Journal of Computer and System Sciences 121 (2021) 57–75

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Measuring what matters: A hybrid approach to dynamic programming with treewidth ☆

Eduard Eiben [a], Robert Ganian [b], Thekla Hamm [b,*], O-joung Kwon [c,d]

[a] Department of Computer Science, Royal Holloway, University of London, Egham, UK
[b] Department of Computer Science, Vienna University of Technology, Vienna, Austria
[c] Department of Mathematics, Incheon National University, South Korea
[d] Discrete Mathematics Group, Institute for Basic Science (IBS), Daejeon, South Korea

ARTICLE INFO

Article history:
Received 7 April 2020
Received in revised form 9 April 2021
Accepted 29 April 2021
Available online 11 May 2021

Keywords:
Parameterized complexity
Treewidth
Rank-width
Fixed-parameter algorithms

ABSTRACT

We develop a framework for applying treewidth-based dynamic programming on graphs with "hybrid structure", i.e., with parts that may not have small treewidth but instead possess other structural properties. Informally, this is achieved by defining a refinement of treewidth which only considers parts of the graph that do not belong to a pre-specified tractable graph class. Our approach allows us to not only generalize existing fixed-parameter algorithms exploiting treewidth, but also fixed-parameter algorithms which use the size of a modulator as their parameter. As the flagship application of our framework, we obtain a parameter that combines treewidth and rank-width to obtain fixed-parameter algorithms for Chromatic Number, Hamiltonian Cycle, and Max-Cut.

1. Introduction

Over the past decades, the use of structural properties of graphs to obtain efficient algorithms for NP-hard computational problems has become a prominent research direction in computer science. Perhaps the best known example of a structural property that can be exploited in this way is the tree-likeness of the inputs, formalized in terms of the decomposition-based structural parameter *treewidth* [61]. It is now well-known that a vast range of fundamental problems admit so-called *fixed-parameter* algorithms parameterized by the treewidth of the input graph – that is, can be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$ on n-vertex graphs of treewidth k (for some computable function f). We say that such problems are FPT parameterized by treewidth.

On the other hand, dense graphs are known to have high treewidth and hence require the use of different structural parameters; the classical example of such a parameter tailored to dense graphs is *clique-width* [16]. Clique-width is asymptotically equivalent to the structural parameter *rank-width* [60], which is nowadays often used instead of clique-width due to a number of advantages (rank-width is much easier to compute [42] and can be used to design more efficient fixed-parameter algorithms than clique-width [35,36]). While rank-width (or, equivalently, clique-width) dominates[1] treewidth and can be used to "lift" fixed-parameter algorithms designed for treewidth to well-structured dense graphs for a number

☆ A preliminary version of this paper appeared as an extended abstract in [23].

* Corresponding author.
E-mail addresses: Eduard.Eiben@rhul.ac.uk (E. Eiben), rganian@gmail.com (R. Ganian), thekla.hamm@tuwien.ac.at (T. Hamm), ojoungkwon@inu.ac.kr (O.-j. Kwon).

[1] We say parameter α dominates parameter β if for each graph class with bounded β, α is also bounded.

https://doi.org/10.1016/j.jcss.2021.04.005

of problems, there are also important problems which are FPT parameterized by treewidth but W[1]-hard (and hence probably not FPT) parameterized by rank-width. The most prominent examples of such problems are Chromatic Number [30], Hamiltonian Cycle [30], and Max-Cut [31].

Another generic type of structure used in algorithmic design is based on measuring the size of a *modulator* (i.e., a vertex deletion set) [12] to a certain graph class. Basic examples of parameters based on modulators include the vertex cover number (a modulator to edgeless graphs) [28,32] and the feedback vertex set number (a modulator to forests) [6,45]. For dense graphs, modulators to graphs of rank-width 1 have been studied [24,51], and it is known that for every constant c one can find a modulator of size at most k to graphs of rank-width c (if such a modulator exists) in time $f(k) \cdot n$ [50]. However, the algorithmic applications of such modulators have remained largely unexplored up to this point.

Our contribution. We develop a class of *hybrid parameters* which combines the foremost advantages of treewidth and modulators to obtain a "best-of-both-worlds" outcome. More specifically, instead of measuring the treewidth of the graph itself or the size of a modulator to a graph class $\mathcal{H}$, we consider the treewidth of a (torso of a) modulator[2] to $\mathcal{H}$. This parameter, which we simply call $\mathcal{H}$-treewidth, allows us to lift previously established tractability results for a vast number of problems from treewidth and modulators to a strictly more general setting. As our first technical contribution, we substantiate this claim with a meta-theorem that formalizes generic conditions under which a treewidth-based algorithm can be generalized to $\mathcal{H}$-treewidth; the main technical tool for the proof is an adaptation of *protrusion replacement* techniques [5, Section 4].

As the flagship application of $\mathcal{H}$-treewidth, we study the case where $\mathcal{H}$ is the class $\mathcal{R}_c$ of graphs of rank-width at most c (an arbitrary constant). $\mathcal{R}_c$-treewidth hence represents a way of lifting treewidth towards dense graphs that lies "between" treewidth and rank-width. We note that this class of parameters naturally incorporates a certain scaling trade-off: $\mathcal{R}_c$-treewidth dominates $\mathcal{R}_{c-1}$-treewidth for each constant c, but the runtime bounds for algorithms using $\mathcal{R}_c$-treewidth are worse than those for $\mathcal{R}_{c-1}$-treewidth.

Our first result for $\mathcal{R}_c$-treewidth is a fixed-parameter algorithm for computing the parameter itself. We then develop fixed-parameter algorithms for Chromatic Number, Hamiltonian Cycle and Max-Cut parameterized by $\mathcal{R}_c$-treewidth; moreover, in 2 out of these 3 cases the parameter dependencies of our algorithms are essentially tight. All these algorithms represent generalizations of:

1. classical fixed-parameter algorithms parameterized by treewidth [22],
2. polynomial-time algorithms on graphs of bounded rank-width [36], and
3. (not previously known) fixed-parameter algorithms parameterized by modulators to graph classes of bounded rank-width.

The main challenge for all of these problems lies in dealing with the fact that some parts of the graph need to be handled using rank-width based techniques, while for others we use treewidth-based dynamic programming. We separate these parts from each other using the notion of *nice $\mathcal{H}$-tree decompositions*. The algorithm then relies on enhancing the known dynamic programming approach for solving the problem on treewidth with a subroutine that not only solves the problem on the part of the graph outside of the modulator, but also serves as an interface by supplying appropriate records to the treewidth-based dynamic programming part of the algorithm. At its core, each of these subroutines boils down to solving an "extended" version of the original problem parameterized by the size of a modulator to $\mathcal{R}_c$; in particular, each subroutine immediately implies a fixed-parameter algorithm for the respective problem when parameterized by a modulator to constant rank-width. To give a specific example for such a subroutine, in the case of Chromatic Number one needs to solve the problem parameterized by a modulator to $\mathcal{R}_c$ where the modulator is furthermore precolored.

To avoid any doubt, we make it explicitly clear that the runtime of all of our algorithms utilizing $\mathcal{R}_c$-treewidth has a polynomial dependency on the input where the degree of this polynomial depends on c (as is necessitated by the W[1]-hardness of the studied problems parameterized by rank-width).

Related work. There are previous works which use a combination of treewidth with *backdoors*, a notion that is closely related to modulators, in order to solve non-graph problems such as Constraint Satisfaction [39], Boolean Satisfiability [38] and Integer Programming [37]. Interestingly, the main technical challenge in all of these papers is the problem of computing the parameter, while using the parameter to solve the problem is straightforward. In the algorithmic results presented in this contribution, the situation is completely reversed: the main technical challenge lies in developing the algorithms (and most notably the subroutines) for solving our targeted problems. Moreover, while the aforementioned three papers focus on solving a single problem, here we aim at identifying and exploiting structural properties that can be used to solve a wide variety of graph problems.

Telle and Saether [63] proposed a generalization of treewidth towards dense graphs that is based on explicitly allowing a specific operation (graph splits). The resulting parameter is not related to modulators, and we show that it is incomparable to $\mathcal{R}_c$-treewidth. A subset of the authors previously studied modulator-based parameters that used graph splits and rank-width. The resulting *well-structured modulators* could either dominate rank-width [26], or (depending on the specific constants used) be used to obtain polynomial kernels for a variety of problems [25]. Since $\mathcal{R}_c$-treewidth lies "between" treewidth and rank-width, it is clear that our class of parameters is different from these previously studied ones.

[2] Formal definitions are provided in Section 3.

We remark that modulators have also been used in a variety of other contexts. For instance, modulators to graphs of bounded treedepth can be used to obtain polynomial kernels for many problems [33,9], and there are also numerous applications of modulators to cluster graphs [21] and natural restrictions of these [34]. The parameterizations considered in this paper are also related to the notion of elimination distance, which has been studied in a variety of contexts [1,10,11, 46,55].

2. Preliminaries

For $i \in \mathbb{N}$, let $[i]$ denote the set $\{1, \ldots, i\}$. All graphs considered in this paper are simple and undirected. We refer to the standard textbook [20] for basic graph terminology.

For a graph G, let $V(G)$ and $E(G)$ denote the vertex set and the edge set of G, respectively. For $S \subseteq V(G)$, let $G[S]$ denote the subgraph of G induced by S. For $v \in V(G)$ and $S \subseteq V(G)$, let $G - v$ be the graph obtained from G by removing v, and let $G - S$ be the graph obtained by removing all vertices in S. For $v \in V(G)$, the set of neighbors of v in G is denoted by $N_G(v)$ (or $N(v)$ when G is clear from the context). For $A \subseteq V(G)$, let $N_G(A)$ denote the set of vertices in $G - A$ that have a neighbor in A. For $v \in V(G)$ and a subgraph H of $G - v$, we say v is *adjacent* to H if it has a neighbor in H. For a vertex set A of a graph G, an *A-path* is a path whose endpoints are contained in A and all the internal vertices are contained in $G - A$.

A set M of vertices in a graph G is called a *modulator* to a graph class $\mathcal{H}$ if $G - M \in \mathcal{H}$. The operation of *collapsing* a vertex set X, denoted $G \circ X$, deletes X from the graph and adds an edge between vertices $u, v \in V(G - X)$ if $uv \notin E(G)$ and there is an u-v path with all internal vertices in $G[X]$.

2.1. Parameterized complexity

A *parameterized problem* $\mathcal{P}$ is a subset of $\Sigma^* \times \mathbb{N}$ for some finite alphabet Σ. Let $L \subseteq \Sigma^*$ be a classical decision problem for a finite alphabet, and let p be a non-negative integer-valued function defined on Σ^*. Then L *parameterized by* p denotes the parameterized problem $\{(x, p(x)) \mid x \in L\}$ where $x \in \Sigma^*$. For a problem instance $(x, k) \in \Sigma^* \times \mathbb{N}$ we call x the main part and k the parameter. A parameterized problem $\mathcal{P}$ is *fixed-parameter tractable* (FPT in short) if a given instance (x, k) can be solved in time $f(k) \cdot |x|^{\mathcal{O}(1)}$ where f is an arbitrary computable function of k; we call algorithms which run in this time *fixed-parameter algorithms*. Similarly, a parameterized problem $\mathcal{P}$ is in the class XP if a given instance (x, k) can be solved in time $|x|^{f(k)}$ where f is an arbitrary computable function of k, and we call algorithms running in this time *XP algorithms*.

Parameterized complexity classes are defined with respect to *fpt-reducibility*. A parameterized problem P is *fpt-reducible* to Q if in time $f(k) \cdot |x|^{O(1)}$, one can transform an instance (x, k) of $\mathcal{P}$ into an instance (x', k') of $\mathcal{Q}$ such that $(x, k) \in \mathcal{P}$ if and only if $(x', k') \in \mathcal{Q}$, and $k' \leq g(k)$, where f and g are computable functions depending only on k. Central to parameterized complexity is the following hierarchy of complexity classes, defined by the closure of canonical problems under fpt-reductions: $\mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \cdots \subseteq \mathsf{XP}$. All inclusions are believed to be strict. In particular, $\mathsf{FPT} \neq \mathsf{W}[1]$ under the Exponential Time Hypothesis [43].

The class W[1] can be considered the analog of NP in parameterized complexity. A major goal in parameterized complexity is to distinguish between parameterized problems which are in FPT and those which are W[1]-hard, i.e., those to which every problem in W[1] is fpt-reducible.

We refer the reader to the respective books [18,22,58] for more details on parameterized complexity. Finally, we recall that a parameter α *dominates* a parameter β if for each graph class where β is bounded, α is also bounded. Two parameters are *incomparable* if neither dominates the other.

2.2. Treewidth

A *tree decomposition* of a graph G is a pair $(T, \{B_t \mid t \in V(T)\})$ where $B_t \subseteq V(G)$ for every $t \in V(T)$ and T is a tree such that:

1. for each edge $\{u, v\} \in E(G)$, there is a $t \in V(T)$ such that $\{u, v\} \subseteq B_t$, and
2. for each vertex $v \in V(G)$, $T[\{t \in V(T) \mid v \in B_t\}]$ is a non-empty (connected) tree.

The *width* of a tree decomposition is $\max_{t \in V(T)} |B_t| - 1$. The *treewidth* [52] of G is the minimum width taken over all tree decompositions of G and it is denoted by $\mathbf{tw}(G)$. We call the elements of $V(T)$ *nodes* and B_t *bags*.

Fact 1 *([4]). There exists an algorithm which, given an n-vertex graph G and an integer k, runs in time $k^{\mathcal{O}(k^3)} \cdot n$, and either outputs a tree decomposition of G of width at most k or correctly determines that $\mathbf{tw}(G) > k$.*

A tree decomposition $(T, \{B_t \mid t \in V(T)\})$ of a graph G is *nice* if T can be rooted such that the following conditions hold:

1. Every node of T has at most two children.

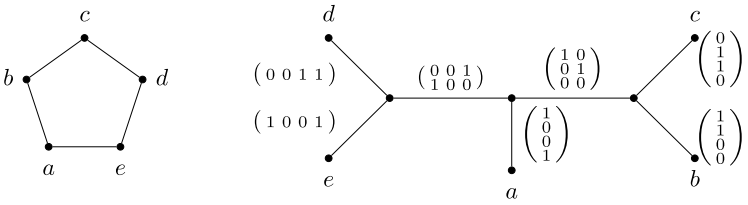

Fig. 1. A rank-decomposition of the graph cycle C_5.

2. If a node t of T has two children t_1 and t_2, then $B_t = B_{t_1} = B_{t_2}$; in this case we call t a *join node*.
3. If a node t of T has exactly one child t', then either of the following holds:
   (a) $|B_t| = |B_{t'}| + 1$, in which case we call t an *introduce node*, or
   (b) $|B_t| = |B_{t'}| - 1$ in which case we call t a *forget node*.
4. If a node t is a leaf, then $|B_t| = 1$; in this case we call t a *leaf node*.

The advantage of nice tree decompositions is that they allow the design of more transparent dynamic programming algorithms, since one only needs to deal with four specific types of nodes. It is well known (and easy to see) that given a tree decomposition of a graph $G = (V, E)$ of width at most k and with $\mathcal{O}(|V|)$ nodes, one can construct in linear time a nice tree decomposition of G with $\mathcal{O}(|V|)$ nodes and width at most k [7].

Given a node t in T, we let Y_t be the set of all vertices contained in the bags of the subtree of T rooted at t, i.e., $Y_t = B_t \cup \bigcup_{p \text{ is separated from the root by } t} B_p$.

2.3. Rank-width

For a graph G and $U, W \subseteq V(G)$, let $\boldsymbol{A}_G[U, W]$ denote the $U \times W$-submatrix of the adjacency matrix over the two-element field GF(2), i.e., the entry $a_{u,w}$, with $u \in U$ and $w \in W$, of $\boldsymbol{A}_G[U, W]$ is 1 if and only if $\{u, w\}$ is an edge of G. The *cut-rank* function ρ_G of a graph G is defined as follows: For a bipartition (U, W) of the vertex set $V(G)$, $\rho_G(U) = \rho_G(W)$ equals the rank of $\boldsymbol{A}_G[U, W]$.

A *rank-decomposition* of a graph G is a pair (T, μ) where T is a tree of maximum degree 3 and $\mu : V(G) \to \{t \mid t \text{ is a leaf of } T\}$ is a bijective function (see Fig. 1). For an edge e of T, the connected components of $T - e$ induce a bipartition (X, Y) of the set of leaves of T. Then the *width* of e in rank-decomposition (T, μ) is $\rho_G(\mu^{-1}(X))$. The *width* of (T, μ) is the maximum width over all edges of T. The *rank-width* of G, $\mathbf{rw}(G)$ in short, is the minimum width over all rank-decompositions of G. We denote by $\mathcal{R}_i$ the class of all graphs of rank-width at most i. A *rooted* rank-decomposition is obtained from a rank-decomposition (T, μ) by subdividing an arbitrary edge of T, and considering the newly created vertex as *root*. For any node $t \in V(T)$ we use G_t to denote the subgraph of G, induced by all vertices associated to leaves of T for which t lies on the unique path from the respective leaf to the root in T.

Unlike clique-width, rank-width can be computed exactly using a fixed-parameter algorithm (which also outputs a corresponding rank-decomposition).

Fact 2 *([42, Theorem 7.3]). There exists an algorithm which, given an n-vertex graph G and an integer k, returns in time $f(k) \cdot n^3$, and either outputs a rank-decomposition of width at most k or correctly determines that $\mathbf{rw}(G) > k$.*

2.4. Monadic Second-Order Logic

Counting Monadic Second-Order Logic (CMSO_1) is a basic tool to express properties of vertex sets in graphs. The syntax of CMSO_1 includes logical connectives $\wedge, \vee, \neg, \Leftrightarrow, \Rightarrow$, variables for vertices and vertex sets, quantifiers $\exists, \forall$ over these variables, and the relations $a \in A$ where a is a vertex variable and A is a vertex set variable; $\text{adj}(a, b)$, where a and b are vertex variables and the interpretation is that a and b are adjacent; equality of variables representing vertices and sets of vertices; $\text{Parity}(A)$, where A is a vertex set variable and the interpretation is that $|A|$ is even.

The CMSO_1 Optimization problem is defined as follows:

CMSO_1-OPT	
Instance:	A graph G, a CMSO_1 formula $\phi(A)$ with a free set variable A, and $\text{opt} \in \{\min, \max\}$.
Task:	Find an interpretation of the set A in G such that G models $\phi(A)$ and A is of minimum/maximum (depending on opt) cardinality.

From the fixed-parameter tractability of computing rank-width [42], the equivalence of rank-width and clique-width [60] and Courcelle's Theorem for graphs of bounded clique-width [15] (see also later work that establishes the result directly for rank-width [35]) it follows that:

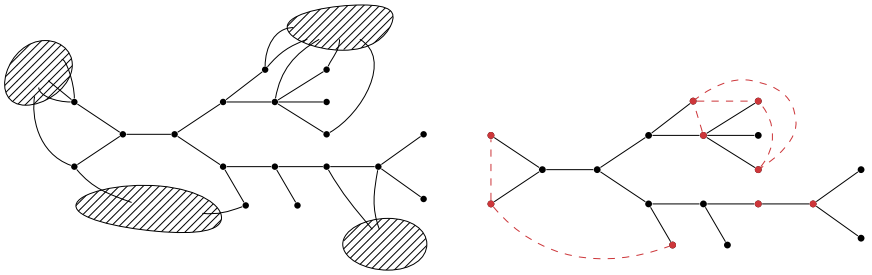

Fig. 2. *Left:* A graph G with a tree as a modulator to $\mathcal{H}$ (the part in $\mathcal{H}$ is depicted hatched). *Right:* The corresponding $\mathcal{H}$-torso.

Fact 3 *([35]).* CMSO_1-OPT *is FPT parameterized by* $\mathbf{rw}(G) + |\phi|$, *where* G *is the input graph and* ϕ *is the* CMSO_1 *formula.*

We refer the reader to the books [14,29] for an in-depth overview of Monadic Second Order logic.

3. $\mathcal{H}$-Treewidth

The aim of $\mathcal{H}$-treewidth is to capture the treewidth of a modulator to the graph class $\mathcal{H}$. However, one cannot expect to obtain a parameter with reasonable algorithmic applications by simply measuring the treewidth of the graph *induced* by a modulator to $\mathcal{H}$ – instead, one needs to measure the treewidth of a so-called *torso*, which adds edges to track how the vertices in the modulator interact through $\mathcal{H}$. To substantiate this, we observe that Hamiltonian Cycle would become NP-hard even on graphs with a modulator that (1) induces an edgeless graph, and (2) is a modulator to an edgeless graph, and where (3) each connected component outside the modulator has boundedly many neighbors in the modulator.

Observation 4. Hamiltonian Cycle *remains* NP*-hard if we allow the input to consist, not only of a graph* G*, but also a modulator* $X \subseteq V(G)$ *to an edgeless graph and for each connected component* C *of* $G[X]$, $|N(C)| \leq 3$.

Proof. Hamiltonian Cycle is known to be NP-hard on cubic bipartite graphs [2]. Setting X to be one side of the bipartition of any graph in this graph class is easily seen to satisfy the given conditions. □

The notion of a *torso* has previously been algorithmically exploited in other settings [37,39,57], and its adaptation is our first step towards the definition of $\mathcal{H}$-treewidth (see also Fig. 2).

Definition 5 *($\mathcal{H}$-Torso).* Let G be a graph and $X \subseteq V(G)$. For a graph class $\mathcal{H}$, $G \circ X$ is an *$\mathcal{H}$-torso* of G if each connected component C of $G[X]$ satisfies $C \in \mathcal{H}$.

Definition 6 *($\mathcal{H}$-Treewidth).* The *$\mathcal{H}$-treewidth* of a graph G is the minimum treewidth of an $\mathcal{H}$-torso of G. We denote the $\mathcal{H}$-treewidth of G by $\mathbf{tw}_{\mathcal{H}}(G)$.

Typically, we will want to consider a graph class $\mathcal{H}$ for which certain problems are polynomial-time tractable. Hence, we will assume w.l.o.g. that $(\emptyset, \emptyset) \in \mathcal{H}$. From the definition it is obvious that then:

Observation 7. *For any graph* G, $\mathbf{tw}_{\mathcal{H}}(G) \leq \mathbf{tw}(G)$.

3.1. Nice $\mathcal{H}$-tree decompositions

Just like for tree decompositions, we can also define a canonical form of decompositions which has properties that are convenient when formulating dynamic programs using $\mathcal{H}$-treewidth. Intuitively, a *nice $\mathcal{H}$-tree decomposition* behaves like a nice tree decomposition on the torso graph (see points 1-3), with the exception that the neighborhoods of the collapsed parts must occur as special *boundary* leaves (see points 4-5).

Definition 8 *(Nice $\mathcal{H}$-tree decomposition).* A *nice $\mathcal{H}$-tree decomposition* of a graph G is a triple $(X, T, \{B_t \mid t \in V(T)\})$ where $X \subseteq V(G)$ such that $G \circ X$ is an $\mathcal{H}$-torso, $(T, \{B_t \mid t \in V(T)\})$ is a rooted tree decomposition of $G \circ X$, and:

1. Every node in T has at most two children.
2. If a node t of T has two children t_1 and t_2, then $B_t = B_{t_1} = B_{t_2}$; in this case we call t a *join node*.
3. If a node t of T has exactly one child t', then either of the following holds:
   (a) $|B_t| = |B_{t'}| + 1$, in which case we call t an *introduce node*, or

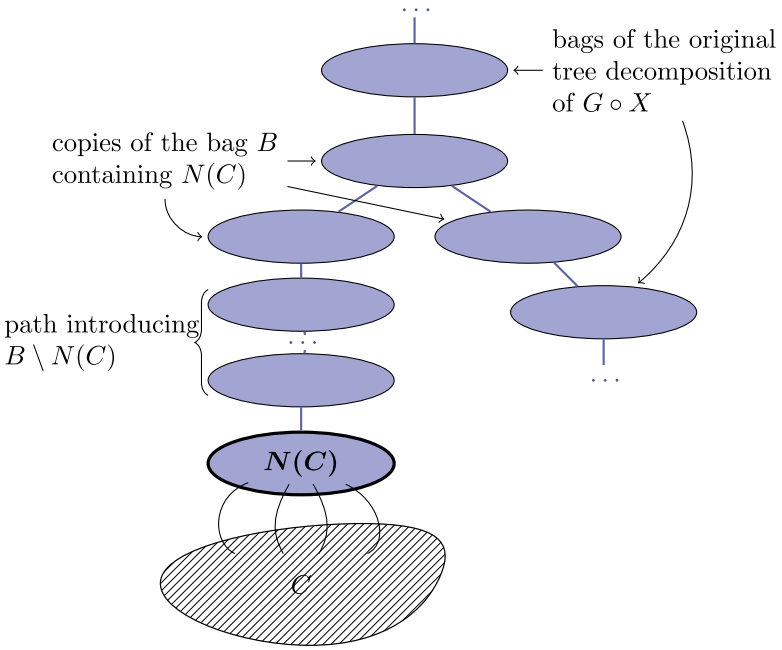

Fig. 3. Construction in the proof of Lemma 9. Part of a the tree of a nice $\mathcal{H}$-tree-decomposition (blue) including a boundary leaf (bold) and a connected component C (hatched) of X. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

(b) $|B_t| = |B_{t'}| - 1$ in which case we call t a *forget node*.

4. If a node t is a leaf, then one of the following holds:
   (a) $|B_t| = 1$; in which case we call t a *simple leaf node*, or
   (b) $B_t = N(C)$ for some connected component C of $G[X]$; in which case we call t a *boundary leaf node*.
5. For each connected component C of $G[X]$ there is a unique leaf t with $B_t = N(C)$.

In line with standard terminology for treewidth, we call the sets B_t *bags*. The *width* of a nice $\mathcal{H}$-tree decomposition is simply the width of $(T, \{B_t \mid t \in V(T)\})$. Below we show that computing a nice $\mathcal{H}$-tree decomposition of bounded width can be reduced to finding an appropriate $\mathcal{H}$-torso.

Lemma 9. *Given an n-vertex graph G and an $\mathcal{H}$-torso U of G with $\mathbf{tw}(U) \leq k$, we can find a nice $\mathcal{H}$-tree decomposition of G with width at most k in time $k^{\mathcal{O}(k^3)} \cdot n$.*

Proof. The following construction is illustrated in Fig. 3.

Compute a nice tree decomposition $(T, \{B_t \mid t \in V(T)\})$ of U with width at most k in time at most $k^{\mathcal{O}(k^3)} \cdot n$ using Fact 1. Let $X := V(G) \setminus V(U)$. By the definition of $\circ$, $U = G \circ X$ and the neighbors of each connected component C of $G[X]$ form a clique in U. Thus, for each C there is some node, say t_C, of T such that $N(C) \subseteq B_t$.

Obtain T' from T by successively performing the following, for each connected component C of $G[X]$: Replace t_C by a new node t' with two children t_1 and t_2, and set $B_{t'} = B_{t_1} = B_{t_2} = B_{t_C}$. Below t_1, attach the descendants of t_C (i.e., a copy of the subtree rooted at t_C taken from the tree that was obtained after the adaptations for the previously considered connected components of $G[X]$), and below t_2 attach a path $s_1, \ldots, s_{|B_t \setminus N(C)|}$. Set $B_{s_1} = B_{t_2} \setminus \{v_1\}$ and $B_{s_{i+1}} = B_{s_i} \setminus \{v_{i+1}\}$ where $\{v_1, \ldots, v_{|B_t \setminus N(C)|}\}$ is some enumeration of $B_t \setminus N(C)$.

It is easy to see that these manipulations result in a nice $\mathcal{H}$-tree decomposition without introducing bags larger than the ones already occurring in $(T, \{B_t \mid t \in V(T)\})$. □

Hence, we can state the problem of computing a decomposition as follows:

$\mathcal{H}$-Treewidth		*Parameter: k*
Instance:	A graph G and an integer k.	
Task:	Find an $\mathcal{H}$-torso U of G such that $\mathbf{tw}(U) \leq k$, or correctly determine that no such $\mathcal{H}$-torso exists.	

3.2. An algorithmic meta-theorem

Before proceeding to the flagship application of $\mathcal{H}$-treewidth where we chose $\mathcal{H}$ to be the class of graphs of bounded rank-width, here we give a generic set of conditions that allow fixed-parameter algorithms for problems parameterized by $\mathcal{H}$-treewidth. Specifically, we consider graph problems that are *finite-state* [13] or have *finite integer index* [5,8,33]. Informally speaking, such problems only transfer a limited amount of information across a small separator in the input graph and hence can be solved "independently" on both sides of such a separator. Since these notions are only used in this section, we provide concise definitions below.

First of all, we will need the notion of boundaried graphs and gluing. A graph $\bar{G}$ is called *t-boundaried* if it contains t distinguished vertices identified as $b_1^G, \ldots, b_t^G$. The *gluing operation* $\oplus$ takes two t-boundaried graphs $\bar{G}$ and $\bar{H}$, creates their disjoint union, and then alters this disjoint union by identifying the boundaries of the two graphs (i.e. by setting $b_i^G = b_i^H$ for each $i \in [t]$).

Consider a decision problem $\mathcal{P}$ whose input is a graph. We say that two t-boundaried graphs $\bar{C}$ and $\bar{D}$ are *equivalent* with respect to $\mathcal{P}$, denoted by $\bar{C} \sim_{\mathcal{P},t} \bar{D}$, if for each t-boundaried graph $\bar{H}$ it holds that

$$\bar{C} \oplus \bar{H} \in \mathcal{P} \text{ if and only if } \bar{D} \oplus \bar{H} \in \mathcal{P}.$$

We say that $\mathcal{P}$ is *finite-state* (or *FS*, in brief) if, for each $t \in \mathbb{N}$, $\sim_{\mathcal{P},t}$ has a finite number of equivalence classes.

Next, consider a decision problem $\mathcal{Q}$ whose input is a graph and an integer. In this case we say that two t-boundaried graphs $\bar{C}$ and $\bar{D}$ are *equivalent* with respect to $\mathcal{Q}$ (denoted by $\bar{C} \sim_{\mathcal{Q},t} \bar{D}$) if there exists an *offset* $\delta(\bar{C}, \bar{D}) \in \mathbb{Z}$ such that for each t-boundaried graph $\bar{H}$ and each $q \in \mathbb{Z}$:

$$(\bar{C} \oplus \bar{H}, q) \in \mathcal{Q} \text{ if and only if } (\bar{D} \oplus \bar{H}, q + \delta(\bar{C}, \bar{D})) \in \mathcal{Q}.$$

We say that $\mathcal{Q}$ has *finite integer index* (or is *FII*, in brief) if, for each $t \in \mathbb{N}$, $\sim_{\mathcal{Q},t}$ has a finite number of equivalence classes.

We note that a great number of natural graph problems are known to be FS or FII. For instance, all problems definable in Monadic Second Order logic are FS [5, Lemma 3.2], while examples of FII problems include Vertex Cover, Independent Set, Feedback Vertex Set, Dominating Set, Connected Dominating Set, and Edge Dominating Set [33].

We say that a FS problem $\mathcal{P}$ is *efficiently extendable* on a graph class $\mathcal{H}$ if there is a fixed-parameter algorithm (parameterized by t) that takes as input two t-boundaried graphs $\bar{G}$ and $\bar{H}$ such that their boundaries are modulators to $\mathcal{H}$ and decides whether $\bar{G} \sim_{\mathcal{P},t} \bar{H}$. Similarly we say that a or FII problem $\mathcal{P}$ is *efficiently extendable* on a graph class $\mathcal{H}$ if there is a fixed-parameter algorithm (parameterized by t) that takes as input two t-boundaried graphs $\bar{G}$ and $\bar{H}$ such that their boundaries are modulators to $\mathcal{H}$ and decides whether $\bar{G} \sim_{\mathcal{P},t} \bar{H}$ *as well as* computing a corresponding offset $\delta(\bar{G}, \bar{H})$.

For our meta-theorem, we will make use of the fact that every FII and FS problem is known to be FPT when parameterized by treewidth. This can be seen in, e.g., the work of Bodlaender and de Fluiter [8], but we also provide a short direct proof that uses the protrusion replacement technique introduced by Bodlaender et al. [5].

Fact 10. *Every FS and every FII graph problem is* FPT *when parameterized by the treewidth of the input graph.*

Proof. Let G be the input graph of an FS or FII problem $\mathcal{P}$. We begin by computing a width-optimal nice tree-decomposition $(T, \{B_t \mid t \in V(T)\})$ of G. Let c be the maximum size of a representative of an equivalence class of $\sim_{\mathcal{P},t}$ [5, Lemma 5.18], and observe that c only depends on $\mathcal{P}$ and the treewidth of G. If G contains at most c-many vertices, we can solve the problem by brute force. Otherwise, there must exist a node $t \in V(T)$ such that $c < |Y_t| \leq 2c$. We now apply the protrusion replacement technique from the proof of Lemma 5.18 in Bodlaender et al. [5] to obtain an equivalent instance of $\mathcal{P}$ where Y_t has been replaced by a new subgraph of order at most c; we remark that while the lemma is only stated for FII, the proof is oblivious to whether the problem is FII or FS. We then proceed along the nodes of the original tree-decomposition tree T until we obtain an equivalent instance whose size is bounded by c, at which point we can invoke the first case in the proof. □

Theorem 11. *Let $\mathcal{P}$ be a FS or FII graph problem and $\mathcal{H}$ be a graph class. If*

1. *$\mathcal{P}$ is efficiently extendable on $\mathcal{H}$, and*
2. *$\mathcal{H}$-*treewidth *is* FPT*,*

then $\mathcal{P}$ is FPT *parameterized by $\mathcal{H}$-treewidth.*

Proof. $\mathcal{P}$ can be solved using the following algorithm. First of all, we use Point 2. to compute an $\mathcal{H}$-torso $G \circ X$ of treewidth k, where k is the $\mathcal{H}$-treewidth of the input graph G.

Now consider for each connected component C of $G[X]$ the boundaried graph $\bar{H} = \overline{G[C \cup N(C)]}$. Since $C \in \mathcal{H}$, and $\mathbf{tw}(G \circ X) \leq k$ and the fact that $N(C)$ forms a clique in $G \circ X$ imply $|N(C)| \leq k$, Point 1. can be used to compare other at most k-boundaried graphs to $\bar{H}$ with respect to $\sim_{\mathcal{P},k}$ in fixed parameter time. We use this to compute a minimum-size

representative of the equivalence class of $\bar{H}$ with respect to $\sim_{\mathcal{P},k}$; note that the procedure used to do so in the general setting of this meta-theorem does not allow us to give any explicit upper bound on the parameter dependency. To this end we use a brute-force enumeration argument that has previously been applied in the kernelization setting [40]: we enumerate all boundaried graphs (in any order with a non-decreasing number of vertices), and for each graph we check whether it is equivalent to $\bar{H}$ with respect to $\sim_{\mathcal{P},k}$. Observe that since the size of a minimum-cardinality representative of any equivalence class of $\sim_{\mathcal{P},k}$ depends only on $\mathcal{P}$ and k, the number of graphs that need to be checked in this way as well as the size of each such graph depends only on $\mathcal{P}$ and k, and hence there exists some function of $\mathcal{P}$ and k which upper-bounds the running time of this brute-force procedure. Let $\bar{G}_\alpha$ be the computed minimum-cardinality representative of the equivalence class of $\bar{H}$. In case $\mathcal{P}$ is FII, Point 1. also allows us to additionally determine the offset $\delta(\bar{H}, \bar{G}_\alpha)$.

In the final step, for each C the algorithm glues $\overline{G \circ X}$ (with boundary $N(C)$) with the computed representative $\bar{G}_\alpha$. Let G' be the graph obtained after processing each connected component C in the above manner. Note that since $\mathbf{tw}(G \circ X) \leq k$ and each $\bar{G}_\alpha$ has size bounded by a function of k, the graph G' has treewidth bounded by a function of k. Indeed, one can extend a tree decomposition of $G \circ X$ by adding each $\bar{G}_\alpha$ into a new bag (a leaf in the decomposition) adjacent to the leaf of the nice $\mathcal{H}$-tree decomposition we obtain containing $N(C)$. At this stage, the algorithm invokes Fact 10 and outputs YES iff $G' \in \mathcal{P}$. In the case of FII problems where the input was (G, ℓ), the algorithm instead outputs YES iff $(G', \ell') \in \mathcal{P}$, where ℓ' is obtained from ℓ by subtracting all the individual offsets $\delta(G[C \cup N(C)], G_\alpha)$. Correctness follows by the fact that $G_\alpha \sim_{\mathcal{P},k} G[C \cup N(C)]$. □

4. $\mathcal{R}_c$-Treewidth

This section focuses on the properties of $\mathcal{R}_c$-treewidth, a hierarchy of graph parameters that represent our flagship application of the generic notion of $\mathcal{H}$-treewidth.

4.1. Comparison to known parameters

It follows from Observation 7 that $\mathcal{R}_c$-treewidth dominates treewidth (for every $c \in \mathbb{N}$).

Similarly, it is obvious that $\mathcal{R}_c$-treewidth dominates the size of a modulator to $\mathcal{R}_c$ (also for every $c \in \mathbb{N}$). The following lemma shows that, for every fixed c, $\mathcal{R}_c$-treewidth is dominated by rank-width.

Lemma 12. *Let $c \in \mathbb{N}$. If $\mathbf{tw}_{\mathcal{R}_c}(G) = k$ then $\mathbf{rw}(G) \leq c + k + 1$.*

Proof. Let the $\mathcal{R}_c$-treewidth of G be witnessed by some nice $\mathcal{R}_c$-tree decomposition $(X, T, \{B_t \mid t \in V(T)\})$ of width k.

We can obtain a rank-decomposition (T', μ) of G from $(X, T, \{B_t \mid t \in V(T))\}$ as follows:

For vertices v of $G \circ X$ such that there is no leaf node $t \in V(T)$ with $B_t = \{v\}$, let $t \in V(T)$ be a forget node with child t' such that $B_t \cup \{v\} = B_{t'}$. Turn t' into a join node by introducing t_1, t_2 with $B_{t_1} = B_{t_2} = B_{t'}$ as children of t', attaching the former child of t' to t_1 and a new leaf node t_v with $B_{t_v} = \{v\}$ below t_2. Note that this preserves the fact that for any $v \in V(G) \setminus X$, $T[\{u \in V(T) \mid v \in B_u\}]$ is a tree. Now we can choose for each $v \in V(G \circ X)$ some $\mu(v) \in V(T)$ such that $B_t = \{v\}$. This defines an injection from $V(G \circ X)$ to the leaves of T. However, not every leaf of T is mapped to by μ. On one hand there are the boundary leaf nodes, below which we will attach subtrees to obtain a rank-decomposition of G. On the other hand there may be $v \in V(G \circ X)$ for which the choice of $\mu(v)$ was not unique, i.e. there is $t \neq \mu(v)$ with $B_t = \{v\}$. For all such v and t we delete all nodes on the root-t-path in T that do not lie on a path from the root to a vertex in $\{\mu(w) \mid w \in V(G \circ X)\} \cup \{t' \mid t'$ boundary leaf node$\}$. This turns μ into an injection from $V(G \circ X)$ to the leaves of T, that is surjective on the non-boundary leaf nodes.

Next, we extend (T, μ) to a rank-decomposition of G by proceeding in the following way for each connected component C of $G[X]$:

Let $t_C \in V(T)$ be the boundary leaf node with $B_t = N(C)$. Since $\mathbf{rw}(C) \leq c$, we find a rank-decomposition (T_C, μ_C) of C with width at most c. Attach T_C below t_C.

Let T' be the tree obtained by performing these modifications for all connected components C. Consider the rank-decomposition of G given by

$$\left(T', v \mapsto \begin{cases} \mu(v) & \text{if } v \in V(G \circ X) \\ \mu_C(v) & \text{if } v \in C \text{ for some } C \text{ as above} \end{cases}\right).$$

We show that its width is at most $c + k + 1$. Any edge e of T' is of one of the following types:

- e corresponds to an edge already contained in T: Then e induces a bipartition (X_G, Y_G) of $V(G)$. Fix $t \in V(T)$ to be the vertex in which e starts.
  Let $x \in X_G$ and $y \in Y_G$ be such that $xy \in E(G)$. Obviously neighbors in X lie within the same connected component of $G[X]$. By construction a rank-decomposition of any such connected component of $G[X]$ is attached completely within one of the two subtrees of T' separated by e. Thus e does not separate neighbors in X. So, we consider $x \in V(G \circ X)$. If

$y \in V(G \circ X)$ then x and y occur together in some bag of the original tree of the tree decomposition, and as remarked earlier this is still the case modified tree. This implies that at least one of $\{x, y\}$ must be present in B_t. If $y \notin V(G \circ X)$, by the construction of T', y corresponds to a leaf $t_y \in V(T')$ in a subtree attached to T rooted at $t_C \in V(T)$ with $B_{t_C} = N(C) \ni x$ and t_C and t_y are not separated by the removal of e. Also, x corresponds to a leaf $t_x \in V(T)$ which is in the subtree of $T' - e$ not containing t_y, i.e. the subtree not containing t_C. This means any subtree containing t_C and t_x also contains t, and since $(T, \{B_t \mid t \in V(T)\})$ is a tree decomposition and $x \in B_{t_x} \cap B_{t_C}$ this means $x \in B_t$.
In both cases at least one of $\{x, y\}$ is in B_t. Since this argument applies for every edge crossing the bipartition (X_G, Y_G), it follows that $\boldsymbol{A}_G[X_G, Y_G]$ may only contain "1" entries in rows and columns that correspond to the vertices in B_t. Since $|B_t| \leq k+1$, it holds that $\boldsymbol{A}_G[X_G, Y_G]$ can be converted into a zero matrix by deleting at most $k+1$ rows plus columns, which is a sufficient condition for $\boldsymbol{A}_G[X_G, Y_G]$ having rank at most $k+1$.

- e corresponds to an edge in a rank-decomposition of some connected component C of $G[X]$: Then e induces a bipartition (X_G, Y_G) of $V(G)$ and a bipartition (X_C, Y_C), where $X_C = X_G \cap C$ and $Y_C = Y_G \cap C$, of C. Since vertices of C are only connected to vertices in $N(C)$ outside of C and $N(C) \subseteq B_t$ for some $t \in V(T)$, we have $\rho_G(X_G) \leq \rho_C(X_C) + |N(C)| \leq c + k + 1$.
- e corresponds to an edge connecting the rank-decomposition of some connected component C of $G[X]$ to T: Then the bipartition induced is $(C, V(G) \setminus C)$ and as $N(C) \subseteq B_t$ for some $t \in V(T)$ $\rho_G(C) \leq |N(C)| \leq k+1$. □

Next, we compare $\mathcal{R}_c$-treewidth to Telle and Saether's notion of *sm*-width [63], which is another parameter that lies between treewidth and rank-width. The rough idea behind the definition of this parameter is to generalize treewidth towards denser graph classes by allowing the use of *graph splits* in an explicit way without increasing the parameter value. For completeness, we provide a formal definition for sm-width below using the general notions of *branch-width* and graph splits.

Definition 13 *(Branch-width [62]).* A *branch decomposition* of the vertex set of a graph G is a binary tree T with $V(G)$ as leaf set. Each edge e in T partitions the vertices of $V(G)$ into two sets (say X_e and $V(G) \setminus X_e$), corresponding to the leaves present in the two connected components of $T - e$.

A *cut function* is a function $c : \mathcal{P}(V(G)) \to \mathbb{N}$ such that $\forall A \subseteq V(G)$ $c(A) = c(V(G) \setminus A)$. The *width* of a branch decomposition with respect to the cut function c is given by $\max_{e \in E(T)} c(X_e)$. The *branch width* of a graph with respect to a cut function is the minimum of the branch width with respect to the cut function over all branch decompositions of the vertex set of this graph.

Definition 14 *(Split).* $A \subseteq V(G)$ is a *split* of a graph G if

1. $|A| \geq 2$, $|V(G) \setminus A| \geq 2$ and
2. $\forall v, w \in A$ $N(v) \cap (V(G) \setminus A) \neq \emptyset \wedge N(w) \cap (V(G) \setminus A) \neq \emptyset$
$\Rightarrow N(v) \cap (V(G) \setminus A) = N(w) \cap (V(G) \setminus A)$.

Definition 15 *(Split-matching-width [63]).* The *split-matching-width* (or sm-width) of a graph G, denoted by $\mathrm{smw}(G)$, is the branch width of G with respect to the cut function given by

$$c(A) = \begin{cases} 1 & \text{if } A \text{ is a split} \\ \max\{|M| \mid M \text{is a matching in } G[A, V(G) \setminus A]\} & \text{otherwise} \end{cases}.$$

For clarity, we remark that omitting the first case for $c(A)$ in the definition of sm-width results in a parameter called *maximum-matching width*, or *mm-width* for short, which is asymptomatically equivalent to treewidth [49].

Lemma 16. *$\mathcal{R}_c$-treewidth and* smw *are incomparable. More formally, the following hold:*

1. *For all $c \in \mathbb{N}$ there is a sequence of graphs $(G_i)_{i \in \mathbb{N}}$ such that for all $i \in \mathbb{N}$ $\mathbf{tw}_{\mathcal{R}_c}(G_i) \geq i$, but $\mathrm{smw}(G_i) \leq c + 3$.*
2. *For all $c \geq 3$ there is a sequence of graphs $(G_i)_{i \in \mathbb{N}}$ such that for all $i \in \mathbb{N}$ $\mathrm{smw}(G_i) \geq i$, but $\mathbf{tw}_{\mathcal{R}_c}(G_i) = 0$.*

Proof. For 1. consider the following construction: For arbitrary c there is a graph H_c such that $\mathbf{tw}(H_c) = c+2$ and $\mathbf{rw}(H_c) = c+1$; notably, this can be achieved by setting H_c to be the $(c+2) \times (c+2)$ square grid [48]. Now let G_i be given by taking $2 \cdot (i+1)$ copies, $H_c^{(1)}, \ldots, H_c^{(2 \cdot (i+1))}$ of H_c and completely connecting all $H_c^{(j)}$ with j even to all $H_c^{(j)}$ with j odd.
The split-matching-width of G_i is at most $c+3$ since each copy of H_c iteratively defines a split and thus can be treated independently and it is known that the maximum matching width of a graph is at most its treewidth $+1$ [64].
On the other hand, we claim that $\mathbf{tw}_{\mathcal{R}_c}(G_i) \geq i$:

- If G_i has no non-trivial $\mathcal{R}_c$-torso, then since G_i is at least $((i+1) \cdot |V(H_c)|)$-degenerate because of the complete connections between odd and even copies of H_c, we know that

$$\mathbf{tw}_{\mathcal{R}_c}(G_i) = \mathbf{tw}(G_i) \geq (i+1) \cdot |V(H_c)| \geq i.$$

- If the $\mathcal{R}_c$-treewidth of G_i is given by $\mathbf{tw}(G_i \circ X)$ and it holds that X only intersects odd (symmetrically only even) copies of H_c then, by definition of $G_i \circ X$, the even (respectively odd) copies of H_c form a clique in $G_i \circ X$ and thus

$$\mathbf{tw}_{\mathcal{R}_c}(G_i) = \mathbf{tw}(G_i \circ X) \geq (i+1) \cdot |V(H_c)| - 1 \geq i.$$

- If the $\mathcal{R}_c$-treewidth of G_i is given by $\mathbf{tw}(G_i \circ X)$ and it holds that X intersects even and odd copies of H_c and $|G - X| \geq i+1$, then $G_i \circ X$ is a clique of size at least $i+1$ and thus

$$\mathbf{tw}_{\mathcal{R}_c}(G_i) = \mathbf{tw}(G_i \circ X) = |V(G_i) \setminus X| - 1 = i.$$

- Otherwise the $\mathcal{R}_c$-treewidth of G_i is given by $\mathbf{tw}(G_i \circ X)$ and it holds that $|G - X| \leq i$. Then by the pigeon hole principle we find a $1 \leq j \leq i$ such that

$$\left|V(H_c^{(j)} \cap X)\right| \geq \frac{|V(G_i)| - i}{2 \cdot (i+1)} = \frac{2 \cdot (i+1) \cdot |V(H_c)| - i}{2 \cdot (i+1)} > |V(H_c)| - 1.$$

  Since $\mathbf{rw}\left(H_c^{(j)}\right) = c+1$, this is a contradiction to $\mathbf{rw}(G[X]) \leq c$.

For 2. let $c \geq 3$ and consider the following construction: Fix a cycle C_i on $3 \cdot i + 5$ vertices. Now let G_i to be the complement graph of C_i. Since C_i has rank-width 2 and the complement graph's rank-width is at most one larger [41], we know $\mathbf{rw}(G_i) \leq 3$ and we can use $V(G_i)$ to build an $\mathcal{R}_c$-torso of treewidth 0. On the other hand, there is no split of G_i and the maximum matching width of a graph G is known to be at least $\frac{1}{3} \cdot (\mathbf{tw}(G)+1)$ [64]. So $\mathrm{smw}(G_i) \geq \frac{1}{3} \cdot (\mathbf{tw}(G_i)+1) \geq i$. □

4.2. Computing $\mathcal{R}_c$-treewidth

Our aim here is to determine the complexity of computing our parameters, i.e., finding a torso of small treewidth. Obtaining such a torso is a base prerequisite for our algorithms. We formalize the problem below.

$\mathcal{R}_c$-Treewidth		*Parameter: k*
Instance:	A graph G and an integer k.	
Task:	Find a $\mathcal{R}_c$-torso U of G such that $\mathbf{tw}(U) \leq k$, or correctly determine that no such $\mathcal{R}_c$-torso exists.	

We show that, for every c, $\mathcal{R}_c$-Treewidth is FPT. The main tool we use to this end is Courcelle's Theorem (Fact 3).

Lemma 17. $\mathcal{R}_c$-Treewidth *is* FPT.

Proof. We begin by noting that there exists an CMSO_1 formula $\phi_c(G)$ that is true iff a graph G has rank-width at most c (and, equivalently, if each connected component of G has rank-width at most c). This was observed already by Kanté, Kim, Kwon and Paul [50] and follows from the fact that the property of having rank-width at most c can be characterized by a finite set of *vertex minors* [59], a property which can be expressed in CMSO_1 [17]. For clarity, we explicitly remark that $|\phi_c(G)|$ depends only on c.

Next, we also note that there exists an CMSO_1 formula ψ'_k (of size depending only on k) that is true iff a graph G has treewidth at most k [54]. Consider now the formula $\psi_k(Z)$ for some fixed $Z \subseteq V(G)$ which operates precisely like ψ'_k, but only on the graph $G \circ Z$. In particular, $\psi_k(Z)$ can be obtained from ψ'_k by:

- For each vertex variable a occurring in ψ'_k, adding a condition forcing $v \notin Z$;
- For each set variable A occurring in ψ'_k, adding a condition forcing $A \cap Z = \emptyset$;
- Replacing each occurrence of $\mathrm{adj}(a, b)$ with an expression that is true if $\mathrm{adj}(a, b)$ or if there exists an a-b path with all internal vertices in $G[Z]$.

Let $\phi_{c,k}(Z) = \phi_c(G[Z]) \wedge \psi_k(Z)$. Consider now a vertex set Z (or, more precisely, an interpretation of Z) which satisfies $\phi_{c,k}$. Since $G[Z]$ satisfies ϕ_c, it holds that each connected component of $G[Z]$ has rank-width at most c, and hence $G \circ Z$ is an $\mathcal{R}_c$-torso of G. Moreover, since $\psi_k(Z)$ is true, $G \circ Z$ has treewidth at most k.

To conclude the proof, note that Fact 3 alongside Lemma 12 provide an FPT algorithm (with parameter k) to find an interpretation of Z satisfying $\phi_{c,k}(Z)$, or correctly identify that no such interpretation exists. □

5. Algorithms exploiting modulators to $\mathcal{R}_c$

For a problem $\mathcal{P}$ to be FPT parameterized by $\mathcal{R}_c$-treewidth, $\mathcal{P}$ must necessarily be FPT parameterized by treewidth and also FPT parameterized by the size of a modulator to $\mathcal{R}_c$. However, it is important to note that neither condition is not sufficient; indeed, one can easily construct artificial problems that are defined in a way which make them trivial in both of the cases outlined above, but become intractable (or even undecidable) once parameterized by $\mathcal{R}_c$-treewidth. A very simple example for this is achieved by setting $\mathcal{P}(c,k)$ to be YES whenever the treewidth or the size of a modulator to $\mathcal{R}_c$ is at most k, and otherwise ask to solve some undecidable problem. That is, after all, why we need the notion of *efficient extendability* in Theorem 11.

Hence, in order to develop fixed-parameter algorithms for Chromatic Number, Hamiltonian Cycle and Max-Cut parameterized by $\mathcal{R}_c$-treewidth, we first need to show that they are not only FPT parameterized by the size of a modulator to $\mathcal{R}_c$, but they are also efficiently extendable. Such a result would be sufficient to employ Theorem 11 together with Lemma 17 in order to establish the desired fixed-parameter tractability results. That is also our general aim in this section, with one caveat: in order to give explicit and tight upper bounds on the parameter dependency of our algorithms, we provide algorithms that solve generalizations of Chromatic Number, Hamiltonian Cycle and Max-Cut parameterized by the size of a modulator to $\mathcal{R}_c$. It will become apparent in the next section that these generalizations precisely correspond to the records required by the treewidth-based dynamic program that will be used in the torso. In other words, the efficient extendability of our problems on $\mathcal{R}_c$ is not proved directly but rather follows as an immediate consequence of our proofs in this section and the correctness of known treewidth-based algorithms.

Notation. Before we proceed to the three algorithms, we introduce some common notation that will be useful to describe the general approach of dynamic programming algorithms using rank-width. For a node t of a rooted rank decomposition (T,μ) of a graph G, let $S_t \subseteq V(G)$ be the set of vertices mapped by μ to leaves that t separates from the root (i.e., S_t contains the "vertices below t"). For disjoint vertex sets S and Q of G, two vertices v and w in S are *twins* with respect to Q if $N(v)\cap Q = N(w)\cap Q$. A *twin class* of S with respect to Q is a maximal subset of S that consists of pairwise twins w.r.t. Q. Twin classes are a useful concept for designing dynamic programming algorithms that run on rank-decompositions, since there are at most $2^{\mathbf{rw}(G)}$ twin classes of the set S_t of vertices assigned in a subtree rooted at an arbitrary node $t \in V(T)$ with respect to the remaining vertices in the graph. For each node t of T we enumerate these twin classes as $R_1^t, R_2^t, \ldots, R_z^t$, where $z = 2^{\mathbf{rw}(G)}$ is a bound on the number of twin classes of S_t w.r.t. $V(G)\setminus(X\cup S_t)$; in the case where there are only $i < z$ twin classes, we let $R_j^t = \emptyset$ for all $i < j \leq z$.

For two children t_1 and t_2 of some node t, we define L_{t_1,t_2} as a matrix whose rows are indexed by the indices of twin classes of S_{t_1} and columns are indexed by the indices of twin classes of S_{t_2} such that for $L_{t_1,t_2}[j_1,j_2]=1$ if $R_{j_1}^{t_1}$ is complete to $R_{j_2}^{t_2}$, and $L_{t_1,t_2}[j_1,j_2]=0$ otherwise. For a child t' of a node t, we define a function $U_{t',t}$ from the indices of twin classes of $S_{t'}$ to the indices of twin classes of S_t such that for $U_{t',t}(j_1)=j_2$ if $R_{j_1}^{t'}$ is fully contained in $R_{j_2}^{t}$. This function is well-defined because every twin class at $S_{t'}$ is fully contained in one of the twin classes at S_t. Informally, for two children t_1 and t_2 of a node t, L_{t_1,t_2} specifies which edges are created between twin classes at t, while $U_{t_1,t}$ and $U_{t_2,t}$ specify how twin-classes are "reshuffled" at t; note that all of these matrices can be pre-computed in polynomial time.

5.1. Chromatic number

In Chromatic Number, we are given a graph G and asked for the smallest number $\chi(G)$ such that the vertex set of G can be properly colored using $\chi(G)$ colors, i.e., the smallest number $\chi(G)$ such that $V(G)$ can be partitioned into $\chi(G)$ independent sets. Our aim in this section is to solve a variant of Chromatic Number on graphs with a k-vertex modulator X to $\mathcal{R}_c$ where X is precolored:

$\mathcal{R}_c$-Precoloring Extension		*Parameter: k*
Instance:	A graph G, a k-vertex modulator $X\subseteq V(G)$ to $\mathcal{R}_c$ and a proper coloring of X.	
Task:	Compute the smallest number of colors required to extend the coloring of X to a proper coloring of G.	

Theorem 18. *$\mathcal{R}_c$-Precoloring Extension can be solved in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$.*

Proof. Let G be a graph together with a k-vertex modulator X to $\mathcal{R}_c$ and a proper coloring of X. Let col_X be the set of colors assigned to at least one vertex in X; without loss of generality $\mathrm{col}_X \subseteq [k]$. Our starting point is a rooted rank-decomposition (T,μ) of $G-X$ of width at most c, which may be computed in time $\mathcal{O}(n^3)$, using Fact 2. On a high level, our algorithm will proceed by dynamic programming along (T,μ) and group colors together based on which twin classes they occur in (analogously as in the XP algorithm for Chromatic Number parameterized by clique-width, due to Kobler and Rotics [53]), but it keeps different (more detailed) records about the at most k colors used in X.

The table. We are now ready to formally define the dynamic programming table that stores the information we require at each node t of T under M_t. The entries in the table for each node will be associated to tuples of $b_1, b_2, \ldots, b_k \subseteq [z]$ and

$d : 2^{[z]} \to [n]$, and contain either 0 or 1; here $2^{[z]}$ denotes the power set of $[z]$. We let $M_t(b_1, b_2, \ldots, b_k, d) = 1$ if there is a proper coloring of G_t such that

- for every $i \in [k]$, the color i appears in twin classes in $\{R^t_j : j \in b_i\}$ and does not appear in other twin classes, and
- for every $Z \subseteq [z]$, $d(Z)$ is the number of colors from $\{k+1, k+2, \ldots, n\}$ that appear in twin classes of $\{R^t_j : j \in Z\}$ and do not appear in other twin classes.

On the other hand, if no such proper coloring exists, we let $M_t(b_1, b_2, \ldots, b_k, d) = 0$.

Observe that by the interpretation of d, for distinct subsets Z_1, Z_2 of $[z]$, $d(Z_1)$ and $d(Z_2)$ count disjoint sets of colors. This provides an easy way to count the total number of colors used. Since all vertices in $G - X$ appear below the root node ro, the minimum number of colors required for a proper coloring of G, given a fixed proper coloring on X will be the minimum value of $\left|\text{col}_X \cup \{ i \in [k] \mid b_i \neq \emptyset \}\right| + \sum_{Z \subseteq [z]} d(Z)$, over all tuples $(b_1, b_2, \ldots, b_k, d)$ whose M_{ro} value is 1. The tables M_t will be filled in a leaves-to-root fashion. Thus, in order to complete the proof, it suffices to show how to fill in the table M_t for each node $t \in V(T)$ based on the entries for t's children.

Leaf nodes. Consider a leaf node t, and let $\mu(v) = t$. To be a proper coloring, v has to have a color that differs from the colors in $N_G(v) \cap X$. So, either we choose a color $i \in [k]$ that does not appear in $N_G(v) \cap X$ – for this we let M_t assign the corresponding tuple ($b_i = 1$ and all other b_j-values are 0 and d maps everything to 0) to 1, or we choose a color outside of $[k]$ – for this we let M_t assign the corresponding tuple (all of b_i-values are 0 and $d(Z) = 0$, except for $d(\{j\}) = 1$ where j is the label of the twin class containing v) to 1. All other tuples are assigned a value of 0 by M_t.

Internal nodes. Let us now consider an internal node t with children t_1, t_2. To correctly decide whether to set $M_t(b^t_1, b^t_2, \ldots, b^t_k, d^t) = 1$ for some tuple $\mathcal{T} = (b^t_1, b^t_2, \ldots, b^t_k, d^t)$ at t, we proceed as described below, for all pairs of tuples $\mathcal{T}_i = (b^{t_i}_1, \ldots, b^{t_i}_k, d^{t_i})$ stored at t_i, $i \in [2]$.

First, for the colors in $[k]$, we have to check whether they are used in the twin classes in $R^{t_1}_{j_1}$ and $R^{t_2}_{j_2}$ that are complete to each other (meaning that an edge will be created connecting vertices of the same color), and if yes then we discard this tuple and proceed to the next. More precisely, we set $M_t(b^t_1, b^t_2, \ldots, b^t_k, d^t) = 0$, if the following does *not* hold:

- (Compatibility 1) For every $i \in [k]$ with $b^{t_1}_i \neq \emptyset$ and $b^{t_2}_i \neq \emptyset$, there are no $j_1 \in b^{t_1}_i$ and $j_2 \in b^{t_2}_i$ such that $L_{t_1,t_2}[j_1, j_2] = 1$.

Second, we need to deal colors in $\{k+1, \ldots, n\}$. Here, we branch on ("guess") how many colors appear in twin classes with indices in $d^{t_1}(Z_1)$ are the same as the colors that appear in twin classes with indices in $d^{t_2}(Z_2)$ for all pairs $Z_1, Z_2 \subseteq [z]$. Specifically, for all $Z_1, Z_2 \subseteq [z]$, let $\tau(Z_1, Z_2)$ be a non-negative integer; this will correspond to the number of common colors in $d^{t_1}(Z_1)$ and $d^{t_2}(Z_2)$. Similarly to (Compatibility 1) we have to check that for two complete twin classes, the same color does not appear. So, for each such mapping $\tau : 2^{[z]} \times 2^{[z]} \to [n]$, we check the following:

- (Compatibility 2) For every $Z_1, Z_2 \subseteq [z]$ with $\tau(Z_1, Z_2) > 0$, there are no $j_1 \in Z_1$ and $j_2 \in Z_2$ such that $L_{t_1,t_2}[j_1, j_2] = 1$.

Observe that if Compatibility 2 would be violated, then $R^{t_1}_{j_1}$ and $R^{t_2}_{j_2}$ contain the same color while being complete to each other, meaning that the result would not be a proper coloring. Hence if Compatibility 2 is violated, we discard and proceed to the next available branch (i.e., choice of τ).

Third, we check whether $\mathcal{T}_1$ and $\mathcal{T}_2$ will be combined into $\mathcal{T}$ with the function τ. Recall that the function $U_{t_i,t}$ provides us with information about which set $R^t_{j'}$ will contain $R^{t_i}_j$. We will use this function to rename our twin classes accordingly. When we perform this renaming, we also update τ, $d^{t_1}(Z)$, $d^{t_2}(Z)$ and obtain new functions τ', $dm^{t_1}(Z)$, $dm^{t_2}(Z)$, respectively. For example, if index 2 were to be renamed to index 1 at node t_1 and $\tau(\{1,2\},\{3,4\}) = 3$, then $\tau'(\{1,2\},\{3,4\})$ becomes 0 (since twin class 2 is empty after renaming) and we apply $\tau'(\{1\},\{3,4\}) \leftarrow \tau(\{1\},\{3,4\}) + \tau(\{1,2\},\{3,4\})$.

- (Renaming) For each $i \in \{1,2\}$ and $j \in [z]$, we rename twin class $R^{t_i}_j$ to $Rm^{t_i}_{U_{t_i,t}(j)}$, and update τ, $d^{t_1}(Z)$, $d^{t_2}(Z)$ to τ', $dm^{t_1}(Z)$, $dm^{t_2}(Z)$ accordingly.

Finally, for each $Z \subseteq [z]$ we check whether the number of colors in $dm^{t_1}(Z)$ and $dm^{t_2}(Z)$ correctly add up to $d^t(Z)$ when taking τ into account. This is carried out by checking that the following equality holds.

$$d^t(Z) = \quad dm^{t_1}(Z) - \sum_{Z' \subseteq [z]} \tau'(Z, Z') + dm^{t_2}(Z) - \sum_{Z' \subseteq [z]} \tau'(Z', Z)$$

$$+ \sum_{Z_1 \subseteq [z]} \Big(\sum_{Z_2 \subseteq [z], Z_1 \cup Z_2 = Z} \tau'(Z_1, Z_2) \Big).$$

Informally, the first row counts the number of colors that are unique to $dm^{t_1}_Z$, i.e., do not appear on any twin classes in t_2. The second row then does the same for $dm^{t_2}_Z$. To correctly determine the actual number of colors in d^t_Z that we obtain by following τ, we now need to add colors which occur in both t_1 and t_2 and which end up precisely in Z – that is what the third row counts. Summarizing, we assign $M_t(b^t_1, b^t_2, \ldots, b^t_k, d^t) = 1$ if all the above conditions are satisfied in at least one branch, and use 0 otherwise.

We conclude the proof by analyzing the running time of the described algorithm. For a leaf node t, we can fill in the table in time $\mathcal{O}(k)$. For an internal node t, there are at most $2^{zk} \cdot n^{2^z}$ different tuples $(b^t_1, b^t_2, \ldots, b^t_k, d^t)$. For each, (Compatibility 1) can be checked in time $\mathcal{O}(k)$. The number of possible functions τ is the same as the number of possible assignments of a number from $[n]$ to each pair $Z_1, Z_2 \subseteq [z]$. Thus, it is bounded by $n^{2^{2z}}$. Finally, updating τ, $d^{t_1}(Z)$, $d^{t_2}(Z)$ to τ', $dm^{t_1}(Z)$, $dm^{t_2}(Z)$ can be done in time $\mathcal{O}(k)$. So, for a given coloring on X, we can find a minimum number of necessary colors in time $2^{zk} n^{\mathcal{O}(1)}$. □

5.2. Hamiltonian Cycle

In Hamiltonian Cycle, we are given an n-vertex graph G and asked whether G contains a cycle of length n as a subgraph. Note that if we restrict G to some proper subset of vertices $Y \subseteq V(G)$, then what remains from a Hamiltonian Cycle in G is a set of paths that start and end in the neighborhood of $V(G) \setminus Y$. Hence, the aim of this section is to solve the following generalization of Hamiltonian Cycle:

$\mathcal{R}_c$-Disjoint Paths Cover		*Parameter: k*
Instance:	A graph G, a k-vertex modulator $X \subseteq V(G)$ to $\mathcal{R}_c$, and $m \leq k$ pairs $(s_1, t_1), \ldots, (s_m, t_m)$ of vertices from X with $s_i \neq t_i$ for all $i \in [m]$.	
Task:	Decide whether there are internally vertex-disjoint paths $P_1, P_2, \ldots, P_m$ in G such that P_i is a path from s_i to t_i and every vertex in $G - X$ belongs to precisely one path in $P_1, P_2, \ldots, P_m$.	

Theorem 19. *$\mathcal{R}_c$-Disjoint Paths Cover can be solved in time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$.*

Proof. Let G be a graph together with a k-vertex modulator X to $\mathcal{R}_c$ and m pairs $(s_1, t_1), \ldots, (s_m, t_m)$ of vertices from X. Our starting point is once again a rooted rank-decomposition (T, μ) of $G - X$ of width at most c, which may be computed in time $\mathcal{O}(n^3)$, using Fact 2. We will obtain a fixed-parameter algorithm for checking the existence of such paths $P_1, \ldots, P_m$ in G by expanding the records used in Espelage, Gurski and Wanke's algorithm [27] for computing Hamiltonian Cycle parameterized by clique-width.

To follow partial solutions on each subgraph G_t, we consider certain generalizations of path-partitions of subgraphs of G. For a subgraph H of G, an *X-lenient path-partition* $\mathcal{P}$ of H is a collection of paths in H that are internally vertex-disjoint and share only endpoints in X such that $\bigcup_{P \in \mathcal{P}} V(P) = V(H)$. For convenience, we consider a path as an ordered sequence of vertices, and for a path $P = v_1 v_2 \cdots v_x$, we define $\ell(P) = v_1$ and $r(P) = v_x$.

The table. The entries in the table for each node will be associated to tuples of D and SP, and contain either 0 or 1; here $D = \{d_{b_1,b_2} \in \{0, 1, \ldots, n\} \mid (b_1, b_2) \in [z] \times [z]\}$. The integer d_{b_1,b_2} will represent the number of paths in an X-lenient path-partition of G_t that are fully contained in $G_t - X$ and whose endpoints are contained in $R^t_{b_1}$ and $R^t_{b_2}$. Let SP be a set such that

- for each $i \in \{1, \ldots, m\}$, $(i, 0, x)$, $(0, i, x)$, (i, i) with some $x \in [z]$ are the only possible tuples in SP,
- each integer in $\{1, \ldots, m\}$ appears at most once as an ℓ among all tuples $(\ell, 0, p)$ or (ℓ, ℓ) in SP, and similarly, each integer in $\{1, \ldots, m\}$ appears at most once as an r among all tuples $(0, r, p)$ or (r, r) in SP.

In short, the tuple $(i, 0, t)$ will indicate the existence of a path starting in s_i and ending in a vertex in R^t_x. Similarly, $(0, i, t)$ indicates the existence of a path starting in t_i and ending in R^t_x. The tuple (i, i) will indicate the existence of a path starting in s_i and ending in t_i. Note that there are at most $(n+1)^{z^2}$ possibilities for D. For an element of SP, there are $2mz + 1$ possible elements in SP, and thus there are at most 2^{2kz+1} possibilities for SP. It implies that the number of possible tuples (D, SP) is bounded by $(n+1)^{z^2} 2^{2kz+1}$.

We define a dynamic programming table M_t such that $M_t(D, SP) = 1$ if there is an X-lenient path-partition $\mathcal{P} = \mathcal{P}_1 \uplus \mathcal{P}_2$ of G_t such that

- $\mathcal{P}_1$ is the subset of $\mathcal{P}$ that consists of all paths fully contained in $G_t - X$,
- for every $d_{b_1,b_2} \in D$, there are exactly d_{b_1,b_2} distinct paths in $\mathcal{P}_1$ whose endpoints are contained in $R^t_{b_1}$ and $R^t_{b_2}$,

- for every (ℓ, r, p) or $(\ell, r) \in SP$, there is a unique path $P \in \mathcal{P}_2$ such that
  - if $\ell = i > 0$, then $\ell(P) = s_i$, and if $r = i > 0$, then $r(P) = t_i$,
  - if $\ell = 0$, then $\ell(P) \in R^t_p$, and if $r = 0$, then $r(P) \in R^t_p$.

In this case, we say that the X-lenient path-partition $\mathcal{P}$ is a partial solution with respect to (D, SP), and also (D, SP) is a characteristic of $\mathcal{P}$. We say that paths in $\mathcal{P}_1$ are normal paths, and paths in $\mathcal{P}_2$ are special paths. The value $M_t(D, SP)$ is 0 if there is no such X-lenient path-partition of G_t. We define $\mathcal{Q}_t$ as the set of all tuples (D, SP) where $M_t(D, SP) = 1$.

Since all vertices in $G - X$ appear below the root node ro, to decide whether there is a desired X-lenient path-partition, it suffices to confirm that there are D and SP such that

- $M_{ro}(D, SP) = 1$,
- for every $d_{b_1,b_2} \in D$, $d_{b_1,b_2} = 0$,
- for every $i \in \{1, 2, \ldots, m\}$, $(i, i) \in SP$.

The tables M_t will be filled in a leaves-to-root fashion. Thus, in order to complete the proof, it suffices to show how to fill in the table M_t for each node $t \in V(T)$ based on the entries for t's children.

Leaf nodes. First consider a leaf node t. Let $J = \{ i \in [m] \mid \{s_i, t_i\} \in E(G) \}$. For every pair $i \in J$, we can branch on whether a s_i-t_i path will be only the edge $\{s_i, t_i\}$, or it will contain also some vertices from $G - X$. For $I \subseteq J$, let us denote by SP_I the set $\{ (i, i) \mid i \in I \}$. Moreover, let a vertex v be assigned to t, and x be the index of the twin class $\{v\}$. If it is adjacent to s_i or t_i for some i, then we may select it as a starting point or an ending point of a desired path or both. So, for every tuple (D, SP) such that $d_{b_i,b_j} = 0$ for all $b_i, b_j \in [z]$, and $SP = SP_I \cup SP_i$, where $I \subseteq J \setminus \{i\}$ and

$$SP_i = \begin{cases} \{(i, 0, x)\} & \text{if } v \in N_G(s_i) \\ \{(0, i, x)\} & \text{if } v \in N_G(t_i) \\ \{(i, i)\} & \text{if both holds,} \end{cases}$$

we give $M_t(D, SP) = 1$. Note, that if $v \in N_G(s_i) \cap N_G(t_i)$, then there is a tuple for each of the three possibilities for SP_i. Also, we could just keep it as a path whose endpoints are not linked to X in the final solution. In this case, for a tuple (D, SP) where

- $d_{x,x} = 1$ and all other $d_{y,y'}$'s have values 0,
- $SP = SP^I$ for some $I \subseteq J$,

we give $M_t(D, SP) = 1$. Other tuples get the value 0.

Internal nodes. Let t be an internal node, and let t_1, t_2 be the children of t. We assume that $\mathcal{Q}_{t_1}$ and $\mathcal{Q}_{t_2}$ were already computed, and we want to generate $\mathcal{Q}_t$ from these sets.

For all pairs of tuples $(D^1, SP^1) \in \mathcal{Q}_{t_1}$ and $(D^2, SP^2) \in \mathcal{Q}_{t_2}$, we will generate all possible tuples (D, SP) such that for an X-lenient path-partition $\mathcal{P}$ with respect to (D, S), its restriction to G_{t_i} has (D^i, SP^i) as a characteristic.

We first test basic conditions that the two partial solutions can be a partial solution in G_t. The starting point or an endpoint of a special path in the final solution should start with one vertex in $G - X$. So, the following condition has to be satisfied.

- (Special terminal) Each integer in $\{1, \ldots, m\}$ appears at most once as an ℓ among all tuples $(\ell, 0, p)$ or (ℓ, ℓ) in $SP^1 \cup SP^2$, and similarly, each integer in $\{1, \ldots, m\}$ appears at most once as an r among all tuples $(0, r, p)$ or (r, r) in $SP^1 \cup SP^2$.

If (Special terminal) condition is not satisfied, then we skip this pair of tuples. In the rest, we assume that (Special terminal) condition is satisfied.

Secondly, we rename twin classes to consider both partial solutions together.

- (Renaming) For each $i \in \{1, 2\}$ and $j \in [z]$, we rename twin class $R^{t_i}_j$ to $Rm^{t_i}_{U_{t_i,t}(j)}$, and update D^1, D^2, SP^1, SP^2 to Dm^1, Dm^2, SPm^1, SPm^2 accordingly.

For example, if index 2 were to be renamed to index 1 at node t_1 and $d^1_{2,3} = 3$, then $dm^1_{2,3}$ in Dm^1 becomes 0 and apply $dm^1_{1,3}$ in Dm^1 to $d^1_{1,3} + d^1_{2,3}$. Similarly, if there was a tuple $(i, 0, 2)$ in SP^1, then we add a tuple $(i, 0, 1)$ to SPm^1. Because this simply change the indices of twin classes, (Special terminal) condition is still valid. Afterwards we take the sum of (Dm^1, SPm^1) and (Dm^2, SPm^2) which correspond to the disjoint union of partial solutions with respect to those tuples. More precisely, we take that

- $D^{begin} = \{d^{begin}_{x,y} | x, y \in [z]\}$ where $d^{begin}_{x,y} = dm^1_{x,y} + dm^2_{x,y}$ for all $x, y \in [z]$ and $dm^1_{x,y} \in Dm^1$ and $dm^2_{x,y} \in Dm^2$, and
- $SP^{begin} = SPm^1 \cup SPm^2$.

Because of the condition (Special terminal), no integer in $\{1, \ldots, m\}$ appears twice in SP^{begin}. Therefore, (D^{begin}, SP^{begin}) is a possible tuple for G_t, which corresponds to exactly the disjoint union of partial solutions with respect to (D^1, SP^1) and (D^2, SP^2). We put it into $\mathcal{Q}_t$.

Next, we consider to merge two paths in an X-lenient path-partition, using edges between $V(G_{t_1}) \setminus X$ and $V(G_{t_2}) \setminus X$. There are three types of merging operations: merging two normal paths, merging a normal path and a special path, and merging two special paths.

- (Merging two normal paths) Suppose there are $x_1, y_1, x_2, y_2 \in [z]$ with $x'_1 = U_{t_1,t}(x_1)$, $y'_1 = U_{t_1,t}(y_1)$, $x'_2 = U_{t_2,t}(x_2)$, $y'_2 = U_{t_2,t}(y_2)$ such that $R^{t_1}_{x_1}$ is complete to $R^{t_2}_{x_2}$ and $d^1_{x_1,y_1} > 0$ and $d^2_{x_2,y_2} > 0$. Then we subtract 1 from each of $d^{begin}_{x'_1,y'_1}$ and $d^{begin}_{x'_2,y'_2}$ and add 1 to $d^{begin}_{y'_1,y'_2}$.
- (Merging a normal path and a special path) Suppose there are $x, x_2, y_2 \in [z]$ with $x' = U_{t_1,t}(x)$, $x'_2 = U_{t_2,t}(x_2)$, $y'_2 = U_{t_2,t}(y_2)$ such that $R^{t_1}_{x}$ is complete to $R^{t_2}_{x_2}$ and $(i, 0, x) \in SP^1$ and $d^2_{x_2,y_2} > 0$. Then we subtract 1 from $d^{begin}_{x'_2,y'_2}$ and replace $(i, 0, x)$ with $(i, 0, y'_2)$ in SP^{begin}. The symmetric operation is also applied.
- (Merging two special paths) Suppose there are $x, y \in [z]$ with $x' = U_{t_1,t}(x)$, $y' = U_{t_2,t}(y)$ such that $R^{t_1}_{x}$ is complete to $R^{t_2}_{y}$ and $(i, 0, x) \in SP^1$ and $(0, i, y) \in SP^2$. Then we remove $(i, 0, x')$ and $(0, i, y')$ from SP^{begin} and add (i, i) to SP^{begin}.

To keep track of the information on how many edges are used, we also have to change the information from (D^1, SP^1) and (D^2, SP^2). We recursively apply this process of merging two paths until we can no longer get a tuple (D', SP') that is not yet in $\mathcal{Q}_t$. Because the number of possible tuples (D, SP) is bounded by $(n+1)^{z^2} 2^{2kz+1}$, this process can be done in time $\mathcal{O}((n+1)^{z^2} 2^{2kz+1})$ for a fixed pair of tuples (D^1, SP^1) and (D^2, SP^2). Considering all pairs of tuples in $\mathcal{Q}_{t_1}$ and $\mathcal{Q}_{t_2}$, we can update $\mathcal{Q}_t$ in time $\mathcal{O}((n+1)^{3z^2} 2^{6kz+3})$. Over all, we can update $\mathcal{Q}_t$ for all t in time $\mathcal{O}((n+1)^{3z^2+1} 2^{6kz+3})$. As z is constant, this problem can be solved in time $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$. □

5.3. Max-Cut

The third problem we consider is Max-Cut, where we are given an integer ℓ together with an n-vertex graph G and asked whether $V(G)$ can be partitioned into sets V_1 and V_2 such that the number of edges with precisely one endpoint in V_1 (called the *cut size*) is at least ℓ.

$\mathcal{R}_c$-Max-Cut Extension		*Parameter:* k
Instance:	A graph G, a k-vertex modulator $X \subseteq V(G)$ to $\mathcal{R}_c$, $s \subseteq X$, and $\ell \in \mathbb{N}$.	
Task:	Is there a partition of $V(G)$ into sets V_1 and V_2 such that $X \cap V_1 = s$ and the number of edges between V_1 and V_2 is at least ℓ.	

Theorem 20. $\mathcal{R}_c$-Max-Cut Extension *can be solved in polynomial time.*

Proof. Let G be a graph together with a k-vertex modulator X to $\mathcal{R}_c$ and a set $s \subseteq X$. Since, we fix the partition of X beforehand, we can assume that X is an independent set and only add the number of edges between s and $X \setminus s$ at the end. We can use Fact 2 to obtain a rank-decomposition (T, μ) of $G - X$ of width at most c in time in $\mathcal{O}(n^3)$.

The general idea is similar to the one for the XP algorithm for Max-Cut [36]: We determine, via a dynamic programming table, how large the largest cut in G_t can be if we assume a certain number of vertices of each twin class to be in the same side of the cut as s.

The table. The dynamic programming table consists of entries for each node t of T, each of which in turn describe for any tuple $(c_1, \ldots, c_z) \in [|R^t_1|] \times \ldots [|R^t_z|]$ what the largest possible cut in G_t is such that c_i vertices of R^t_i are in the same side of the cut as s. Formally,

$$M_t(c_1, \ldots, c_z) = \max_{\substack{s \subseteq C \subseteq V(G) \\ \forall 1 \leq i \leq z \, |C \cap R^t_i| = c_i}} |\{e \in E(G) \mid e \cap C \neq \emptyset \wedge e \cap (V(G) \setminus C) \neq \emptyset\}|.$$

Leaf nodes. Actually, the only adaptation that has to be made for the consideration of $s \subseteq X$ concerns the leaf nodes. Consider a leaf node t and let $v = \mu^{-1}(t)$. There are two possibilities for a cut $C \supseteq s$ of G_t: Either we include v into the side of the cut containing s, this corresponds to defining $M_t(1) = |N_G(v) \cap (X \setminus s)|$, or we include v into the side of the cut not containing s, this corresponds to defining $M_t(0) = |N_G(v) \cap X|$.

Internal nodes. Let us now consider an internal node t with children t_1 and t_2. A cut of G_t then always is the union of a cut in G_{t_1} and a cut in G_{t_2}. The edges that contribute to the value of these cuts within G_{t_1} and G_{t_2} respectively still contribute to the value in G_t. However, edges with an endpoint in $V(G_{t_1})$ and an endpoint in $V(G_{t_2})$ are not taken into account in G_{t_1} and G_{t_2}. The number of these edges are not dependent on the specific cutsets within G_{t_1} and G_{t_2}, but the size of their intersection with each of the twin classes. Thus, each entry of the dynamic programming table at t is given rise to by a pair of two entries $M_{t_1}(c_1, \ldots, c_{z_{t_1}})$ and $M_{t_2}(d_1, \ldots, d_{z_{t_2}})$ of the dynamic programming table at t_1 and t_2, in the following way:

$$\begin{aligned} &M_t(c_{U_{t_1,t}(1)} + d_{U_{t_2,t}(1)}, \ldots, c_{U_{t_1,t}(z_{t_1})} + d_{U_{t_2,t}(z_{t_2})}) \\ &= M_{t_1}(c_1, \ldots, c_{z_{t_1}}) + M_{t_2}(d_1, \ldots, d_{z_{t_2}}) \\ &\quad + \sum_{i=1}^{z} |R^t_{U_{t_1,t}(i)}| \cdot |R^t_{U_{t_2,t}(i)}| - c_{U_{t_1,t}(i)} \cdot d_{U_{t_2,t}(i)}, \end{aligned}$$

for certain $c_1, \ldots, c_{z_{t_1}}, d_1, \ldots, d_{z_{t_2}}$. To determine the correct $c_1, \ldots, c_{z_{t_1}}, d_1, \ldots, d_{z_{t_2}}$ we can take the maximum over all possible choices.

We conclude the proof by analyzing the running time of the described algorithm. For a leaf node t, we can fill the table in time in $\mathcal{O}(k)$. For an internal node t, there are at most $n^z \leq n^{2^c}$ different tuples $(c_1, \ldots, c_z)$ and at most $n^z \leq n^{2^c}$ possibilities to write each of the c_i as a 2-sum. □

6. Algorithmic applications of $\mathcal{R}_c$-treewidth

In this section, we show that Chromatic Number, Hamiltonian Cycle and Max-Cut are FPT parameterized by $\mathcal{R}_c$-treewidth. As our starting point, recall that each of these problems admits a fixed-parameter algorithm when parameterized by treewidth which is based on leaves-to-root dynamic programming along the nodes of a nice tree decomposition. Notably, the algorithms are based on defining a certain *record* $\delta(P, Q)$ (for vertex sets P, Q) such that $\delta(B_t, Y_t)$ captures all the relevant information required to solve the problem on $G[Y_t]$ and to propagate this information from a node t to its parent. The algorithms compute these records on the leaves of the tree decomposition by brute force, and then dynamically update these records while traversing a nice tree decomposition towards the root; once the record $\delta(B_r, Y_r)$ is computed for the root r of the decomposition, the algorithm outputs the correct answer. We refer to the standard textbooks for a detailed description of dynamic programming along nice tree decompositions [18,22] and to Subsections 6.1 - 6.3 for an overview of the definition of the records used for the target problems.

Our general strategy for solving these problems will be to replicate the records employed by the respective dynamic programming algorithm $\mathbb{A}$ used for treewidth, but *only for the tree of the nice $\mathcal{R}_c$-tree decomposition* of the input graph G. Recall that aside from the "standard" simple leaf nodes, trees of nice $\mathcal{R}_c$-tree decompositions also contain boundary leaf nodes, which serve as separators between the torso and a connected component C with rank-width at most c. For $\mathbb{A}$ to work correctly with the desired running time, we need to compute the record for each boundary leaf node using a subprocedure that exploits the bounded rank-width of C; in particular, we will see that this amounts to solving the problems defined in Section 5. Before proceeding to the individual problems, we provide a formalization and proof for the general ideas outlined above.

Lemma 21. *Let $\mathcal{P}$ be a graph problem which can be solved via a fixed-parameter algorithm $\mathbb{A}$ parameterized by treewidth, where $\mathbb{A}$ runs in time $f(k') \cdot n'^a$ and operates by computing a certain record δ in a leaves-to-root fashion along a provided nice width-k' tree decomposition of the n'-vertex input graph.*

Let $\mathcal{Q}$ be obtained from $\mathcal{P}$ by receiving the following additional information in the input:

- *a nice $\mathcal{R}_c$-tree decomposition $(X, T, \{B_t \mid t \in V(T)\})$ of width k for the input n-vertex graph G, and*
- *for each boundary leaf node t corresponding to the neighborhood of a connected component C of $G[X]$, the record $\delta(B_t, B_t \cup C)$.*

Then, $\mathcal{Q}$ can be solved in time $f(k) \cdot n^a$.

Proof. Consider the algorithm $\mathbb{B}$ which computes all the records δ for simple leaf nodes by calling the respective subroutine used in $\mathbb{A}$, uses the records $\delta(B_t, B_t \cup C)$ for the boundary leaf nodes, and then proceeds to compute the records in a dynamic leaves-to-root fashion in exactly the same way as $\mathbb{A}$. It is clear that $\mathbb{B}$ terminates in the claimed runtime.

To argue correctness, assume for a contradiction that $\mathbb{B}$ outputs incorrectly. Now consider the tree decomposition $(T', \{B'_t \mid t \in V(T')\})$ of G obtained by attaching, to each boundary leaf t_C in T (where $B_{t_C} = N(C)$ for some connected component C in $G[X]$), a new leaf whose bag contains $N(C) \cup C$. While the width of such a decomposition may be linear in the number of vertices, $\mathbb{A}$ must still output the correct solution, and observe that the records computed by $\mathbb{A}$ on $(T', \{B'_t \mid t \in V(T')\})$ must precisely match those computed by $\mathbb{B}$ on $(X, T, \{B_t \mid t \in V(T)\})$. Hence, we would in this case conclude that $\mathbb{A}$ also outputs incorrectly, a contradiction. □

6.1. Chromatic number

Chromatic Number is W[1]-hard parameterized by rank-width [30] but can be solved in time $2^{\mathcal{O}(\mathbf{tw}(G)\cdot\log \mathbf{tw}(G))}\cdot n$ on n-vertex graphs when a minimum-width tree decomposition is provided with the input [47]; moreover, it is known that this runtime is essentially tight, since under ETH the problem cannot be solved in time $2^{o(\mathbf{tw}(G)\cdot\log \mathbf{tw}(G))}\cdot n^{\mathcal{O}(1)}$ [56].

It is well known that the chromatic number is at most $\mathbf{tw}(G)+1$ (this can be observed, e.g., by greedily coloring a vertex when it is introduced in the nice tree decomposition). One possible way of defining records in order to achieve a runtime of $2^{\mathcal{O}(\mathbf{tw}(G)\cdot\log \mathbf{tw}(G))}\cdot n$ is to track, for each proper coloring of vertices in a bag B_t, the minimum number of colors required to extend such a coloring to Y_t [47]. Formally, let S_t be the set of all colorings of B_t with colors $[\mathbf{tw}(G)+1]$, and let $\alpha(B_t, Y_t): S_t \to \mathbb{Z}$ be defined as follows:

- $\alpha(B_t, Y_t)(s) = -1$ if s is not a proper coloring of $G[B_t]$.
- $\alpha(B_t, Y_t)(s) = q$ if q is the minimum number of colors used by any proper coloring of $G[Y_t]$ which extends s.

Using Theorem 18, we can compute such $\alpha(B_t, Y_t)(s)$ for every proper coloring s of B_t. Hence, combining Theorem 18 and Lemma 21, we obtain:

Theorem 22. Chromatic Number *can be solved in time* $2^{\mathcal{O}(k\log(k))}\cdot n^{\mathcal{O}(1)}$ *if a nice* $\mathcal{R}_c$*-tree decomposition of width* k *is provided as input.*

6.2. Hamiltonian Cycle

Hamiltonian Cycle is W[1]-hard parameterized by rank-width [30] but can be solved in time $2^{\mathcal{O}(\mathbf{tw}(G)\cdot\log \mathbf{tw}(G))}\cdot n$ on n-vertex graphs when a minimum-width tree decomposition is provided with the input via standard dynamic programming (see, e.g., an introduction to advances made for the problem by Ziobro and Pilipczuk [65]). This algorithm can be improved to run in time $2^{\mathcal{O}(\mathbf{tw}(G))}\cdot n$ by applying the advanced *rank-based approach* of Cygan, Kratsch and Nederlof [19] to prune the number of records. To simplify our exposition, here we focus on extending the standard dynamic programming algorithm which yields a slightly super-exponential runtime.

One possibility for defining the records for Hamiltonian Cycle is to track all possible ways one can cover Y_t by paths that start and end in B_t (intuitively, this corresponds to what remains of a hypothetical solution if we "cut off" everything above Y_t) [22]. Formally, let ${B_t}^{\diamond}$ be defined as follows:

- if $|B_t| > 2$, then ${B_t}^{\diamond}$ is the set of graphs with at most $|B_t|$ edges and degree at most 2 over vertex set B_t;
- if $|B_t| = 2$, then ${B_t}^{\diamond}$ contains three (multi)graphs over vertex set B_t: the edgeless graph, the graph with one edge, and the multigraph with two edges and no loops;
- if $|B_t| = 1$, then ${B_t}^{\diamond}$ contains an edgeless graph and a graph with a single loop, both over the single vertex in B_t;
- if $|B_t| = 0$, then ${B_t}^{\diamond} = \{\text{YES}, \text{NO}\}$.

We let $\beta(B_t, Y_t): {B_t}^{\diamond} \to \{0,1\}$, where for $Q \in {B_t}^{\diamond}$ we set $\beta(B_t, Y_t)(Q) = 1$ if and only if there exists a set P of paths in $G[Y_t]$ and a bijection that maps each $(v_1, \ldots, v_\ell) \in P$ to an edge $(v_1, v_\ell) \in E(Q)$ such that each vertex $v \in G[Y_t \setminus B_t]$ is contained in precisely one path in P. In the special case where $B_t = \emptyset$, our records explicitly state whether $G[Y_t]$ contains a Hamiltonian cycle or not.

As before, we can now shift our attention to the problem of computing our records in boundary leaf nodes. If $|B_t| \leq 1$, then, because B_t is a separator in G, either $Y_t = B_t$, in which case $\beta(B_t, Y_t)$ is trivial to compute, or $\beta(B_t, Y_t)$ can be filled by simply solving Hamiltonian cycle on $G[Y_t]$ in polynomial time using known algorithms [36]. In all other cases, we loop over all of the at most k^{2k}-many graphs $Q \in {B_t}^{\diamond}$ and for each such Q we need to check whether $G[Y_t] - B_t$ can be covered by internally vertex-disjoint paths connecting the pairs of vertices in B_t that form the endpoints of the edges in Q. Moreover, note that in this case Q does not contains any loops. Hence, this is precisely the $\mathcal{R}_c$-Disjoint Paths Cover problem defined in Subsection 5.2 and combining Theorem 19 and Lemma 21, we obtain:

Theorem 23. Hamiltonian Cycle *can be solved in time* $2^{\mathcal{O}(k\log(k))}\cdot n^{\mathcal{O}(1)}$ *if a nice* $\mathcal{R}_c$*-tree decomposition of width* k *is provided as input.*

6.3. Max-Cut

Max-Cut is another problem that is W[1]-hard parameterized by rank-width [31] but admits a simple fixed-parameter algorithm parameterized by treewidth – notably, it can be solved in time $2^{\mathcal{O}(\mathbf{tw}(G))}\cdot n$ on n-vertex graphs when a minimum-width tree decomposition is provided with the input via standard dynamic programming [18,22].

The simplest way of defining the records for Max-Cut is to keep track of all possible ways the bag B_t can be partitioned into V_1 and V_2, and for each entry in our table we keep track of the maximum number of crossing edges in Y_t compatible with that entry. Formally, let $\gamma(B_t, Y_t): 2^{B_t} \to \mathbb{N}_0$, where for each $s \in 2^{B_t}$ it holds that $\gamma(B_t, Y_t)(s)$ is the maximum cut size

that can be achieved in $G[Y_t]$ by any partition (V_1, V_2) satisfying $V_1 \cap B_t = s$. As before, from Theorem 20 and Lemma 21, we obtain:

Theorem 24. Max-Cut *can be solved in time* $2^k \cdot n^{\mathcal{O}(1)}$*, if a nice* $\mathcal{R}_c$*-tree decomposition of width* k *is provided on the input.*

7. Concluding remarks

While the technical contribution of this paper mainly focused on $\mathcal{R}_c$-tree-width, a parameter that allows us to lift fixed-parameter algorithms parameterized by treewidth to well-structured dense graph classes, it is equally viable to consider $\mathcal{H}$-treewidth for other choices of $\mathcal{H}$. Naturally, one should aim at graph classes where problems of interest become tractable, but it is also important to make sure that a (nice) $\mathcal{H}$-tree decomposition can be computed efficiently (i.e., one needs to obtain analogues to our Lemma 17). Examples of graph classes that may be explored in this context include split graphs, interval graphs, and more generally graphs of bounded mim-width [3,44]. Very recently, Jansen, de Kroon and Wlodarczyk have obtained algorithms that can obtain approximately-optimal $\mathcal{H}$-tree decompositions for a number of interesting choices of $\mathcal{H}$, such as interval graphs and more generally classes defined by forbidden induced subgraphs or minors [46].

Acknowledgments

Robert Ganian and Thekla Hamm acknowledge support by the Austrian Science Fund (FWF), projects P31336 and Y1329. Thekla Hamm also acknowledges support by the Austrian Science Fund (FWF) project W1255-N23. O-joung Kwon was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2017R1A2B4005020), and also supported by the Institute for Basic Science (IBS-R029-C1).

The authors also wish to thank the anonymous reviewers for their helpful comments, especially regarding Fact 10.

Declaration of competing interest

Conflicts of interest: None

References

[1] A. Agrawal, L. Kanesh, F. Panolan, M.S. Ramanujan, S. Saurabh, An FPT algorithm for elimination distance to bounded degree graphs, in: M. Bläser, B. Monmege (Eds.), 38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference), in: LIPIcs, vol. 187, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 5:1–5:11.
[2] T. Akiyama, T. Nishizeki, N. Saito, $\mathcal{NP}$-completeness of the hamiltonian cycle problem for bipartite graphs, J. Inf. Process. 3 (1980) 73.
[3] R. Belmonte, M. Vatshelle, Graph classes with structured neighborhoods and algorithmic applications, Theor. Comput. Sci. 511 (2013) 54–65.
[4] H.L. Bodlaender, A linear-time algorithm for finding tree-decompositions of small treewidth, SIAM J. Comput. 25 (6) (1996) 1305–1317.
[5] H.L. Bodlaender, F.V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, D.M. Thilikos, (Meta) kernelization, J. ACM 63 (5) (Nov. 2016) 44:1–44:69.
[6] H.L. Bodlaender, B.M.P. Jansen, S. Kratsch, Preprocessing for treewidth: a combinatorial analysis through kernelization, SIAM J. Discrete Math. 27 (4) (2013) 2108–2142.
[7] H.L. Bodlaender, T. Kloks, Efficient and constructive algorithms for the pathwidth and treewidth of graphs, J. Algorithms 21 (2) (1996) 358–402.
[8] H.L. Bodlaender, B. van Antwerpen-de Fluiter, Reduction algorithms for graphs of small treewidth, Inf. Comput. 167 (2) (2001) 86–119.
[9] M. Bougeret, I. Sau, How much does a treedepth modulator help to obtain polynomial kernels beyond sparse graphs?, Algorithmica 81 (10) (2019) 4043–4068.
[10] J. Bulian, A. Dawar, Graph isomorphism parameterized by elimination distance to bounded degree, Algorithmica 75 (2) (2016) 363–382.
[11] J. Bulian, A. Dawar, Fixed-parameter tractable distances to sparse graph classes, Algorithmica 79 (1) (2017) 139–158.
[12] L. Cai, Parameterized complexity of vertex colouring, Discrete Appl. Math. 127 (3) (2003) 415–429.
[13] B. Courcelle, The monadic second-order logic of graphs. I. Recognizable sets of finite graphs, Inf. Comput. 85 (1) (1990) 12–75.
[14] B. Courcelle, J. Engelfriet, Graph Structure and Monadic Second-Order Logic - a Language-Theoretic Approach, Encyclopedia of Mathematics and Its Applications, vol. 138, Cambridge University Press, 2012.
[15] B. Courcelle, J.A. Makowsky, U. Rotics, Linear time solvable optimization problems on graphs of bounded clique-width, Theory Comput. Syst. 33 (2) (2000) 125–150.
[16] B. Courcelle, S. Olariu, Upper bounds to the clique width of graphs, Discrete Appl. Math. 101 (1) (2000) 77–114.
[17] B. Courcelle, S. Oum, Vertex-minors, monadic second-order logic, and a conjecture by Seese, J. Comb. Theory, Ser. B 97 (1) (2007) 91–126.
[18] M. Cygan, F.V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh, Parameterized Algorithms. Texts in Computer Science, Springer, 2013.
[19] M. Cygan, S. Kratsch, J. Nederlof, Fast hamiltonicity checking via bases of perfect matchings, J. ACM 65 (3) (2018) 12:1–12:46.
[20] R. Diestel, Graph Theory, 2nd edition, Graduate Texts in Mathematics, vol. 173, Springer Verlag, New York, 2000.
[21] M. Doucha, J. Kratochvíl, Cluster vertex deletion: a parameterization between vertex cover and clique-width, in: B. Rovan, V. Sassone, P. Widmayer (Eds.), Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS, 2012, Bratislava, Slovakia, August 27–31, 2012. Proceedings, in: Lecture Notes in Computer Science, vol. 7464, Springer, 2012, pp. 348–359.
[22] R.G. Downey, M.R. Fellows, Fundamentals of Parameterized Complexity, Texts in Computer Science, Springer, 2013.
[23] E. Eiben, R. Ganian, T. Hamm, O. Kwon, Measuring what matters: a hybrid approach to dynamic programming with treewidth, in: Proc. MFCS 2019, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 42:1–42:15.
[24] E. Eiben, R. Ganian, O. Kwon, A single-exponential fixed-parameter algorithm for distance-hereditary vertex deletion, J. Comput. Syst. Sci. 97 (2018) 121–146.
[25] E. Eiben, R. Ganian, S. Szeider, Meta-kernelization using well-structured modulators, Discrete Appl. Math. 248 (2018) 153–167.
[26] E. Eiben, R. Ganian, S. Szeider, Solving problems on graphs of high rank-width, Algorithmica 80 (2) (2018) 742–771.

[27] W. Espelage, F. Gurski, E. Wanke, Deciding clique-width for graphs of bounded tree-width, J. Graph Algorithms Appl. 7 (2) (2003) 141–180.
[28] M.R. Fellows, D. Lokshtanov, N. Misra, F.A. Rosamond, S. Saurabh, Graph layout problems parameterized by vertex cover, in: Proc. ISAAC 2008, in: Lecture Notes in Computer Science, vol. 5369, Springer, 2008, pp. 294–305.
[29] J. Flum, M. Grohe, Parameterized Complexity Theory, XIV of Texts in Theoretical Computer Science. An EATCS Series, vol. XIV, Springer Verlag, Berlin, 2006.
[30] F.V. Fomin, P.A. Golovach, D. Lokshtanov, S. Saurabh, Intractability of clique-width parameterizations, SIAM J. Comput. 39 (5) (2010) 1941–1956.
[31] F.V. Fomin, P.A. Golovach, D. Lokshtanov, S. Saurabh, Almost optimal lower bounds for problems parameterized by clique-width, SIAM J. Comput. 43 (5) (2014) 1541–1563.
[32] F.V. Fomin, B.M.P. Jansen, M. Pilipczuk, Preprocessing subgraph and minor problems: when does a small vertex cover help?, J. Comput. Syst. Sci. 80 (2) (2014) 468–495.
[33] J. Gajarský, P. Hlinený, J. Obdrzálek, S. Ordyniak, F. Reidl, P. Rossmanith, F.S. Villaamil, S. Sikdar, Kernelization using structural parameters on sparse graph classes, J. Comput. Syst. Sci. 84 (2017) 219–242.
[34] R. Ganian, Improving vertex cover as a graph parameter, Discret. Math. Theor. Comput. Sci. 17 (2) (2015) 77–100.
[35] R. Ganian, P. Hlinený, On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width, Discrete Appl. Math. 158 (7) (2010) 851–867.
[36] R. Ganian, P. Hlinený, J. Obdrzálek, A unified approach to polynomial algorithms on graphs of bounded (bi-)rank-width, Eur. J. Comb. 34 (3) (2013) 680–701.
[37] R. Ganian, S. Ordyniak, M.S. Ramanujan, Going beyond primal treewidth for (M)ILP, in: Proc. AAAI 2017, AAAI Press, 2017, pp. 815–821.
[38] R. Ganian, M.S. Ramanujan, S. Szeider, Backdoor treewidth for SAT, in: Proc. SAT 2017, in: Lecture Notes in Computer Science, vol. 10491, Springer, 2017, pp. 20–37.
[39] R. Ganian, M.S. Ramanujan, S. Szeider, Combining treewidth and backdoors for CSP, in: Proc. STACS 2017, in: LIPIcs, vol. 66, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, pp. 36:1–36:17.
[40] R. Ganian, F. Slivovsky, S. Szeider, Meta-kernelization with structural parameters, J. Comput. Syst. Sci. 82 (2) (2016) 333–346.
[41] P. Hlinený, S. Oum, D. Seese, G. Gottlob, Width parameters beyond tree-width and their applications, Comput. J. 51 (3) (2008) 326–362.
[42] P. Hliněný, S. Oum, Finding branch-decompositions and rank-decompositions, SIAM J. Comput. 38 (3) (June 2008) 1012–1032.
[43] R. Impagliazzo, R. Paturi, F. Zane, Which problems have strongly exponential complexity?, J. Comput. Syst. Sci. 63 (4) (2001) 512–530.
[44] L. Jaffke, O. Kwon, J.A. Telle, Mim-width II. The feedback vertex set problem, Algorithmica 82 (1) (2020) 118–145.
[45] B.M.P. Jansen, H.L. Bodlaender, Vertex cover kernelization revisited - upper and lower bounds for a refined parameter, Theory Comput. Syst. 53 (2) (2013) 263–299.
[46] B.M.P. Jansen, J.J.H. de Kroon, M. Wlodarczyk, Vertex deletion parameterized by elimination distance and even less, in: 53rd Annual ACM Symposium on Theory of Computing, STOC 2021, June 21–25, 2021, online, 2021, in press, see also https://arxiv.org/abs/2103.09715.
[47] K. Jansen, P. Scheffler, Generalized coloring for tree-like graphs, Discrete Appl. Math. 75 (2) (1997) 135–155.
[48] V. Jelínek, The rank-width of the square grid, Discrete Appl. Math. 158 (7) (2010) 841–850.
[49] J. Jeong, S.H. Sæther, J.A. Telle, Maximum matching width: new characterizations and a fast algorithm for dominating set, Discrete Appl. Math. 248 (2018) 114–124.
[50] M.M. Kanté, E.J. Kim, O. Kwon, C. Paul, An FPT algorithm and a polynomial kernel for linear rankwidth-1 vertex deletion, Algorithmica 79 (1) (2017) 66–95.
[51] E.J. Kim, O. Kwon, A polynomial kernel for distance-hereditary vertex deletion, in: Proc. WADS 2017, in: Lecture Notes in Computer Science, vol. 10389, Springer, 2017, pp. 509–520.
[52] T. Kloks, Treewidth: Computations and Approximations, LNCS, vol. 842, Springer Verlag, Berlin, 1994.
[53] D. Kobler, U. Rotics, Edge dominating set and colorings on graphs with fixed clique-width, Discrete Appl. Math. 126 (2–3) (2003) 197–221.
[54] D. Lapoire, Recognizability equals monadic second-order definability for sets of graphs of bounded tree-width, in: Proc. STACS 98, in: Lecture Notes in Computer Science, vol. 1373, Springer, 1998, pp. 618–628.
[55] A. Lindermayr, S. Siebertz, A. Vigny, Elimination distance to bounded degree on planar graphs, in: J. Esparza, D. Král' (Eds.), 45th International Symposium on Mathematical Foundations of Computer Science, MFCS, 2020, August 24-28, 2020, Prague, Czech Republic, in: LIPIcs, vol. 170, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 65:1–65:12.
[56] D. Lokshtanov, D. Marx, S. Saurabh, Slightly superexponential parameterized problems, SIAM J. Comput. 47 (3) (2018) 675–702.
[57] D. Marx, P. Wollan, Immersions in highly edge connected graphs, SIAM J. Discrete Math. 28 (1) (2014) 503–520.
[58] R. Niedermeier, Invitation to Fixed-Parameter Algorithms, Oxford Lecture Series in Mathematics and Its Applications, vol. 31, Oxford University Press, Oxford, 2006.
[59] S. Oum, Rank-width and vertex-minors, J. Comb. Theory, Ser. B 95 (1) (2005) 79–100.
[60] S. Oum, P.D. Seymour, Approximating clique-width and branch-width, J. Comb. Theory, Ser. B 96 (4) (2006) 514–528.
[61] N. Robertson, P.D. Seymour, Graph minors. III. Planar tree-width, J. Comb. Theory, Ser. B 36 (1) (1984) 49–64.
[62] N. Robertson, P.D. Seymour, Graph minors. X. Obstructions to tree-decomposition, J. Comb. Theory, Ser. B 52 (2) (1991) 153–190.
[63] S.H. Sæther, J.A. Telle, Between treewidth and clique-width, Algorithmica 75 (1) (2016) 218–253.
[64] M. Vatshelle, New width parameters of graphs, PhD thesis, University of Bergen, 2012.
[65] M. Ziobro, M. Pilipczuk, Finding hamiltonian cycle in graphs of bounded treewidth: experimental evaluation, in: Proc. SEA 2018, in: LIPIcs, vol. 103, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, pp. 29:1–29:14.