nk^3)$.*

Proof. In the first step, the algorithm mimics algorithm D but avoids the computation of D_ϕ and D_ρ. This requires, at each node $t \in T$, the constant-time computations described in Lemmas 4 and 5 for each pair $u, v \in V_t$. Thus, the running time is dominated by running the all-pairs shortest paths computation in Step L4, which runs in time $O(|V(H)|^3)$. ◀

We note that the performance of the algorithm is dwarfed by the requirements of actually computing a tree decomposition, so for the polynomial dependency on k it is crucial that T be provided as part of the input.

This finishes the proof of Theorem 1. Note that the computations of algorithm L can be performed during the top-down traversal in Steps D6–D8, just after $\mathrm{dist}_{\rho(t)}$ is found. Thus, a unified presentation of the algorithm could be given in only two traversals.

4 Conclusion

Our constructions do extend readily to *directed* graphs, and parameters like directed eccentricity, source radius, etc. can be computed within the same time bound as their undirected counterparts. We choose to claim this here without proof, since a more general presentation that encompasses directed graphs incurs considerable expository overhead without providing much insight outside of what is already present in the appendix of [1]. The most notable changes arise in the computation of *round-trip distance* from u to v, which is the minimum of the sums of the lengths of directed paths uPv and vQu. To compute these values, we need to change the definition of distance profile vectors to matrices: For partition $\pi = (U, \{v_1, \ldots, v_r\}, W)$ consider the $r \times r$ matrix M with $m_{ij} = d$ if there is a vertex $u \in U$ such that there exist U-internal shortest directed paths v_iPu and uPv_j with $d = l(v_iPu) + l(uPv_j)$. The set D_π then contains all matrices with elements bounded by R that are realised by some $u \in U$. The total size of this set is bounded by $(R+2)^{r^2}$. When v_r is forgotten in the tree decomposition, the entries m_{ir} and m_{rj} account for directed U-internal paths of length $m_{ir} + m_{ij}$. The resulting time running time, suppressing several details, becomes $n \exp O(k^2 \log \operatorname{diam} G)$, much like the bound of $O(k^2 n \log^{k^2-1} n)$ of [1]. In both constructions, we notice an exponential dependency on the square of the treewidth.

Our bounds rely on the fact that we store merely the *existence* of vertex pairs at certain distances, not the *number* of such pairs. Thus, our constructions have nothing to contribute to various graph distance measures that involve counting, such as the Wiener index, the median, or closeness centrality.

The main question opened up by our algorithms is the complexity of computing superconstant diameter. In particular, an algorithm with running time $n \exp O(k)$ could exist even for diameter $\omega(1)$. However, we conjecture that the diameter of a graph with treewidth and diameter k cannot be computed in time $n^{2-\epsilon} \exp o(k \log k)$.

Acknowledgements. I am happy that Virginia Vassilevska Williams told me about this question.

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.

2 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.

3 Sergio Cabello and Christian Knauer. Algorithms for bounded treewidth with orthogonal range searching. *Comput. Geom.*, 42(9):815–824, 2009.

4 Shiva Chaudhuri and Christos D. Zaroliagis. Shortest path queries in digraphs of small treewidth. *Algorithmica*, 27(3):212–226, 2000.

5 David Eppstein. Subgraph isomorphism in planar graphs and related problems. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1995. San Francisco, California*, pages 632–640. ACM/SIAM, 1995.

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

7 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms for graphs of bounded treewidth are probably optimal. In *Proceedings of the Twenty-Second Annual ACM–SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 777–789. SIAM, 2011.

8 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. ACM, 2013.

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