2014 IEEE Annual Symposium on Foundations of Computer Science

Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth

Daniel Lokshtanov*, Marcin Pilipczuk*, Michał Pilipczuk*and Saket Saurabh*†
*University of Bergen, Norway.
Email: {daniello, marcin.pilipczuk, michal.pilipczuk}@ii.uib.no
†Institute of Mathematical Sciences, India.
saket@imsc.res.in

***Abstract*—We give a fixed-parameter tractable algorithm that, given a parameter k and two graphs G_1, G_2, either concludes that one of these graphs has treewidth at least k, or determines whether G_1 and G_2 are isomorphic. The running time of the algorithm on an n-vertex graph is $2^{\mathcal{O}(k^5 \log k)} \cdot n^5$, and this is the first fixed-parameter algorithm for GRAPH ISOMORPHISM parameterized by treewidth.**

Our algorithm in fact solves the more general *canonization* problem. We namely design a procedure working in $2^{\mathcal{O}(k^5 \log k)} \cdot n^5$ time that, for a given graph G on n vertices, either concludes that the treewidth of G is at least k, or finds an isomorphism-invariant *construction term* — an algebraic expression that encodes G together with a tree decomposition of G of width $\mathcal{O}(k^4)$. Hence, a canonical graph isomorphic to G can be constructed by simply evaluating the obtained construction term, while the isomorphism test reduces to verifying whether the computed construction terms for G_1 and G_2 are equal.

***Index Terms*—graph isomorphism; canonization; parameterized algorithms; treewidth**

1. INTRODUCTION

GRAPH ISOMORPHISM is one of the most fundamental graph problems: given two graphs G_1, G_2, decide whether G_1 and G_2 are *isomorphic*, i.e., there exists a bijection ϕ between $V(G_1)$ and $V(G_2)$ such that $uv \in E(G_1)$ if and only if $\phi(u)\phi(v) \in E(G_2)$. Despite extensive research on the topic, it is still unknown whether the problem can be solved in polynomial time. On the other hand, there are good reasons to believe that GRAPH ISOMORPHISM is not NP-hard either, since NP-hardness of GRAPH ISOMORPHISM would imply a collapse of the polynomial hierarchy [31].

A significant amount of effort has been put into understanding and broadening the spectrum of classes of graphs where polynomial-time isomorphism tests can be designed. Perhaps the most important example is the classic algorithm of Babai and Luks [2], [25], which solves GRAPH ISOMORPHISM on graphs of maximum degree d in time $\mathcal{O}(n^{\mathcal{O}(d)})$. On the other hand, following polynomial-time solvability of GRAPH ISOMORPHISM on planar graphs [18], [19], [20], [34] it has been investigated how more general topological constraints can be exploited to design efficient algorithms for the problem. Isomorphism tests for graphs of genus g working in time $\mathcal{O}(n^{\mathcal{O}(g)})$ were proposed independently by Filotti and Mayer [14] and by Miller [26]. These results were improved by Ponomarenko [29], who gave an $\mathcal{O}(n^{f(|H|)})$ algorithm for graphs excluding a fixed graph H as a minor, for some function f. The result of Ponomarenko implies also that GRAPH ISOMORPHISM can be solved in polynomial time on graphs of constant treewidth. A simple algorithm for graphs of treewidth k running in time $\mathcal{O}(n^{k+4.5})$ was independently given by Bodlaender [4]. Finally, we mention the result of Arnborg and Proskurowski [1], who gave canonical representation of partial 2- and 3-trees.

Recently, Grohe and Marx [16], [17] obtained a structure theorem for graphs excluding a fixed graph H as a topological minor. This theorem roughly states that such graphs can be decomposed along small separators into parts that are either H-minor-free, or of almost bounded degree (in terms of $|H|$). Using previous algorithms for H-minor-free graphs [29] and bounded-degree graphs [2], [25], they managed to show that GRAPH ISOMORPHISM can be solved in $\mathcal{O}(n^{f(|H|)})$ time for H-topological-minor-free graphs, for some function f. Note that this result generalizes both the algorithms for GRAPH ISOMORPHISM on minor free-graphs [29] and on bounded degree graphs [2], [25]. The work of Grohe and Marx constitutes the current frontier of polynomial-time solvability of GRAPH ISOMORPHISM.

Observe that in all the aforementioned results the exponent of the polynomial depends on the considered parameter, be it the maximum degree, genus, treewidth, or the size of the excluded (topological) minor. In the field of parameterized complexity such algorithms are called XP algorithms (for *slice-wise polynomial*), and are often compared to the more efficient FPT algorithms (for *fixed-parameter tractable*), where the running time is required to be of the form $f(k) \cdot n^c$. Here k is the parameter, f is a computable function, and c is a universal constant independent of k. One of the main research directions in parameterized complexity is to consider problems that admit XP algorithms and determine whether they admit an FPT algorithm; we refer to the monographs [13], [15] for more information on parameterized complexity. It is therefore natural to ask which of the the aforementioned results on GRAPH ISOMORPHISM can be improved to fixed-parameter tractable algorithms.

Prior to this work very little was known about such improve-

D. Lokshtanov is supported by the BeHard grant under the recruitment programme of the of Bergen Research Foundation. The research of M. Pilipczuk and M. Pilipczuk leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 267959. S. Saurabh is supported by PARAPPROX, ERC starting grant no. 306992.

ments. In particular, the existence of FPT algorithms for GRAPH ISOMORPHISM parameterized by maximum degree, genus or treewidth of the input graph have remained intriguing open problems. A reader familiar with the algorithmic aspects of treewidth might find it surprising that the existence of an FPT algorithm for GRAPH ISOMORPHISM parameterized by treewidth is a difficult problem. The parameter has been studied intensively during the last 25 years, and is quite well-understood. Many problems that are very hard on general graphs become polynomial time, or even linear-time solvable on graphs of constant treewidth. For the vast majority of these problems, designing an FPT algorithm parameterized by treewidth boils down to designing a straightforward dynamic programming algorithm over the decomposition. For GRAPH ISOMORPHISM this is not the case, even the relatively simple $\mathcal{O}(n^{k+4.5})$ time algorithm of Bodlaender [4] is structurally quite different from most algorithms on bounded treewidth graphs. This might be the reason why GRAPH ISOMORPHISM was one of very few remaining problems of fundamental nature, whose fixed-parameter tractability when parameterized by treewidth was unresolved.

Therefore, fixed-parameter tractability of GRAPH ISOMORPHISM parameterized by treewidth has been considered an important open problem in parameterized complexity for years. This question (and its weaker variants for width measures lower-bounded by treewidth) was asked explicitly in [8], [9], [16], [21], [24], [27], [35], and appears on the open problem list of the recent edition of the monograph of Downey and Fellows [13].

Most of the related work on the parameterized complexity of GRAPH ISOMORPHISM with respect to width measures considers parameters that are always at least as large as treewidth. The hope has been that insights gained from these considerations might eventually lead to settling the main question. In particular, fixed-parameter tractable algorithms for GRAPH ISOMORPHISM has been given for the following parameters: tree-depth [9], feedback vertex set number [24], connected path distance width [27], and rooted tree distance width [35]. Very recent advances by Otachi and Schweitzer [28] give FPT algorithms for parameterizations by root-connected tree distance width and by connected strong treewidth. Even though all these parameters are typically much larger than treewidth, already settling fixed-parameter tractability for them required many new ideas and considerable technical effort. This supports the statement by Kawarabayashi and Mohar [21] that "[...] even for graphs of bounded treewidth, the graph isomorphism problem is not trivial at all".

Our results and techniques.: In this paper we answer the question of fixed-parameter tractability of GRAPH ISOMORPHISM parameterized by treewidth in the affirmative, by proving the following theorem:

Theorem 1.1. *There exists an algorithm that, given an integer k and two graphs G_1, G_2 on n vertices, works in time $2^{\mathcal{O}(k^5 \log k)} \cdot n^5$ and either correctly concludes that $\mathbf{tw}(G_1) \geq k$ or $\mathbf{tw}(G_2) \geq k$, or determines whether G_1 and G_2 are isomorphic.*

In fact, we prove a stronger statement, that is, we provide a *canonization* algorithm for graphs of bounded treewidth. For this, in Section 6 we define so-called *construction terms*: algebraic expressions encoding construction procedures for graphs of bounded treewidth. Thus, construction terms can be seen as an alternative definition of treewidth via graph grammars (see Lemma 6.1).

Theorem 1.2. *There exists an algorithm that, given a graph G and a positive integer k, in time $2^{\mathcal{O}(k^5 \log k)} \cdot n^5$ either correctly concludes that $\mathbf{tw}(G) \geq k$, or outputs an isomorphism-invariant term $\mathbf{t}$ that constructs G and uses at most $\mathcal{O}(k^4)$ labels. Moreover, this term has length at most $\mathcal{O}(k^4 \cdot n)$.*

Thus, the proof of Theorem 1.1 boils down to applying the algorithm of Theorem 1.2 to both G_1 and G_2, and verifying whether the obtained terms are equal. Theorem 1.2 implies also that graphs of bounded treewidth admit an FPT canonization algorithm in the sense of Grohe and Marx [16], [17], that is, an algorithm that, given G, constructs a canonical graph $\mathfrak{c}(G)$ on the vertex set $\{1, 2, \ldots, |V(G)|\}$ isomorphic to G, such that $\mathfrak{c}(G_1) = \mathfrak{c}(G_2)$ whenever G_1 and G_2 are isomorphic. To construct $\mathfrak{c}(G)$ we just need to evaluate the canonical construction term. Let us point out that for all the aforementioned classes where GRAPH ISOMORPHISM can be solved in XP time, corresponding XP canonization algorithms were also developed: for bounded-degree graphs by Babai and Luks [2], for H-minor-free graphs by Ponomarenko [29], and for H-topological-minor-free graphs by Grohe and Marx [16], [17].

We remark that the approach to treewidth and tree decompositions via graph grammars and tree automata dates back to the earliest works on this subject, and is the foundation of intensive research on links between treewidth and monadic second-order logic; we refer to a recent monograph of Courcelle and Engelfriet [10] for a more thorough introduction. Unfortunately there is currently no agreed standard notation for these concepts. In order to ensure clarity of notation, we introduce our own formalism in Section 6.1.

We now sketch the main ideas behind the proof of Theorem 1.2. The starting point is the classic algorithm of Bodlaender [4] that resolves isomorphism of graphs of treewidth k in time $\mathcal{O}(n^{k+4.5})$. Essentially, this algorithm considers all the $(k+1)$-tuples of vertices of each of the graphs as potential bags of a tree decomposition, and tries to assemble both graphs from these building blocks in the same manner using dynamic programming. It turns out that with a slight modification, the algorithm of Bodlaender can be extended to solve the canonization problem as well. We now direct our attention to speeding up the algorithm.

Our idea is the following: if we were able to constrain ourselves to a small enough family of potential bags, then basically the same algorithm restricted to the pruned family of states would work in FPT time. For this to work we need the family to be of size $f(k) \cdot n^c$ for some function f and constant c. Furthermore, we would need an algorithm that given as input the graph G, computes this family in FPT time. Finally, we would need this family of bags to be *isomorphism invariant*.

Informally, we want the pruned family of bags to only depend on the (unlabelled) graph G, and not on the labelling of vertices of G given as input. A definition of what we mean by isomorphism invariance is given in the preliminaries.

Therefore, the goal is to find a family $\mathcal{B} \subseteq 2^{V(G)}$ of potential bags that is on one hand isomorphism-invariant and reasonably small, and on the other hand it is rich enough to contain all the bags of some tree decomposition of width at most $g(k)$, for some function g. Coping with this task is the main contribution of this paper, and this is achieved in Theorems 3.4, 4.2, and 5.5. We remark that the very recent, independent work of Otachi and Schweitzer [28] also observes that finding an isomorphism-invariant family of potential bags of size $f(k) \cdot n^c$ is sufficient for designing an isomorphism test; however, their approach for proving this initial step is completely different.

The crucial idea of our construction of a small isomorphism-invariant family of bags is to start with the classic 4-approximation algorithm for treewidth given in the Graph Minors series by Robertson and Seymour [30]; we also refer to the textbook of Kleinberg and Tardos [22] for a comprehensive exposition of this algorithm. Since a good understanding of this algorithm is necessary for our considerations, let us recall it briefly.

The algorithm of Robertson and Seymour constructs a tree decomposition of the input graph G in a top-down manner. More precisely, it is a recursive procedure that at each point maintains a separator S of size at most $3k + \mathcal{O}(1)$, which separates the part of G we are currently working with (call it H) from the rest. The output of the procedure is a tree decomposition of H with the set S as the top adhesion. At each recursive call the algorithm proceeds as follows. If S is small, the algorithm adds to S an arbitrarily chosen vertex $u \in V(H)$. If S is large, the algorithm attempts to break S into two roughly equal-sized pieces using a separator X of size at most $k+1$. The fact that the graph has treewidth at most k ensures that such a separator X will always exist. The new set S', defined as $S \cup \{u\}$ or $S \cup X$, becomes the top-most bag. Below this bag we attach tree decompositions obtained from the recursive calls for instances $(H := G[N[Z]], S := N(Z))$, where Z iterates over the family of vertex sets of the connected components of $H \setminus S'$. The crucial point is that if S is large (of size roughly $3k$) and X is of size at most $k+1$, then every such component Z will neighbour at most $|S|$ vertices of S', and hence the size of S will not increase over the course of the recursion.

Our high-level plan is to modify this algorithm so that it works in an (almost) isomorphism-invariant manner, and then return all the produced bags as the family $\mathcal{B}$. At a glance, it seems that the algorithm inherently uses two 'very non-canonical' operations: adding an arbitrarily chosen vertex u and breaking S using an arbitrarily chosen separator X. Especially canonizing the choice of the separator X seems like a hard nut to crack. We circumvent this obstacle in the following manner: we take X to be the union of 'all possible' separators that break S evenly. The problem that now arises is that it is not clear how to bound the number of neighbours in $S' = S \cup X$ that can be seen from a connected component we want to recurse on; recall that we needed to bound this number by $|S|$, in order to avoid an explosion of the bag sizes throughout the course of the algorithm. The crucial technical insight of the paper, proved in Lemma 3.2, is that if one defines 'all possible separators' in a very careful manner, then this bound holds. The proof of Lemma 3.2 relies on a delicate argument that exploits submodularity of vertex separations.

Even if the problem of canonizing the choice of X is solved, we still need to cope with the arbitrary choice of u in case S is small. It turns out that the problem appears essentially only if the set S is very well connected: for every two vertices $x, y \in S$, the vertex flow between x and y is more than k. In other words, the set S constitutes a *clique separator* in the *k-improved graph of G*, i.e., a graph derived from G by making every pair of vertices with vertex flow more than k adjacent. It is known that for the sake of computing tree decompositions of width k one can focus on the k-improved graph rather than the original one (see e.g. [5] and Lemma 2.6). Therefore, our algorithm will work without any problems providing that the k-improved graph of the input one does not admit any clique separators. The produced tree decomposition has width $2^{\mathcal{O}(k \log k)}$, due to a possibly exponential number of separators breaking S at each step, and is isomorphism-invariant up to the choice of a single vertex from which the whole procedure begins. By running the algorithm from every possible starting point and computing the union of the families of bags of all the obtained decompositions, we obtain an isomorphism-invariant family of potential bags of size $\mathcal{O}(n^2)$. This result is obtained in Theorem 3.4, which summarizes the case when no clique separators are present.

However, the behaviour of clique separators in the graph has been well understood already in the 1980s, starting from the work of Tarjan [32], and studied intensively from a purely graph-theoretical viewpoint. It turns out that all inclusion-wise minimal clique separators of a graph form a tree-like structure, giving raise to so-called *clique minimal separator decomposition*, which decomposes the graph into pieces that do not admit any clique separators. These pieces are often called *atoms*. Most importantly for us, the set of atoms of a graph is isomorphism-invariant, and can be computed in polynomial time. Therefore, in the general case we can compute the clique minimal separator decomposition of the k-improved graph of the input graph, run the algorithm for the case of no clique separators on each atom separately, and finally output the union of all the obtained families. This result is obtained in Theorem 4.2. We refer to the introductory paper of Berry et al. [3] for more information on clique separators.

The family $\mathcal{B}$ given by Theorem 4.2 is essentially already sufficient for running the modified algorithm of Bodlaender [4] on it, and thus resolving fixed-parameter tractability of GRAPH ISOMORPHISM parameterized by treewidth. However, the bags contained in the family $\mathcal{B}$ may be of size as much as $2^{\mathcal{O}(k \log k)}$. Our canonization algorithm considers every permutation of every candidate bag. Hence, this would result in a double-exponential dependence on k in the running time. In Section 5 we demonstrate how to reduce this dependence to $2^{\mathrm{poly}(k)}$. More precisely, we prove that instead of the original family $\mathcal{B}$, we can consider a modified family $\mathcal{B}'$ constructed as follows: for every $B \in \mathcal{B}$, we replace B with all the subsets of B that have size $\mathcal{O}(k^4)$. Thus, every bag of $\mathcal{B}$ gives raise to $\binom{2^{\mathcal{O}(k \log k)}}{\mathcal{O}(k^4)} = 2^{\mathcal{O}(k^5 \log k)}$ sets in $\mathcal{B}'$, and hence $|\mathcal{B}'| \leq 2^{\mathcal{O}(k^5 \log k)} \cdot |\mathcal{B}|$. However, now every bag of $\mathcal{B}'$ has only $2^{\mathcal{O}(k^4 \log k)}$ possible permutations, instead of a number that is double-exponential in k. In this manner, we can trade a possible explosion of the size of the constructed family for a polynomial upper bound on the cardinality of its members. This trade-off is achieved in Theorem 5.5, and leads to a better time complexity of the canonization algorithm.

Organization of the paper.: Section 2 contains preliminaries. Sections 3, 4, 5 contain proofs of Theorems 3.4, 4.2, 5.5, respectively. In Section 6 we utilize Theorem 5.5 to present the canonization algorithm, i.e., to prove Theorems 1.1 and 1.2. In particular, Section 6.1 introduces the formalism of construction terms. Proofs of results denoted by (♠) could be found in the longer version of the paper available at arxiv.

2. Preliminaries

In most cases, we use standard graph notation, see e.g. [12].

Separations, separators, and clique separators.: We recall here standard definitions and facts about separations and separators in graphs.

Definition 2.1 (separation). A pair (A, B) where $A \cup B = V(G)$ is called a *separation* if $E(A \setminus B, B \setminus A) = \emptyset$. The *separator* of (A, B) is $A \cap B$ and the *order* of a separation (A, B) is $|A \cap B|$.

Let $X, Y \subseteq V(G)$ be two not necessarily disjoint subsets of vertices. Then a separation (A, B) is an $X - Y$ *separation* if $X \subseteq A$ and $Y \subseteq B$. The classic Menger's theorem states that for given X, Y, the minimum order of an $X - Y$ separation is equal to maximum vertex-disjoint flow between X and Y in G. This minimum order (denoted further $\mu(X, Y)$) can be computed in polynomial time, using for instance the Ford-Fulkerson algorithm. Moreover, among the $X - Y$ separations of minimum order there exists a unique one with inclusion-wise minimal A, and a unique one with inclusion-wise minimal B. We will call these minimum-order $X - Y$ separations *pushed towards* X and *pushed towards* Y, respectively. It is known that the Ford-Fulkerson algorithm can actually find the minimum order $X - Y$ separations that are pushed towards X and Y within the same running time.

For two vertices $x, y \in V(G)$, by $\mu(x, y)$ we denote the minimum order of a separation (A, B) in G such that $x \in A \setminus B$ and $y \in B \setminus A$. Note that if $xy \in E(G)$ then such a separation does not exist; in such a situation we put $\mu(x, y) = +\infty$. Again, classic Menger's theorem states that for nonadjacent x and y, the value $\mu(x, y)$ is equal to the maximum number of internally vertex-disjoint $x - y$ paths that can be chosen in the graph, and this value can be computed in polynomial time using the Ford-Fulkerson algorithm. The notions of minimum-order separations pushed towards x and y are defined analogously as before. If the graph we are referring to is not clear from the context, we put it in the subscript by the symbol μ.

We emphasize here that, contrary to $X - Y$ separations, in an $x - y$ separation we require $x \in A \setminus B$ and $y \in B \setminus A$, that is, the separator $A \cap B$ cannot contain x nor y. This, in particular, applies to minimum-order $x - y$ separations pushed towards x or y.

Definition 2.2 (clique separation). A separation (A, B) is called a *clique separation* if $A \setminus B \neq \emptyset$, $B \setminus A \neq \emptyset$, and $A \cap B$ is a clique in G.

Note that an empty set of vertices is also a clique, hence any separation with an empty separator is in particular a clique separation. We will say that a graph is *clique separator free* if it does not admit any clique separation. Such graphs are often called also *atoms*, see e.g. [3]. From the previous remark it follows that every clique separator free graph is connected.

Tree decompositions.: In this paper it is most convenient to view tree decompositions of graphs as rooted. The following notation originates in Marx and Grohe [17], and we use an extended version borrowed from [11].

Let T be a rooted tree and let t be any non-root node of T. The parent of t in T will be denoted by $\mathrm{parent}(t)$. A node s is a *descendant* of t, denoted $s \preceq t$, if t lies on the unique path connecting s to the root. We will also say that t is an *ancestor* of s. Note that in this notation every node is its own descendant as well as its own ancestor.

Definition 2.3 (tree decomposition). A *tree decomposition* of a graph G is a pair (T, β), where T is a rooted tree and $\beta \colon V(T) \to 2^{V(G)}$ is a mapping such that:

- for each node $v \in V(G)$, the set $\{t \in V(G) \mid v \in \beta(t)\}$ induces a nonempty and connected subtree of T,
- for each edge $e \in E(G)$, there exists $t \in V(T)$ such that $e \subseteq \beta(t)$.

Sets $\beta(t)$ for $t \in V(T)$ are called the *bags* of the decomposition, while sets $\beta(s) \cap \beta(t)$ for $st \in E(T)$ are called the *adhesions*. We sometimes implicitly identify a node of T with the bag associated with it. The *width* of a tree decomposition T is equal to its maximum bag size decremented by one, i.e., $\max_{t \in V(T)} |\beta(t)| - 1$. The *adhesion width* of T is equal to its maximum adhesion size, i.e., $\max_{st \in E(T)} |\beta(s) \cap \beta(t)|$. We define also additional mappings as follows:

$$\gamma(t) = \bigcup_{u \preceq t} \beta(u),$$

$$\sigma(t) = \begin{cases} \emptyset & \text{if t is the root of } T \\ \beta(t) \cap \beta(\mathrm{parent}(t)) & \text{otherwise,} \end{cases}$$

$$\alpha(t) = \gamma(t) \setminus \sigma(t).$$

The *treewidth* of a graph, denoted $\mathbf{tw}(G)$, is equal to the minimum width of its tree decomposition. Let us remark that in this paper we will be mostly working with graphs of treewidth *less than* k, while most of the literature on the subject considers graphs of treewidth *at most* k. This irrelevant detail will help us avoid clumsy additive constants in many arguments.

If $\mathcal{B} \subseteq 2^{V(G)}$ is a family of subsets of vertices, then we say that $\mathcal{B}$ *captures* a tree decomposition (T, β) if $\beta(t) \in \mathcal{B}$

for each $t \in V(T)$. In this context we often call $\mathcal{B}$ a *family of potential bags*.

Graphs of treewidth at most k are known to be *k-degenerate*, that is, every subgraph of a graph of treewidth at most k has a vertex of degree at most k. This in particular implies that an n-vertex graph of treewidth at most k can have at most kn edges.

Let G be a graph and let $S \subseteq V(G)$. We say that a separation (A, B) in G is *α-balanced for S* if $|(A \setminus B) \cap S|, |(B \setminus A) \cap S| \leq \alpha|S|$. The following lemma states that graphs of bounded treewidth provide balanced separations of small order.

Lemma 2.4 ([6]). *Let G be a graph with $\mathbf{tw}(G) < k$ and let $S \subseteq V(G)$ be a subset of vertices. Then there exists a $\frac{2}{3}$-balanced separation for S of order at most k.*

Improved graph.: For a positive integer k, we say that a graph H is *k-complemented* if the implication $(\mu_H(x, y) \geq k) \Rightarrow (xy \in E(H))$ holds for every pair of vertices $x, y \in V(H)$. For every graph G we can construct a *k-improved graph $G^{\langle k \rangle}$* by having $V(G^{\langle k \rangle}) = V(G)$ and $xy \in E(G^{\langle k \rangle})$ if and only if $\mu_G(x, y) \geq k$. Observe that $G^{\langle k \rangle}$ is a supergraph of G, since $\mu_G(x, y) = +\infty$ for all x, y that are adjacent in G. Moreover, observe that every k-complemented supergraph of G must be also a supergraph of $G^{\langle k \rangle}$. It appears that $G^{\langle k \rangle}$ is k-complemented itself, and hence it is the unique minimal k-complemented supergraph of G.

Lemma 2.5 ([7]). *For every graph G and positive integer k, the graph $G^{\langle k \rangle}$ is k-complemented. Consequently, it is the unique minimal k-complemented supergraph of G.*

The following lemma formally relates tree decompositions of a graph and its improved graph, and states that for the sake of computing a tree decomposition of small width we may focus on the improved graph.

Lemma 2.6 (♠). *Let k be a positive integer and let G be a graph. Then every tree decomposition (T, β) of G that has width less than k, is also a tree decomposition of $G^{\langle k \rangle}$. In particular, if $\mathbf{tw}(G) < k$ then $\mathbf{tw}(G) = \mathbf{tw}(G^{\langle k \rangle})$.*

The idea of focusing on the improved graph dates back to the work of Bodlaender on a linear time FPT algorithm for treewidth [5]. Actually, Bodlaender was using a weaker variant an improved graph, where an edge is added only if the vertices in question share at least k common neighbors. The main point was that this weaker variant can be computed in linear time [5] with respect to the size of the graph. In our work we use the stronger variant, and we can afford spending more time on computing the improved graph.

Lemma 2.7 (♠). *There exists an algorithm that, given a positive integer k and a graph G on n vertices, works in $\mathcal{O}(k^2 n^3)$ time and either correctly concludes that $\mathbf{tw}(G) \geq k$, or computes $G^{\langle k \rangle}$.*

a) Isomorphisms and isomorphism-invariance.: We say that two graphs G_1, G_2 are *isomorphic* if there exists a bijection $\phi \colon V(G_1) \to V(G_2)$, called *isomorphism*, such that $xy \in E(G_1) \Leftrightarrow \phi(x)\phi(y) \in E(G_2)$ for all $x, y \in V(G_1)$. We will often say that some object $\mathcal{D}(G)$ (e.g., a set of vertices, a family of sets of vertices), whose definition depends on the graph, is *isomorphism-invariant* or *canonical*. By this we mean that for any graph G' that is isomorphic to G with isomorphism ϕ, it holds that $\mathcal{D}(G') = \phi(\mathcal{D}(G))$, where $\phi(\mathcal{D}(G))$ denotes the object $\mathcal{D}(G)$ with all the vertices of G replaced by their images under ϕ. The precise meaning of this term will be always clear from the context. Most often, isomorphism-invariance of some definition or of the output of some algorithm will be obvious, since the description does not depend on the internal representation of the graph, nor uses any tie-breaking rules for choosing arbitrary objects.

We also extend the notion of isomorphism-invariance to *structures*, which are formed by the considered graph G together with some object $\mathcal{E}$ (e.g., a vertex, a set of vertices, a subgraph). Such structures usually represent the graph together with an initial set of choices made by some algorithm, for instance the starting vertex from which the construction of a tree decomposition begins. We say that some object $\mathcal{D}(G, \mathcal{E})$, whose definition is dependant on the structure $(G, \mathcal{E})$, is *invariant under isomorphism of* $(G, \mathcal{E})$, if $\mathcal{D}(G', \mathcal{E}') = \phi(\mathcal{D}(G, \mathcal{E}))$ for any structure $(G', \mathcal{E}')$ such that ϕ is an isomorphism between G and G' that additionally satisfies $\phi(\mathcal{E}) = \mathcal{E}'$. Again, the precise definition of this term will be always clear from the context.

3. The case of no clique separators

Let G be graph and let $S \subseteq V(G)$ be any subset of vertices. The following definition will be a crucial technical ingredient in our reasonings.

Definition 3.1. Suppose that (S_L, S_R) is a partition of S. We say that a separation (A, B) of G is *stable for* (S_L, S_R) if (A, B) is a minimum-order $S_L - S_R$ separation. A separation (A, B) is *S-stable* if it is stable for some partition of S.

Note that $(S_L, V(G))$ and $(V(G), S_R)$ are both $S_L - S_R$ separations, and they have orders $|S_L|$ and $|S_R|$, respectively. Hence, if (A, B) is a separation that is stable for (S_L, S_R), then in particular $|A \cap B| \leq \min(|S_L|, |S_R|)$.

The following lemma will be the main tool for construction of an isomorphism invariant family of candidate bags.

Lemma 3.2. *Let G be a graph, let $S \subseteq V(G)$ be any subset of vertices, and let $\mathcal{F}$ be any finite family of S-stable separations. Define*

$$X := S \cup \bigcup_{(A,B) \in \mathcal{F}} A \cap B.$$

Suppose that Z is the vertex set of any connected component of $G \setminus X$. Then $|N(Z)| \leq |S|$.

Proof. We proceed by induction w.r.t. $|\mathcal{F}|$. For $\mathcal{F} = \emptyset$ we have $N(Z) \subseteq X = S$, so the claim is trivial.

Assume then that $\mathcal{F} = \mathcal{F}' \cup \{(A, B)\}$ for some family $\mathcal{F}'$ with $|\mathcal{F}'| < |\mathcal{F}|$, and let X' be defined in the same manner for $\mathcal{F}'$ as X is for $\mathcal{F}$. As (A, B) is S-stable, there is some a partition (S_L, S_R) of S such that (A, B) is a minimum-order $S_L - S_R$ separation. Let Z be the vertex set of any connected component of $G \setminus X$, and let $Z' \supseteq Z$ be the vertex set of the connected component of $G \setminus X'$ that contains Z. Since $G[Z]$ is connected and $Z \cap (A \cap B) = \emptyset$, we have that either $Z \subseteq A \setminus B$ or $Z \subseteq B \setminus A$; without loss of generality assume the former.

Let $(C, D) = (V(G) \setminus Z', N[Z'])$. Observe that (C, D) is a separation in G. Moreover, since $Z' \cap S = \emptyset$, then (C, D) is a $S - Z'$ separation, so in particular also a $S - Z$ separation. Furthermore, observe that $C \cap D = N(Z')$, so by the induction hypothesis we obtain $|C \cap D| = |N(Z')| \leq |S|$. We can partition $V(G)$ into 9 parts, where the part a vertex belongs to depends on its membership to $A \setminus B$, $A \cap B$, or $B \setminus A$, and its membership to $C \setminus D$, $C \cap D$, or $D \setminus C$. Let us call these parts $Q_{1,1}, Q_{1,2}, \ldots, Q_{3,3}$.

Observe now that $Z \subseteq (A \setminus B) \cap (D \setminus C) = Q_{1,3}$. Since $X \setminus X' \subseteq A \cap B$ and $C \cap D = N(Z') \subseteq X'$, we have that $N(Z) \subseteq (A \cap C \cap D) \cup (Z' \cap A \cap B) = Q_{1,2} \cup Q_{2,2} \cup Q_{2,3}$. On the other hand, by induction hypothesis we have $|N(Z')| = |Q_{1,2} \cup Q_{2,2} \cup Q_{3,2}| \leq |S|$. Hence, to prove that $|N(Z)| \leq |S|$, it suffices to prove that $|Q_{1,2} \cup Q_{2,2} \cup Q_{2,3}| \leq |Q_{1,2} \cup Q_{2,2} \cup Q_{3,2}|$, or, equivalently, $|Q_{2,3}| \leq |Q_{3,2}|$.

To this end, consider a pair of subsets $(L, R) = (A \cup D, B \cap C)$. We first claim that (L, R) is a separation. Indeed, if $u \in L \setminus R = Q_{1,1} \cup Q_{1,2} \cup Q_{1,3} \cup Q_{2,3} \cup Q_{3,3}$ and $v \in R \setminus L = Q_{3,1}$, then existence of an edge uv would contradict the fact that both (A, B) and (C, D) are separations. Now we claim that (L, R) is a $S_L - S_R$ separation. Indeed, we have $S_L \subseteq A \subseteq L$, and moreover $S_R \subseteq B$ and $S_R \subseteq S \subseteq C$ implies that $S_R \subseteq B \cap C = R$. Since (A, B) is a minimum-order $S_L - S_R$ separation, we have

$$|Q_{2,1} \cup Q_{2,2} \cup Q_{3,2}| = |L \cap R| \geq |A \cap B| = |Q_{2,1} \cup Q_{2,2} \cup Q_{2,3}|.$$

Hence indeed $|Q_{2,3}| \leq |Q_{3,2}|$ and we are done. □

Lemma 3.2 is used in the following result, which encapsulates one step of the construction of an isomorphism-invariant family of candidate bags. In the sequel, we will use the following parameters:

$$\begin{aligned} \tau &= 6k, \\ \rho &= \tau + 2(k-1) \cdot \binom{\tau}{2} = \mathcal{O}(k^3), \\ \zeta &= \rho + 2k \cdot \binom{\rho}{k+1}^2 = 2^{\mathcal{O}(k \log k)}. \end{aligned}$$

Lemma 3.3 (♠). *Let k be a positive integer and let H be a connected graph that is k-complemented. Let $S \subseteq V(H)$ be a subset of vertices such that (a) $\emptyset \neq S \subsetneq V(H)$, (b) $|S| \leq \rho$, (c) S does* not *induce a clique, (d) $H \setminus S$ is connected, and (e) $S = N_H(V(H) \setminus S)$. There exists an algorithm that either correctly concludes that $\mathbf{tw}(H) \geq k$, or finds a set X with the following properties:*

(i) $X \supsetneq S$, that is, X is a proper superset of S;
(ii) $|X| \leq \zeta$; and
(iii) if Z is the vertex set of any connected component of $H \setminus X$, then $|N(Z)| \leq \rho$.

The algorithm runs in $2^{\mathcal{O}(k \log k)} \cdot |V(H)|$ time and the constructed set X is invariant with respect to isomorphisms of the structure (H, S).

Armed with Lemma 3.3, we may proceed to the main result of this section, that is, the enumeration of an isomorphism-invariant family of bags that captures a tree decomposition of reasonably small width.

Theorem 3.4 (♠). *Let k be a positive integer, and let G be a graph on n vertices that is clique-separator free (in particular, connected), and k-complemented. There exists an algorithm that computes an isomorphism-invariant family of potential bags $\mathcal{B} \subseteq 2^{V(G)}$ with the following properties:*

(i) $|B| \leq \zeta$ for each $B \in \mathcal{B}$;
(ii) $|\mathcal{B}| = \mathcal{O}(n^2)$;
(iii) assuming that $\mathbf{tw}(G) < k$, the family $\mathcal{B}$ captures some tree decomposition of G that has width at most $\zeta + 1 = 2^{\mathcal{O}(k \log k)}$ and adhesion width at most $\rho = \mathcal{O}(k^3)$.

Moreover, the algorithm runs in $2^{\mathcal{O}(k \log k)} \cdot n^3$ time.

4. Taming clique separators

The main tool that we will use in this section is a decomposition theorem that breaks the graph using minimal clique separators into pieces that cannot be decomposed further. It appears that such a decomposition can be done, with a set of its bags defined in a unique manner. The idea of decomposing a graph using clique separators dates back to the work of Tarjan [32], and has been studied intensively thereafter. We refer to an introductory article of Berry et al. [3] for more details. The following theorem states all the properties of this decomposition in the language of tree decompositions.

Theorem 4.1 (see e.g. [3]). *Let G be a connected graph. There exists a tree decomposition $(T^\star, \beta^\star)$ of G, called* clique minimal separator decomposition*, with the following properties:*

- *for every $t \in V(T^\star)$, $G[\beta^\star(t)]$ is clique-separator free;*
- *each adhesion of $(T^\star, \beta^\star)$ is a clique in G.*

Moreover, $T^\star$ has at most $n-1$ nodes, and the bags of $(T^\star, \beta^\star)$ are exactly all the inclusion-wise maximal induced subgraphs of G that are clique-separator free. Consequently, the family of bags of $(T^\star, \beta^\star)$ is isomorphism-invariant. Finally, the decomposition $(T^\star, \beta^\star)$ can be computed in $\mathcal{O}(nm)$ time.

We remark that the exact shape of the clique minimal separator decomposition is not isomorphism-invariant: the construction procedure depends on the order in which inclusion-wise minimal clique separators of the graph are considered. However, the family of bags of this decomposition is isomorphism-invariant, since these bags may be characterized as all the inclusion-wise maximal induced subgraphs of G that are clique-separator free. In other words, all the possible runs of the decomposition algorithm yield the same family of bags, just arranged in a different manner.

Theorem 4.1 enables us to conveniently extend Theorem 3.4 to graphs that may contain clique separations.

Theorem 4.2 (♠). *Let k be a positive integer, and let G be a graph on n vertices that is k-complemented. There exists an algorithm that computes an isomorphism-invariant family of bags $\mathcal{B}$ with the following properties:*

(i) $|B| \leq \zeta$ *for each* $B \in \mathcal{B}$*;*
(ii) $|\mathcal{B}| \leq \mathcal{O}(k^2n^2)$*;*
(iii) assuming that $\mathbf{tw}(G) < k$*, the family* $\mathcal{B}$ *captures some tree decomposition of* G *that has width at most* $\zeta+1 = 2^{\mathcal{O}(k \log k)}$ *and adhesion width at most* $\rho = \mathcal{O}(k^3)$.

Moreover, the algorithm runs in $2^{\mathcal{O}(k \log k)} \cdot n^3$ *time.*

5. Reducing bag sizes

Definition 5.1. Let G be a graph, let $\mathcal{B} \subseteq 2^{V(G)}$ be a family of candidate bags, and let q be a positive integer. Then

$$\mathcal{B}^{\leq q} := \{X \subseteq V(G) \ : \ |X| \leq q \text{ and } \exists_{B\in\mathcal{B}} X \subseteq B\}.$$

Note that if family $\mathcal{B}$ is isomorphism-invariant, then so does $\mathcal{B}^{\leq q}$.

The following lemma will be the crucial technical insight of this section. Intuitively it states that by focusing on the family $\mathcal{B}^{\leq q}$ for large enough q, instead of the original $\mathcal{B}$, we still capture some tree decomposition of the graph that has a reasonably small width. The crucial point here is that the candidate bags of $\mathcal{B}^{\leq q}$ are much smaller than those of $\mathcal{B}$. Since the canonization algorithm of Section 6 is essentially considering all permutations of all the bags, reducing the bag size will be useful for speeding it up.

Lemma 5.2 (♠). *Let* G *be a connected graph of treewidth less than* k*, and let* $\mathcal{B} \subseteq 2^{V(G)}$ *be a family of candidate bags that captures some tree decomposition of* G *that has width at most* k' *and adhesion width at most* ℓ*, where* $k \leq \ell \leq k'$*. Then the family* $\mathcal{B}^{\leq (k+1)\ell}$ *captures some tree decomposition of* G *that has width at most* $(k+1)\ell - 1$*.*

We also neeed the following definition.

Definition 5.3 (connectivity-sensitive tree decomposition). We say that a tree decomposition (T, β) of a connected graph G is *connectivity-sensitive* (*cs-tree decomposition*, for short), if the following conditions are satisfied for every $t \in V(T)$:

- $G[\alpha(t)]$ is connected, and
- $\sigma(t) = N_G(\alpha(t))$.

Actually, one can see that the tree decomposition constructed in the proof of Theorem 3.4 is connectivity sensitive, so the family $\mathcal{B}$ actually captures a cs-tree decomposition of the graph with required width and adhesion width. A similar conclusion, however, is not so clear in the case of Theorem 4.2. Fortunately, it is easy to see that every tree decomposition of a connected graph can be turned into a cs-tree decomposition without increasing the widths.

Lemma 5.4 (♠). *If a connected graph* G *admits a tree decomposition* (T, β) *of width* k *and adhesion width* ℓ*, then* G *admits also a cs-tree decomposition* (T', β') *of width at most* k *and adhesion width at most* ℓ*. Moreover, every bag appearing in* (T', β') *is a subset of some bag of* (T, β)*.*

Using Lemma 5.2, we can further refine Theorem 4.2. The new property (iv) is a technical condition that will be used later.

Theorem 5.5. *Let* k *be a positive integer, and let* G *be a graph on* n *vertices that is connected and* k*-complemented. There exists an algorithm that computes an isomorphism-invariant family of bags* $\mathcal{B}$ *with the following properties:*

(i) $|B| \leq (k+1)\rho \in \mathcal{O}(k^4)$ *for each* $B \in \mathcal{B}$*;*
(ii) $|\mathcal{B}| \leq 2^{\mathcal{O}(k^5 \log k)} \cdot n^2$*;*
(iii) assuming that $\mathbf{tw}(G) < k$*, the family* $\mathcal{B}$ *captures some tree decomposition of* G *that has width at most* $(k+1)\rho - 1 \in \mathcal{O}(k^4)$*;*
(iv) family $\mathcal{B}$ *is closed under taking subsets.*

Moreover, the algorithm runs in $2^{\mathcal{O}(k^5 \log k)} \cdot n^3$ *time.*

Proof. We run the algorithm of Theorem 4.2 on the graph G to obtain an isomorphism-invariant family $\mathcal{B}_0$. Then, we output the family $\mathcal{B} := \mathcal{B}_0^{\leq (k+1)\rho}$. Observe that since $|B| \leq \zeta$ for each $B \in \mathcal{B}_0$, then each $B \in \mathcal{B}_0$ gives rise to at most $\sum_{i=0}^{(k+1)\rho} \binom{\zeta}{i} \in 2^{\mathcal{O}(k^5 \log k)}$ sets in the output family $\mathcal{B}$. This justifies the bound on $|\mathcal{B}|$ (property (ii)) and on the running time. Properties (i) and (iv) follow directly from the construction, and property (iii) follows from Lemma 5.2. □

6. Canonization

In this section we utilize the isomorphism-invariant family of candidate bags constructed in Theorems 3.4, 4.2, and 5.5 to give a canonization algorithm for graphs of bounded treewidth running in FPT time. First we introduce a concept that we call *construction terms*, which is an alternative definition of treewidth and tree decompositions via graph grammars. Our canonization algorithm then produces a canonical expression (construction term) that builds the graph; the isomorphism tests boils down to verifying equality of these canonical expressions.

A. Construction terms

The formalization given in this section has been known in the graph grammar literature from eighty's. We refer to [6], [33] for a review on these topics. We would also like to mention that the materials presented in this subsection is not much more than a formalization of what is commonly known as nice tree decomposition [23]. We give the details here to make the presentation self-content.

For a positive integer q we denote by $[q] = \{1, 2, \ldots, q\}$. For a function f by $f[x \to y]$ we denote a function defined as follows:

$$f[x \to y](z) = \begin{cases} y & \text{if } z = x, \\ f(z) & \text{otherwise.} \end{cases}$$

Note that this definition is correct regardless whether x was in the domain of f or not. If x belongs to the domain of f, then by $f[x \to \bot]$ we denote the function $f \setminus \{(x, f(x))\}$, i.e., f with x deleted from the domain.

Let k be a positive integer and let $\Sigma = \{1, 2, \ldots, k\}$ be an alphabet of k labels. We now define a family $\mathbb{T}$ of terms; each term $\mathbf{t} \in \mathbb{T}$ will have a prescribed subset $\mathbf{used}(\mathbf{t}) \subseteq \Sigma$ of labels *used* by $\mathbf{t}$, and a graph $\mathbf{bag}(\mathbf{t})$ with vertex set $\mathbf{used}(\mathbf{t})$.

- We have a *leaf term* $\mathfrak{l} \in \mathbb{T}$, with $\mathbf{used}(\mathfrak{l}) = \emptyset$ and $\mathbf{bag}(\mathfrak{l})$ being the empty graph.

- If $\mathbf{t} \in \mathbb{T}$ and $i \in \Sigma \setminus \mathbf{used}(\mathbf{t})$, then we can create an *introduce term* $\mathfrak{i}_i(\mathbf{t}) \in \mathbb{T}$, with $\mathbf{used}(\mathfrak{i}_i(\mathbf{t})) = \mathbf{used}(\mathbf{t}) \cup \{i\}$ and $\mathbf{bag}(\mathfrak{i}_i(\mathbf{t}))$ being $\mathbf{bag}(\mathbf{t})$ with an isolated vertex i introduced.
- If $\mathbf{t} \in \mathbb{T}$ and $i \in \mathbf{used}(\mathbf{t})$, then we can create a *forget term* $\mathfrak{f}_i(\mathbf{t}) \in \mathbb{T}$, with $\mathbf{used}(\mathfrak{f}_i(\mathbf{t})) = \mathbf{used}(\mathbf{t}) \setminus \{i\}$ and $\mathbf{bag}(\mathfrak{f}_i(\mathbf{t})) = \mathbf{bag}(\mathbf{t}) \setminus \{i\}$.
- If $\mathbf{t} \in \mathbb{T}$, $i, j \in \mathbf{used}(\mathbf{t})$, $i \neq j$ and $ij \notin E(\mathbf{bag}(\mathbf{t}))$, then we can create an *introduce edge term* $\mathfrak{e}_{i,j}(\mathbf{t}) \in \mathbb{T}$, with $\mathbf{used}(\mathfrak{e}_{i,j}(\mathbf{t})) = \mathbf{used}(\mathbf{t})$ and $\mathbf{bag}(\mathfrak{e}_{i,j}(\mathbf{t}))$ being $\mathbf{bag}(\mathbf{t})$ with edge ij added.
- Let $q \geq 2$ be any integer. Suppose that there are terms $\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_q \in \mathbb{T}$ such that
  - $\mathbf{used}(\mathbf{t}_1) = \mathbf{used}(\mathbf{t}_2) = \ldots = \mathbf{used}(\mathbf{t}_q)$, and
  - all the graphs $\mathbf{bag}(\mathbf{t}_1), \mathbf{bag}(\mathbf{t}_2), \ldots, \mathbf{bag}(\mathbf{t}_q)$ are edgeless.

  Then we can create a *join term* $\mathfrak{j}(\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_q) \in \mathbb{T}$, with

  $$\mathbf{used}(\mathfrak{j}(\mathbf{t}_1, \ldots, \mathbf{t}_q)) = \mathbf{used}(\mathbf{t}_1) = \ldots = \mathbf{used}(\mathbf{t}_q),$$

  and $\mathbf{bag}(\mathfrak{j}(\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_q))$ being the edgeless graph on vertex set $\mathbf{used}(\mathfrak{j}(\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_q))$.

The family $\mathbb{T}$ comprises all the terms that can be built from leaf terms using introduce, forget, introduce edge, and join terms. Note that the join terms can have arbitrarily large arity, but has to be at least 2. We define the *length* of the term $\mathbf{t}$, denoted $|\mathbf{t}|$, as the total number of operators $\mathfrak{l}, \mathfrak{i}_i, \mathfrak{f}_i, \mathfrak{e}_{i,j}, \mathfrak{j}$ used in it.

Terms from $\mathbb{T}$ have a natural interpretation as expressions building graphs with at most k distinguished vertices. More formally, with every term $\mathbf{t} \in \mathbb{T}$ we associate a pair $\mathfrak{G}[\mathbf{t}] = (G[\mathbf{t}], \lambda[\mathbf{t}])$, called a *labelled graph*, where $G[\mathbf{t}]$ is a graph and $\lambda[\mathbf{t}]$ is bijection between some subset of $V(G[\mathbf{t}])$ of cardinality $|\mathbf{used}(\mathbf{t})|$, and $\mathbf{used}(\mathbf{t})$. The bijection $\lambda[\mathbf{t}]$ is also called the *labelling*. We maintain the invariant that $\lambda[\mathbf{t}]$ is an isomorphism between the graph induced by its domain in $G[\mathbf{t}]$ and the graph $\mathbf{bag}(\mathbf{t})$; this invariant follows by a trivial induction from the definition to follow. The labelled graph $\mathfrak{G}[\cdot]$ is defined as follows:

- If $\mathbf{t} = \mathfrak{l}$, then $G[\mathfrak{l}]$ is an empty graph and $\lambda[\mathfrak{l}]$ is an empty function.
- If $\mathbf{t} = \mathfrak{i}_i(\mathbf{t}')$ for some $\mathbf{t}' \in \mathbb{T}$ and $i \in \Sigma$, then $G[\mathbf{t}]$ is equal to $G[\mathbf{t}']$ with a new independent vertex v introduced, and $\lambda[\mathbf{t}] = \lambda[\mathbf{t}'][v \to i]$.
- If $\mathbf{t} = \mathfrak{f}_i(\mathbf{t}')$ for some $\mathbf{t}' \in \mathbb{T}$ and $i \in \Sigma$, then $G[\mathbf{t}] = G[\mathbf{t}']$ and $\lambda[\mathbf{t}] = \lambda[\mathbf{t}'][\lambda[\mathbf{t}']^{-1}(i) \to \bot]$.
- If $\mathbf{t} = \mathfrak{e}_{i,j}(\mathbf{t}')$ for some $\mathbf{t}' \in \mathbb{T}$ and $i, j \in \Sigma$, $i \neq j$, then $G[\mathbf{t}]$ is equal to $G[\mathbf{t}']$ with an edge between $\lambda[\mathbf{t}']^{-1}(i)$ and $\lambda[\mathbf{t}']^{-1}(j)$ introduced, and $\lambda[\mathbf{t}] = \lambda[\mathbf{t}']$. Recall that $ij \notin E(\mathbf{bag}(\mathbf{t}'))$, so by the induction hypothesis we have that $\lambda[\mathbf{t}']^{-1}(i)$ and $\lambda[\mathbf{t}']^{-1}(j)$ are not adjacent in $G[\mathbf{t}']$.
- Suppose $\mathbf{t} = \mathfrak{j}(\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_q)$ for some $\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_q \in \mathbb{T}$. Then $G[\mathbf{t}]$ is constructed by taking the disjoint union of $G[\mathbf{t}_1], G[\mathbf{t}_2], \ldots, G[\mathbf{t}_q]$, and, for every $i \in \mathbf{used}(\mathbf{t}_1) = \mathbf{used}(\mathbf{t}_2) = \ldots = \mathbf{used}(\mathbf{t}_q)$, identifying all vertices $\{\lambda[\mathbf{t}_j]^{-1}(i) : j = 1, 2, \ldots, q\}$ into one vertex. This identified vertex is assigned label i in the labelling $\lambda[\mathbf{t}]$.

We now say that $\mathbf{t}$ is a *construction term* for graph G if $\mathbf{used}(\mathbf{t}) = \emptyset$ and $G[\mathbf{t}]$ is isomorphic to G. As the reader probably suspects, construction terms and tree decompositions are tightly related.

Lemma 6.1 (♠). *A graph G has treewidth less than k if and only if it admits a construction term that constructs it and uses at most k labels.*

Let $\mathbb{O} = \{\mathfrak{i}_i : i \in \Sigma\} \cup \{\mathfrak{f}_i : i \in \Sigma\} \cup \{\mathfrak{e}_{i,j} : i, j \in \Sigma, i \neq j\} \cup \{\mathfrak{l}, \mathfrak{j}\}$ be the set of operators used in the terms of $\mathbb{T}$. Let us introduce an arbitrary linear order $\trianglelefteq$ on the elements of $\mathbb{O}$: for instance first come operators $\mathfrak{i}_i$, sorted by i, then operators $\mathfrak{f}_i$, sorted by i, then operators $\mathfrak{e}_{i,j}$, sorted lexicographically by (i, j), and finally operators $\mathfrak{l}$ and $\mathfrak{j}$. Given this order, we may inductively define a linear order $\trianglelefteq$ on the terms from $\mathbb{T}$ as follows. Let $\mathbf{t}_1, \mathbf{t}_2$ be two terms, and let $o_1, o_2 \in \mathbb{O}$ be the top-most operations used in $\mathbf{t}_1$ and $\mathbf{t}_2$, respectively. Then relation $\trianglelefteq$ between $\mathbf{t}_1$ and $\mathbf{t}_2$ is defined inductively based on the definition for terms of smaller depth.

- If $o_1 \neq o_2$, then $\mathbf{t}_1 \triangleleft \mathbf{t}_2$ if $o_1 \triangleleft o_2$, and $\mathbf{t}_1 \triangleright \mathbf{t}_2$ if $o_1 \triangleright o_2$.
- If $o_1 = o_2 = \mathfrak{l}$, then $\mathbf{t}_1 = \mathbf{t}_2$.
- If $o_1 = o_2 \notin \{\mathfrak{l}, \mathfrak{j}\}$, then let $\mathbf{t}_1 = o(\mathbf{t}'_1)$ and $\mathbf{t}_2 = o(\mathbf{t}'_2)$, where $o = o_1 = o_2$. If $\mathbf{t}'_1 = \mathbf{t}'_2$ then $\mathbf{t}_1 = \mathbf{t}_2$, if $\mathbf{t}'_1 \triangleleft \mathbf{t}'_2$ then $\mathbf{t}_1 \triangleleft \mathbf{t}_2$, and if $\mathbf{t}'_1 \triangleright \mathbf{t}'_2$ then $\mathbf{t}_1 \triangleright \mathbf{t}_2$.
- Suppose $o_1 = o_2 = \mathfrak{j}$, and let the arity of the join operation in $\mathbf{t}_1, \mathbf{t}_2$ be equal to q_1, q_2, respectively. Let $\mathbf{t}_1 = \mathfrak{j}(\mathbf{t}_{1,1}, \mathbf{t}_{1,2}, \ldots, \mathbf{t}_{1,q_1})$ and $\mathbf{t}_2 = \mathfrak{j}(\mathbf{t}_{2,1}, \mathbf{t}_{2,2}, \ldots, \mathbf{t}_{2,q_2})$. Since terms $\mathbf{t}_{1,j}$ and $\mathbf{t}_{2,j}$ has smaller depth than $\mathbf{t}_1$ and $\mathbf{t}_2$, respectively, the order $\trianglelefteq$ is already defined for them. Hence, we may compare sequences $(\mathbf{t}_{1,1}, \mathbf{t}_{1,2}, \ldots, \mathbf{t}_{1,q_1})$ and $(\mathbf{t}_{2,1}, \mathbf{t}_{2,2}, \ldots, \mathbf{t}_{2,q_2})$ lexicographically. If $(\mathbf{t}_{1,1}, \mathbf{t}_{1,2}, \ldots, \mathbf{t}_{1,q_1}) \triangleleft (\mathbf{t}_{2,1}, \mathbf{t}_{2,2}, \ldots, \mathbf{t}_{2,q_2})$ then $\mathbf{t}_1 \triangleleft \mathbf{t}_2$, if $(\mathbf{t}_{1,1}, \mathbf{t}_{1,2}, \ldots, \mathbf{t}_{1,q_1}) \triangleright (\mathbf{t}_{2,1}, \mathbf{t}_{2,2}, \ldots, \mathbf{t}_{2,q_2})$ then $\mathbf{t}_1 \triangleright \mathbf{t}_2$, and if $(\mathbf{t}_{1,1}, \mathbf{t}_{1,2}, \ldots, \mathbf{t}_{1,q_1}) = (\mathbf{t}_{2,1}, \mathbf{t}_{2,2}, \ldots, \mathbf{t}_{2,q_2})$ then $\mathbf{t}_1 = \mathbf{t}_2$.

Note here that two join terms that differ only in the order of arguments are considered different, even though they construct the same labelled graph. The term where the arguments are sorted nondecreasingly is considered the smallest.

B. Constructing a canonical construction term

We are finally ready to prove the main result of this paper.

Proof of Theorem 1.2. Let $k' = (k+1)\rho$. Firstly, without loss of generality we assume that G is connected. For disconnected graphs we can apply the algorithm to each connected component $G_1, G_2, \ldots, G_p$ separately, obtaining terms $\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_p$, then sort these terms nondecreasingly so that $\mathbf{t}_1 \trianglelefteq \mathbf{t}_2 \trianglelefteq \ldots \trianglelefteq \mathbf{t}_p$, and output the term $\mathbf{t} := \mathfrak{j}(\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_p)$. Thus, providing that the construction for a connected graph is isomorphism-invariant, then due to the sorting step so is the construction for disconnected graphs.

We now compute the k-improved graph $G^{\langle k \rangle}$, using Lemma 2.7. If computation of this graph revealed that $\mathbf{tw}(G) \geq k$, then we provide a negative answer to the whole algorithm.

Now, we apply Theorem 5.5 to the graph $G^{\langle k\rangle}$, obtaining an isomorphism-invariant family of candidate bags $\mathcal{B}$. Observe that since the definition of $G^{\langle k\rangle}$ is invariant w.r.t. isomorphisms of G, and the definition of $\mathcal{B}$ is invariant w.r.t. isomorphisms of $G^{\langle k\rangle}$, then the family $\mathcal{B}$ is invariant w.r.t. isomorphisms of G.

Assume for a moment that G has a tree decomposition of width less than k. Then, by Lemma 2.6, so does $G^{\langle k\rangle}$. Consequently, by Theorem 5.5 we have that $\mathcal{B}$ captures some tree decomposition of $G^{\langle k\rangle}$ that has width at most $k'-1$. Since $G^{\langle k\rangle}$ is a supergraph of G, this tree decomposition is also a tree decomposition of G. By Lemma 5.4 and property (iv) of Theorem 5.5, we can further infer that $\mathcal{B}$ captures some cs-tree decomposition of G of width at most $k'-1$. Let us denote this cs-tree decomposition by (T,β).

The plan for the rest of the proof is as follows. We provide a dynamic programming algorithm that exploits the family $\mathcal{B}$ to compute a term $\mathbf{t}$ that constructs G. From the algorithm it will be clear that the definition of $\mathbf{t}$ is isomorphism-invariant. It is possible that the computation of $\mathbf{t}$ fails, but only if $\mathbf{tw}(G)\geq k$: using the captured cs-tree decomposition (T,β), we will argue that the algorithm computes some feasible construction term, providing that $\mathbf{tw}(G)<k$. Hence, in case of failure we can safely report that $\mathbf{tw}(G)\geq k$.

Let us define the family of states $\mathbb{S}$ as the family of all the triples (B,λ,Z), where

- $B\in\mathcal{B}$;
- λ is an injective function from B to $[k']$;
- $Z=\emptyset$ or Z is the vertex set of a connected component of $G\setminus B$.

Observe that $|\mathbb{S}|\leq|\mathcal{B}|\cdot k'!\cdot(n+1)=2^{\mathcal{O}(k^5\log k)}\cdot n^3$. For every state $I=(B,\lambda,Z)\in\mathbb{S}$, we compute a term $\mathbf{t}[I]$ that constructs the labeled graph $\mathfrak{G}[I]:=(G[B\cup Z]\setminus\binom{B}{2},\lambda)$, i.e., the graph $G[B\cup Z]$ with all the edges inside B cleared, and with labelling λ on B. The definition of $\mathbf{t}[I]$ will be invariant w.r.t. isomorphisms of the structure $\mathfrak{G}[I]$. Computation of $\mathbf{t}[I]$ can possibly fail, in which case we denote it by $\mathbf{t}[I]=\bot$. The output term $\mathbf{t}$ is simply defined as $\mathbf{t}[\emptyset,\emptyset,V(G)]$, and using the captured cs-tree decomposition (T,β) we will make sure that $\mathbf{t}[\emptyset,\emptyset,V(G)]\neq\bot$ in case $\mathbf{tw}(G)<k$.

To make sure that the inductive definition of $\mathbf{t}[I]$ is well-defined, we also define the *potential* Φ of a state $I=(B,\lambda,Z)$ similarly as in Theorem 3.4: $\Phi(B,\lambda,Z)=2|Z|+|B|$. The definition of $\mathbf{t}[I]$ depends only on the terms for states with a strictly smaller potential. Since the potential is always nonnegative, the definition is valid.

Before we proceed to the definition of $\mathbf{t}[I]$, let us introduce one more helpful definition. We often run into situations where we would like to compute the canonical term for a triple (B,λ,Z) that is not necessarily a state according to our definition, because Z consists several connected components of $G\setminus B$ rather than at most one. To cope with such situations, we define operator $\mathtt{break}[B,\lambda,Z]$. Formally, operator $\mathtt{break}[B,\lambda,Z]$ can be applied to triples (B,λ,Z) where $B\in\mathcal{B}$, λ is an injective function from B to $[k']$, and Z comprises vertex sets of some (possibly zero) connected components of $G\setminus B$. The behaviour of $\mathtt{break}[B,\lambda,Z]$ is defined as follows:

- If $Z=\emptyset$ or $G[Z]$ is connected (equivalently, $(B,\lambda,Z)\in\mathbb{S}$), then we simply put $\mathtt{break}[B,\lambda,Z]=\mathbf{t}[B,\lambda,Z]$.
- If $G[Z]$ consists of more than one connected component, then let $Z_1,Z_2,\ldots,Z_p$ be the vertex sets of these connected components. Let $\mathbf{t}_i=\mathbf{t}[B,\lambda,Z_i]$ for $i=1,2,\ldots,p$. If any of the terms $\mathbf{t}_i$ is equal to $\bot$, then we put $\mathtt{break}[B,\lambda,Z]=\bot$. Otherwise, by sorting the terms if necessary, assume that $\mathbf{t}_1\trianglelefteq\mathbf{t}_2\trianglelefteq\ldots\trianglelefteq\mathbf{t}_p$. Then $\mathtt{break}[B,\lambda,Z]=\mathfrak{j}(\mathbf{t}_1,\mathbf{t}_2,\ldots,\mathbf{t}_p)$.

Observe that the join operation is valid, since we assumed that term $\mathbf{t}_i$ constructs $(G[B\cup Z_i]\setminus\binom{B}{2},\lambda)$, where all the edges between the vertices of B are cleared. We naturally extend the notation $\mathfrak{G}[\cdot]$ to triples that can be arguments of the operator $\mathtt{break}[\cdot]$.

We now proceed to the definition of $\mathbf{t}[I]$ for a state $I=(B,\lambda,Z)\in\mathbb{S}$. We generate a family $\mathcal{C}$ of candidates for $\mathbf{t}[I]$. We put $\mathbf{t}[I]=\bot$ if $\mathcal{C}=\emptyset$, and otherwise $\mathbf{t}[I]$ is defined as the $\trianglelefteq$-minimum element of $\mathcal{C}$. Elements of $\mathcal{C}$ reflect possible ways of obtaining the term constructing $\mathfrak{G}[I]$ from simpler terms.

Firstly, if $Z=B=\emptyset$, then we take $\mathcal{C}=\{\mathfrak{l}\}$.

Assume now that B contains some vertex u that is not adjacent to any vertex of Z. Then, for every such vertex u, we add to $\mathcal{C}$ the term $\mathbf{t}_{\mathfrak{i},u}:=\mathfrak{i}_{\lambda(u)}(\mathbf{t}[I'])$ for $I'=(B\setminus u,\lambda[u\to\bot],Z)$. Formally, we add this term only if $I'\in\mathbb{S}$ and $\mathbf{t}[I]\neq\bot$; the same remark holds also for the other elements of $\mathcal{C}$ to follow. Observe that if $\mathbf{t}[I']$ constructs $\mathfrak{G}[I']$, then $\mathbf{t}_{\mathfrak{i},u}$ constructs $\mathfrak{G}[I]$.

Then, for every vertex $v\in Z$ we consider the possibility that v has just been forgotten. Formally, for each $v\in Z$ and each label $i\in[k']\setminus\lambda(B)$, we add to $\mathcal{C}$ the following term:

$$\mathbf{t}_{\mathfrak{f},v,i}:=\mathfrak{f}_i(\mathfrak{e}_{i,j_1}(\mathfrak{e}_{i,j_2}(\ldots\mathfrak{e}_{i,j_q}(\mathtt{break}[I'])\ldots))),\qquad(1)$$

where $I'=(B\cup\{v\},\lambda[v\to i],Z\setminus v)$ and $j_1<j_2<\ldots<j_q$ are labels in λ of neighbors of v in B in the graph G. Again, if $\mathtt{break}[\cdot]$ cannot be applied to I' or if $\mathtt{break}[I']=\bot$, then we do not add this candidate. Observe that if $\mathtt{break}[I']$ constructs $\mathfrak{G}[I']$, then $\mathbf{t}_{\mathfrak{f},v,i}$ constructs $\mathfrak{G}[I]$.

This concludes the definition of the term $\mathbf{t}[I]$; observe that the definition depends only on the definitions for states with strictly smaller potential, as was promised. It can be easily seen by induction that the definition is invariant with respect to isomorphisms of the structure $(G,\mathfrak{G}[I])$, due to taking the $\trianglelefteq$-minimum from an invariant family of candidates. The following claim shows that, in the end, we obtain a meaningful term provided that $\mathbf{tw}(G)<k$.

Claim 6.2 (♠). *If $\mathbf{tw}(G)<k$ then $\mathbf{t}[\emptyset,\emptyset,V(G)]\neq\bot$.*

We are left with establishing the upper bound on the length of the output term, and analysing the running time of the algorithm. To achieve this goal, we inductively bound the lengths of the terms produced by the algorithm. For a state $I=(B,\lambda,Z)$, define $\phi(I)$ as follows:

$$\phi(I)=(k'+2)\cdot\max(2|Z|-1,0)+|B|+2.\qquad(2)$$

Observe that if $|Z|\geq 1$, then $\phi(I)\leq(k'+2)\cdot 2|Z|$.

Claim 6.3 (♠). *For any $I\in\mathbb{S}$, if $\mathbf{t}[I]\neq\bot$ then $|\mathbf{t}[I]|\leq\phi(I)$.*

Claim 6.3 implies that each term $\mathbf{t}[I]$ computed by the algorithm has length at most $\mathcal{O}(k'n)$, which in particular proves the claimed upper bound on the length of the output term. For the analysis of the running time of the algorithm, observe that for each state $I \in \mathbb{S}$ we consider $\mathcal{O}(k'n)$ possible candidates for $\mathbf{t}[I]$. Each of these candidates is constructed in $\mathcal{O}(kn)$ time, since we possibly need to partition $G[Z \setminus \{v\}]$ into connected components. Moreover, each of these candidates has length at most $\mathcal{O}(k'n)$, by Claim 6.3. It follows that the $\trianglelefteq$-minimum among these candidates can be selected in $\mathcal{O}((k'n)^2)$ time. Since $|\mathbb{S}| = 2^{\mathcal{O}(k^5 \log k)} \cdot n^3$, we conclude that the whole algorithm works in time $2^{\mathcal{O}(k^5 \log k)} \cdot n^5$. $\square$

Theorem 1.1 follows directly from our canonization algorithm. We remark Grohe and Marx [16], [17] view canonization as an algorithm $\mathfrak{c}(\cdot)$ that, given a graph G, outputs an isomorphic graph $\mathfrak{c}(G)$ such that $\mathfrak{c}(G_1) = \mathfrak{c}(G_2)$ whenever G_1 and G_2 are isomorphic. Theorem 1.2 gives an FPT canonization algorithm for graph of bounded treewidth also in this sense. We simply need to compute the canonical term $\mathbf{t}$ constructing G, and then output the graph $G[\mathbf{t}]$ on the vertex set $[n]$, where the vertices are numbered according to pre-order in term $\mathbf{t}$ of operations when they become forgotten.

Acknowledgements: We are grateful to Yota Otachi and Pascal Schweitzer for sharing with us their manuscript [28] and helpful comments on our work.

References

[1] S. Arnborg and A. Proskurowski. Canonical representations of partial 2- and 3-trees. *BIT*, 32(2):197–214, 1992.

[2] L. Babai and E. M. Luks. Canonical labeling of graphs. In *STOC*, pages 171–183, 1983.

[3] A. Berry, R. Pogorelcnik, and G. Simonet. An introduction to clique minimal separator decomposition. *Algorithms*, 3(2):197–215, 2010.

[4] H. L. Bodlaender. Polynomial algorithms for Graph Isomorphism and Chromatic Index on partial k-trees. *J. Algorithms*, 11(4):631–643, 1990.

[5] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[6] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998.

[7] H. L. Bodlaender. Necessary edges in k-chordalisations of graphs. *J. Comb. Optim.*, 7(3):283–290, 2003.

[8] H. L. Bodlaender, E. D. Demaine, M. R. Fellows, J. Guo, D. Hermelin, D. Lokshtanov, M. Müller, V. Raman, J. M. M. van Rooij, and F. A. Rosamond. Open problems in parameterized and exact computation — IWPEC 2008. *Technical Report UU-CS-2008-017, Department of Information and Computing Sciences, Utrecht University*, 2008.

[9] A. Bouland, A. Dawar, and E. Kopczyński. On tractable parameterizations of Graph Isomorphism. In *IPEC*, pages 218–230, 2012.

[10] B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic — A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.

[11] M. Cygan, D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. Minimum bisection is fixed parameter tractable. In *STOC*, pages 323–332, 2014.

[12] R. Diestel. *Graph Theory*. Springer, 2005.

[13] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[14] I. S. Filotti and J. N. Mayer. A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus (working paper). In *STOC*, pages 236–243, 1980.

[15] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.

[16] M. Grohe and D. Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. *CoRR*, abs/1111.1109, 2011.

[17] M. Grohe and D. Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. In *STOC*, pages 173–192, 2012.

[18] J. E. Hopcroft and R. E. Tarjan. Isomorphism of planar graphs. In *Complexity of Computer Computations*, pages 131–152, 1972.

[19] J. E. Hopcroft and R. E. Tarjan. A $v \log v$ algorithm for isomorphism of triconnected planar graphs. *J. Comput. Syst. Sci.*, 7(3):323–331, 1973.

[20] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *STOC*, pages 172–184, 1974.

[21] K. Kawarabayashi and B. Mohar. Graph and map isomorphism and all polyhedral embeddings in linear time. In *STOC*, pages 471–480, 2008.

[22] J. M. Kleinberg and É. Tardos. *Algorithm design*. Addison-Wesley, 2006.

[23] T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

[24] S. Kratsch and P. Schweitzer. Isomorphism for graphs of bounded feedback vertex set number. In *SWAT*, pages 81–92, 2010.

[25] E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Syst. Sci.*, 25(1):42–65, 1982.

[26] G. L. Miller. Isomorphism testing for graphs of bounded genus. In *STOC*, pages 225–235, 1980.

[27] Y. Otachi. Isomorphism for graphs of bounded connected-path-distance-width. In *ISAAC*, pages 455–464, 2012.

[28] Y. Otachi and P. Schweitzer. Reduction techniques for Graph Isomorphism in the context of width parameters. *CoRR*, abs/1403.7238, 2014.

[29] I. Ponomarenko. The isomorphism problem for classes of graphs closed under contraction. *Journal of Soviet Mathematics*, 55(2):1621–1643, 1991.

[30] N. Robertson and P. D. Seymour. Graph Minors XIII. The Disjoint Paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

[31] U. Schöning. Graph Isomorphism is in the low hierarchy. *J. Comput. Syst. Sci.*, 37(3):312–323, 1988.

[32] R. E. Tarjan. Decomposition by clique separators. *Discrete Mathematics*, 55(2):221–232, 1985.

[33] R. van Bevern, M. R. Fellows, S. Gaspers, and F. A. Rosamond. Myhill-nerode methods for hypergraphs. In *ISAAC*, volume 8283 of *Lecture Notes in Computer Science*, pages 372–382, 2013.

[34] H. Weinberg. A simple and efficient algorithm for determining isomorphism of planar triply connected graphs. *Circuit Theory*, 13:142–148, 1966.

[35] K. Yamazaki, H. L. Bodlaender, B. de Fluiter, and D. M. Thilikos. Isomorphism for graphs of bounded distance width. *Algorithmica*, 24(2):105–127, 1999.