Optimal Centrality Computations Within Bounded Clique-Width Graphs

Guillaume Ducoffe

Optimal centrality computations within bounded clique-width graphs *

Guillaume Ducoffe
guillaume.ducoffe@ici.ro[1,2]

[1]National Institute for Research and Development in Informatics, Romania
[2]University of Bucharest, Romania

Abstract

Given an n-vertex m-edge graph G of clique-width at most k, and a corresponding k-expression, we present algorithms for computing some well-known centrality indices (eccentricity and closeness) that run in $\mathcal{O}(2^{\mathcal{O}(k)}(n+m)^{1+\epsilon})$ time for any $\epsilon > 0$. Doing so, we can solve various distance problems within the same amount of time, including: the diameter, the center, the Wiener index and the median set. Our running-times match conditional lower bounds of Coudert et al. (*SODA'18*) under the Strong Exponential-Time Hypothesis. On our way, we get a distance-labeling scheme for n-vertex m-edge graphs of clique-width at most k, using $\mathcal{O}(k \log^2 n)$ bits per vertex and constructible in $\tilde{\mathcal{O}}(k(n+m))$ time from a given k-expression. Doing so, we match the label size obtained by Courcelle and Vanicat (*DAM 2016*), while we considerably improve the dependency on k in their scheme. As a corollary, we get an $\tilde{\mathcal{O}}(kn^2)$-time algorithm for computing All-Pairs Shortest-Paths on n-vertex graphs of clique-width at most k, being given a k-expression. This partially answers an open question of Kratsch and Nelles (*STACS'20*). Our algorithms work for graphs with non-negative vertex-weights, under two different types of distances studied in the literature. For that, we introduce a new type of orthogonal range query as a side contribution of this work, that might be of independent interest.

*This work was supported by project PN-19-37-04-01 "New solutions for complex problems in current ICT research fields based on modelling and optimization", funded by the Romanian Core Program of the Ministry of Research and Innovation (MCI) 2019-2022. It was also supported by Grant TC ICUB-SSE 15109-26.07.2021, "The complexity landscape of Maximum Matching". Results of this paper were partially presented at the IPEC'21 conference [27].

1 Introduction

For any undefined graph terminology, see [2, 25]. Unless stated otherwise, all graphs considered in this work are simple and connected. We here consider *clique-width*, which is one of the most studied parameters in Graph Theory, superseded only by treewidth. Roughly, clique-width is a measure of the closeness of a graph to a cograph (*a.k.a.*, P_4-free graph). We postpone its formal definition until Sec. 2. The clique-width was shown to be bounded on many important subclasses of perfect graphs [4, 5, 22, 44, 53], and beyond [9, 6, 10, 7, 8, 23, 54, 58, 60]. For instance, distance-hereditary graphs, and so, trees, have clique-width at most three [44]. Every graph of bounded treewidth also has bounded clique-width, but the converse is not true [15]. Indeed, unlike for treewidth, there are dense graphs of bounded clique-width (*e.g.*, the complete graphs). This generality comes at some cost: whereas the celebrated Courcelle's theorem asserts that any problem expressible in MSO_2 logic can be solved in FPT linear time on bounded treewidth graphs [17], the same is true for bounded clique-width graphs only for the problems expressible in the more restricted MSO_1 logic [19]. Fomin et al. showed this to be unavoidable, in the sense that there are problems expressible in MSO_2 logic that are $W[1]$-hard in the clique-width [33, 34, 35]. We refer to [30] for other algorithmic applications of clique-width in parameterized complexity.

Our focus is about the so-called "FPT in P" program. Here the goal is, for some problem solvable in $\mathcal{O}(m^{q+o(1)})$ time on arbitrary m-edge graphs, to design an $\mathcal{O}(f(k)m^{p+o(1)})$-time algorithm, for some $p < q$, within the class of graphs where some fixed parameter is at most k (one usually seeks for $p = 1$ and $f(k) = k^{\mathcal{O}(1)}$). The idea of using tools and methods from parameterized complexity in order to solve faster certain polynomial-time solvable problems has been here and there in the literature for a while (*e.g.*, see [47]). Nevertheless it was only recently that such idea was better formalized [42], in part motivated by some surprising results obtained for treewidth [1]. Indeed, on the positive side, the treewidth does help in solving faster many important problems in P, that is, in $\tilde{\mathcal{O}}(k^{\mathcal{O}(1)}n)$ time on graphs and matrices of treewidth at most k [36, 49]. But for other such problems, any truly subquadratic-time parameterized algorithm requires *exponential* dependency on the treewidth. For example, given a graph G with a non-negative weight function on its edge-set (resp., on its vertex-set), the weight of a path equals the sum of the weights of all its edges (resp., of all its vertices). For unweighted graphs, this is exactly the number of edges (resp, the number of edges plus one). The distance $d_G(u, v)$ between two vertices u and v is equal to the least weight of a uv-path. Finally, the *diameter* of G is defined as $diam(G) = \max_{u,v \in V(G)} d_G(u, v)$. Abboud et al. proved that under the Strong Exponential-Time Hypothesis (SETH), for any $\epsilon > 0$, there is no $\mathcal{O}(2^{o(k)}n^{2-\epsilon})$-time algorithm for computing the diameter of n-vertex *unweighted* graphs of treewidth at most k [1]. An algorithm for this problem on *weighted* graphs, running in $\mathcal{O}(2^{\mathcal{O}(k)}n^{1+\epsilon})$ time for any $\epsilon > 0$, was proved recently in [11] by using the orthogonal range query framework of Cabello and Knauer [13].

Insofar, clique-width has received less attention than treewidth in the nascent field of FPT in P. Perhaps one good reason for that is that, for most problems on edge-weighted graphs, clique-width provably does *not* help [51]. This is because we may regard any graph as a weighted clique, where each non-edge got replaced by an edge of sufficiently large weight. Note however that most conditional lower bounds in the literature hold even for unweighted graphs (this is the case for the diameter and the other distance problems that we here study). Furthermore, in a recent paper Kratsch and Nelles [52] have evidenced that some applications of clique-width to unweighted graphs could be extended to vertex-weighted graphs. We give further evidence for that in our work. One other well-known drawback of clique-width is that, unlike for treewidth, the parameterized complexity of computing it is a wide open problem [14]. Until very recently, the best-known approximation algorithms for clique-width were running in $\mathcal{O}(n^3)$-time [55]. However, this has now been improved to $\mathcal{O}(n^2)$ for constant clique-width graphs [32]. Furthermore, on many subclasses of bounded clique-width graphs, there exist linear-time algorithms in order to compute a so called "k-expression", for some $k = \mathcal{O}(1)$, with the latter certifying the clique-width of the graph to be at most k [44, 54]. Therefore, the study of graph problems in P parameterized by clique-width may be regarded as a unifying framework for all such subclasses. In this respect, Coudert et al. obtained $\tilde{\mathcal{O}}(k^{\mathcal{O}(1)}(n+m))$-time algorithms for triangle and cycle problems on n-vertex m-edge graphs of clique-width at most k [16]. However, they also observed that assuming SETH, even on n-vertex cubic graphs of clique-width at most k, for any $\epsilon > 0$, there is

no $\mathcal{O}(2^{o(k)}n^{2-\epsilon})$-time algorithm for computing the diameter. Unlike for treewidth, it was open until this paper whether there does exist a parameterized quasi-linear-time algorithm for this problem on bounded clique-width graphs that matches their conditional lower bound. Indeed, we are only aware of a linear-time algorithm for computing the diameter of bounded clique-width graphs in [20], but with a super-exponential dependency on the clique-width in the runtime, due to the use of Courcelle's theorem. The work of Coudert et al. has also been continued in [29, 28, 51] and especially in [52], where the authors obtained an $\mathcal{O}((kn)^2)$-time algorithm for All-Pairs Shortest Paths (APSP) on n-vertex graphs of clique-width at most k.

Results. We provide new insights on the fine-grained complexity of polynomial-time solvable distance problems within bounded clique-width graphs. As in all previous works in this area, all our algorithmic results require a k-expression to be given in the input. Specifically, let $G = (V, E, w)$ be such that $|V| = n$, $|E| = m$, and $w : V \to \mathbb{N}$ is a vertex-weight function. The *eccentricity* of a vertex u, denoted $e_G(u)$, is its largest distance to any other vertex; its inverse is sometimes called the graph centrality of u [46]. The *closeness centrality* of u, denoted $C_G(u)$, equals $1/\sum_v d_G(u, v)$ [57]. For a discussion about these centrality measures, and others, and their role in social network analysis, we refer to [24]. Our main contribution is an algorithm for computing all eccentricities, and closeness centralities within the n-vertex m-edge graphs of clique-width at most k, being given a k-expression, that runs in $\mathcal{O}(2^{\mathcal{O}(k)}(n + m)^{1+\epsilon})$ time for any $\epsilon > 0$ (Theorem 4.1).

We point out that the diameter of a graph is its largest eccentricity. The radius of a graph is its least eccentricity, and its center is the set of all vertices whose eccentricity equals the radius. Therefore, our result for computing all eccentricities implies, for any $\epsilon > 0$, an $\mathcal{O}(2^{\mathcal{O}(k)}(n + m)^{1+\epsilon})$-time algorithm for computing the diameter, the radius, and the center of a graph of clique-width at most k, if a k-expression is given. To the best of our knowledge, it is the first algorithm to match the conditional lower bound of Coudert et al. Previously, the only known algorithms for these problems were applications of Courcelle's theorem [19]. The Wiener index $W(G)$ of a graph G is the sum of all its distances, while its median set contains all the vertices of maximal closeness centrality. In the same way, our result for computing the closeness centrality implies, for any $\epsilon > 0$, an $\mathcal{O}(2^{\mathcal{O}(k)}(n + m)^{1+\epsilon})$-time algorithm for computing both the Wiener index and the median set of a graph of clique-width at most k, if a k-expression is given. Our runtimes are also optimal under SETH for the Wiener index, and so, for the closeness centrality (the conditional lower bound is the same as for the diameter problem, see the discussion in Sec. 4).

Recall that our results hold for vertex-weighted graphs. A related problem, studied in location theory, is given an unweighted graph $G = (V, E)$ and a cost function $p : V \to \mathbb{N}$, to compute for every vertex u its p-eccentricity (resp., its total p-distance sum), defined as $e_p(u) := \max_v p(v)d_G(u, v)$ (resp., as $TD_p(u) := \sum_v p(v)d_G(u, v)$). Note that the total distance for unweighted graphs is nothing but the inverse of closeness centrality. Our approach can also be applied to that case (Theorem 5.1).

Finally, as a byproduct of our techniques, we obtain a new *distance labeling scheme* for bounded clique-width graph classes which outperforms the state of the art [20][1]. See our Theorem 3.2 for details. In doing so, we get an $\tilde{\mathcal{O}}(kn^2)$-time algorithm in order to solve All-Pairs Shortest-Paths within n-vertex vertex-weighted graphs of clique-width at most k (Corollary 3.5). This improves on the previously best-known $\mathcal{O}((kn)^2)$-time algorithm, and it almost completely solves an open problem from Kratsch and Nelles [52] who asked whether there exists an $\mathcal{O}(kn^2)$-time algorithm for this problem.

Overview of our techniques. Roughly, the standard approach for bounded clique-width graphs is to process a k-expression sequentially. It is possible to transform a k-expression into a so called *partition tree*, a purely combinatorial object that has been used in [18] in order to derive a new characterization of the clique-width. – We formally define clique-width and partition trees in Sec. 2. – Doing so, it becomes easier and more transparent to apply standard algorithmic approaches, for trees, to the k-expressions. In particular, it is known that every bounded clique-width graph has a balanced edge-cut of bounded neighbourhood diversity (*i.e.*, whose edges can be partitioned in a bounded number of complete bipartite graphs) [3, 26]. As a side contribution of this work, we show how to compute such balanced cuts in parameterized linear time in the

[1] In all fairness, the labeling scheme of Courcelle and Vanicat can be applied to many more problems than just the computation of the distances in the graph.

clique-width from a given partition tree. – Note that the original runtime from [3] is unknown to us as we were unable to find this reference. – This procedure of recursively finding such an edge-cut produces a special type of centroid decomposition of a partition tree, with algorithmic applications to several distance problems on bounded clique-width graphs. While such a divide-and-conquer approach can hardly be considered as 'new', its usefulness in the fine-grained complexity study of polynomial-time solvable problems on bounded clique-width graphs has remained to be demonstrated until our work. We expect several other results to be found with this approach, in a similar way to what has been done for bounded treewidth graphs in [49].

The distance-labeling scheme of Theorem 3.2 follows almost directly from our centroid decomposition of a partition tree, that is why we chose to present it first in the paper. In order to compute the centrality indices, we combine this centroid decomposition with two other tools. One is the range query framework of Cabello and Knauer [13] that we use to compute some distance information (depending on the centrality index) between the vertices that are on different sides of an edge-cut of small neighbourhood diversity. To our best knowledge, our work is the first (but admittedly, simple) application of this framework to edge-cuts. We also augment this framework with a new type of orthogonal range query, with applications to the fast computation of all p-eccentricities and total p-distances, see Sec. 5. Our second tool is inspired from prior works on bounded treewidth graphs [1, 11] and Cunningham's split decomposition [21]. Specifically, we design some *edge*-weighted gadgets in order to preserve the distances of the original graph in the two subgraphs resulting from the removal of an edge-cut of bounded neighbourhood diversity. Adding weighted edges is problematic because the diameter problem cannot even be solved in truly subquadratic time within edge-weighted graphs of bounded clique-width. To address this issue, we restrict our addition of weighted edges to ensure that when we further partition the graph via more edge-cuts, the weighted edges are not included in these edge-cuts. To do this, we partition the vertices of our gadgets into at most $\mathcal{O}(\log n)$ clusters of only $\mathcal{O}(k^2)$ vertices each, so that weighted edges are only added between pairs of vertices in the same cluster. Then, we ensure that no cluster is ever separated by an edge-cut computed from the partition tree. Doing so, we are still able to ensure that we can find *unweighted* edge-cuts that satisfy the requirement of being both balanced and of small neighborhood diversity. We stress that to prove correctness of our construction, we had to carefully analyze the structure of a partition tree, which is arguably the most technical part of our analysis. While it is tempting to make our gadgets vertex-weighted (*e.g.*, by properly subdividing the weighted edges), we did not find a satisfying way to do that without increasing the neighbourhood diversity of some of the cuts.

Notations. Throughout the remainder of the paper, we shall write $G = (V, E)$ for an unweighted graph, and $G = (V, E, w)$ for a vertex-weighted graph, where $w : V \to \mathbb{N}$. The neighbour set of a vertex $v \in V$, resp. of a subset $S \subseteq V$, is defined as $N_G(v) = \{u \in V \mid uv \in E\}$, resp. as $N_G(S) = \bigcup_{v \in S} N_G(v) \setminus S$. We may also define the distance between a vertex $v \in V$ and a subset $S \subseteq V$ as $d_G(v, S) = d_G(S, v) = \min_{u \in S} d_G(u, v)$, and the distance between two subsets S, S' as $d_G(S, S') = \min_{u \in S, v \in S'} d_G(u, v)$. Note that if $S = \emptyset$ then, $d_G(v, S) = d_G(S, S') = +\infty$ for any v and S'. Recall that for every fixed vertex s, we can compute in $\mathcal{O}((n + m) \log n)$ time all the distances $d_G(s, v)$, for $v \in V$, using Dijkstra algorithm. It is folklore that for every fixed subset S, we can adapt Dijkstra algorithm in order to compute in $\mathcal{O}((n + m) \log n)$ time all the distances $d_G(S, v)$, for $v \in V$ (that consists, roughly, in replacing S by a single new vertex s). Throughout the paper, we call it a modified Dijkstra algorithm. We shall introduce additional terminology wherever needed in the paper.

Results of this paper were partially presented at the IPEC'21 conference [27].

2 Clique-width and partition trees

First, we recall two equivalent definitions of clique-width [18]. The following definitions can be extended to weighted graphs simply by ignoring all the weights.

Clique-width expressions. A k-labeled graph is a triple $G = (V, E, \ell)$ where $\ell : V \to \{1, 2, \ldots, k\}$ is called a labeling function. A clique-width k-expression (for short, a k-expression) is an algebraic expression where the four allowed operations are: $i(v)$: we add a new isolated vertex with label $\ell(v) = i$; $G_1 \oplus G_2$:

we make the disjoint union of two k-labeled graphs; $\eta(i,j)$: we add a join (complete bipartite subgraph) between all vertices with label i and all vertices with label j; $\rho(i,j)$: for all vertices v s.t. $\ell(v) = i$, we set $\ell(v) = j$. The generated graph is the one obtained from the k-expression by deleting all the labels. We say that a graph $G = (V, E)$ has clique-width at most k if it is the graph generated by some k-expression. For instance, $1(a)2(b)\eta(1,2)\rho(1,3)1(c)\eta(1,2)\rho(2,3)2(d)\eta(1,2)$ is a 3-expression generating the four-node path P_4 with nodes a, b, c, d. In particular, the clique-width of P_4 is at most three. This is in fact an equality, as the graphs of clique-width at most two are exactly the cographs [44]. We denote by $cw(G)$ the clique-width of the graph G. The size of a k-expression is its number of operations. If the generated graph has order n and m edges, and there is no unnecessary operation $\rho(i,j)$ nor $\eta(i,j)$ – which we will assume to be the case throughout the remainder of this paper –, then the k-expression has size in $\mathcal{O}(n+m)$ (*e.g.*, see [38], where Fürer proved this result in a more general setting).

Partition tree. It is useful to represent a k-expression as a parse tree. We stress that the subtree rooted at any node in the parse tree represents the k-expression of some labelled subgraph of G. By iteratively contracting the edges incident to non-branching nodes of a parse tree, we get a so-called partition tree, whose nodes are mapped to the partition in at most k label classes of the vertices in the corresponding labelled subgraph of G. Formally, given a graph $G = (V, E)$, a partition tree is a pair (T, f) where T is a rooted tree whose inner nodes have at least two children, and f is a function mapping every node of T to a partial partition of V, such that:

- for every node $a \in V(T)$, $f(a)$ is a partition of some vertex-subset $A \subseteq V$;
- for every vertex $v \in V$, there is a leaf node $a_v \in V(T)$ s.t. $f(a_v) = \{\{v\}\}$;
- for every inner node $a \in V(T)$, let $b_1, b_2, \ldots, b_d$ be its children. If $f(a)$ is a partition of A, and in the same way for every $1 \leq i \leq d$, $f(b_i)$ is a partition of B_i, then the vertex-subsets $B_1, B_2, \ldots, B_d$ are pairwise disjoint and $A = \bigcup_{i=1}^{d} B_i$. Furthermore, for every $1 \leq i \leq d$, for every subset $X_i \in f(b_i)$, there is $X \in f(a)$ s.t. $X_i \subseteq X$ (we say that $\bigcup_{i=1}^{d} f(b_i)$ refines $f(a)$). Finally, for every $1 \leq i < j \leq d$, for every adjacent vertices $v_i \in B_i$ and $v_j \in B_j$, if $v_i \in X$ and $v_j \in Y$, for some $X, Y \in f(a)$, then we have $X \neq Y$ and $X \times Y \subseteq E$ (we say that the partition is compatible with the edge-incidence relation in the graph G).

The *width* of a partition tree is equal to $\max_{a \in V(T)} |f(a)|$. A graph has clique-width at most k if and only if it admits a partition tree of width at most k [18].

Note that if we naively store a partition tree (T, f), then storing explicitly all the labels $f(a)$, for $a \in V(T)$, would require $\mathcal{O}(n^2)$ space. Instead, for every $a \in V(T)$, for every $X \in f(a)$, we may create a new vertex (a, X); then if b_i is a child of a, for every $X_i \in f(b_i)$ s.t. $X_i \subseteq X$, we add an arc between (a, X) and (b_i, X_i). This is called in [18] the representation graph of (T, f) and it only requires $\mathcal{O}(kn)$ space if the width is at most k.

Lemma 2.1 ([18])**.** *There is an algorithm that transforms a k-expression of size L into the representation graph of a width-k partition tree in $\mathcal{O}(kL)$ time.*

In particular, given a k-expression for an n-vertex m-edge graph G, we can construct the representation graph of a width-k partition tree in $\mathcal{O}(k(n+m))$ time.

Relation with k-modules. For a graph $G = (V, E)$, a subset $M \subseteq V$ is a module if we have $N_G(u) \setminus M = N_G(v) \setminus M$ for every vertices $u, v \in M$. A k-module is some $M \subseteq V$ that can be partitioned into k subsets, denoted $M_1, M_2, \ldots, M_k$, in such a way that for every $1 \leq i \leq k$, M_i is a module in the subgraph $G[(V \setminus M) \cup M_i]$. Some relations between clique-width and k-modules were explored in [56]. We make the following useful observation, whose proof is inspired by [56, Theorem 7].

Lemma 2.2. *The following two properties hold for every partition tree (T, f) of a graph $G = (V, E)$:*

1. *For every node $a \in V(T)$, let $A = \bigcup f(a)$ be the vertex-subset of which $f(a)$ is a partition. Then, A is a $|f(a)|$-module of G, with a corresponding partition of A being $f(a)$.*

2. *Let $a_1, a_2, \ldots, a_p$ be some children nodes of some $a' \in V(T)$ and, for each $1 \leq i \leq p$, let $A_i = \bigcup f(a_i)$ be the vertex-subset of which $f(a_i)$ is a partition. Then, $A = \bigcup_{i=1}^{p} A_i$ is a $|f(a')|$-module of G, with a corresponding partition of A being $\{X' \cap A \mid X' \in f(a')\}$.*

Proof. We prove these two above statements simultaneously, by induction on the depth of the nodes. For the base case, let us assume $a \in V(T)$ to be the root of T. In particular, $A = V$. In this situation, every $X \in f(a)$ is a trivial module of $G \setminus (V \setminus X) = G[X]$. Hence, $A = V$ is a k-module of G with a corresponding partition being $f(a)$. Then, let $a_1, a_2, \ldots, a_p$ be children nodes of some $a' \in V(T)$, and let us assume by induction that $A' = \bigcup f(a')$ is a $|f(a')|$-module of G, with a corresponding partition being $f(a')$. Recall that $A = \bigcup_{i=1}^{p} A_i$ is the union of all the subsets partitioned by the $f(a_i)$'s. By the refinement property we have $A \subseteq A'$, and therefore $\Phi(A) = \{X' \cap A \mid X' \in f(a')\}$ is a partition of A. Let us prove that A is a $|\Phi(A)|$-module of G, with a corresponding partition being $\Phi(A)$ (Property 2 of the lemma). Equivalently, we are left proving that for every $X \in \Phi(A)$, for every $u, v \in X$ we have $N_G(u) \setminus A = N_G(v) \setminus A$. For that, recall that there is a $X' \in f(a')$ s.t. $X \subseteq X'$. By our induction hypothesis, X' is a module of $G \setminus (A' \setminus X')$. Therefore, $N_G(u) \setminus A' = N_G(v) \setminus A'$. In order to prove that X is a module of $G \setminus (A \setminus X)$, it now suffices to prove that we have $N_G(u) \cap (A' \setminus A) = N_G(v) \cap (A' \setminus A)$. Let $w \in A' \setminus A$ be s.t. $uw \in E$. The refinement property implies the existence of some node $b \notin \{a_1, a_2, \ldots, a_p\}$ s.t. b is another child of a', $f(b)$ is a partition of some vertex-subset B that is disjoint from A, and $w \in B$. Then, since $uw \in E$, the compatibility property implies the existence of some $Y' \in f(a')$ s.t. $Y' \neq X'$, $w \in Y'$ and $X' \times Y' \subseteq E$. In particular, every vertex of X', and so, of X, is adjacent to w. This implies $N_G(u) \cap (A' \setminus A) = N_G(v) \cap (A' \setminus A)$. Finally, let us prove that for every child a of a', $A = \bigcup f(a)$ is also a $|f(a)|$-module of G, with a corresponding partition being $f(a)$ (Property 1 of the lemma). By setting $p = 1$, we first get that A is a $|f(a')|$-module, with a corresponding partition being $\Phi(A) = \{X' \cap A \mid X' \in f(a')\}$. Then, we are done by the refinement property because every $X \in f(a)$ must be contained into some subset $X' \cap A \in \Phi(A)$. □

Finally, recall that a cut of $G = (V, E)$ is a bipartition $(A, V \setminus A)$ of its vertex-set. The *neighbourhood diversity* of a cut is the least k s.t. A is a k-module of G. By Lemma 2.2, each node of a width-k partition tree defines a cut of neighbourhood diversity at most k.

3 Distance-labeling scheme

We describe our distance oracle for bounded clique-width graph classes. For technical reasons, we need to make it work also for unconnected graphs. While it is likely that we could process each connected component separately, we did not explore this possibility since it was leading to more complicated updates of the partition trees (see the proof of Theorem 3.2 below).

Given a possibly unconnected graph G, the *distance* $d_G(u, v)$ between $u, v \in V$ is equal to: $+\infty$ if u and v are on different connected components of G, and to the smallest weight of a uv-path in G otherwise. A *distance-labeling scheme* consists in some encoding function $C_G : V \to \{0,1\}^*$ and some decoding function $D_G : \{0,1\}^* \times \{0,1\}^* \to \mathbb{N} \cup \{+\infty\}$ s.t. $d_G(u, v) = D_G(C_G(u), C_G(v))$ for every vertices u and v. We are interested in minimizing the total pre-processing time in order to compute the labels $C_G(v)$, for all vertices v, and the query time in order to compute the distance given two labels. It is often the case that D_G runs in time polynomial in the size of the labels. Then, the objective is to minimize $\max_{v \in V} |C_G(v)|$.

The following result is due to Courcelle and Vanicat:

Theorem 3.1 ([20])**.** *The family of n-vertex bounded clique-width unweighted graphs enjoys an exact distance labeling scheme using labels of length $\mathcal{O}(\log^2 n)$ bits. Moreover, the distance between two vertices can be computed in $\mathcal{O}(\log^2 n)$ time.*

The hidden dependency in the clique-width is a stack of exponentials [40]. We improve the latter while keeping optimal bit size and improved query time, namely:

Theorem 3.2. *For a vertex-weighted graph, let W denote the maximum weight. The family of n-vertex m-edge vertex-weighted graphs of clique-width at most k enjoys an exact distance labeling scheme using labels*

of length $\mathcal{O}(k \log n \log(nW))$ bits (resp., $\mathcal{O}(k \log^2 n)$ bits if the graph is unweighted). Moreover, all the labels can be pre-computed in $\mathcal{O}(k(n+m)\log^2 n)$ time if a k-expression is given (resp., in $\mathcal{O}(k(n+m)\log n)$ time if the graph is unweighted), and the distance between two vertices can be computed in $\mathcal{O}(k \log n)$ time.

For the related problem of adjacency queries, we refer to [50] for a data structure in $\mathcal{O}(kn)$ space for the n-vertex graphs of clique-width at most k.

Recall that $d_G(v, S) = d_G(S, v) = \min_{u \in S} d_G(u, v)$. In particular, $d_G(v, S) = +\infty$ if S is empty. We will need the following result:

Lemma 3.3. *Let $G = (V, E, w)$ be a vertex-weighted graph (possibly not connected) and let $(A, V \setminus A)$ be a cut of neighbourhood diversity k. Furthermore, let $A_1, A_2, \ldots, A_k$ be a partition of A s.t. for every $1 \leq i \leq k$, A_i is a module of $G \setminus (A \setminus A_i)$. For $1 \leq i \leq k$, let $B_i = N_G(A_i) \setminus A$. The following hold for every $u, v \in V$:*

- *if $u \in A,\ v \notin A$ then $d_G(u, v) = \min\{d_G(u, A_i) + d_G(B_i, v) \mid 1 \leq i \leq k\}$;*
- *if $u, v \in A$ then $d_G(u, v) = \min\{d_{G[A]}(u, v)\} \cup \{d_G(u, A_i) + d_G(B_i, v) \mid 1 \leq i \leq k\}$;*
- *if $u, v \notin A$ then $d_G(u, v) = \min\{d_{G[V \setminus A]}(u, v)\} \cup \{d_G(u, A_i) + d_G(B_i, v) \mid 1 \leq i \leq k\}$.*

Proof. We may assume u and v to be in a same connected component of G. Indeed if it is not the case then we claim that, for any $1 \leq i \leq k$, we have $d_G(u, A_i) = +\infty$ or $d_G(B_i, v) = +\infty$; in particular, the lemma holds true in this special case. In order to prove this claim, there are two simple cases to consider: either $B_i = \emptyset$, and then $d_G(B_i, v) = +\infty$, or $B_i \neq \emptyset$. In the latter case, $A_i \cup B_i$ is connected, and therefore we must have $d_G(u, A_i) = +\infty$ or $d_G(B_i, v) = +\infty$. From now on, we implicitly assume the existence of a uv-path. Then, for *any* u and v, for every $1 \leq i \leq k$ s.t. $B_i \neq \emptyset$, since there is a complete join between A_i and B_i there always exists a uv-path of length $d_G(u, A_i) + d_G(B_i, v)$ (recall that if $B_i = \emptyset$, then we have $d_G(u, A_i) + d_G(B_i, v) = d_G(B_i, v) = +\infty$). In particular, $d_G(u, v) \leq \min\{d_G(u, A_i) + d_G(B_i, v) \mid 1 \leq i \leq k\}$. Then, we consider all three cases of the lemma. If $u \in A,\ v \notin A$ then on any shortest uv-path, there must be some edge $u'v'$ s.t. $u' \in A,\ v' \notin A$. In particular, $u' \in A_i$ for some $1 \leq i \leq k$, and then $v' \in B_i$. We get $d_G(u, v) \geq d_G(u, A_i) + d_G(B_i, v)$. As a result, $d_G(u, v) = \min\{d_G(u, A_i) + d_G(B_i, v) \mid 1 \leq i \leq k\}$. If $u, v \in A$ then, either there exists a shortest uv-path which is fully contained into A, that implies $d_G(u, v) = d_{G[A]}(u, v)$, or *every* shortest uv-path must intersect $V \setminus A$. In the latter sub-case, we fix a shortest uv-path and we scan it from u until we find an edge $u'v'$ s.t. $u' \in A, v' \notin A$. We deduce as before that we have in this sub-case $d_G(u, v) = \min\{d_G(u, A_i) + d_G(B_i, v) \mid 1 \leq i \leq k\}$. In the same way, if $u, v \notin A$ then either there exists a shortest uv-path which is fully out of A, that implies $d_G(u, v) = d_{G[V \setminus A]}(u, v)$, or *every* shortest uv-path must intersect A. In the latter sub-case, we fix a shortest uv-path and we scan it from v until we find an edge $v'u'$ s.t. $u' \in A, v' \notin A$. We get $d_G(u, v) = \min\{d_G(u, A_i) + d_G(B_i, v) \mid 1 \leq i \leq k\}$. □

Our scheme for bounded clique-width graphs mimics one very well-known for trees which is based on the centroid decomposition [41]. Specifically, let $w : V(T) \to \mathbb{N}$ assign non-negative weights to the nodes of some tree T. A w-centroid is a node c s.t. every subtree of $T \setminus \{c\}$ has weight at most $w(T)/2$. Such node always exists and a centroid can be computed in linear time by using a standard dynamic programming approach [43] (simply orient each edge toward the heaviest subtree, then find a sink). We also need the following easy lemma:

Lemma 3.4. *Let $w : V(T) \to \mathbb{N}$ assign non-negative weights to the nodes of some tree T. If c is a w-centroid of a tree T, then the components of $T \setminus \{c\}$ can be partitioned in linear time in two forest F_1, F_2 s.t. $\max\{w(F_1), w(F_2)\} \leq 2w(T)/3$.*

Proof. If $w(c) \geq w(T)/3$ then the result holds for any bipartition of the components of $T \setminus \{c\}$. From now on we assume that we did not fall in this pathological case. In particular, $T \setminus \{c\}$ is unconnected (otherwise, $w(T \setminus \{c\}) > 2w(T)/3 > w(T)/2$, a contradiction). Let $T_1, T_2, \ldots, T_d$ be the subtrees of $T \setminus \{c\}$. We define i_0 as the least index i s.t. $\sum_{j=1}^{i} w(T_j) > 2w(T)/3$. Note that $i_0 > 1$ since we assume c to be a w-centroid. Then, there are two cases.

- If $\sum_{j=i_0}^{d} w(T_j) \leq 2w(T)/3$ then we are done by setting $F_1 = \bigcup_{j<i_0} T_j,\ F_2 = \bigcup_{j\geq i_0} T_j$.
- Otherwise, we get $w(T) + w(T_{i_0}) \geq \sum_{j\leq i_0} w(T_j) + \sum_{j\geq i_0} w(T_j) > 4w(T)/3$, and so, $w(T_{i_0}) > w(T)/3$. We set $F_1 = T_{i_0},\ F_2 = \bigcup_{j\neq i_0} T_j$.

In both cases, we get the desired partition in two forests of respective weights at most $2w(T)/3$. □

We are now ready to prove the main result of this section:

Proof of Theorem 3.2. We fix some width-k partition tree (T, f), that takes $\mathcal{O}(k(n+m))$ time to compute by using Lemma 2.1. Let $w : V(T) \to \{0, 1\}$ be s.t. $w(a) = 1$ if and only if a is a leaf. Observe that $w(T) = n$ since there is a one-to-one mapping between the vertices in V and the leaves of T. In order to construct the labels $C_G(v)$, for all $v \in V$ (encoding function), we next define a recursive procedure onto the weighted partition tree.

In what follows, let us assume $n > 1$ (otherwise, there is nothing to be done). We compute in $\mathcal{O}(|V(T)|)$ time, and so in $\mathcal{O}(n)$ time, a w-centroid c. Note that if $n = 2$, then T is composed of a root and of two leaves; then, a good choice for the w-centroid c is to take the root. In particular, we may assume c to be an internal node. Otherwise, $n \geq 3$, and so, since $w(T) = n$, we *must* have that c is an internal node. Then, let $a_1, a_2, \ldots, a_d$ be the children of c. We denote C (resp. A_i) the subset of vertices of which $f(c)$ (resp., $f(a_i)$) is a partition. Furthermore, let T_c (resp., let T_{a_i}) be the subtree rooted at c (resp., at a_i). By Lemma 3.4 we can bipartition the trees $T \setminus T_c,\ T_{a_1}, T_{a_2}, \ldots, T_{a_d}$ into two forests F_1, F_2 of respective total weights $\leq 2n/3$. In particular, since c is internal, and so $w(c) = 0$, both forests are non-empty. Up to re-ordering the children nodes of c, we may assume one of those forests, say F_1, to be equal to $\bigcup_{j=1}^{p} T_{a_j}$, for some $p \leq d$. For short, we name $A := \bigcup_{j=1}^{p} A_j$. Doing so, we define the cut $(A, V \setminus A)$, whose two sides can be determined in $\mathcal{O}(n)$ time by traversing the disjoint subtrees $T_{a_1}, T_{a_2}, \ldots, T_{a_p}$.

By Lemma 2.2, A is a k-module of G, with a corresponding partition being $\Phi(A) = \{X \cap A \mid X \in f(c)\}$ (or $f(a_1)$ if $p = 1$). Note that such a partition can be readily derived in $\mathcal{O}(n)$ time from either $f(c)$ (if $p > 1$) or $f(a_1)$ (if $p = 1$). In turn, being given the representation graph of (T, f), we can compute $f(c)$ and $f(a_1)$ in $\mathcal{O}(kn)$ time by traversing the subtrees rooted at nodes c and a_1. Let $X_1, X_2, \ldots, X_k$ be a partition of A s.t., for every $1 \leq i \leq k$, X_i is a module of $G \setminus (A \setminus X_i)$. Furthermore, for every $1 \leq i \leq k$, let $Y_i := N_G(X_i) \setminus A$ (neighbour sets in $V \setminus A$). Since the subsets X_i are pairwise disjoint we can compute $Y_1, Y_2, \ldots, Y_k$ in total $\mathcal{O}(m)$ time. Finally, for every $1 \leq i \leq k$, for every $v \in V$, we compute $d_G(v, X_i)$ and $d_G(v, Y_i)$. It takes $\mathcal{O}((m+n)\log n)$ time per subset, using a modified Dijkstra's algorithm, and so total time in $\mathcal{O}(k(m+n)\log n)$ (resp., if the graph is unweighted, then it takes $\mathcal{O}(m+n)$ time per subset, using a modified BFS, and so total time in $\mathcal{O}(k(m+n))$). We end up applying recursively the same procedure as above on the disjoint (possibly unconnected) subgraphs $G[A]$ and $G[V \setminus A]$. For that, we need to build a partition tree for each subgraph.

- For $G[A]$, we take $T_A = T_{a_1}$ if $p = 1$, otherwise we take $T_A = T_c \setminus (\bigcup_{j>p} T_j)$. Then, for every $b \in V(T_A)$, we set $f_A(b) = \{X \cap A \mid X \in f(b)\}$. Observe that if $b \neq c$ then $f_A(b) = f(b)$. Hence, the representation graph of (T_A, f_A) can be computed from the representation graph of (T, f) in $\mathcal{O}(kn)$ time.
- For $G[V \setminus A]$, a natural choice would be to take the subtree $T_{V\setminus A} = T \setminus (\bigcup_{j=1}^{p} T_{a_j})$. Then, for every $b \in V(T_{V\setminus A})$, we set $f_{V\setminus A}(b) = \{X \setminus A \mid X \in f(b)\}$. Again, we observe that the representation graph of $(T_{V\setminus A}, f_{V\setminus A})$ can be computed from the representation graph of (T, f) in $\mathcal{O}(kn)$ time (*i.e.*, by dynamic programming on the path from c to the root of T, removing on the way all groups fully into A). However, doing so, we may not respect all properties of a partition tree. Specifically, if $d = p$ then c has become a leaf-node and it must be removed. But then, its father node c' may have only one child b left. If that is the case, then either c' is the root of T and then we choose $T_{V\setminus A} = T_b$, or we choose the father node of c' as the new father node of b, removing on our way the node c'. Note that we do *not* modify $f_{V\setminus A}(b)$ during this procedure. Finally, if $d = p + 1$ then c only has one child a_d left. We proceed similarly as in the previous case. That is, either c was the root of T and then we set $T_{V\setminus A} = T_{a_d}$, or we choose the father node of c as the new father node of a_d, removing on our way the node c. Note that doing so, we do *not* modify the partition $f_{V\setminus A}(a_d)$.

The above procedure recursively defines a so called w-centroid decomposition $T^{(w)}$. The latter is a binary rooted tree, whose root is labeled by the cut $(A, V \setminus A)$. Its left and right subtrees are w-centroid decompositions of $G[A]$ and $G[V \setminus A]$ respectively. Note that by construction, the depth of $T^{(w)}$ is in $\mathcal{O}(\log n)$. Furthermore, there is a one-to-one mapping between the leaves of $T^{(w)}$ and the vertices of G. For every vertex $v \in V$, its label $C_G(v)$ contains each cut on its path until the root of $T^{(w)}$, and the $2k$ distances computed for each cut. – Infinite distances may be encoded as some special character. – Here, we stress that all these distances are computed in some induced subgraphs of G, and not in G itself (unless it is for the first cut, at the root). Since the depth of $T^{(w)}$ is in $\mathcal{O}(\log n)$, each $C_G(v)$ stores $\mathcal{O}(k \log n)$ distances, and so it has a bit size in $\mathcal{O}(k \log n \log(Wn))$ (resp, in $\mathcal{O}(k \log^2 n)$ if the graph is unweighted). Furthermore, as $G[A]$ and $G[V \setminus A]$ are disjoint, every recursive stage of the procedure takes $\mathcal{O}(k(n+m) \log n)$ time (resp., $\mathcal{O}(k(n+m))$ time). Hence, the total pre-processing time in order to compute $C_G(v)$, for all $v \in V$, is in $\mathcal{O}(k(n+m) \log^2 n)$ (resp., in $\mathcal{O}(k(n+m) \log n)$ if G is unweighted).

We are left describing D_G (decoding). Let $u, v \in V$ be arbitrary. Their least common ancestor in $T^{(w)}$ corresponds to some cut $(A^j, A^{j-1} \setminus A^j)$ s.t. $u \in A^j$, $v \in A^{j-1} \setminus A^j$. Consider *all* the cuts on the path between their least common ancestor and the root of $T^{(w)}$. We call the latter $(A^0, V \setminus A^0), (A^1, A^0 \setminus A^1), \ldots, (A^j, A^{j-1} \setminus A_j)$. Since up to reverting their two sides, all these cuts have neighbourhood diversity at most k, then we may apply Lemma 3.3 $j+1$ times in order to compute $d_G(u, v)$ (*i.e.*, in $G, G[A^0], G[A^1], \ldots, G[A^{j-1}]$). Note that $j = \mathcal{O}(\log n)$. Finally, since for each cut considered, the $2k$ distances that are required in order to apply this lemma are stored in $C_G(u)$ and $C_G(v)$, it takes $\mathcal{O}(k)$ time per cut, and so, the final query time is in $\mathcal{O}(k \log n)$. □

Recall that All-Pairs Shortest-Paths in an n-vertex graph of clique-width at most k can be solved in $\mathcal{O}((kn)^2)$ time [52]. As a by-product of our Theorem 3.2, we observe below that we can improve the dependency on k, but at the price of a poly-logarithmic overhead in the running time.

Corollary 3.5. *For every n-vertex vertex-weighted graph $G = (V, E, w)$, if $cw(G) \leq k$ and a k-expression is given, then we can solve All-Pairs Shortest-Paths for G in $\mathcal{O}(k(n \log n)^2)$ time (resp., in $\mathcal{O}(kn^2 \log n)$ time if G is unweighted).*

Proof. We start applying Theorem 3.2 in order to compute a distance-labeling scheme with $\mathcal{O}(k \log n)$ query time. Then, we consider all pairs $u, v \in V$ (there are $\mathcal{O}(n^2)$ such pairs) and we compute $d_G(u, v)$ in $\mathcal{O}(k \log n)$ time. □

4 Centrality indices and beyond

We refine our strategy for Theorem 3.2 in order to prove the main result of this paper:

Theorem 4.1. *For every connected n-vertex m-edge vertex-weighted graph $G = (V, E, w)$, if $cw(G) \leq k$ and a k-expression is given, then we can compute in $\mathcal{O}(2^{\mathcal{O}(k)}(n+m)^{1+\epsilon})$ time, for any $\epsilon > 0$: all the eccentricities, and all the closeness centralities.*

Corollary 4.2. *For every connected n-vertex m-edge vertex-weighted graph $G = (V, E, w)$, if $cw(G) \leq k$ and a k-expression is given, then we can compute the diameter, radius, center, Wiener index and median set of G in $\mathcal{O}(2^{\mathcal{O}(k)}(n+m)^{1+\epsilon})$ time, for any $\epsilon > 0$.*

Recall that Coudert et al. proved that assuming SETH, for any $\epsilon > 0$, there is no $\mathcal{O}(2^{o(k)}(n+m)^{2-\epsilon})$-time algorithm for computing the diameter within cubic graphs of clique-width at most k [16]. Therefore, our results for the diameter (and so, for the eccentricities) are optimal under SETH. Our results for the Wiener index (and so, for the closeness centrality) are also optimal under SETH. Indeed, since the pathwidth of a graph is an upper bound for its clique-width [31], then it follows from [1] that it is already "SETH-hard", in the unweighted case, to decide in $\mathcal{O}(2^{o(k)}(n+m)^{2-\epsilon})$ time whether the diameter is either two or three. It is well-known that $diam(G) \leq 2$ if and only if for every $v \in V$ of degree $\delta_G(v)$, $TD(v) = 2(n-1) - \delta_G(v)$ [11]. In particular, $diam(G) \leq 2$ if and only if $W(G) = 2n(n-1) - 2m$.

Additional notations. In Sec. 4.2, 4.3 and 4.4 we need to also allow *edge-weights*, due to some technicalities in our final proof of Theorem 4.1. Such a graph is denoted by $G = (V, E, w, \alpha)$, where $\alpha : E \to \mathbb{N}$. Then, the weight of a path is the sum of the weights of all its vertices and edges. The special case of vertex-weighted graphs is retrieved by setting $\alpha(e) = 0$ for every $e \in E$. Finally, we call a cut $(A, V \setminus A)$ *unweighted* if all edges between A and $V \setminus A$ have a zero weight. The neighbourhood diversity of a cut is the same in G as in the underlying unweighted graph obtained from G by removing all the weights.

4.1 Minimal partition of k-modules

First, it is not hard to show that every k-module has a partition in a least number of subsets. In what follows, we will often use a few simple properties of this minimal partitioning.

Lemma 4.3. *Every vertex-subset A in a vertex-weighted graph $G = (V, E, w)$ admits a unique partition $A_1, A_2, \ldots, A_k$ with the following two properties:*

1. *For every $1 \leq i \leq k$, for every $u_i, v_i \in A_i$, we have $N_G(u_i) \setminus A = N_G(v_i) \setminus A$. In particular, A is a k-module of G.*
2. *For every $k' < k$, A is* not *a k'-module of G.*

We call it the minimal partition of A, and it can be computed in linear time.

Proof. Let $G' = G \setminus E(A)$ be the graph obtained from G by removing all edges with their two ends in A. Two vertices are called false twins if they have exactly the same neighbours in G'. This is an equivalence relation over V, whose equivalence classes are sometimes called "twin classes". We claim that if A is a k'-module, with a corresponding partition being $A_1, A_2, \ldots, A_{k'}$, then for every $1 \leq i \leq k'$, all the vertices of A_i must belong to the same twin class. Indeed, for every $u_i, v_i \in A_i$ we get $N_{G'}(u_i) = N_G(u_i) \setminus A = N_G(v_i) \setminus A = N_{G'}(v_i)$. Then, the minimal partition of A is composed of all the non-empty intersections of A with the twin classes of G'. The twin classes of a graph can be computed in linear time by using classic partition refinement techniques [45]. □

4.2 Orthogonal range queries

We then need to recall some basics about the framework introduced in [13] by Cabello and Knauer. Let $P \subseteq \mathbb{R}^k$ be a static set of k-dimensional points. We assume each point $\overrightarrow{p} \in P$ to be assigned a value $g(\overrightarrow{p})$. A box is the Cartesian product of k intervals. Note that we allow each interval to be unbounded and/or open or partially open. Roughly, given a box $\mathcal{R}$, a range query on P asks for either reporting or counting all points in $P \cap \mathcal{R}$, or for some specific point(s) in this intersection maximizing a given objective function. Here, we consider the following types of range queries:

- (*Maximum range query*) Given some box $\mathcal{R}$, find some $\overrightarrow{p} \in P \cap \mathcal{R}$ maximizing $g(\overrightarrow{p})$;
- (*Sum range query*) Given some box $\mathcal{R}$, compute $\sum_{\overrightarrow{p} \in P \cap \mathcal{R}} g(\overrightarrow{p})$.
- (*Count range query*) Given some box $\mathcal{R}$, compute $|P \cap \mathcal{R}|$.

Lemma 4.4 ([11])**.** *For every k-dimensional point set P of size n, for any $\epsilon > 0$, we can construct in $\mathcal{O}(2^{\mathcal{O}(k)} n^{1+\epsilon})$ time, a data structure, sometimes called a k-dimensional range tree, that allows to answer any maximum range query, sum range query or count range query in $\mathcal{O}(2^{\mathcal{O}(k)} n^{\epsilon})$ time.*

In the following Lemma 4.5 we give a new simple application of Lemma 4.4 to distance problems in graphs, namely:

Lemma 4.5. *Let $G = (V, E, w, \alpha)$ be a connected n-vertex m-edge graph with respective vertex- and edge-weight functions w and α, let $(A, V \setminus A)$ be an unweighted cut of neighbourhood diversity at most k, and let $A' \subseteq A$, $B' \subseteq V \setminus A$. For any $\epsilon > 0$, after a pre-processing in $\mathcal{O}(km + 2^{\mathcal{O}(k)} n^{1+\epsilon})$ time, for every vertex*

$u \in A'$ we can compute the values $\max_{v\in B'} d_G(u,v)$ and $\sum_{v\in B'} d_G(u,v)$ in $\tilde{\mathcal{O}}(2^{\mathcal{O}(k)}n^\epsilon)$ time; in the same way, for every vertex $v \in B'$ we can compute the values $\max_{u\in A'} d_G(v,u)$ and $\sum_{u\in A'} d_G(v,u)$ in $\mathcal{O}(2^{\mathcal{O}(k)}n^\epsilon)$ time.

Proof. Let $A_1, A_2, \ldots, A_k$ be the minimal partition of A. By Lemma 4.3, we can compute it in $\mathcal{O}(m)$ time. For $1 \leq i \leq k$, let $B_i = N_G(A_i) \setminus A$. Note that since the subsets A_i are pairwise disjoint, we can compute $B_1, B_2, \ldots, B_k$ in total $\mathcal{O}(m)$ time. Observe that there is at most one index i s.t. $B_i = \emptyset$ (otherwise, we can merge all groups A_j s.t. $B_j = \emptyset$ into one, thus contradicting the minimality of the partition of A). W.l.o.g., if such index exists then it must be $i = k$. We want to exclude this index, if it exists, in order to avoid handling with arithmetic over infinite values. So, let $k' = k$ if $B_k \neq \emptyset$, otherwise let $k' = k - 1$. For every $1 \leq i \leq k'$, for every $u \in A'$, we compute $d_G(u, A_i)$. In the same way, for every $1 \leq i \leq k'$, for every $v \in B'$, we compute $d_G(B_i, v)$. It takes $\mathcal{O}(k'm \log n) = \mathcal{O}(km \log n)$ time in total if we use Dijkstra's single-source shortest-path algorithm. Then, for every $v \in B'$ and for every $1 \leq i \leq k'$, we create a k'-dimensional point $\overrightarrow{p}(v,i)$: whose first coordinate is the index i, followed by the values $d_G(B_i, v) - d_G(B_j, v)$, $1 \leq j \leq k'$, $j \neq i$. Set $g(\overrightarrow{p}(v,i)) = d_G(B_i, v)$. Finally, let P contain all these $k'|B'|$ points. We add all points in P into some k'-dimensional range tree, that takes $\mathcal{O}(2^{\mathcal{O}(k)}n^{1+\epsilon})$ time for any $\epsilon > 0$ by Lemma 4.4.

Now, let $u \in A'$ be fixed, and assume that we want to compute the values $\max_{v\in B'} d_G(u,v)$ and $\sum_{v\in B'} d_G(u,v)$. Note that if we subdivide each edge e with positive weight and we assign weight $\alpha(e)$ to the resulting vertex then, $A \cup E(A)$ is a $(k'+1)$-module in the resulting graph, with $E(A)$ representing the edges in G with their both ends in A (no such vertex has a neighbour in the other side of the cut). By Lemma 3.3 (applied to the resulting graph), for every $v \in B'$, we have $d_G(u,v) = \min\{d_G(u, A_i) + d_G(B_i, v) \mid 1 \leq i \leq k\}$. Since $d_G(B_k, v) = +\infty$ if $B_k = \emptyset$, we also have $d_G(u,v) = \min\{d_G(u, A_i) + d_G(B_i, v) \mid 1 \leq i \leq k'\}$. We (virtually) partition B' into $C_1, C_2, \ldots C_{k'}$ so that, for every $1 \leq i \leq k'$, $v \in C_i$ if and only if the least index j s.t. $d_G(u,v) = d_G(u, A_j) + d_G(B_j, v)$ is equal to i. Specifically, we design boxes $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_{k'}$ so that $p(v,j) \in \mathcal{R}_i \iff j = i$ and $v \in C_i$. Note that if we can do so, then:

$$\begin{aligned}
\max_{v\in B'} d_G(u,v) &= \max_{1\leq i\leq k'} \max_{v\in C_i} d_G(u,v) \\
&= \max_{1\leq i\leq k'} \left(d_G(u, A_i) + \max\{d_G(B_i, v) \mid v \in C_i\}\right) \\
&= \max_{1\leq i\leq k'} \left(d_G(u, A_i) + \max\{g(\overrightarrow{p}(v,j)) \mid \overrightarrow{p}(v,j) \in \mathcal{R}_i\}\right).
\end{aligned}$$

In particular, we are left doing k' maximum range queries. In the same way:

$$\begin{aligned}
\sum_{v\in B'} d_G(u,v) &= \sum_{i=1}^{k'} \sum_{v\in C_i} d_G(u,v) \\
&= \sum_{i=1}^{k'} \sum_{v\in C_i} \left(d_G(u, A_i) + d_G(B_i, v)\right) \\
&= \sum_{i=1}^{k'} \left[d_G(u, A_i) \cdot |C_i| + \sum_{v\in C_i} d_G(B_i, v)\right] \\
&= \sum_{i=1}^{k'} \left[d_G(u, A_i) \cdot |P \cap \mathcal{R}_i| + \sum \{g(\overrightarrow{p}(v,j)) \mid \overrightarrow{p}(v,j) \in \mathcal{R}_i\}\right].
\end{aligned}$$

In particular, we are left doing k' sum range queries and k' count range queries. Hence, being given $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_{k'}$, we are done in $\mathcal{O}(2^{\mathcal{O}(k)}n^\epsilon)$ time for any $\epsilon > 0$ by Lemma 4.4.

For every $1 \leq i \leq k'$, the box $\mathcal{R}_i$ is defined as follows. Let $\overrightarrow{p} = (p_1, p_2, \ldots, p_{k'})$ be a k'-dimensional point

of P. We have $\overrightarrow{p} \in \mathcal{R}_i$ if and only if:

$$\begin{cases} p_1 = i \\ \forall 1 \leq j \leq i-1,\ p_{j+1} < (d_G(u, A_j) - d_G(u, A_i)) \\ \forall i+1 \leq j \leq k',\ p_j \leq (d_G(u, A_j) - d_G(u, A_i)) \,. \end{cases}$$

Indeed, we have: $d_G(u, A_i) + d_G(B_i, v) \leq d_G(u, A_j) + d_G(B_j, v)$ if and only if $d_G(B_i, v) - d_G(B_j, v) \leq (d_G(u, A_j) - d_G(u, A_i))$. Furthermore, by construction, if $j < i$ then $d_G(B_i, v) - d_G(B_j, v)$ is exactly the $(j+1)^{th}$ coordinate of $\overrightarrow{p}(v, i)$ (in which case we want the inequality to be strict by the definition of C_i), and if $j > i$ then $d_G(B_i, v) - d_G(B_j, v)$ is exactly the j^{th} coordinate of this point. For the vertices $v \in B'$, we proceed similarly as above, that is, we create a point-set P' from A' and we put them in some separate k'-dimensional range tree. □

4.3 Distance-preservers with weighted edges

Our next objective consists in adding some weighted subsets to the two sides of a cut in order to preserve the distances from the original graph. Recall that for every two subsets X and Y, $d_G(X, Y) = \min_{x \in X, y \in Y} d_G(x, y)$. Our construction below is inspired by Cunningham's split decomposition [21].

Definition 4.1. Given $G = (V, E, w, \alpha)$ connected, with respective vertex- and edge-weight functions w and α, let $(A, V \setminus A)$ be an unweighted cut of neighbourhood diversity at most k. Let $A_1, A_2, \ldots, A_k$ be the minimal partition of A and, for every $1 \leq i \leq k$, let $B_i = N_G(A_i) \setminus A$. W.l.o.g., either all the B_i's are nonempty, or B_k is the unique empty set amongst the B_i's. We set $k' = k$ if $B_k \neq \emptyset$, and $k' = k - 1$ otherwise.

- For every $1 \leq i \leq k'$, let $b_{ii} \in B_i$ be of minimum weight. For every $1 \leq i < j \leq k'$, let also $b_{ij} \in B_i$, $b_{ji} \in B_j$ be the ends of a shortest B_iB_j-path (possibly, $b_{ij} = b_{ji}$). The graph H_A is obtained from $G[A \cup \{b_{ij} \mid 1 \leq i, j \leq k'\}]$ by adding, for every $1 \leq i < j \leq k'$ s.t. $b_{ij} \neq b_{ji}$, an edge $b_{ij}b_{ji}$ of weight $d_G(B_i, B_j) - w(b_{ij}) - w(b_{ji})$.
- For every $1 \leq i \leq k'$, let $a_{ii} \in A_i$ be of minimum weight. For every $1 \leq i < j \leq k'$, let also $a_{ij} \in A_i$, $a_{ji} \in A_j$ be the ends of a shortest A_iA_j-path. The graph H_B is obtained from $G[(V \setminus A) \cup \{a_{ij} \mid 1 \leq i, j \leq k'\}]$ by adding, for every $1 \leq i < j \leq k'$, an edge $a_{ij}a_{ji}$ of weight $d_G(A_i, A_j) - w(a_{ij}) - w(a_{ji})$.

Below, we observe that it is rather straightforward to compute these two above subgraphs H_A and H_B in parameterized almost linear time:

Lemma 4.6. *Given $G = (V, E, w, \alpha)$ connected, with respective vertex- and edge-weight functions w and α, let $(A, V \setminus A)$ be an unweighted cut of neighbourhood diversity at most k. The gadget subgraphs H_A and H_B (see Definition 4.1) can be constructed in $\tilde{\mathcal{O}}(k^2 n + km)$ time.*

Proof. Consider all the subsets A_i, B_i, as they were defined in Definition 4.1. As we already observed in the proof of Lemma 4.5, these $2k$ subsets can be created in $\mathcal{O}(m)$ time by using partition refinement techniques. Then, it suffices to compute the vertices a_{ij}, b_{ij}, and the distances $d_G(A_i, A_j), d_G(B_i, B_j)$. For every fixed i, we choose for b_{ii} any vertex of minimum weight in B_i. Then, we execute a modified Dijkstra's single-source shortest-path algorithm in order to compute $d_G(v, B_i)$ for every $v \in V$. It takes $\tilde{\mathcal{O}}(m)$ time. For every $j > i$, let b_{ji} be a vertex of B_j minimizing its distance to B_i. We choose for b_{ij} a closest vertex to b_{ji} in B_i. It can be done in total $\mathcal{O}(kn)$ time (for all $j > i$) by dynamic programming on the shortest-path forest output by our modified Dijkstra's algorithm. We do the same in order to compute the vertices a_{ij}, a_{ji}. □

The following two properties are crucial in our proofs of Theorem 4.1.

Lemma 4.7. *Given $G = (V, E, w, \alpha)$ connected, with respective vertex- and edge-weight functions w and α, let $(A, V \setminus A)$ be an unweighted cut of neighbourhood diversity at most k. Let H_A, H_B and k be as in Definition 4.1. Then, for every $u, v \in A$ we have $d_G(u, v) = d_{H_A}(u, v)$. Similarly, for every $u, v \notin A$ we have $d_G(u, v) = d_{H_B}(u, v)$.*

Proof. We only detail the proof for $u, v \in A$. First, we prove that $d_{H_A}(u,v) \leq d_G(u,v)$. Indeed, if there exists a uv-path of weight $d_G(u,v)$ which is fully into A, then this path also exists in H_A. Otherwise, every shortest uv-path in G must intersect $V \setminus A$. Let us fix a shortest uv-path P in G. We scan P from u until we find the first edge xy s.t. $x \in A, y \notin A$. Similarly, we scan P from v until we find the first edge st s.t. $s \in A,\ t \notin A$. There exist i, j s.t. $x \in A_i, s \in A_j$, and so, $y \in B_i, t \in B_j$. We have $d_G(y,t) \geq d_G(B_i, B_j)$, and this is in fact an equality because P is a shortest uv-path and there are complete joins between A_i and B_i, respectively between A_j and B_j. Then, we may replace all the yt-subpath in P by either the edge $b_{ij}b_{ji}$ or (if $b_{ij} = b_{ji}$, in particular if $i = j$) simply by b_{ij}. Doing so, we obtain a uv-path of H_A of weight equal to $d_G(u,v)$. Conversely, we prove that $d_{H_A}(u,v) \geq d_G(u,v)$. Indeed, consider any uv-path P' of H_A. We can replace every edge $b_{ij}b_{ji} \in E(P')$ by a shortest $b_{ij}b_{ji}$-path in G. Doing so, we transform P' into a uv-path of G without changing the weight. The proof for $u, v \notin A$ is similar as what we did above. □

Our approach only works for unweighted cuts or more generally for cuts such that all the edge-weights are equal. Indeed, let us consider a cut $(A, V \setminus A)$ of neighbourhood diversity at most k such that, for some $1 \leq i \leq k$, there exist edges of different weights between A_i and B_i. Then, it is no more true that for every shortest uv-path, with $u, v \in A$ arbitrary, for any two consecutive edges xy, ts of the cut, with $x \in A_i$ and $s \in A_j$ for some $1 \leq j \leq k$, we must always have $d(y,t) = d(B_i, B_j)$. It implies that Lemma 4.7 does not hold for arbitrary cuts. In particular, if we want to apply the procedure of Definition 4.1 recursively, for some cuts in the gadget subgraphs H_A and H_B, then we must have both ends of each weighted edge on a same side of the cut. The next lemma shows that restricting ourselves to such cuts does not cause an explosion of their neighbourhood diversity.

Lemma 4.8. *Given $G = (V, E, w, \alpha)$ connected, with respective vertex- and edge-weight functions w and α, let $(A, V \setminus A)$ be an unweighted cut of neighbourhood diversity at most k. Let H_A, H_B and k be as in Definition 4.1.*

1. *For any $A' \subseteq A$, if A' is a k-module of G then it is a k-module of H_A.*
2. *For any $B' \subseteq V \setminus A$, if B' is a k-module of G then it is a k-module of H_B; if $A \cup B'$ is a k-module of G then $B' \cup \{a_{ij} \mid 1 \leq i, j \leq k'\}$ is a k-module of H_B.*

Proof. Let $A_1, A_2, \ldots, A_k$ and $B_1, B_2, \ldots, B_k$ be as in Definition 4.1. By minimality of the partition of A, there are no two indices i and j s.t. $A_i \cup A_j$ is a module of $G \setminus (A \setminus (A_i \cup A_j))$ (otherwise, we could have merged these two groups into one). We prove the properties of the lemma separately.

- Let us first assume that $A' \subseteq A$ is a k-module of G. Clearly, A' is also a k-module of $G[A \cup \{b_{i,j} \mid 1 \leq i, j \leq k'\}]$. Furthermore, the only edges of $E(H_A) \setminus E(G)$ are those $b_{ij}b_{ji}$, and we always have $b_{ij}, b_{ji} \in V(H_A) \setminus A'$. As a result, we obtain $N_{H_A}(u) \setminus A' = (N_G(u) \cap V(H_A)) \setminus A'$, and so, A' keeps the property of being a k-module in H_A.
- In the same way, let us now assume that $B' \subseteq V \setminus A$ is a k-module of G. The subset B' is also a k-module of the subgraph $G[(V \setminus A) \cup \{a_{ij} \mid 1 \leq i, j \leq k'\}]$. Furthermore, the only edges of $E(H_B) \setminus E(G)$ are those $a_{ij}a_{ji}$, and we always have $a_{ij}, a_{ji} \in V(H_B) \setminus B'$. Hence, the same as before, B' keeps the property of being a k-module in H_B.
- Finally, let $B' \subseteq V \setminus A$ be s.t. $B' \cup A$ is a k-module of G. In particular, $B' \cup \{a_{ij} \mid 1 \leq i, j \leq k'\} \subseteq B' \cup A$ is a k-module of $G[(V \setminus A) \cup \{a_{ij} \mid 1 \leq i, j \leq k'\}]$. We already mentioned that the only edges of $E(H_B) \setminus E(G)$ are those $a_{ij}a_{ji}$. Since we always have $a_{ij}, a_{ji} \in B' \cup \{a_{ij} \mid 1 \leq i, j \leq k'\}$, the subset $B' \cup \{a_{ij} \mid 1 \leq i, j \leq k'\}$ keeps the property to be a k-module in H_B.

□

4.4 Proofs of the main results

Proof of Theorem 4.1. We revisit the scheme of Theorem 3.2. That is, we fix some width-k partition tree (T, f), that takes $\mathcal{O}(k(n+m))$ time by using Lemma 2.1. Furthermore, we pre-process the tree T in order to compute in $\mathcal{O}(1)$ time, for any two nodes $a, a' \in V(T)$, their least common ancestor; it can be done in $\mathcal{O}(n)$ time [48]. Finally, let $w : V(T) \to \{0,1\}$ be s.t. $w(a) = 1$ if and only if a is a leaf. In what follows, we mimic the recursive construction of a w-centroid decomposition $T^{(w)}$ of T, as it was defined in the proof of Theorem 3.2.

The algorithm. We consider a more general problem for which we are given some tuple $\langle r, H, U, T^U, f^U, \mathcal{L}\rangle$. Let us detail each of the components of this input:

1. Here, H is a weighted graph with non-negative real vertex-weights and with non-negative real edge-weights (initially, $H = G$). Roughly, H is a supergraph of some induced subgraph of G which is augmented with additional vertices and weighted edges in order to preserve the distances in G.

2. The value r represents the recursion level of the algorithm (initially, $r = 0$), that is roughly the number of cuts of G traversed by the algorithm. Note that the order of the graphs H considered is decreasing exponentially with r (see the complexity analysis at the end of the proof).

3. The vertex-subset U is such that $U \subseteq V \cap V(H)$ (initially, $U = V$). We further impose to have $H[U] = G[U]$, and that for every $u, v \in U$ we have $d_G(u, v) = d_H(u, v)$. In particular, all the edges of $H[U]$ are unweighted. The objective of the algorithm is to compute, for every vertex of U, its maximum distance in G to a vertex of U, respectively the sum of all its distances in G to the vertices of U. For that, intuitively, we embed U in the vertex- and edge-weighted graph H.

4. The rooted tree (T^U, f^U) is a width-k-partition tree of $G[U]$ (initially, $T^U = T$ and $f^U = f$). We further assume that T^U was constructed from a rooted subtree of T by repeatedly contracting internal nodes with only one child. Roughly, (T^U, f^U) is just a compression of (T, f) where we iteratively removed useless branches and contracted degree-one nodes so that the resulting tree stays of order linear in the size of the input subset U. In particular, all the ancestor-descendant relations in T^U are also ancestor-descendant relations in T. Furthermore, for every node $b \in V(T^U)$ we impose $f^U(b) = \{X \cap U \mid X \in f(b) \text{ and } X \cap U \neq \emptyset\}$. Note that in lieu of (T^U, f^U), we are given the representation graph of this partition tree (as defined in Sec. 2). Throughout the algorithm, we use partition trees in order to compute edge-cuts, from which we recursively partition the vertex-set of G. Roughly, since we compute all these cuts indirectly from the same fixed partition tree (T, f) of G, we ensure that all these cuts are pairwise non-crossing (two cuts are crossing if any side of one cut intersects any side of the other cut). This non-crossing property allows us to reinterpret every cut of G considered as an unweighted balanced cut of some vertex- and edge-weighted graph H.

5. Finally, $H \setminus U$ is a disjoint union of $r' \leq r$ subgraphs of order $\mathcal{O}(k^2)$, that we shall name "clusters" in what follows. Roughly, each cluster is a substitution gadget for one side of some cut of G that was already considered at some earlier recursion level. To each cluster W_i, we associate some node c_i of the original tree T. Roughly, c_i corresponds to some balanced cut, computed at an earlier recursive stage, and the cluster W_i resulted from the procedure of Definition 4.1 applied to this cut. So, in particular, we impose that any edge between two vertices that are on different clusters (resp., between a vertex in a cluster and a vertex of U) must be unweighted. All the pairs (W_i, c_i) are stored in the list $\mathcal{L}$ (initially, $\mathcal{L}$ is the empty list).

The output of the algorithm is, for every $u \in U$, the values $\max_{v \in U} d_H(u, v)$ and $\sum_{v \in U} d_H(u, v)$.

For that, let $n_r := |V(H)|$ and $m_r := |E(H)|$. We may assume that $|U| \geq \lambda k^2 \log n$, for some sufficiently large constant λ. Indeed, if it not the case then we may compute by brute-force all the desired values. Our algorithm has at most $\mathcal{O}(\log n)$ recursive stages, and therefore, in this case we have $n_r = |U| + \mathcal{O}(k^2 \log n) = \mathcal{O}(k^2 \log n)$. In particular, we can perform the brute-force computation in $\mathcal{O}(k^6 \log^3 n)$ time (base case of the recursion).

Thus from now on, let us assume $|U| = \Omega(k^2 \log n)$. We compute a w-centroid c in T^U. This can be done in $\mathcal{O}(|V(T^U)|) = \mathcal{O}(n_r)$ time. Since $w(T^U) = |U| > 3$, this node c cannot be a leaf. Let $a_1, a_2, \ldots, a_d$ be the children of c. As before, we denote by C (resp. A_i) the subset of vertices of which $f^U(c)$ (resp., $f^U(a_i)$) is a partition, and by T^U_c (resp., $T^U_{a_i}$) the subtree rooted at c (resp., at a_i). Here, we stress that $C \subseteq U$ (resp., $A_i \subseteq U$). By using Lemma 3.4, we may partition $T^U \setminus \{c\}$ in two non-empty forests of respective weights $\leq 2|U|/3$. Furthermore, we may assume one of our two forests to contain exactly $T^U_{a_1}, T^U_{a_2}, \ldots, T^U_{a_p}$ for some $p \leq d$. Then, let $A = \bigcup_{j=1}^{p} A_j$ (computable in $\mathcal{O}(|U|) = \mathcal{O}(n_r)$ time by traversal of T^U). We compute the following cut of H:

- The subsets A and $U \setminus A$ are on separate sides of the cut.
- For every $(W_j, c_j) \in \mathcal{L}$, there are two cases. If there exists some index i s.t. the least common ancestor of c_j and a_i in T is a *strict* descendant of c (a child of c in T, or a descendant of one of these children), then we put W_j on the same side of the cut as A. Otherwise, we put W_j on the same side of the cut as $U \setminus A$.

Let us give some intuition for both cases above. Roughly, we mimic the computation of some cut of G which disconnects A from $U \setminus A$. Both sides of this cut correspond to some subtrees T_1, T_2 of T. Note in particular that T_1 is the smallest subtree of T containing c and $T_{a_1}, T_{a_2} \ldots, T_{a_p}$. Similarly, each cluster of H is a substitution gadget for one side of some cut of G, already considered at an earlier recursion level, and as a result it can also be mapped to some subtree of T. We put a cluster on the same side as A (resp., as $U \setminus A$) if and only if its corresponding subtree in T is a subtree of T_1 (resp., of T_2). Note that, for each $(W_j, c_j) \in \mathcal{L}$, we can decide in which case we are as follows. For every $1 \leq i \leq p$, we compute the least common ancestor s_i of c_j and a_i in T. Then, for every $1 \leq i \leq p$, we compute the least common ancestor of s_i and c in T. Given the pre-computed least-common ancestor data structure for T, this can be done in total $\mathcal{O}(p)$ time, and so in $\mathcal{O}(|U|) = \mathcal{O}(n_r)$ time per cluster. Overall, since we have $|\mathcal{L}| = r' = \mathcal{O}(\log n)$, we can compute this above cut in $\mathcal{O}(n_r \log n)$ time. Let $(A', V(H) \setminus A')$ be this cut, where $A \subseteq A'$. By construction, it is unweighted. We prove below (see the Correctness part of the proof) that A' is a k-module of H. Then, we apply Lemma 4.5 in order to compute, for every $u \in A$, the values $\max_{v \in U \setminus A} d_H(u, v)$ and $\sum_{v \in U \setminus A} d_H(u, v)$ (resp., for every $v \in U \setminus A$, the values $\max_{u \in A} d_H(v, u)$ and $\sum_{u \in A} d_H(v, u)$). It takes $\mathcal{O}(2^{\mathcal{O}(k)}(n_r + m_r)^{1+\epsilon})$ time for any $\epsilon > 0$.

We are left computing for every $u \in A$, the values $\max_{u' \in A} d_H(u, u')$ and $\sum_{u' \in A} d_H(u, u')$ (resp., for every $v \in U \setminus A$, the values $\max_{v' \in U \setminus A} d_H(v, v')$ and $\sum_{v' \in U \setminus A} d_H(v, v')$). For that, we construct the gadget subgraphs H_A and H_B, as in Definition 4.1. By Lemma 4.6, it can be done in $\tilde{\mathcal{O}}(k^2 n_r + k m_r)$ time. Let (T^A, f^A) and (T^B, f^B) be width-k partition trees of $G[A]$ and $G[U] \setminus A$. Recall (see the proof of Theorem 3.2) that the trees T^A and T^B can be computed in $\mathcal{O}(|U|)$ time from T^U as follows: we start with $T^U_c \setminus \left(\bigcup_{i>p} T^U_{a_i}\right)$ and $T^U \setminus \left(\bigcup_{i=1}^{p} T^U_{a_i}\right)$, then we remove useless leaves and/or we repeatedly contract internal nodes with only one child. The corresponding partition function f^A, resp. f^B, is obtained from f^U by removal in the partition at each node of all the vertices out of A, resp. by removal of all the vertices in A. Hence, being given the representation graph of (T^U, f^U), the representation graphs of (T^A, f^A) and (T^B, f^B) can be computed in $\mathcal{O}(k|U|)$ time. Let $\mathcal{L}_A$ contain every $(W_j, c_j) \in \mathcal{L}$ s.t. $W_j \subseteq A'$; we also add in $\mathcal{L}_A$ a new cluster $(V(H_A) \setminus A', c)$. In the same way, let $\mathcal{L}_B$ contain every $(W_j, c_j) \in \mathcal{L}$ s.t. $W_j \subseteq V(H) \setminus A'$; we also add in $\mathcal{L}_B$ a new cluster $(V(H_B) \setminus B', c)$, where $B' = V(H) \setminus A'$. We end up calling our algorithm recursively for the inputs $\langle r+1, H_A, A, T^A, f^A, \mathcal{L}_A \rangle$ and $\langle r+1, H_B, U \setminus A, T^B, f^B, \mathcal{L}_B \rangle$.

Correctness. There are two properties to check in order to prove the validity of our approach. The first such property is that, being given the two gadget subgraphs H_A and H_B resulting from H, the distances in H (and so, in G) are preserved. This follows from Lemma 4.7. The second property to be checked is that we always compute a cut $(A', V(H) \setminus A')$ of neighbourhood diversity at most k. We prove it by induction on r. Specifically, we prove the following slightly stronger property.

Property 1. *For every $\langle r, H, U, T^U, f^U, \mathcal{L} \rangle$, let $s_1, s_2, \ldots, s_q$ be children of some node s in T^U. Let S_i be the subset of U of which $f^U(s_i)$ is a partition, and set $S = \bigcup_{i=1}^{q} S_i$. Finally, let S' be the union of S with*

all subsets W_j, for $(W_j, c_j) \in \mathcal{L}$, s.t. the least common ancestor in T of c_j and some node s_i is a strict descendant of s. Then, S' is a k-module of H.

If $r = 0$, then since $\mathcal{L} = \emptyset$ this directly follows from Lemma 2.2. Let us assume the property to be true for $\langle r, H, U, T^U, f^U, \mathcal{L}\rangle$. In what follows, we analyse the cuts in H_A and H_B, respectively.

(*Gadget subgraph H_A*). Let $s_1, s_2, \ldots, s_q$ be children nodes of some s in T^A. Observe that T^A is a subtree of T^U (equal to either $T^U_c \setminus \left(\bigcup_{i>p} T^U_{a_i}\right)$, or $T^U_{a_1}$ if $p = 1$). In particular, $s_1, s_2, \ldots, s_q$ are also children nodes of s in T^U. Let $W_{r'+1} := V(H_A) \setminus A'$ be the only cluster of $\mathcal{L}_A$ that is not contained in $\mathcal{L}$ (constructed using the procedure of Definition 4.1 in order to create H_A). We define S'_0 as the union of S with all the clusters W_j, for $(W_j, c_j) \in \mathcal{L}$, s.t. the least common ancestor in T of c_j and some node s_i is a strict descendant of s. By the induction hypothesis, S'_0 is a k-module of H.

Claim 4.9. *For every $(W_j, c_j) \in \mathcal{L}$, we have $W_j \subseteq S'_0 \Longrightarrow (W_j, c_j) \in \mathcal{L}_A$.*

Proof. Since s is a node of T^A it is a descendant of c. There are two cases, depending on whether $s = c$.

- If $s \neq c$ (see Fig. 1), then s is a descendant of some a_i, for $1 \leq i \leq p$ (possibly, $s = a_i$). In this situation, c_j is a strict descendant of s, and so of a_i.

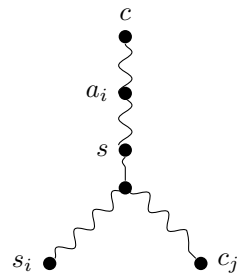

Figure 1: To the proof of Claim 4.9. Case $s \neq c$.

- Otherwise, $s = c$ (see Fig. 2). Then, the nodes $s_1, s_2, \ldots, s_q$ must be a subset of the nodes $a_1, a_2, \ldots, a_p$. This implies that the least common ancestor of c_j and some a_i is a strict descendant of $s = c$.

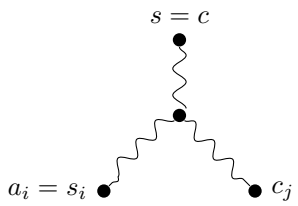

Figure 2: To the proof of Claim 4.9. Case $s = c$.

Therefore, in both cases, there exists an i s.t. c_j and a_i have a least common ancestor in T which is a strict descendant of c. ◇

Then, by Claim 4.9, $S'_0 \subseteq A'$. By Lemma 4.8, S'_0 is a k-module of H_A. Finally, since $(W_{r'+1}, c) \in \mathcal{L}_A$ and *all* nodes of T^A are descendants of c, we get $W_{r'+1} \not\subseteq S'$, and so, $S' = S'_0$.

(*Gadget subgraph H_B*). Let $s_1, s_2, \ldots, s_q$ be children nodes of some s in T^B. By construction, in T^U, the node s is a common ancestor of all the nodes $s_1, s_2, \ldots, s_q$ (it may not be their father node since we possibly

contracted internal nodes in order to create T^B). Let $W_{r'+1} := V(H_B) \setminus B'$ be the only cluster of $\mathcal{L}_B$ that is not contained in $\mathcal{L}$ (constructed using the procedure of Definition 4.1 in order to create H_B). In our analysis below, we will often use the following observation: when creating T^B from $T^U \setminus \left(\bigcup_{i=1}^{p} T^U_{a_i}\right)$ only two nodes may be removed, namely, c (if it has at most one child left) or its father node in T^U (if c becomes a leaf and it has exactly one sibling in T^U). There are now two cases to be considered.

- We first assume that, for every $1 \leq i \leq q$, the least common ancestor of c and s_i is an ancestor of s (possibly, s itself). In particular, $f^B(s_i) = f^U(s_i)$.

  Claim 4.10. *$s_1, s_2, \ldots, s_q$ are also children nodes of s in T^U.*

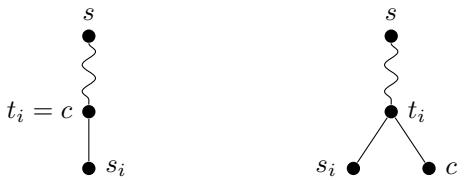

Figure 3: The two cases of Claim 4.10.

Proof. Suppose for the sake of contradiction that s is not the father of s_i, for some $1 \leq i \leq q$. In particular, the original father node of s_i, let us call it t_i, got removed when we created T^B (see Fig. 3 for an illustration). But then, t_i should be either c, or the father node of c in T^U. As a result, s_i and c would have a least common ancestor in T which is a strict descendant of s, a contradiction. ◇

The remainder of the proof is now essentially the same as what we did above for the gadget subgraph H_A. Specifically, let S'_0 be the union of S with all the clusters W_j, for $(W_j, c_j) \in \mathcal{L}$, s.t. the least common ancestor in T of c_j and some node s_i is a strict descendant of s. By the induction hypothesis, S'_0 is a k-module of H. Furthermore, the following result (similar to Claim 4.9) is true:

Claim 4.11. *If $(W_j, c_j) \in \mathcal{L}$ is s.t. $W_j \subseteq S'_0$, then $(W_j, c_j) \in \mathcal{L}_B$.*

Proof. Suppose for the sake of contradiction $(W_j, c_j) \in \mathcal{L}_A$. In particular, for some $1 \leq i' \leq p$, c_j and $a_{i'}$ have a common ancestor in T which is a strict descendant of c. There also exists an $1 \leq i \leq q$ s.t. c_j and s_i have a common ancestor which is a strict descendant of s. Since both s and c are ancestors of c_j, one of these two nodes is an ancestor of the other. But s cannot be a strict ancestor of c (otherwise, the least common ancestor of s_i and c would be a strict descendant of s). Therefore, c is an ancestor of s. Then, we consider two sub-cases.

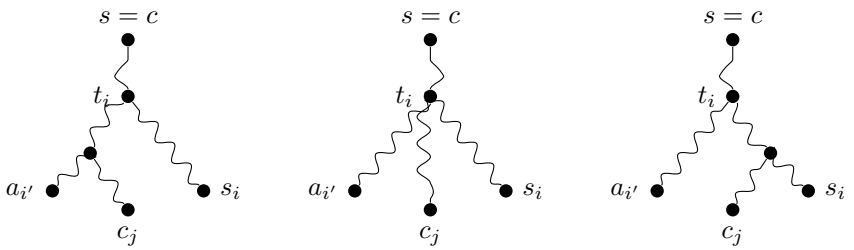

Figure 4: To the proof of Claim 4.11. Case $s = c$.

– First, let us assume $s = c$ (see Fig. 4). Observe that $s_i \neq a_{i'}$ (otherwise, $s_i \notin V(T^B)$). Then, by Claim 4.10, s_i and $a_{i'}$ are sibling nodes in T^U. Recall that the least common ancestor of c_j and s_i in T, resp. of c_j and $a_{i'}$ in T, must be a strict descendant of $s = c$. As a result, the least common ancestor of s_i and $a_{i'}$ in T, call it t_i, must be also a strict descendant of c in T. This implies that t_i got removed at some earlier recursive stage. But this is impossible, because at the stage when t_i got removed, this node still had at least two children (being ancestors of s_i and $a_{i'}$, respectively). A contradiction.

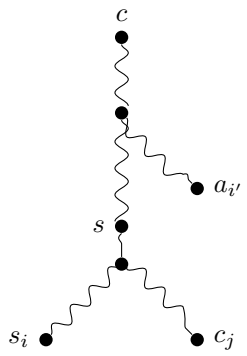

Figure 5: To the proof of Claim 4.11. Case $s \neq c$.

– From now on, we assume $s \neq c$ (see Fig. 5). We further observe that $a_{i'}$ cannot be a descendant of s (*i.e.*, because $s \in V(T^U)$ and $a_{i'}$ is a child of c in T^U). Therefore, the least common ancestor of $a_{i'}$ and c_j should be on the sc-path in T. In fact, this least common ancestor must be $a_{i'}$ itself (otherwise, the least common ancestor of s and $a_{i'}$ would be a strict descendant of c, that still exists in T^U because it has at least two children, thus contradicting again that $a_{i'}$ is a child of c in T^U). In particular, since $a_{i'}$ is onto the sc-path in T, $s \in T^U_{a_{i'}}$. But then, it contradicts our assumption that $s \in T^B$.

Summarizing, in both sub-cases we derive a contradiction. ◇

By the above Claim 4.11, $S'_0 \subseteq V(H) \setminus A'$. Hence, by Lemma 4.8, S'_0 is also a k-module of H_B. Observe that $S'_0 \subseteq S' \subseteq S'_0 \cup W_{r'+1}$. Finally, since we have $(W_{r'+1}, c) \in \mathcal{L}_B$ *and* by the hypothesis, no s_i has a least common ancestor with c which is a strict descendant of s, we *cannot* have $W_{r'+1} \subseteq S'$. As a result, $S' = S'_0$.

- Otherwise, let us assume w.l.o.g. that the least common ancestor of c and s_1 in T is a strict descendant of s. Let us call it t_1.

Claim 4.12. *t_1 is a child of s in T^U.*

Proof. There are two sub-cases (see Fig. 6). First, let us assume $s_1 = t_1$. If s_1 is not a child of s in T^U then its former father node, call it s'_1, got removed when we created T^B. Then, either $s'_1 = c$, or s'_1 is the father of c in T^U. In both cases, this contradicts our assumption that s_1 is an ancestor of c in T. Thus, from now on, let us assume $t_1 \neq s_1$. Since the father of s_1 in T^B is s, t_1 got removed at some earlier recursive stage. In fact, this must be when we created T^B because we have $s_1, c \in V(T^U)$ (otherwise, if it were done earlier, we could have not removed t_1 since it still had at least two children). Then again, either $t_1 = c$, or t_1 is the father of c in T^U. Suppose for the sake of contradiction that s is not the father of t_1 in T^U. Then, at least two nodes got removed from $T^U_c \setminus \left(\bigcup_{i=1}^{p} T^U_{a_i}\right)$ in order to create T^B. This can happen only if c became a leaf, and then the two nodes removed must be c and its father in T^U. But then, we should have $t_1 = c$, that contradicts the fact that c became a leaf. ◇

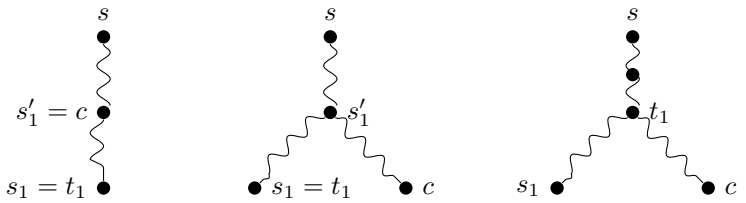

Figure 6: The different sub-cases in the proof of Claim 4.12.

We can also prove, as another intermediate claim (similar to the above Claim 4.12), that every node $s_i,\ i > 1$, is a child of s in T^U. Indeed, if it were not the case for some s_i then its father node t_i in T^U got removed when we created T^B. We either have $t_i = c$ or t_i is the father of c in T^U. In particular, t_1 is an ancestor of t_i because it is also an ancestor of c and (by Claim 4.12) a child of s in T^U (recall that s_i, and so, t_i is a descendant of s). However, since $s_1, s_i \in V(T^B)$, this would contradict the removal of t_1 from T^B. Overall, we proved as claimed that t_1 and $s_2, s_3, \ldots, s_q$ are children of s in T^U. In particular, $f^U(t_1) = A \cup S_1$, while for every $2 \leq i \leq q$, $f^U(s_i) = f^B(s_i) = S_i$. Let S'_0 be the union of $A \cup S$ with all the clusters W_j, for $(W_j, c_j) \in \mathcal{L}$, s.t. the least common ancestor in T of c_j and some node s_i is a strict descendant of s. Note that the least common ancestor of c_j and s_1 is a strict descendant of s if and only if the least common ancestor of t_1 and c_j also is. Therefore, by the induction hypothesis, S'_0 is a k-module of H. Furthermore,

Claim 4.13. Every $(W_j, c_j) \in \mathcal{L} \cap \mathcal{L}_A$ *satisfies* $W_j \subseteq S'_0$.

Proof. We refer to Fig. 7 for an illustration. If $(W_j, c_j) \in \mathcal{L} \cap \mathcal{L}_A$, then there is an $1 \leq i' \leq p$ s.t. the least common ancestor of c_j and $a_{i'}$ is a strict descendant of c. In particular, the least common ancestor of s_1 and c_j is a strict descendant of s. ◇

We get by Claim 4.13 that $A' \subseteq S'_0$. Let $B' = S'_0 \setminus A'$. Since $A' \cup B'$ is a k-module of H, by Lemma 4.8, $W_{r'+1} \cup B'$ is a k-module of H_B. Finally, we observe that $S' = W_{r'+1} \cup B'$.

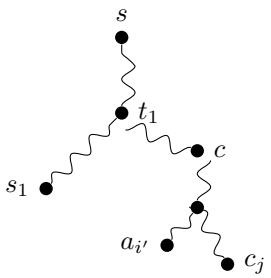

Figure 7: To the proof of Claim 4.13.

Complexity analysis. By induction, for every $r \geq 0$, for every $\langle r, H, U, T^U, f^U, \mathcal{L} \rangle$, we have $|U| \leq (2/3)^r n$. In particular, the depth of the recursion tree is $\mathcal{O}(\log n)$ (as it was anticipated when we presented above the algorithm). Furthermore, for any fixed r, if we consider the sets U of all the inputs $\langle r, H, U, T^U, f^U, \mathcal{L} \rangle$, then we get a (possibly partial) partition of V. In particular, the sum of all the values $n_r = |V(H)|$, over all the inputs $\langle r, H, U, T^U, f^U, \mathcal{L} \rangle$ that are at the same recursion level r, is at most $n + n \times \mathcal{O}(rk^2) = \mathcal{O}(k^2 n \log n)$. In the same way, the sum of all the values $m_r = |E(H)|$, over all the inputs $\langle r, H, U, T^U, f^U, \mathcal{L} \rangle$ that are at the same recursion level r, is at most $m + n \times \mathcal{O}(k^2 r) = \mathcal{O}(k^2 n \log n + m)$.

Processing $\langle r, H, U, T^U, f^U, \mathcal{L}\rangle$ takes $\mathcal{O}(2^{\mathcal{O}(k)}(n_r+m_r)^{1+\epsilon})$ time for any $\epsilon > 0$ if we exclude the recursive calls. Therefore, the total running time at any fixed recursive stage, and so also for the whole algorithm, is in $\mathcal{O}(2^{\mathcal{O}(k)}(n+m)^{1+\epsilon})$ for any $\epsilon > 0$. □

5 Facility location problems on bounded clique-width graphs

In this section we consider unweighted graphs, where the distance between two vertices u and v is classically defined as the minimum number of edges on a uv-path. Our last result in the paper is as follows:

Theorem 5.1. *For every connected n-vertex m-edge graph $G = (V, E)$, if $cw(G) \leq k$ and a k-expression is given, then for any $\epsilon > 0$, we can compute in $\mathcal{O}(2^{\mathcal{O}(k)}(n+m)^{1+\epsilon})$ time: all the p-eccentricities and all the total p-distances, for every cost function $p : V \to \mathbb{N}$.*

Despite its apparent similarity with Theorem 4.1, Theorem 5.1 has some special features. To see why, let us assume that two vertices u, v are disconnected by a join with respective sides X, Y. Then, $d(u, v) = d(u, X) + 1 + d(Y, v)$,[2] and therefore for any fixed u, in order to maximize $d(u, v)$ it suffices to find such a v maximizing $d(v, Y)$. However, this is no more true if we have a cost function p; indeed, we now want to maximize $p(v) \cdot (d(u, X) + 1) + p(v)d(v, Y)$.

For that, we first prove that:

Lemma 5.2. *Let F be a set of n linear functions $f_i : t \to a_i \cdot t + b_i$, where $a_i, b_i \geq 0$. Then after an $\mathcal{O}(n \log n)$-time pre-processing, for any $x \geq 0$ we can compute $\max_{1 \leq i \leq n}\{a_i \cdot x + b_i\}$ in $\mathcal{O}(\log n)$ time.*

Proof. Let f_i, f_j satisfy $a_i \leq a_j$ and $b_i \leq b_j$. Since we have $f_i(t) \leq f_j(t)$ for every $t \geq 0$, we can safely discard f_i from F. In particular, we may assume all coefficients a_i (resp., b_i) to be pairwise different. To perform all such removals in total $\mathcal{O}(n \log n)$ time, let $F = (f_1, f_2, \ldots, f_n)$ be lexicographically ordered by non-decreasing values of (a_i, b_i). Doing so, we can remove all duplicates. Then, we consider the functions f_i in order and we put them in some min-heap with as for key the value b_i. At any step j, the functions f_i that are already in the min-heap are those for which either $a_i < a_j$ or $a_i = a_j$ and $b_i < b_j$. In order to detect and remove all such functions for which we have $b_i \leq b_j$, it suffices (since they all satisfy $a_i \leq a_j$) to repeatedly looking at the minimum-key element into the heap.

Define $t_1 = 0$ and, for every $i > 1$, $t_i = \frac{b_{i-1} - b_i}{a_i - a_{i-1}}$. Note that for $i > 1$, we have $f_{i-1}(t) > f_i(t)$ for every $0 \leq t < t_i$ while we have $f_{i-1}(t) \leq f_i(t)$ for every $t \geq t_i$. In particular, if $t_i \geq t_{i+1}$ then, we claim that we can safely discard f_i from F. Indeed, for $0 \leq t < t_i$ we have $f_{i-1}(t) > f_i(t)$ while for every $t \geq t_i \geq t_{i+1}$ we have $f_{i+1}(t) \geq f_i(t)$. Consider the following algorithm. All functions of F are put in a doubly-linked list, where they are kept ordered by increasing values of a_i. We start from the head of the list and we proceed as follows until we reach the bottom of it. Let f_i be the function considered at a given step of the algorithm (initially, $f_i = f_1$ is the function minimizing a_1 or, equivalently, the one maximizing b_1). If f_i is the current head of the list, then we go to its successor function f_{i+1} in the list. Otherwise, let f_{i-1} be its predecessor function into the list. If $t_i > t_{i-1}$, then we also go to f_{i+1}. Otherwise, we discard f_{i-1}, and we still consider f_i at the next step. In this latter case, note that we need to reset $t_i := \frac{b_{i-2} - b_i}{a_i - a_{i-2}}$, where f_{i-2} is the new predecessor function of f_i into the list (formerly, the predecessor function of f_{i-1}). This algorithm is correct by our previous claim and it runs in $\mathcal{O}(n)$ time. Therefore, from now on we may assume to have $t_1 = 0 < t_2 < \ldots < t_n$.

Finally, let $x \geq 0$ be arbitrary. Let i be the largest index such that $x \geq t_i$ (computed in $\mathcal{O}(\log n)$ time by binary search). We claim to have $a_i \cdot x + b_i = \max_j\{a_j \cdot x + b_j\}$. Indeed, suppose by contradiction there exists a j s.t. $a_j \cdot x + b_j > a_i \cdot x + b_i$. If $j < i$ then, assume j to be maximum with this property. Since we have $t_{j+1} \leq x$, we obtain $f_{j+1}(x) \geq f_j(x)$, that contradicts either the maximality of j or that $a_j \cdot x + b_j > a_i \cdot x + b_i$. Otherwise, $j > i$ and we assume this index to be minimized. Since we have $x < t_j$, we obtain $f_{j-1}(x) > f_j(x)$, thus contradicting either the minimality of j or that $a_j \cdot x + b_j > a_i \cdot x + b_i$. □

We combine this lemma with some insights of Cabello about range trees [12]:

[2] This is a slightly different formula than in Lemma 3.3, which is for vertex-weighted graphs.

Lemma 5.3 ([12]). *Given a set P of n points in $\mathbb{R}^d$, there is a family of sets $\mathcal{P} = \{P_i \mid i \in I\}$ and a data structure with the following properties for any $\epsilon > 0$:*

- *$P_i \subset P$ for each $P_i \in \mathcal{P}$;*
- *all the sets of $\mathcal{P}$ together have $\mathcal{O}(2^{\mathcal{O}(d)}n^{1+\epsilon})$ points, counting with multiplicity; that is, $\sum_{P_i \in \mathcal{P}} |P_i| = \mathcal{O}(2^{\mathcal{O}(d)}n^{1+\epsilon})$;*
- *for each box $\mathcal{R} \subset \mathbb{R}^d$, the data structure finds in $\mathcal{O}(2^{\mathcal{O}(d)}n^{\epsilon})$ time indices $I_{\mathcal{R}} \subset I$ s.t. $|I_{\mathcal{R}}| = \mathcal{O}(2^{\mathcal{O}(d)}n^{\epsilon})$ and $P \cap \mathcal{R} = \dot{\bigcup}_{i \in I_{\mathcal{R}}} P_i$;*
- *the family $\mathcal{P}$ and the data structure can be computed in $\mathcal{O}(2^{\mathcal{O}(d)}n^{1+\epsilon})$ time.*

Doing so, we get:

Corollary 5.4. *Let P be a set of n points in $\mathbb{R}^d$ where each point $p \in P$ is associated an ordered pair $(a(p), b(p))$ of nonnegative real numbers. We can construct a data structure in $\mathcal{O}(2^{\mathcal{O}(d)}n^{1+\epsilon})$ time, for any $\epsilon > 0$, such that, for any box $\mathcal{R}$ and nonnegative $x \geq 0$, a point $p \in P \cap \mathcal{R}$ maximizing $a(p) \cdot x + b(p)$ can be output in $\mathcal{O}(2^{\mathcal{O}(d)}n^{\epsilon})$ time.*

Proof. We construct the family $\mathcal{P}$ and the data structure of Lemma 5.3, then we apply Lemma 5.2 to each set $P_i \in \mathcal{P}$. □

We propose a new version of Lemma 4.5 where, roughly, we use Corollary 5.4 instead of Lemma 4.4.

Lemma 5.5. *Let $G = (V, E, \alpha)$ be a connected n-vertex m-edge graph, where $\alpha : E \to \mathbb{N}$, and let $p \geq 0$ be some vertex-weight function. Let also $(A, V \setminus A)$ be an unweighted cut of neighbourhood diversity at most k, and let $A' \subseteq A$, $B' \subseteq V \setminus A$. For any $\epsilon > 0$, after a pre-processing in $\mathcal{O}(km + 2^{\mathcal{O}(k)}n^{1+\epsilon})$ time, for every vertex $u \in A'$ we can compute the values $\max_{v \in B'} p(v) \cdot d_G(u, v)$ and $\sum_{v \in B'} p(v) \cdot d_G(u, v)$ in $\mathcal{O}(2^{\mathcal{O}(k)}n^{\epsilon})$ time; in the same way, for every vertex $v \in B'$ we can compute the values $\max_{u \in A'} p(u) \cdot d_G(v, u)$ and $\sum_{u \in A'} p(u) \cdot d_G(v, u)$ in $\mathcal{O}(2^{\mathcal{O}(k)}n^{\epsilon})$ time.*

Proof. Let $A_1, A_2, \ldots, A_k$ be the minimal partition of A. By Lemma 4.3, we can compute it in $\mathcal{O}(m)$ time. For $1 \leq i \leq k$, let $B_i = N_G(A_i) \setminus A$. Note that since the subsets A_i are pairwise disjoint, we can compute $B_1, B_2, \ldots, B_k$ in total $\mathcal{O}(m)$ time. Observe that there is at most one index i s.t. $B_i = \emptyset$ (otherwise, we can merge all groups A_j s.t. $B_j = \emptyset$ into one, thus contradicting the minimality of the partition of A). W.l.o.g., if such index exists then it must be $i = k$. We want to exclude this index, if it exists, in order to avoid handling with arithmetic over infinite values. So, let $k' = k$ if $B_k \neq \emptyset$, otherwise let $k' = k - 1$. For every $1 \leq i \leq k'$, for every $u \in A'$, we compute $d_G(u, A_i)$. In the same way, for every $1 \leq i \leq k'$, for every $v \in B'$, we compute $d_G(B_i, v)$. It takes $\mathcal{O}(k'm) = \mathcal{O}(km)$ time in total if we use the single-source shortest-path algorithm of Thorup [59]. Then, for every $v \in B'$ and for every $1 \leq i \leq k'$, we create a k'-dimensional point $\overrightarrow{p}(v, i)$: whose first coordinate is the index i, followed by the values $d_G(B_i, v) - d_G(B_j, v)$, $1 \leq j \leq k'$, $j \neq i$. Let P contain all these $k'|B'|$ points.

We add all points in P into three different k'-dimensional range trees, namely: we set $a(\overrightarrow{p}(v, i)) = p(v)$, $b(\overrightarrow{p}(v, i)) = p(v)d_G(B_i, v)$ and then we apply Corollary 5.4; we set $g_1(\overrightarrow{p}(v, i)) = p(v)d_G(B_i, v)$ and then we apply Lemma 4.4; we set $g_2(\overrightarrow{p}(v, i)) = p(v)$ and then we apply Lemma 4.4. It takes $\mathcal{O}(2^{\mathcal{O}(k)}n^{1+\epsilon})$ time for any $\epsilon > 0$. Now, let $u \in A'$ be fixed, and assume that we want to compute the values $\max_{v \in B'} p(v)d_G(u, v)$ and $\sum_{v \in B'} p(v)d_G(u, v)$. Note that if we replace each edge e by a path of length $\alpha(e)$ then, $A \cup E(A)$ is a $(k' + 1)$-module in the resulting graph, with $E(A)$ representing the edges in G with their both ends in A (no such vertex has a neighbour in the other side of the cut). By Lemma 3.3 (applied to the resulting graph, and slightly modified for unweighted graphs), for every $v \in B'$, we have $d_G(u, v) = \min\{d_G(u, A_i) + 1 + d_G(B_i, v) \mid 1 \leq i \leq k\}$. Since $d_G(B_k, v) = +\infty$ if $B_k = \emptyset$, we also have $d_G(u, v) = \min\{d_G(u, A_i) + 1 + d_G(B_i, v) \mid 1 \leq i \leq k'\}$. We (virtually) partition B' into $C_1, C_2, \ldots C_{k'}$ so that, for every $1 \leq i \leq k'$, $v \in C_i$ if and

only if the least index j s.t. $d_G(u,v) = d_G(u,A_j) + 1 + d_G(B_j,v)$ is equal to i. Specifically, we design boxes $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_{k'}$ so that $p(v,j) \in \mathcal{R}_i \iff j = i$ and $v \in C_i$. Note that if we can do so, then:

$$\begin{aligned}
\max_{v \in B'} p(v) d_G(u,v) &= \max_{1 \leq i \leq k'} \max_{v \in C_i} p(v) d_G(u,v) \\
&= \max_{1 \leq i \leq k'} \max\{p(v) \cdot (d_G(u,A_i)+1) + p(v) d_G(B_i,v) \mid v \in C_i\} \\
&= \max_{1 \leq i \leq k'} \max\{a(\overrightarrow{p}(v,j)) \cdot (d_G(u,A_i)+1) + b(\overrightarrow{p}(v,j)) \mid \overrightarrow{p}(v,j) \in \mathcal{R}_i\}.
\end{aligned}$$

In particular, we are left applying Corollary 5.4 for k' range queries. In the same way:

$$\begin{aligned}
\sum_{v \in B'} p(v) d_G(u,v) &= \sum_{i=1}^{k'} \sum_{v \in C_i} p(v) d_G(u,v) \\
&= \sum_{i=1}^{k'} \sum_{v \in C_i} (p(v)(d_G(u,A_i)+1) + p(v) d_G(B_i,v)) \\
&= \sum_{i=1}^{k'} \left[(d_G(u,A_i)+1) \cdot \sum_{v \in C_i} p(v) + \sum_{v \in C_i} p(v) d_G(B_i,v) \right] \\
&= \sum_{i=1}^{k'} \Big[(d_G(u,A_i)+1) \cdot \sum \{g_1(\overrightarrow{p}(v,j)) \mid \overrightarrow{p}(v,j) \in \mathcal{R}_i\} \\
&\quad + \sum \{g_2(\overrightarrow{p}(v,j)) \mid \overrightarrow{p}(v,j) \in \mathcal{R}_i\} \Big].
\end{aligned}$$

In particular, we are left doing k' sum range queries, but on two separate range trees. Hence, being given $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_{k'}$, we are done in $\mathcal{O}(2^{\mathcal{O}(k)} n^{\epsilon})$ time for any $\epsilon > 0$ by Lemma 4.4.

For every $1 \leq i \leq k'$, the box $\mathcal{R}_i$ is defined as follows. Let $\overrightarrow{p} = (p_1, p_2, \ldots, p_{k'})$ be a k'-dimensional point of P. We have $\overrightarrow{p} \in \mathcal{R}_i$ if and only if:

$$\begin{cases}
p_1 = i \\
\forall 1 \leq j \leq i-1,\ p_{j+1} < (d_G(u,A_j) - d_G(u,A_i)) \\
\forall i+1 \leq j \leq k',\ p_j \leq (d_G(u,A_j) - d_G(u,A_i)).
\end{cases}$$

Indeed, we have: $d_G(u,A_i) + 1 + d_G(B_i,v) \leq d_G(u,A_j) + 1 + d_G(B_j,v)$ if and only if $d_G(u,A_i) + d_G(B_i,v) \leq d_G(u,A_j) + d_G(B_j,v)$, if and only if $d_G(B_i,v) - d_G(B_j,v) \leq (d_G(u,A_j) - d_G(u,A_i))$. Furthermore, by construction, if $j < i$ then $d_G(B_i,v) - d_G(B_j,v)$ is exactly the $(j+1)^{th}$ coordinate of $\overrightarrow{p}(v,i)$ (in which case we want the inequality to be strict by the definition of C_i), and if $j > i$ then $d_G(B_i,v) - d_G(B_j,v)$ is exactly the j^{th} coordinate of this point. For the vertices $v \in B'$, we proceed similarly as above, that is, we create a point-set P' from A' and we put them in some separate k'-dimensional range trees. □

Theorem 5.1 now follows from the exact same proof as for Theorem 4.1, but where we use Lemma 5.5 rather than Lemma 4.5. □

6 Open problems

We would find it interesting to extend our framework to other centrality indices, such as the computation of betweenness centrality [37]. To our best knowledge, this problem is open also for bounded treewidth graphs.

Shrub-depth is a well-studied "low-depth" variation of clique-width [39]. We observe that its algorithmic applications to polynomial-time solvable problems have yet to be explored. In particular, given a (d,m)-tree model for a graph $G = (V,E)$, can we compute its diameter $diam(G)$ in $\mathcal{O}(poly(d,m) \cdot (|V| + |E|)^{2-\epsilon})$ time, for some $\epsilon > 0$?

References

[1] A. Abboud, V. Vassilevska Williams, and J. R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 377–391. SIAM, 2016.

[2] J. A. Bondy and U. S. R. Murty. *Graph theory*, volume 244 of *Graduate Texts in Mathematics*. Springer-Verlag London, 2008.

[3] R. Borie, J. Johnson, V. Raghavan, and J. Spinrad. Robust polynomial time algorithms on clique-width k graphs. 2002.

[4] A. Brandstädt, K. K. Dabrowski, S. Huang, and D. Paulusma. Bounding the clique-width of H-free split graphs. *Discrete Applied Mathematics*, 211:30–39, 2016.

[5] A. Brandstädt, K. K. Dabrowski, S. Huang, and D. Paulusma. Bounding the Clique-Width of H-Free Chordal Graphs. *Journal of Graph Theory*, 86(1):42–77, 2017.

[6] A. Brandstädt, F. F. Dragan, H.-O. Le, and R. Mosca. New graph classes of bounded clique-width. *Theory of Computing Systems*, 38(5):623–645, 2005.

[7] A. Brandstadt, J. Engelfriet, H.-O. Le, and V. V. Lozin. Clique-width for 4-vertex forbidden subgraphs. *Theory of Computing Systems*, 39(4):561–590, 2006.

[8] A. Brandstädt, T. Klembt, and S. Mahfud. P_6-and triangle-free graphs revisited: structure and bounded clique-width. *Discrete Mathematics & Theoretical Computer Science*, 8(1), 2006.

[9] A. Brandstädt, H.-O. Le, and R. Mosca. Gem-and co-gem-free graphs have bounded clique-width. *International Journal of Foundations of Computer Science*, 15(01):163–185, 2004.

[10] A. Brandstädt, H.-O. Le, and R. Mosca. Chordal co-gem-free and (P_5, gem)-free graphs have bounded clique-width. *Discrete Applied Mathematics*, 145(2):232–241, 2005.

[11] K. Bringmann, T. Husfeldt, and M. Magnusson. Multivariate Analysis of Orthogonal Range Searching and Graph Distances. *Algorithmica*, pages 1–24, 2020.

[12] S. Cabello. Computing the inverse geodesic length in planar graphs and graphs of bounded treewidth. Technical Report 1908.01317, arXiv, 2019.

[13] S. Cabello and C. Knauer. Algorithms for graphs of bounded treewidth via orthogonal range searching. *Computational Geometry*, 42(9):815–824, 2009.

[14] D. G. Corneil, M. Habib, J.-M. Lanlignel, B. Reed, and U. Rotics. Polynomial Time Recognition of Clique-Width ≤ 3 Graphs. In *Latin American Theoretical INformatics Symposium (LATIN)*, volume 1776 of *Lecture Notes in Computer Science*, pages 126–134. Springer, 2000.

[15] D. G. Corneil and U. Rotics. On the relationship between clique-width and treewidth. *SIAM Journal on Computing*, 34(4):825–847, 2005.

[16] D. Coudert, G. Ducoffe, and A. Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. *ACM Transactions on Algorithms (TALG)*, 15(3):1–57, 2019.

[17] B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

[18] B. Courcelle, P. Heggernes, D. Meister, C. Papadopoulos, and U. Rotics. A characterisation of clique-width through nested partitions. *Discrete Applied Mathematics*, 187:70–81, 2015.

[19] B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

[20] B. Courcelle and R. Vanicat. Query efficient implementation of graphs of bounded clique-width. *Discrete Applied Mathematics*, 131(1):129–150, 2003.

[21] W. H. Cunningham. Decomposition of directed graphs. *SIAM Journal on Algebraic Discrete Methods*, 3(2):214–228, 1982.

[22] K. K. Dabrowski and D. Paulusma. Classifying the clique-width of H-free bipartite graphs. *Discrete Applied Mathematics*, 200:43–51, 2016.

[23] K. K. Dabrowski and D. Paulusma. Clique-width of graph classes defined by two forbidden induced subgraphs. *The Computer Journal*, 59(5):650–666, 2016.

[24] K. Das, S. Samanta, and M. Pal. Study on centrality measures in social networks: a survey. *Social network analysis and mining*, 8(1):1–11, 2018.

[25] R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2010. 4*th* edition.

[26] F. F. Dragan and C. Yan. Collective tree spanners in graphs with bounded parameters. *Algorithmica*, 57(1):22–43, 2010.

[27] G. Ducoffe. Optimal Centrality Computations Within Bounded Clique-Width Graphs. In P. A. Golovach and M. Zehavi, editors, *International Symposium on Parameterized and Exact Computation (IPEC 2021)*, volume 214 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

[28] G. Ducoffe and A. Popa. The b-matching problem in distance-hereditary graphs and beyond. In *International Symposium on Algorithms and Computation (ISAAC)*, volume 123 of *Leibniz International Proceedings in Informatics*, pages 30:1–30:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[29] G. Ducoffe and A. Popa. The use of a pruned modular decomposition for maximum matching algorithms on some graph classes. In *International Symposium on Algorithms and Computation (ISAAC)*, volume 123 of *Leibniz International Proceedings in Informatics*, pages 6:1–6:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[30] W. Espelage, F. Gurski, and E. Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 1 of *Lecture Notes in Computer Science*, pages 117–128. Springer, 2001.

[31] M. R. Fellows, F. A. Rosamond, U. Rotics, and S. Szeider. Clique-width is NP-complete. *SIAM Journal on Discrete Mathematics*, 23(2):909–939, 2009.

[32] F. Fomin and T. Korhonen. Fast fpt-approximation of branchwidth. Technical Report 2111.03492, arXiv, 2021.

[33] F. V. Fomin, P. A. Golovach, D. Lokshtanov, and S. Saurabh. Intractability of clique-width parameterizations. *SIAM Journal on Computing*, 39(5):1941–1956, 2010.

[34] F. V. Fomin, P. A. Golovach, D. Lokshtanov, and S. Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM Journal on Computing*, 43(5):1541–1563, 2014.

[35] F. V. Fomin, P. A. Golovach, D. Lokshtanov, S. Saurabh, and M. Zehavi. Clique-width III: Hamiltonian Cycle and the Odd Case of Graph Coloring. *ACM Transactions on Algorithms*, 15(1):9, 2019.

[36] F. V. Fomin, D. Lokshtanov, S. Saurabh, M. Pilipczuk, and M. Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Transactions on Algorithms*, 14(3):34:1–34:45, 2018.

[37] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977.

[38] M. Fürer. A natural generalization of bounded tree-width and bounded clique-width. In *Latin American Symposium on Theoretical Informatics*, pages 72–83. Springer, 2014.

[39] J. Gajarskỳ, M. Lampis, and S. Ordyniak. Parameterized algorithms for modular-width. In *International Symposium on Parameterized and Exact Computation*, pages 163–176. Springer, 2013.

[40] C. Gavoille and C. Paul. Distance labeling scheme and split decomposition. *Discrete Mathematics*, 273(1-3):115–130, 2003.

[41] C. Gavoille, D. Peleg, S. Pérennes, and R. Raz. Distance labeling in graphs. *Journal of Algorithms*, 53(1):85–112, 2004.

[42] A. C. Giannopoulou, G. B. Mertzios, and R. Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *Theoretical computer science*, 689:67–95, 2017.

[43] A. Goldman. Optimal center location in simple networks. *Transportation science*, 5(2):212–221, 1971.

[44] M. C. Golumbic and U. Rotics. On the clique-width of some perfect graph classes. *International Journal of Foundations of Computer Science*, 11(03):423–443, 2000.

[45] M. Habib, R. McConnell, C. Paul, and L. Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234(1-2):59–84, 2000.

[46] P. Hage and F. Harary. Eccentricity and centrality in networks. *Social networks*, 17(1):57–63, 1995.

[47] T. Hagerup, J. Katajainen, N. Nishimura, and P. Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *Journal of Computer and System Sciences*, 57(3):366–375, 1998.

[48] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[49] Y. Iwata, T. Ogasawara, and N. Ohsaka. On the power of tree-depth for fully polynomial FPT algorithms. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 96 of *Leibniz International Proceedings in Informatics*, pages 41:1–41:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

[50] S. Kamali. Compact representation of graphs of small clique-width. *Algorithmica*, 80(7):2106–2131, 2018.

[51] S. Kratsch and F. Nelles. Efficient and adaptive parameterized algorithms on modular decompositions. In *European Symposia on Algorithms (ESA)*, pages 55:1–55:15, 2018.

[52] S. Kratsch and F. Nelles. Efficient Parameterized Algorithms for Computing All-Pairs Shortest Paths. In *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*, volume 154 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[53] V. Lozin and D. Rautenbach. Chordal bipartite graphs of bounded tree-and clique-width. *Discrete Mathematics*, 283(1-3):151–158, 2004.

[54] J. A. Makowsky and U. Rotics. On the clique-width of graphs with few P_4's. *International Journal of Foundations of Computer Science*, 10(03):329–348, 1999.

[55] S. Oum and P. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006.

[56] M. Rao. Clique-width of graphs defined by one-vertex extensions. *Discrete Mathematics*, 308(24):6157–6165, 2008.

[57] G. Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966.

[58] K. Suchan and I. Todinca. On powers of graphs of bounded NLC-width (clique-width). *Discrete Applied Mathematics*, 155(14):1885–1893, 2007.

[59] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM (JACM)*, 46(3):362–394, 1999.

[60] J.-M. Vanherpe. Clique-width of partner-limited graphs. *Discrete mathematics*, 276(1-3):363–374, 2004.