Optimal centrality computations within bounded clique-width graphs

Guillaume Ducoffe

Optimal centrality computations within bounded clique-width graphs

Guillaume Ducoffe
National Institute for Research and Development in Informatics, Romania
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 run-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.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Fixed parameter tractability; Theory of computation → Shortest paths

Keywords and phrases Clique-width, Centralities computation, Facility Location problems, Distance-labeling scheme, Fine-grained complexity in P, Graph theory

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.16

International Symposium on Parameterized and Exact Computation (IPEC).
Editors: Meirav Zehavi and Petr Golovach; Article No. 16; pp. 16:1–16:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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, 40, 48], and beyond [9, 6, 10, 7, 8, 23, 49, 53, 54]. For instance, distance-hereditary graphs, and so, trees, have clique-width at most three [40]. 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 [31, 32, 33]. We refer to [29] 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 [42]). Nevertheless it was only recently that such idea was better formalized [38], 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 [34, 44]. 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 [46]. 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 [47] 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]. To date, the best-known approximation algorithms for clique-width run in $\mathcal{O}(n^3)$-time [50]. Still, 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 [40, 49]. 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 [28, 27, 46] and especially in [47], 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 $w : V \to \mathbb{N}$. 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 [41]. The *closeness centrality* of u, denoted $C_G(u)$, equals $1/\sum_v d_G(u, v)$ [52]. 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 8).

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 17).

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 4 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 7). 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 [47] 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 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 [44].

The distance-labeling scheme of Theorem 4 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

[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.

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'. We shall introduce additional terminology wherever needed in the paper.

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 [40]. 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 [35], where Fürer proved this result in a more general setting).

Partition tree. It is useful to represent a k-expression as a parse tree. 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 collection of subsets of vertices with equal label in their rooted subtree. 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 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 [51]. We make the following useful observation, whose proof is inspired by [51, Theorem 7].

▶ **Lemma 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 $\{X' \cap A \mid X' \in f(a')\}$.*

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, 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 4 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** ([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 [37]. We improve the latter while keeping optimal bit size and improved query time, namely:

▶ **Theorem 4.** *For a vertex-weighted graph, let W denote the maximum weight. The family of n-vertex m-edge 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 [45] 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 5.** *Let $G = (V, E, w)$ be a 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 $d_G(u, v) = \min\{d_{G[V \setminus A]}(u, v)\} \cup \{d_G(u, A_i) + d_G(B_i, v) \mid 1 \leq i \leq k\}$.*

Our scheme for bounded clique-width graphs mimics one very well-known for trees which is based on the centroid decomposition [36]. 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 [39] (simply orient each edge toward the heaviest subtree, then find a sink). We also need the following easy lemma:

▶ **Lemma 6.** *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. $\max\{w(F_1), w(F_2)\} \leq 2w(T)/3$.*

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

Proof of Theorem 4. We fix some width-k partition tree (T, f), that takes $\mathcal{O}(k(n+m))$ time to compute by using Lemma 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 6 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, 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 $f(c)$ or $f(a_1)$. 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. 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 5 $j+1$ 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 [47]. As a by-product of our Theorem 4, 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 7.** *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 4 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 4 in order to prove the main result of this paper:

▶ **Theorem 8.** *For every connected n-vertex m-edge 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 9.** *For every connected n-vertex m-edge 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 [30], 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 $W(G) = 2n(n-1) - 2m$.

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 10.** *Every vertex-subset A in a 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.

Additional notations. From this point on we need to also allow *edge-weights*, due to some technicalities in our final proof of Theorem 8. 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.

4.2 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 11** ([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 12 we give a new simple application of Lemma 11 to distance problems in graphs, namely:

▶ **Lemma 12.** *Let $G = (V, E, w, \alpha)$ be a connected n-vertex m-edge graph, 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.*

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 13.** *Given $G = (V, E, w, \alpha)$ connected, 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. 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 14.** *Given $G = (V, E, w, \alpha)$ connected, 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 13) can be constructed in $\tilde{\mathcal{O}}(k^2n + km)$ time.*

The following two properties are crucial in our proofs of Theorem 8.

▶ **Lemma 15.** *Given $G = (V, E, w, \alpha)$ connected, let $(A, V \setminus A)$ be an unweighted cut of neighbourhood diversity at most k. Let H_A, H_B be as in Definition 13. 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 $d_G(u, v) = d_{H_B}(u, v)$.*

Our approach only works for unweighted cuts. In particular, if we want to apply the procedure of Definition 13 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 16.** *Given $G = (V, E, w, \alpha)$ connected, let $(A, V \setminus A)$ be an unweighted cut of neighbourhood diversity at most k. Let H_A, H_B be as in Definition 13.*

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.*

4.4 Proofs of the main results

Sketch Proof of Theorem 8. We revisit the scheme of Theorem 4. That is, we fix some width-k partition tree (T, f), that takes $\mathcal{O}(k(n+m))$ time by using Lemma 1. 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 [43]. Furthermore, 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 of T.

The algorithm. We consider a more general problem for which we are given as input some tuple $\langle r, H, U, T^U, f^U, \mathcal{L}\rangle$. Let us detail each of the components of this input. Here, H is a graph with non-negative real vertex-weights and non-negative integer edge-weights (initially, $H = G$). The value r represents the recursion level of the algorithm (initially, $r = 0$).

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 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. 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 $f^U(b) = \{X \cap U \mid X \in f(b)\}$. Note that in lieu of (T^U, f^U), we are given the representation graph of this partition tree (as defined in Sec. 2). Finally, $H \setminus U$ is partitioned in $r' \leq r$ subgraphs of order $\mathcal{O}(k^2)$, that we shall name "clusters" in what follows (there may exist edges between different clusters, but they must be unweighted). 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 13 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 each $u \in U$, $\max_{v \in U} d_H(u, v)$ and $\sum_{v \in U} d_H(u, v)$.

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 (base case of the recursion). Then, we compute a w-centroid c in T^U. 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 6, 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 $A = \bigcup_{j=1}^{p} A_j$. 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 $1 \leq i \leq p$ 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 $U \setminus A$.

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.

Let $(A', V(H) \setminus A')$ be the resulting cut, where $A \subseteq A'$. By construction, it is unweighted. We prove that A' is a k-module of H. Then, we apply Lemma 12 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 $\sum_{u \in A} d_H(v, u)$). 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 13 (*i.e.*, w.r.t. the above cut $(A', V(H) \setminus A')$). 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 15. 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 by induction on r, using Lemma 16, that:

▶ **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.*

Complexity analysis. By induction, the depth of the recursion tree is $\mathcal{O}(\log n)$. Furthermore, the sum of all the orders $|V(H)| + |E(H)|$, 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 $\mathcal{O}(k^2 n \log n + m)$. Therefore, by Lemmas 14 and 12 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

Our last result in the paper is as follows:

▶ **Theorem 17.** *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 8, Theorem 17 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 18.** *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.*

Combined with some insights of Cabello about range trees [12], we get:

▶ **Corollary 19.** *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.*

▶ **Lemma 20.** *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.*

Theorem 17 now follows from the exact same proof as for Theorem 8, but where we use Lemma 20 rather than Lemma 12.

[2] This is a slightly different formula than in Lemma 5, which is for vertex-weighted graphs. Here we adapt the formula for unweighted graphs, where the distance between two vertices u and v is classically defined as the minimum number of edges on a uv-path.

References

1 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.

2 John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory*, volume 244 of *Graduate Texts in Mathematics*. Springer-Verlag London, 2008.

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

4 Andreas Brandstädt, Konrad K Dabrowski, Shenwei Huang, and Daniël Paulusma. Bounding the clique-width of H-free split graphs. *Discrete Applied Mathematics*, 211:30–39, 2016.

5 Andreas Brandstädt, Konrad K Dabrowski, Shenwei Huang, and Daniël Paulusma. Bounding the Clique-Width of H-Free Chordal Graphs. *Journal of Graph Theory*, 86(1):42–77, 2017.

6 Andreas Brandstädt, Feodor F Dragan, Hoàng-Oanh Le, and Raffaele Mosca. New graph classes of bounded clique-width. *Theory of Computing Systems*, 38(5):623–645, 2005.

7 Andreas Brandstadt, Joost Engelfriet, Hoang-Oanh Le, and Vadim V Lozin. Clique-width for 4-vertex forbidden subgraphs. *Theory of Computing Systems*, 39(4):561–590, 2006.

8 Andreas Brandstädt, Tilo Klembt, and Suhail Mahfud. P_6-and triangle-free graphs revisited: structure and bounded clique-width. *Discrete Mathematics & Theoretical Computer Science*, 8(1), 2006.

9 Andreas Brandstädt, Hoàng-Oanh Le, and Raffaele Mosca. Gem-and co-gem-free graphs have bounded clique-width. *International Journal of Foundations of Computer Science*, 15(01):163–185, 2004.

10 Andreas Brandstädt, Hoàng-Oanh Le, and Raffaele 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 Derek G. Corneil, Michel Habib, Jean-Marc Lanlignel, Bruce Reed, and Udi 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. doi:10.1007/10719839_14.

15 Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM Journal on Computing*, 34(4):825–847, 2005. doi:10.1137/S0097539701385351.

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 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

18 Bruno Courcelle, Pinar Heggernes, Daniel Meister, Charis Papadopoulos, and Udi Rotics. A characterisation of clique-width through nested partitions. *Discrete Applied Mathematics*, 187:70–81, 2015. doi:10.1016/j.dam.2015.02.016.

19 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

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

21 William H. Cunningham. Decomposition of directed graphs. *SIAM Journal on Algebraic Discrete Methods*, 3(2):214–228, 1982. doi:10.1137/0603021.

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

23 Konrad K Dabrowski and Daniël 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 Reinhard Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2010. 4*th* edition. doi:10.1007/978-3-662-53622-3.

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

27 Guillaume Ducoffe and Alexandru 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. doi:10.4230/LIPIcs.ISAAC.2018.30.

28 Guillaume Ducoffe and Alexandru 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. doi:10.4230/LIPIcs.ISAAC.2018.6.

29 Wolfgang Espelage, Frank Gurski, and Egon 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. doi:10.1007/3-540-45477-2_12.

30 Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is NP-complete. *SIAM Journal on Discrete Mathematics*, 23(2):909–939, 2009. doi:10.1137/070687256.

31 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM Journal on Computing*, 39(5):1941–1956, 2010. doi:10.1137/080742270.

32 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM Journal on Computing*, 43(5):1541–1563, 2014. doi:10.1137/130910932.

33 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: Hamiltonian Cycle and the Odd Case of Graph Coloring. *ACM Transactions on Algorithms*, 15(1):9, 2019. doi:10.1145/3280824.

34 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michal Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Transactions on Algorithms*, 14(3):34:1–34:45, 2018. doi:10.1145/3186898.

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

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

37 Cyril Gavoille and Christophe Paul. Distance labeling scheme and split decomposition. *Discrete Mathematics*, 273(1-3):115–130, 2003.

38 Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *Theoretical computer science*, 689:67–95, 2017. doi:10.1016/j.tcs.2017.05.017.

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

40 Martin Charles Golumbic and Udi Rotics. On the clique-width of some perfect graph classes. *International Journal of Foundations of Computer Science*, 11(03):423–443, 2000. doi:10.1142/S0129054100000260.

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

42 Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *Journal of Computer and System Sciences*, 57(3):366–375, 1998.

43 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

44 Yoichi Iwata, Tomoaki Ogasawara, and Naoto 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. doi:10.4230/LIPIcs.STACS.2018.41.

45 Shahin Kamali. Compact representation of graphs of small clique-width. *Algorithmica*, 80(7):2106–2131, 2018.

46 Stefan Kratsch and Florian Nelles. Efficient and adaptive parameterized algorithms on modular decompositions. In *European Symposia on Algorithms (ESA)*, pages 55:1–55:15, 2018. doi:10.4230/LIPIcs.ESA.2018.55.

47 Stefan Kratsch and Florian 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. doi:10.4230/LIPIcs.STACS.2020.38.

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

49 Johann A. Makowsky and Udi Rotics. On the clique-width of graphs with few P_4's. *International Journal of Foundations of Computer Science*, 10(03):329–348, 1999. doi:10.1142/S0129054199000241.

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

51 Michaël Rao. Clique-width of graphs defined by one-vertex extensions. *Discrete Mathematics*, 308(24):6157–6165, 2008. doi:10.1016/j.disc.2007.11.039.

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

53 Karol Suchan and Ioan Todinca. On powers of graphs of bounded NLC-width (clique-width). *Discrete Applied Mathematics*, 155(14):1885–1893, 2007.

54 Jean-Marie Vanherpe. Clique-width of partner-limited graphs. *Discrete mathematics*, 276(1-3):363–374, 2004.