Discrete Mathematics and Theoretical Computer Science . . . . . .
. . . . . **vol. 26:2 #1 (2024)**

Composing dynamic programming tree-decomposition-based algorithms

Julien Baste

Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

revisions 14th Mar. 2023, 25th Jan. 2024; accepted 6th Feb. 2024.

Given two integers ℓ and p as well as ℓ graph classes $\mathcal{H}_1, \ldots, \mathcal{H}_\ell$, the problems $\mathsf{GraphPart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell, p)$, $\mathsf{VertPart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell)$, and $\mathsf{EdgePart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell)$ ask, given graph G as input, whether $V(G)$, $V(G)$, $E(G)$ respectively can be partitioned into ℓ sets $S_1, \ldots, S_\ell$ such that, for each i between 1 and ℓ, $G[S_i] \in \mathcal{H}_i$, $G[S_i] \in \mathcal{H}_i$, $(V(G), S_i) \in \mathcal{H}_i$ respectively. Moreover in $\mathsf{GraphPart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell, p)$, we request that the number of edges with endpoints in different sets of the partition is bounded by p. We show that if there exist dynamic programming tree-decomposition-based algorithms for recognizing the graph classes $\mathcal{H}_i$, for each i, then we can constructively create a dynamic programming tree-decomposition-based algorithms for $\mathsf{GraphPart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell, p)$, $\mathsf{VertPart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell)$, and $\mathsf{EdgePart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell)$. We apply this approach to known problems. For well-studied problems, like VERTEX COVER and GRAPH q-COLORING, we obtain running times that are comparable to those of the best known problem-specific algorithms. For an exotic problem from bioinformatics, called DISPLAYGRAPH, this approach improves the known algorithm parameterized by treewidth.

Keywords: graph partition, treewidth, parameterized complexity, dynamic programming, dynamic programming core model

1 Introduction

In one of the first graph partition problems, one is asked, given a graph G and an integer k, whether $V(G)$ can be partitioned into ℓ sets $V_1, \ldots, V_\ell$ such that the number of edges between two different sets is small, see for instance Goldschmidt and Hochbaum (1994). These problems have many applications starting from clustering genes by Sharan et al. (2003), through optimizing financial problems by Mezuman and Weiss (2012), parallel scientific computing to image segmentation by Grady and Schwartz (2006) and Torres and Monteiro (2012), and analysis of social networks by Qian et al. (2010). The above specified graph partitioning problem favors cutting small sets of isolated vertices in the input graph as shown by Shi and Malik (2000) and Wu and Leahy (1993). In order to avoid this kind of solution which is often undesirable for many practical applications, restrictions are often imposed on the sets V_i, $i \in [1, \ell]$. The most natural restriction is to require the partition to be balanced as done by Andreev and Racke (2006). Another one, used in image segmentation, is to consider normalized cuts as done by Shi and Malik (2000), that is, cuts that maximize the similarity within the sets while minimizing the dissimilarity between the

ISSN 1365–8050

sets. In social networks, the graph clustering problem is a graph partition problem where the graphs $G[V_i]$, $i \in [1, \ell]$, are required to be dense as studied by Schaeffer (2007). In this paper, we consider the graph partition problem in a general form defined in the following way. Given ℓ graph classes $\mathcal{H}_1, \ldots, \mathcal{H}_\ell$ and an integer p, the $\mathsf{GraphPart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell, p)$ problem consists in, given a graph G, determining whether $V(G)$ can be partitioned into ℓ sets $V_1, \ldots, V_\ell$ such that $\{\{u, v\} \in E(G) \mid u \in V_i, v \in V_j, i \neq j\}$, *i.e.*, the set of *transversal* edges, is of size at most p and $G[V_i] \in \mathcal{H}_i$ for each $i \in [1, \ell]$.

Coloring problems are special kinds of graph partition problems where the number of transversal edges is not relevant anymore. So, in the $\mathsf{VertPart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell)$ problem, the task is to determine whether the vertex set of the input graph G can be partitioned into ℓ sets $V_1, \ldots, V_\ell$ such that $G[V_i] \in \mathcal{H}_i$ for each $i \in [1, \ell]$. The most famous coloring problem is the GRAPH 3-COLORABILITY problem corresponding to $\mathsf{VertPart}(\mathcal{I}, \mathcal{I}, \mathcal{I})$ where $\mathcal{I}$ is the class of edgeless graphs. This problem is one of the first problems proved to be NP-hard by Karp (1972) and has attracted a lot of attention. While GRAPH 3-COLORABILITY is the best known, several other graph classes are also under study. For instance, Yang and Yuan (2004), Rajasingh and Shanthi (2013), and Yuan and Wang (2003) consider the induced matching partition where each vertex set of the partition should induce a graph of maximum degree 1. Chang et al. (2004) focus on $\mathsf{VertPart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell)$ where ℓ is a fixed integer, $\mathcal{H}_1 = \ldots = \mathcal{H}_\ell = \mathcal{R}$, and $\mathcal{R}$ is either the class of every tree or the class of every forest. These problems are called TREE ARBORICITY when $\mathcal{R}$ is the class of every tree and VERTEX ARBORICITY when $\mathcal{R}$ is the class of every forest. They provide polynomial time algorithms for block-cactus graph, series-parallel graphs, and cographs. Yang and Yuan (2007) focus on planar graphs of diameter two. As shown by Janssen et al. (2019), these problems have, in particular, applications in bioinformatics for constructing phylogenetic trees.

The treewidth of a graph is a structural parameter that measures the similarity of the graph to a forest, see Section 2 for the formal definitions. Courcelle (1990) shows that every problem that can be expressed in monadic second-order logic can be solved in FPT-time parameterized by treewidth, *i.e.*, in time $f(\mathsf{tw}) \cdot n^{\mathcal{O}(1)}$ for some function f where n (resp. tw) denotes the number of vertices (resp. the treewidth) of the input graph. Rao (2007) shows that if there is a monadic second-order logic formula that recognizes a graph class $\mathcal{H}$, then for any fixed integer ℓ, $\mathsf{VertPart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell)$, with $\mathcal{H}_i = \mathcal{H}$ for all $i \in [1, \ell]$, can be solved in polynomial time on graphs of bounded treewidth. If Courcelle (1990) and Rao (2007) provide powerful meta-algorithms, the claimed running times may be far from being optimal. For instance, Courcelle (1990) provides an $2^{2^{\mathcal{O}(\mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$ algorithm for GRAPH 3-COLORING when it is well known that an $2^{\mathcal{O}(\mathsf{tw})} \cdot n^{\mathcal{O}(1)}$ algorithm exists, see for instance (Cygan et al., 2015, Theorem 7.9).

Recently, treewidth has found several applications in bioinformatics when dealing with the so-called display graphs as illustrated in the work of Bryant and Lagergren (2006), Scornavacca et al. (2014), and Baste et al. (2017). In order to solve the DISPLAYGRAPH problem, Janssen et al. (2019) want to determine whether a given graph of bounded treewidth is a positive instance of $\mathsf{VertPart}(\mathcal{T}, \mathcal{T})$ where $\mathcal{T}$ is the class of all trees. Using Courcelle's theorem, they provide a $2^{2^{\mathcal{O}(\mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$ algorithm. Using the approach developed in this paper, we obtain, as Corollary 14, an algorithm running in time $2^{\mathcal{O}(\mathsf{tw})} \cdot n^{\mathcal{O}(1)}$ for this same problem.

The dynamic programming core model is a formalism introduced by Baste et al. (2022). It was first introduced in order to construct meta-algorithms for what are called diverse problems. It provides a formalism for dynamic programming algorithms that process a tree decomposition, once, in a bottom-up approach. This kind of algorithm is indeed widely used when working with treewidth. Therefore the

dynamic programming core model allows us to manipulate most of the known algorithms that process a tree decomposition.

In the present paper, we use the expressive power of this formalism and show that, with some enhancement, it can be used to easily provide algorithms, with good running times, that solve the graph partition problems parameterized by treewidth. Roughly speaking, given ℓ graph classes $\mathcal{H}_1, \ldots, \mathcal{H}_\ell$, we show that solving $\mathsf{GraphPart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell, p)$ or $\mathsf{VertPart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell)$ is not much harder than recognizing each $\mathcal{H}_i$, using a dynamic programming tree-decomposition-based algorithm. Moreover, we provide, in Theorem 10 and Theorem 11, the explicit running time needed for solving $\mathsf{GraphPart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell, p)$ and $\mathsf{VertPart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell)$ respectively as a function of the running time needed for recognizing each $\mathcal{H}_i$. We provide, in Theorem 12, similar result for the case where we want to partition the edge set of the graph, that is, for the graph problem $\mathsf{EdgePart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell)$ that, given a graph G, consists in determining whether $E(G)$ can be partitioned into ℓ sets $S_1, \ldots, S_\ell$ such that $(V(G), S_i) \in \mathcal{H}_i$ for each $i \in [1, \ell]$.

The main feature of our contribution is to present a meta-approach that provides easy-to-build and efficient algorithms for exotic problems. Moreover the running time obtained for known problems are comparable to the best-known algorithms specifically designed for each given known problem.

In Section 2, we introduce the notations and useful definitions. Section 3 is devoted to the definition of the dynamic programming core model together with some examples of dynamic cores. The main results are given in Section 4. In Section 5, we show how these results can be applied to reproduce known results and to provide unknown algorithms for exotic problems. We provide a short conclusion in Section 6.

2 Preliminaries

We denote by $\mathbb{N}$ the set of nonnegative integers. Given two integers a and b we define $[a, b]$ the set of every integer c such that $a \leq c \leq b$. Let G be a graph. Let ℓ be an integer and $A = (m_1, \ldots, m_\ell)$ be a ℓ-tuple. For each $i \in [1, \ell]$, we use the notation $A.(i)$ to refer to the i-th coordinate of A, *i.e.*, in this case to m_i. Note that the coordinates are numbered from 1 to ℓ. Given a set S, we denote by 2^S the set of every subset of S. Given an alphabet Σ, we denote by $\mathcal{W}_\Sigma$ the set of every finite words over Σ. We denote by Γ the set of three special letters denoted "(", ")", and ",".

We use $V(G)$ and $E(G)$ to denote the vertex and edge sets, respectively, of the graph G. Through out this paper, we assume that vertices are represented as elements of $\mathbb{N}$. Given a set $S \subseteq V(G)$, we denote by $G[S]$ the subgraph of G induced by S. Given a set $S \subseteq E(G)$, we denote by $G[S]$ the graph $(V(G), S)$. Given two sets $S_1, S_2 \subseteq V(G)$, we denote by $\mathsf{edge}_G(S_1, S_2)$ the set of every edge of G with one endpoint in S_1 and the other endpoint in S_2. We also denote by $\mathcal{G}$ the set of every graph. We denote by $\mathcal{I}$ the class of edgeless graphs. Given an integer p, we denote by $\mathcal{G}_p$ the class of every graph with at most p vertices. We also denote by $\mathcal{T}$ the set of every tree and by $\mathcal{F}$ the set of every forest. Given a tree T rooted at r, for each $t \in V(T)$, we denote by $\mathsf{child}(t)$ the set of every child of t in T and by $\mathsf{desc}(t)$ the set of every descendent of t in T.

A *rooted tree decomposition* of a graph G is a tuple $\mathcal{D} = (T, r, \mathcal{X})$, where T is a tree rooted at $r \in V(T)$ and $\mathcal{X} = \{X_t \mid t \in V(T)\}$ is a collection of subsets of $V(G)$ such that:

- $\bigcup_{t \in V(T)} X_t = V(G)$,
- for every edge $\{u, v\} \in E$, there is a $t \in V(T)$ such that $\{u, v\} \subseteq X_t$, and

- for each $\{x, y, z\} \subseteq V(T)$ such that z lies on the unique path between x and y in T, $X_x \cap X_y \subseteq X_z$.

The *width* of a tree decomposition $\mathcal{D} = (T, r, \mathcal{X})$, denoted by $\mathsf{w}(\mathcal{D})$, is defined as $\max_{t \in V(T)} |X_t| - 1$. The *treewidth* of a graph G, denoted by $\mathsf{tw}(G)$, is the smallest integer w such that there exists a tree decomposition of G of width at most w. We also define $Y_t = X_t \cup \bigcup_{t' \in \mathsf{child}(t)} X_{t'}$ and $Z_t = X_t \cup \bigcup_{t' \in \mathsf{desc}(t)} X_{t'}$.

It is well known, see for instance Kloks (1994), that given a rooted tree decomposition $\mathcal{D} = (T, r, \mathcal{X})$, we can, without loss of generality, assume that $X_r = \emptyset$, that, for each $t \in V(T)$, t has at most 2 children and that $|Y_t| \leq |X_t| + 1$. In the following we always assume that the rooted tree decompositions have these properties.

Given a graph G, a rooted tree decomposition $\mathcal{D} = (T, r, \mathcal{X})$ of G, and a set $S \subseteq V(G)$, we define $\mathcal{D}[S]$ to be $(T, r, \{X_t \cap S \mid t \in V(T)\})$. Note that $\mathcal{D}[S]$ is a rooted tree decomposition of $G[S]$.

3 Dynamic programming core model

In this section we define and use an improvement of the dynamic programming core model introduced by Baste et al. (2022). The main idea of this model is to formalize what is a dynamic programming algorithm based on a tree decomposition. This will allow us to manipulate these algorithms in their generic form in order to construct meta-algorithms.

Definition 1 (Dynamic Core)**.** *A* dynamic core $\mathfrak{C}$ *over an alphabet* Σ *is a set of four functions:*

- $\mathsf{Accept}_{\mathfrak{C}} : \mathcal{G} \to 2^{\mathcal{W}_\Sigma}$,
- $\mathsf{Process}_{\mathfrak{C},0} : \mathcal{G} \to 2^{\mathcal{W}_\Sigma}$,
- $\mathsf{Process}_{\mathfrak{C},1} : \mathcal{G} \times \mathcal{G} \to 2^{\mathcal{W}_\Sigma^2}$, *and*
- $\mathsf{Process}_{\mathfrak{C},2} : \mathcal{G} \times \mathcal{G} \times \mathcal{G} \to 2^{\mathcal{W}_\Sigma^3}$.

In the following, we always assume that the associated alphabet is implicitly given when a dynamic core is mentioned and we denote by $\Sigma_{\mathfrak{C}}$ the alphabet associated to a dynamic core $\mathfrak{C}$. Given a dynamic core $\mathfrak{C}$, a graph G and a rooted tree decomposition $\mathcal{D} = (T, r, \mathcal{X})$ of G, the *data* of $\mathfrak{C}$ associated to $(G, \mathcal{D})$ are, for each $t \in V(T)$:

$$\mathsf{Accept}_{\mathfrak{C},G,\mathcal{D}} = \mathsf{Accept}_{\mathfrak{C}}(G[X_r])$$

$$\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t) = \begin{cases} \mathsf{Process}_{\mathfrak{C},0}(G[X_t]) & \text{if } \mathsf{child}(t) = \emptyset, \\ \mathsf{Process}_{\mathfrak{C},1}(G[X_t], G[X_{t'}]) & \text{if } \mathsf{child}(t) = \{t'\}, \text{ and} \\ \mathsf{Process}_{\mathfrak{C},2}(G[X_t], G[X_{t'}], G[X_{t''}]) & \text{if } \mathsf{child}(t) = \{t', t''\}. \end{cases}$$

We would like to highlight that Definition 1 is the main addition of this paper, concerning the definition of the dynamic programming core model, compared to Baste et al. (2022). The functions $\mathsf{Process}_{\mathfrak{C},0}$, $\mathsf{Process}_{\mathfrak{C},1}$, and $\mathsf{Process}_{\mathfrak{C},2}$ can be viewed as the rules to update the table of the given dynamic programming algorithm, and so allow to easily and naturally construct a dynamic core from a dynamic programming algorithm that is based on a tree decomposition. Note that these rules are given independently of the tree decomposition as it is usual to do for a dynamic programming algorithm. The definitions

of $\mathsf{Accept}_{\mathfrak{C},G,\mathcal{D}}$ and $\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}$, if we consider $\Sigma = \{0,1\}$, are similar to the ones initially defined by Baste et al. (2022).

Note that, in this work, we assume that $X_r = \emptyset$ and so $\mathsf{Accept}_{\mathfrak{C},G,\mathcal{D}} = \mathsf{Accept}_{\mathfrak{C}}((\emptyset,\emptyset))$. We still keep the general notation, with X_r, to keep the setting as flexible as possible. Given a function $f : \mathcal{K} \to \mathcal{J}$ and an input $I \in \mathcal{K}$, we denote by $\tau(f, I)$ the time needed to compute $f(I)$. Given a dynamic core $\mathfrak{C}$, a graph G and a rooted tree decomposition $\mathcal{D} = (T, r, \mathcal{X})$ we let:

$$\tau_n(\mathfrak{C}, G, \mathcal{D}, t) = \begin{cases} \max_{A \subseteq Y_t} \tau(\mathsf{Process}_{\mathfrak{C},0}, G[A \cap X_t]) & \text{if } \mathsf{child}(t) = \emptyset, \\ \max_{A \subseteq Y_t} \tau(\mathsf{Process}_{\mathfrak{C},1}, (G[A \cap X_t], G[A \cap X_{t'}])) & \text{if } \mathsf{child}(t) = \{t'\}, \text{ and} \\ \max_{A \subseteq Y_t} \tau(\mathsf{Process}_{\mathfrak{C},2}, (G[A \cap X_t], G[A \cap X_{t'}], G[A \cap X_{t''}])) & \text{if } \mathsf{child}(t) = \{t', t''\}. \end{cases}$$

We also define $\tau_g(\mathfrak{C}, G, \mathcal{D}) = \sum_{t \in V(T)} \tau_n(\mathfrak{C}, G, \mathcal{D}, t)$ and $\mathsf{size}(\mathfrak{C}, G, \mathcal{D}, t) = \max_{A \subseteq Y_t} |\mathsf{Process}_{\mathfrak{C},G[A],\mathcal{D}[A]}(t)|$. The n of τ_n stands for node, and the g of τ_g stands for global. Note that, for each $t \in V(T)$, $\mathsf{Process}_{\mathfrak{C},G[Y_t],\mathcal{D}[Y_t]}(t) = \mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)$. We say that a dynamic core $\mathfrak{C}$ is *polynomial*, if for each graph G, each rooted tree decomposition $\mathcal{D} = (T, r, \mathcal{X})$ of G, and each $t \in V(T)$, $\mathsf{size}(\mathfrak{C}, G, \mathcal{D}, t)$ and $\tau_n(\mathfrak{C}, G, \mathcal{D}, t)$ are polynomial in $|V(G)| + |V(T)|$.

Definition 2. *Let $\mathfrak{C}$ be a dynamic core, G be a graph in $\mathcal{G}$, and $\mathcal{D} = (T, r, \mathcal{X})$ be a rooted tree decomposition of G. A* $(\mathfrak{C}, G, \mathcal{D})$-witness *is a function $\alpha : V(T) \to \Sigma_{\mathfrak{C}}^*$ such that $\alpha(r) \in \mathsf{Accept}_{\mathfrak{C},G,\mathcal{D}}$ and for each $t \in V(T)$,*

$$\left.\begin{array}{ll} \text{if } \mathsf{child}(t) = \emptyset, & \alpha(t) \\ \text{if } \mathsf{child}(t) = \{t'\}, & (\alpha(t), \alpha(t')) \\ \text{if } \mathsf{child}(t) = \{t', t''\}, & (\alpha(t), \alpha(t'), \alpha(t'')) \end{array}\right\} \in \mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)$$

The witness provided in Definition 2 can be seen as a proof of the correctness of the algorithms we can produce using a given dynamic core.

Definition 3. *Let $\mathcal{H}$ be a class of graphs. We say that a dynamic core* $\mathfrak{C}$ solves $\mathcal{H}$ *if for each graph $G \in \mathcal{G}$, and each rooted tree decomposition $\mathcal{D}$ of G, $G \in \mathcal{H}$ if and only if a $(\mathfrak{C}, G, \mathcal{D})$-witness exists.*

As explained by Baste et al. (2022) and summarized in Theorem 4, a dynamic core can be seen as an algorithm producer. Moreover the running time of the produced algorithms is directly connected to the definition of the associated dynamic core.

Theorem 4 (Baste et al. (2022))**.** *Let $\mathcal{H}$ be a class of graphs and $\mathfrak{C}$ be a dynamic core that solves $\mathcal{H}$. Given a graph $G \in \mathcal{G}$ and a rooted tree decomposition $\mathcal{D} = (T, r, \mathcal{X})$ of G, one can decide whether $G \in \mathcal{H}$ in time $\mathcal{O}\left(\sum_{t \in V(T)} |\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)| + \tau_g(\mathfrak{C}, G, \mathcal{D})\right)$.*

3.1 Some examples of dynamic core

In this section we provide a few examples of dynamic cores. We start by a trivial dynamic core that allows us to produce an algorithm that recognizes that a graph has no edge. This dynamic core solves $\mathcal{I}$, the class of graphs with no edges.

Observation 5. *$\mathcal{I}$ can be solved by a polynomial dynamic core $\mathfrak{C}$.*

Proof: We define $\mathfrak{C}$ such that for each $G, G', G'' \in \mathcal{G}$,

$$\begin{aligned}
\mathsf{Accept}_{\mathfrak{C}}(G) &= \{\top\} \\
\mathsf{Process}_{\mathfrak{C},0}(G) &= \{\top \mid E(G) = \emptyset\} \\
\mathsf{Process}_{\mathfrak{C},1}(G, G') &= \{(\top, \top) \mid E(G) = \emptyset\} \\
\mathsf{Process}_{\mathfrak{C},2}(G, G', G'') &= \{(\top, \top, \top) \mid E(G) = \emptyset\}
\end{aligned}$$

In this case, $\Sigma_{\mathfrak{C}} = \{\top\}$ where $\top$ represents the fact that the already explored part does not contain any edge. Given $G \in \mathcal{I}$ and $\mathcal{D} = (T, r, \mathcal{X})$ a rooted tree decomposition of G, a $(\mathfrak{C}, G, \mathcal{D})$-witness α is such that, for each $t \in V(T)$, $\alpha(t) = \top$.

For the running time, note that given a graph G and a rooted tree decomposition $\mathcal{D} = (T, r, \mathcal{X})$ of G, then for each $t \in V(T)$, $|\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)| \leq 1$ and $\tau_n(\mathfrak{C}, G, \mathcal{D}, t) = \mathcal{O}(|X_t|)$. □

We also provide a slightly more involved dynamic core that solves $\mathcal{G}_p$ for some integer p, *i.e.*, the class of graphs with at most p vertices.

Observation 6. *Let p be an integer. $\mathcal{G}_p$ can be solved by a polynomial dynamic core $\mathfrak{C}$.*

Proof: We define $\mathfrak{C}$ such that for each $G, G', G'' \in \mathcal{G}$,

$$\begin{aligned}
\mathsf{Accept}_{\mathfrak{C}}(G) &= \{q \mid q \in [0, p]\} \\
\mathsf{Process}_{\mathfrak{C},0}(G) &= \{0\} \\
\mathsf{Process}_{\mathfrak{C},1}(G, G') &= \{(q, q') \mid q' \geq 0,\ q \leq p, \text{ and } q = q' + |V(G') \setminus V(G)|\} \\
\mathsf{Process}_{\mathfrak{C},2}(G, G', G'') &= \{(q, q', q'') \mid q', q'' \geq 0,\ q \leq p, \text{ and } q = q' + q'' + |(V(G') \cup V(G'')) \setminus V(G)|\}
\end{aligned}$$

It is now an easy task to show that $\mathfrak{C}$ solves $\mathcal{G}_p$. In this case, given a graph $G \in \mathcal{G}_p$ and a rooted tree decomposition $\mathcal{D} = (T, r, \mathcal{X})$, a possible $(\mathfrak{C}, G, \mathcal{D})$-witness α is such that, for each $t \in V(T)$, $\alpha(t) = |Z_t \setminus X_t|$. Simply note that we only count the number of vertices in the part that has already been completely explored and forgotten and that $X_r = \emptyset$.

For the running time, note that given a graph G and a rooted tree decomposition $\mathcal{D} = (T, r, \mathcal{X})$ of G, then, for each $t \in V(T)$, $|\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)| \leq (p+1)^2$ and $\tau_n(\mathfrak{C}, G, \mathcal{D}, t) = \mathcal{O}(|\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)|)$. The lemma follows. □

Observations 5 and 6 show how to construct a dynamic core for trivially solvable problems. We mostly provided these observations as pedagogical examples. One can then get the intuition that most of the dynamic programming algorithms parameterized by treewidth can be translated into dynamic cores. Indeed, such an algorithm creates a dynamic programming table for each node of the tree decomposition. Theses tables are updated depending of dynamic programming tables of the children of the node taken into consideration. Transposing how these updates are done into a consistent definition of $\mathsf{Process}_{\mathfrak{C},0}$, $\mathsf{Process}_{\mathfrak{C},1}$, or $\mathsf{Process}_{\mathfrak{C},2}$, depending on the number of children of the node taken in consideration, will then provide a dynamic core for the problem.

The rank-based approach, developed by Bodlaender et al. (2015), provides, in particular, a deterministic algorithm that solves FEEDBACK VERTEX SET in time $2^{\mathcal{O}(\mathsf{tw})} \cdot n^{\mathcal{O}(1)}$, where n (resp. tw) stands for the

size (resp. treewidth) of the input graph. From this algorithm, one can easily obtain a dynamic core for recognizing if a graph is a tree. We omit the proof of it as it requires to reintroduce several tools presented by Bodlaender et al. (2015) that are out of the scope of this paper.

Observation 7. *The class $\mathcal{T}$ of trees can be solved by a dynamic core $\mathfrak{C}$ such that, for each graph G, each rooted tree decomposition $\mathcal{D} = (T, r, \mathcal{X})$ of G, and each $t \in V(T)$:*

- $|\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)| = 2^{\mathcal{O}(\mathsf{w}(\mathcal{D}))} \cdot n^{\mathcal{O}(1)}$
- $\tau_g(\mathfrak{C}, G, \mathcal{D}) = 2^{\mathcal{O}(\mathsf{w}(\mathcal{D}))} \cdot |V(T)| \cdot n^{\mathcal{O}(1)}$

Note in particular that the dynamic core provided in Observation 7 is not polynomial.

3.2 Union and intersection of dynamic core

In this section we provide some simple combinations of dynamic cores. More precisely, we show how to take the union and the intersection of two dynamic cores. This will allow us, in Theorems 10, 11, and 12, to consider the union (resp. intersection) of recognizable classes without having to prove each time that the considered union (resp. intersection) is recognizable. Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be two graph classes and let $\mathfrak{C}_1$ (resp. $\mathfrak{C}_2$) be a dynamic core that solves $\mathcal{H}_1$ (resp. $\mathcal{H}_2$). We would like to stress that, in order to solve $\mathcal{H}_1 \cup \mathcal{H}_2$ or $\mathcal{H}_1 \cap \mathcal{H}_2$, the naive procedure, consisting of using $\mathfrak{C}_1$ and then using $\mathfrak{C}_2$, would be more efficient with regard to the running time but will not produce a dynamic core. As the main theorems of the paper, namely Theorems 10, 11, and 12, rely on the knowledge of a dynamic core, this naive procedure will not suit.

Lemma 8. *Let ℓ be an integer, let $\mathcal{H}_1, \ldots, \mathcal{H}_\ell$ be graph classes and let, for each $i \in [1, \ell]$, $\mathfrak{C}_i$ be a dynamic core that solves $\mathcal{H}_i$. There exists a dynamic core $\mathfrak{C}$ that solves $\mathcal{H} = \bigcap_{i\in[1,\ell]} \mathcal{H}_i$ such that, for each graph G, each rooted tree decomposition $\mathcal{D} = (T, r, \mathcal{X})$ of G, and each $t \in V(T)$:*

- $|\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)| = \prod_{i\in[1,\ell]} |\mathsf{Process}_{\mathfrak{C}_i,G,\mathcal{D}}(t)|$
- $\tau_g(\mathfrak{C}, G, \mathcal{D}) = \sum_{i\in[1,\ell]} \tau_g(\mathfrak{C}_i, G, \mathcal{D}) + \mathcal{O}\left(\sum_{t\in V(T)} |\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)|\right)$

Proof: We define $\mathfrak{C}$ such that for each $G, G', G'' \in \mathcal{G}$,

$$\begin{aligned}
\mathsf{Accept}_{\mathfrak{C}}(G) &= \{(m_1, \ldots, m_\ell) \mid \forall i \in [1, \ell], m_i \in \mathsf{Accept}_{\mathfrak{C}_i}(G)\},\\
\mathsf{Process}_{\mathfrak{C},0}(G) &= \{(m_1, \ldots, m_\ell) \mid \forall i \in [1, \ell], m_i \in \mathsf{Process}_{\mathfrak{C}_i,0}(G)\},\\
\mathsf{Process}_{\mathfrak{C},1}(G, G') &= \{((m_1, \ldots, m_\ell), (m'_1, \ldots, m'_\ell)) \mid\\
&\qquad \forall i \in [1, \ell], (m_i, m'_i) \in \mathsf{Process}_{\mathfrak{C}_i,1}(G, G')\},\\
\mathsf{Process}_{\mathfrak{C},2}(G, G', G'') &= \{((m_1, \ldots, m_\ell), (m'_1, \ldots, m'_\ell), (m''_1, \ldots, m''_\ell)) \mid\\
&\qquad \forall i \in [1, \ell], (m_i, m'_i, m''_i) \in \mathsf{Process}_{\mathfrak{C}_i,2}(G, G', G'')\}.
\end{aligned}$$

We now prove that $\mathfrak{C}$ solves $\mathcal{H}$. First note that $\Sigma_{\mathfrak{C}} = \Gamma \cup \bigcup_{i\in[1,\ell]} \Sigma_{\mathfrak{C}_i}$. Let G be a graph and $\mathcal{D} = (T, r, \mathcal{X})$ be a rooted tree decomposition of G.

Assume first that $G \in \mathcal{H}$, then by definition of $\mathcal{H}$, for each $i \in [1,\ell]$, $G \in \mathcal{H}_i$. Let, for each $i \in [1,\ell]$, $\alpha_i : V(T) \to \Sigma^*_{\mathfrak{C}_i}$ be a $(\mathfrak{C}_i, G, \mathcal{D})$-witness. Note that it exists as $\mathfrak{C}_i$ solves $\mathcal{H}_i$ and $G \in \mathcal{H}_i$. We define $\alpha : V(T) \to \Sigma^*_{\mathfrak{C}}$ such that for each $t \in V(T)$, $\alpha(t) = (\alpha_1(t), \ldots, \alpha_\ell(t))$. By construction of $\mathfrak{C}$, α is a $(\mathfrak{C}, G, \mathcal{D})$-witness.

Assume now that there exists a $(\mathfrak{C}, G, \mathcal{D})$-witness $\alpha : V(T) \to \Sigma^*_{\mathfrak{C}}$. Then by definition of $\mathfrak{C}$, for each $t \in V(T)$, $\alpha(t)$ is a ℓ-tuple. Let define ℓ functions $\alpha_i : V(T) \to \Sigma^*_{\mathfrak{C}}$, $i \in [1,\ell]$, such that for each $i \in [1,\ell]$ and each $t \in V(T)$, $\alpha_i(t) = \alpha(t).(i)$. Then, by definition of $\mathfrak{C}$, for each $i \in [1,\ell]$, α_i is a $(\mathfrak{C}_i, G, \mathcal{D})$-witness and so $G \in \mathcal{H}_i$. Thus $G \in \mathcal{H}$.

Let us now analyze the needed running time for this algorithm. Let G be a graph and $\mathcal{D} = (T, r, \mathcal{X})$ be a rooted tree decomposition of G. For each $t \in V(T)$, we have, by definition, $|\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)| = \prod_{i\in[1,\ell]} |\mathsf{Process}_{\mathfrak{C}_i,G,\mathcal{D}}(t)|$. In order to construct the data of $\mathfrak{C}$ associated to $(G, \mathcal{D})$, we need first to construct the data of $\mathfrak{C}_i$ associated to $(G, \mathcal{D})$ for each $i \in [1,\ell]$ and then to combine them. Thus, we have

$$\tau_g(\mathfrak{C}, G, \mathcal{D}) = \sum_{i\in[1,\ell]} \tau_g(\mathfrak{C}_i, G, \mathcal{D}) + \mathcal{O}\left(\sum_{t\in V(T)} |\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)|\right).$$

□

Lemma 9. *Let ℓ be an integer, let $\mathcal{H}_1, \ldots, \mathcal{H}_\ell$ be graph classes and let, for each $i \in [1,\ell]$, $\mathfrak{C}_i$ be a dynamic core that solves $\mathcal{H}_i$. There exists a dynamic core $\mathfrak{C}$ that solves $\mathcal{H} = \bigcup_{i\in[1,\ell]} \mathcal{H}_i$ such that, for each graph G, each rooted tree decomposition $\mathcal{D} = (T, r, \mathcal{X})$ of G, and each $t \in V(T)$:*

- $|\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)| = \sum_{i\in[1,\ell]} |\mathsf{Process}_{\mathfrak{C}_i,G,\mathcal{D}}(t)|$
- $\tau_g(\mathfrak{C}, G, \mathcal{D}) = \sum_{i\in[1,\ell]} \tau_g(\mathfrak{C}_i, G, \mathcal{D}) + \mathcal{O}\left(\sum_{t\in V(T)} |\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)|\right)$

Proof: Let $\perp$ be an unused letter. We define $\mathfrak{C}$ such that for each $G, G', G'' \in \mathcal{G}$,

$$\begin{aligned}
\mathsf{Accept}_{\mathfrak{C}}(G) &= \{(m_1, \ldots, m_\ell) \mid \exists i \in [1,\ell], m_i \in \mathsf{Accept}_{\mathfrak{C}_i}(G)\},\\
\mathsf{Process}_{\mathfrak{C},0}(G) &= \{(m_1, \ldots, m_\ell) \mid \\
&\qquad \exists i \in [1,\ell], m_i \in \mathsf{Process}_{\mathfrak{C}_i,0}(G)\\
&\qquad \forall j \in [1,\ell] \setminus \{i\}, m_j = \perp\},\\
\mathsf{Process}_{\mathfrak{C},1}(G, G') &= \{((m_1, \ldots, m_\ell), (m'_1, \ldots, m'_\ell)) \mid \\
&\qquad \exists i \in [1,\ell], (m_i, m'_i) \in \mathsf{Process}_{\mathfrak{C}_i,1}(G, G')\\
&\qquad \forall j \in [1,\ell] \setminus \{i\}, (m_j, m'_j) = (\perp, \perp)\}, and\\
\mathsf{Process}_{\mathfrak{C},2}(G, G', G'') &= \{((m_1, \ldots, m_\ell), (m'_1, \ldots, m'_\ell), (m''_1, \ldots, m''_\ell)) \mid \\
&\qquad \forall i \in [1,\ell], (m_i, m'_i, m''_i) \in \mathsf{Process}_{\mathfrak{C}_i,2}(G, G', G'')\\
&\qquad \forall j \in [1,\ell] \setminus \{i\}, (m_j, m'_j, m''_j) = (\perp, \perp, \perp)\}.
\end{aligned}$$

We prove, using the same kind of argumentation as for Theorem 8, that $\mathfrak{C}$ solves $\mathcal{H}$. Note that in this case, the letter $\perp$ is used, in particular, for each coordinate i such that $G \notin \mathcal{H}_i$. □

4 Main theorem

In this section we show, given two integers ℓ and p, ℓ graph classes $\mathcal{H}_1, \ldots, \mathcal{H}_\ell$ and ℓ dynamic cores $\mathfrak{C}_1, \ldots, \mathfrak{C}_\ell$ such that, for each $i \in [1, \ell]$, $\mathfrak{C}_i$ solves $\mathcal{H}_i$, how to construct dynamic cores that solve $\mathsf{GraphPart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell, p)$, $\mathsf{VertPart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell)$, and $\mathsf{EdgePart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell)$.

We start by the graph partition problem, that is the most involved case.

Theorem 10. *Let ℓ and p be two integers, let $\mathcal{H}_1, \ldots, \mathcal{H}_\ell$ be graph classes and let, for each $i \in [1, \ell]$, $\mathfrak{C}_i$ be a dynamic core that solves $\mathcal{H}_i$. There exists a dynamic core $\mathfrak{C}$ that solves $\mathcal{H} = \mathsf{GraphPart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell, p)$ such that, for each graph G, each rooted tree decomposition $\mathcal{D} = (T, r, \mathcal{X})$ of G, and each $t \in V(T)$:*

- $|\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)| \leq \ell^{|Y_t|} \cdot (p+1)^2 \cdot \prod_{i \in [1,\ell]} \mathsf{size}(\mathfrak{C}_i, G, \mathcal{D}, t)$
- $\tau_g(\mathfrak{C}, G, \mathcal{D}) = \sum_{t \in V(T)} \Big(\mathcal{O}(|\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)|) + 2^{|Y_t|} \cdot \sum_{i \in [1,\ell]} \tau_n(\mathfrak{C}_i, G, \mathcal{D}, t) \Big)$

Proof: We define $\mathfrak{C}$ such that for each $G, G', G'' \in \mathcal{G}$,

$$
\begin{aligned}
\mathsf{Accept}_{\mathfrak{C}}(G) = \{((m_1, V_1), \ldots, (m_\ell, V_\ell), q) \mid & \\
& q \leq p, \\
& V_1, \ldots, V_\ell \text{ is a partition of } V(G), \text{ and} \\
& \forall i \in [1, \ell], m_i \in \mathsf{Accept}_{\mathfrak{C}_i}(G[V_i]) \}, & \\
\mathsf{Process}_{\mathfrak{C},0}(G) = \{((m_1, V_1), \ldots, (m_\ell, V_\ell), 0) \mid & \\
& V_1, \ldots, V_\ell \text{ is a partition of } V(G) \text{ and} \\
& \forall i \in [1, \ell], m_i \in \mathsf{Process}_{\mathfrak{C}_i,0}(G[V_i]) \}, & \\
\mathsf{Process}_{\mathfrak{C},1}(G, G') = \{(((m_1, V_1), \ldots, (m_\ell, V_\ell), q), ((m'_1, V'_1), \ldots, (m'_\ell, V'_\ell), q')) \mid & \\
& U_1, \ldots, U_\ell \text{ is a partition of } V(G) \cup V(G'), \\
& \forall i \in [1, \ell], V_i = U_i \cap V(G), \\
& \forall i \in [1, \ell], V'_i = U_i \cap V(G'), \\
& \forall i \in [1, \ell], (m_i, m'_i) \in \mathsf{Process}_{\mathfrak{C}_i,1}(G[V_i], G'[V'_i]), \\
& q \leq p, \text{ and} \\
& q = q' + \sum_{i \in [1,\ell]} |\mathsf{edge}_G(U_i \setminus V(G), (V(G) \cup V(G')) \setminus U_i)| \},
\end{aligned}
$$

$$\begin{aligned}\mathsf{Process}_{\mathfrak{C},2}(G,G',G'') = \{(((m_1,V_1),\dots,(m_\ell,V_\ell),q),((m'_1,V'_1),\dots,(m'_\ell,V'_\ell),q'),\\ ((m''_1,V''_1),\dots,(m''_\ell,V''_\ell),q'')) \mid\\ U_1,\dots,U_\ell \text{ is a partition of } V(G)\cup V(G')\cup V(G''),\\ \forall i\in[1,\ell], V_i = U_i\cap V(G),\\ \forall i\in[1,\ell], V'_i = U_i\cap V(G'),\\ \forall i\in[1,\ell], V''_i = U_i\cap V(G''),\\ \forall i\in[1,\ell], (m_i,m'_i,m''_i)\in \mathsf{Process}_{\mathfrak{C}_i,2}(G[V_i],G'[V'_i],G''[V''_i]),\\ q\leq p, \text{ and}\\ q = q'+q''+\sum_{i\in[1,\ell]} |\mathsf{edge}_G(U_i\setminus V(G),(V(G)\cup V(G')\cup V(G''))\setminus U_i)|\ \}.\end{aligned}$$

We now prove that $\mathfrak{C}$ solves $\mathcal{H} = \mathsf{GraphPart}(\mathcal{H}_1,\dots,\mathcal{H}_\ell,p)$. First note that $\Sigma_{\mathfrak{C}} = \Gamma\cup\mathbb{N}\cup\bigcup_{i\in[1,\ell]}\Sigma_{\mathfrak{C}_i}$. Let G be a graph and $\mathcal{D}=(T,r,\mathcal{X})$ be a rooted tree decomposition of G.

Assume first that $G\in\mathcal{H}$. Then, by definition, there exists $V_1,\dots,V_\ell$, partition of $V(G)$ such that for each $i\in[1,\ell]$, $G[V_i]\in\mathcal{H}_i$ and $\sum_{1\leq i<j\leq\ell}|\mathsf{edge}_G(V_i,V_j)|\leq p$. As, for each $i\in[1,\ell]$, $\mathcal{D}[V_i]$ is a rooted tree decomposition of $G[V_i]$ and $G[V_i]\in\mathcal{H}_i$, there exists $\alpha_i : V(T)\to\Sigma^*_{\mathfrak{C}_i}$, a $(\mathfrak{C}_i,G[V_i],\mathcal{D}[V_i])$-witness. Note that this witness exists as $G[V_i]\in\mathcal{H}_i$. We define $\alpha : V(T)\to\Sigma^*_{\mathfrak{C}}$. such that for each $t\in V(T)$,

$$\alpha(t) = ((\alpha_1(t),X_t\cap V_1),\dots,(\alpha_\ell(t),X_t\cap V_\ell),\sum_{i\in[1,\ell]}|\mathsf{edge}_G((Z_t\cap V_i)\setminus X_t, Z_t\setminus V_i)|).$$

By construction of $\mathfrak{C}$, α is a $(\mathfrak{C},G,\mathcal{D})$-witness.

Assume now that there exists a $(\mathfrak{C},G,\mathcal{D})$-witness $\alpha : V(T)\to\Sigma^*_{\mathfrak{C}}$. Then by definition of $\mathfrak{C}$, for each $t\in V(T)$, $\alpha(t)$ is a $(\ell+1)$-tuple where the ℓ first coordinates are pairs with the shape (m,V) where $m\in\Sigma^*_{\mathfrak{C}}$ and $V\subseteq V(G)$, and where $\alpha(t).(\ell+1)$ is an integer in $[0,p]$. For each $i\in[1,\ell]$, let $V_i = \bigcup_{t\in V(T)}\alpha(t).(i).(2)$, and let $\alpha_i : V(T)\to\Sigma^*_{\mathfrak{C}}$ be such that for each $t\in V(T)$, $\alpha_i(t) = \alpha(t).(i).(1)$. Then by definition of $\mathfrak{C}$, for each $i\in[1,\ell]$, α_i is a $(\mathfrak{C}_i,G[V_i],\mathcal{D}_i)$-witness, and so, $G[V_i]\in\mathcal{H}_i$. Note also that by definition of $\mathsf{Accept}_{\mathfrak{C}}$, $\mathsf{Process}_{\mathfrak{C},0}$, $\mathsf{Process}_{\mathfrak{C},1}$, and $\mathsf{Process}_{\mathfrak{C},2}$, the partition selected by α at step $t\in V(T)\setminus\{r\}$ is consistent with the one selected at step t' where t' is the parent of t. Combined with the definition of a tree decomposition, we obtain that $(V_1,\dots,V_\ell)$ is a partition of $V(G)$. Moreover, as α is a $(\mathfrak{C},G,\mathcal{D})$-witness, we have that $\alpha(r).(\ell+1)\leq p$, and so $G\in\mathcal{H}$.

Let us now analyze the needed running time for this algorithm. Let G be a graph and $\mathcal{D}=(T,r,\mathcal{X})$ be a rooted tree decomposition of G. Then for each partition $V_1,\dots,V_\ell$ of Y_t, there is at most $(p+1)^2$ ways to combine p' and p'' and so there are at most $(p+1)^2\cdot\prod_{i\in[1,\ell]}\mathsf{Size}(\mathfrak{C}_i,G,\mathcal{D},t)$ ways to construct an element of $\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)$ consistent with the partition. Moreover, we have $\ell^{|Y_t|}$ possible partitions of the set $|Y_t|$ into ℓ sets. Thus $|\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)|\leq\ell^{|Y_t|}\cdot(p+1)^2\cdot\prod_{i\in[1,\ell]}\mathsf{size}(\mathfrak{C}_i,G,\mathcal{D},t)$. In order to construct the data of $\mathfrak{C}$ associated to $(G,\mathcal{D})$, we first need, for each $i\in[1,\ell]$, to construct the data of $\mathfrak{C}_i$ associated to $(G[V],\mathcal{D}[V],t)$ for each $t\in V(T)$ and $V\subseteq Y_t$, and then, for each $t\in V(T)$ try every

partition of Y_t and combine the corresponding data accordingly. Thus, we have

$$\forall t \in V(T),\ \tau_n(\mathfrak{C}, G, \mathcal{D}, t) = \mathcal{O}(|\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)|) + 2^{|Y_t|} \cdot \sum_{i\in[1,\ell]} \tau_n(\mathfrak{C}_i, G, \mathcal{D}, t) \text{ and}$$

$$\tau_g(\mathfrak{C}, G, \mathcal{D}) = \sum_{t\in V(T)} \tau_n(\mathfrak{C}, G, \mathcal{D}, t).$$

The theorem follows. □

Coloring problems are graph partition problems where it is not needed to keep track of the number of transversal edges. Thus the dynamic cores we present for the coloring problems are simpler than the one providing for the graph partition problems.

Theorem 11. *Let ℓ be an integer, let $\mathcal{H}_1, \ldots, \mathcal{H}_\ell$ be graph classes and let, for each $i \in [1,\ell]$, $\mathfrak{C}_i$ be a dynamic core that solves $\mathcal{H}_i$. There exists a dynamic core $\mathfrak{C}$ that solves $\mathcal{H} = \mathsf{VertPart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell)$ such that, for each graph G, each rooted tree decomposition $\mathcal{D} = (T, r, \mathcal{X})$ of G, and each $t \in V(T)$:*

- $|\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)| \leq \ell^{|Y_t|} \cdot \prod_{i\in[1,\ell]} \mathsf{size}(\mathfrak{C}_i, G, \mathcal{D}, t)$
- $\tau_g(\mathfrak{C}, G, \mathcal{D}) = \sum_{t\in V(T)} \left(\mathcal{O}(|\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)|) + 2^{|Y_t|} \cdot \sum_{i\in[1,\ell]} \tau_n(\mathfrak{C}_i, G, \mathcal{D}, t)\right)$

Proof: Using the same base as for Theorem 10, we define $\mathfrak{C}$ such that for each $G, G', G'' \in \mathcal{G}$,

$$\begin{aligned}
\mathsf{Accept}_{\mathfrak{C}}(G) = \{&((m_1, V_1), \ldots, (m_\ell, V_\ell)) \mid \\
&V_1, \ldots, V_\ell \text{ is a partition of } V(G) \text{ and} \\
&\forall i \in [1,\ell], m_i \in \mathsf{Accept}_{\mathfrak{C}_i}(G[V_i])\ \},\\
\mathsf{Process}_{\mathfrak{C},0}(G) = \{&((m_1, V_1), \ldots, (m_\ell, V_\ell)) \mid \\
&V_1, \ldots, V_\ell \text{ is a partition of } V(G) \text{ and} \\
&\forall i \in [1,\ell], m_i \in \mathsf{Process}_{\mathfrak{C}_i,0}(G[V_i])\ \},\\
\mathsf{Process}_{\mathfrak{C},1}(G, G') = \{&(((m_1, V_1), \ldots, (m_\ell, V_\ell)), ((m'_1, V'_1), \ldots, (m'_\ell, V'_\ell))) \mid \\
&U_1, \ldots, U_\ell \text{ is a partition of } V(G) \cup V(G'), \\
&\forall i \in [1,\ell], V_i = U_i \cap V(G), \\
&\forall i \in [1,\ell], V'_i = U_i \cap V(G'), \text{ and} \\
&\forall i \in [1,\ell], (m_i, m'_i) \in \mathsf{Process}_{\mathfrak{C}_i,1}(G[V_i], G'[V'_i])\ \},\\
\mathsf{Process}_{\mathfrak{C},2}(G, G', G'') = \{&(((m_1, V_1), \ldots, (m_\ell, V_\ell)), ((m'_1, V'_1), \ldots, (m'_\ell, V'_\ell)), ((m''_1, V''_1), \ldots, (m''_\ell, V''_\ell))) \mid \\
&U_1, \ldots, U_\ell \text{ is a partition of } V(G) \cup V(G') \cup V(G''), \\
&\forall i \in [1,\ell], V_i = U_i \cap V(G), \\
&\forall i \in [1,\ell], V'_i = U_i \cap V(G'), \\
&\forall i \in [1,\ell], V''_i = U_i \cap V(G''), \text{ and} \\
&\forall i \in [1,\ell], (m_i, m'_i, m''_i) \in \mathsf{Process}_{\mathfrak{C}_i,2}(G[V_i], G'[V'_i], G''[V''_i])\ \}.
\end{aligned}$$

The proof that $\mathfrak{C}$ solves $\mathcal{H} = \mathsf{VertPart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell)$ is omitted as it is similar to the one provided for Theorem 10 at the difference that this time we do not keep track of the transversal edges. □

Edge partitioning problems are really similar to coloring problem at the difference that the subgraphs we consider are induced by a set of edges instead of a set of vertices.

Theorem 12. *Let ℓ be an integer, let $\mathcal{H}_1, \ldots, \mathcal{H}_\ell$ be graph classes and let, for each $i \in [1, \ell]$, $\mathfrak{C}_i$ be a dynamic core that solves $\mathcal{H}_i$. There exists a dynamic core $\mathfrak{C}$ that solves $\mathcal{H} = \mathsf{EdgePart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell)$ such that, for each graph G, each rooted tree decomposition $\mathcal{D} = (T, r, \mathcal{X})$ of G, and each $t \in V(T)$:*

- $|\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)| \leq \ell^{|E(G[Y_t])|} \cdot \prod_{i\in[1,\ell]} \mathsf{size}(\mathfrak{C}_i, G, \mathcal{D}, t)$
- $\tau_g(\mathfrak{C}, G, \mathcal{D}) = \sum_{t\in V(T)} \Big(\mathcal{O}(|\mathsf{Process}_{\mathfrak{C},G,\mathcal{D}}(t)|) + 2^{|E(G[Y_t])|} \cdot \sum_{i\in[1,\ell]} \tau_n(\mathfrak{C}_i, G, \mathcal{D}, t)\Big)$

Proof: We define $\mathfrak{C}$ such that for each $G, G', G'' \in \mathcal{G}$,

$$
\begin{aligned}
\mathsf{Accept}_{\mathfrak{C}}(G) = \{((m_1, E_1), \ldots, (m_\ell, E_\ell)) \mid\ & \\
& E_1, \ldots, E_\ell \text{ is a partition of } E(G) \text{ and} \\
& \forall i \in [1, \ell], m_i \in \mathsf{Accept}_{\mathfrak{C}_i}(G[E_i]) \},
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{Process}_{\mathfrak{C},0}(G) = \{((m_1, E_1), \ldots, (m_\ell, E_\ell)) \mid\ & \\
& E_1, \ldots, E_\ell \text{ is a partition of } E(G) \text{ and} \\
& \forall i \in [1, \ell], m_i \in \mathsf{Process}_{\mathfrak{C}_i,0}(G[E_i]) \},
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{Process}_{\mathfrak{C},1}(G, G') = \{(((m_1, E_1), \ldots, (m_\ell, E_\ell)), ((m'_1, E'_1), \ldots, (m'_\ell, E'_\ell))) \mid\ & \\
& U_1, \ldots, U_\ell \text{ is a partition of } E(G) \cup E(G'), \\
& \forall i \in [1, \ell], E_i = U_i \cap E(G), \\
& \forall i \in [1, \ell], E'_i = U_i \cap E(G'), \text{ and} \\
& \forall i \in [1, \ell], (m_i, m'_i) \in \mathsf{Process}_{\mathfrak{C}_i,1}(G[E_i], G'[E'_i]) \},
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{Process}_{\mathfrak{C},2}(G, G', G'') = \{(((m_1, E_1), \ldots, (m_\ell, E_\ell)), ((m'_1, E'_1), \ldots, (m'_\ell, E'_\ell)), ((m''_1, E''_1), \ldots, (m''_\ell, E''_\ell))) \mid\ & \\
& U_1, \ldots, U_\ell \text{ is a partition of } E(G) \cup E(G') \cup E(G''), \\
& \forall i \in [1, \ell], E_i = U_i \cap E(G), \\
& \forall i \in [1, \ell], E'_i = U_i \cap E(G'), \\
& \forall i \in [1, \ell], E''_i = U_i \cap E(G''), \text{ and} \\
& \forall i \in [1, \ell], (m_i, m'_i, m''_i) \in \mathsf{Process}_{\mathfrak{C}_i,2}(G[E_i], G'[E'_i], G''[E''_i]) \}.
\end{aligned}
$$

The proof that $\mathfrak{C}$ solves $\mathcal{H} = \mathsf{EdgePart}(\mathcal{H}_1, \ldots, \mathcal{H}_\ell)$ is omitted as it is, again, similar to the one provided for Theorem 10 at the difference that this time we partition over the edges instead of the vertices and there are no transversal edges to consider. □

5 Applications

In this section we show how our results lead to significant simplification when looking for algorithms parameterized by treewidth. We first confront our approach against well-known problems, namely VERTEX COVER and GRAPH q-COLORING, showing that we obtain comparable running time. We then show how this leads to quickly obtain algorithms for exotic problems, $\mathsf{VertPart}(\mathcal{T},\mathcal{T})$ in our example, for which describing an algorithm in the usual way would have been long and tedious.

VERTEX COVER, corresponding to $\mathsf{VertPart}(\mathcal{G}_k,\mathcal{I})$ for some integer k, is well known to be solvable in time $2^{\mathsf{w}}\cdot n^{\mathcal{O}(1)}$ when a tree decomposition of width w of the input graph is given, see for instance (Cygan et al., 2015, Theorem 7.9), while, as shown by Impagliazzo et al. (2001), no algorithm running in time $2^{o(\mathsf{tw})}\cdot n^{\mathcal{O}(1)}$ can solve it unless ETH fails. Combining Observation 5, Observation 6, and Theorem 11, we obtain a dynamic core that solves VERTEX COVER. Moreover, combined with Theorem 4, we obtain an algorithm solving VERTEX COVER in time $2^{\mathsf{w}}\cdot n^{\mathcal{O}(1)}$ when a tree decomposition of width w of the input graph is given.

More generally, deletion problems are problems that attract a lot of attention and that can be considered as coloring problems. Indeed, given a graph class $\mathcal{H}$, the $\mathcal{H}$-DELETION corresponds to the problem $\mathsf{VertPart}(\mathcal{G}_k,\mathcal{H})$, for some integer k. Moreover, we show, in Observation 6 that $\mathcal{G}_p$-RECOGNITION, for some integer p, has a polynomial dynamic core. Combining Observation 6 with Theorem 11, we obtain that if there exists a dynamic core such that recognizing $\mathcal{H}$ can be done in time $2^{f(\mathsf{tw})}\cdot n^{\mathcal{O}(1)}$ for some function f, then $\mathcal{H}$-DELETION can be solved in time $2^{\mathcal{O}(\mathsf{tw}+f(\mathsf{tw}))}\cdot n^{\mathcal{O}(1)}$.

The most basic and well-known coloring problem is GRAPH q-COLORING. Again we obtain an asymptotically optimal algorithm, see for instance (Cygan et al., 2015, Theorem 14.6 and Theorem 14.41), by combining Observation 5 and Theorem 11.

Corollary 13. GRAPH q-COLORING *can be solved in time* $q^{\mathsf{w}}\cdot n^{\mathcal{O}(1)}$ *when a tree decomposition of width* w *of the input graph is given.*

Proof: Given a fixed integer q, the GRAPH q-COLORING problem corresponds to $\mathsf{VertPart}(\mathcal{H}_1,\ldots,\mathcal{H}_q)$ where for each $i\in[1,q]$, $\mathcal{H}_i=\mathcal{I}$. The result follows from the combination of Observation 5 and Theorem 11. □

As discussed in the introduction, finding an algorithm for $\mathsf{VertPart}(\mathcal{T},\mathcal{T})$ parameterized by the treewidth of the input graph is a question of interest in bioinformatics. By combining Observation 7, Theorem 11, and Theorem 4, we obtain an efficient algorithm solving $\mathsf{VertPart}(\mathcal{T},\mathcal{T})$.

Corollary 14. *There exists an algorithm that solves* $\mathsf{VertPart}(\mathcal{T},\mathcal{T})$ *in time* $2^{\mathcal{O}(\mathsf{tw})}\cdot n^{\mathcal{O}(1)}$.

To the best of our knowledge, the only other algorithm parameterized by treewidth for $\mathsf{VertPart}(\mathcal{T},\mathcal{T})$ was known using Courcelle's theorem and run in time $2^{2^{\mathcal{O}(\mathsf{tw})}}\cdot n^{\mathcal{O}(1)}$.

6 Conclusion

In this paper, we provide a generic tool for solving graph partition problems, coloring problems, and edge partition problems parameterized by the treewidth of the input graph. In particular, the developed approach provides a way, different than Courcelle's theorem, to determine whether a problem is FPT parameterized

by treewidth and also provides a first estimation of the expected running time of an algorithm solving the given problem. We would like to highlight the quality of these estimations as, for some well-known problems, they are asymptotically optimal.

In this conclusion we want to stress that when solving graph partition problems, we count the number of transversal edges. The illustrated technique allows, for instance, with some small modifications, to count separately transversal edges between different vertex sets of the partition. One can ask for instance for a partition of the vertex set of the input graph into three sets V_1, V_2, V_3 such that there are $k_{1,2}$ edges between V_1 and V_2, at most $k_{2,3}$ edges edges between V_2 and V_3, and no edge between V_1 and V_3 for some integers $k_{1,2}$ and $k_{2,3}$.

This approach also allows us to work with balanced partition. In this case, we first need to obtain the size of the input graph before constructing the dynamic core. Indeed we will need to intersect each graph class we consider with the class of graph of size q (or $q+1$ if needed) for some correctly calculated q. For normalized cuts, this trick will not work as the target graph classes are not fixed before the algorithm starts to run.

More generally, we believe that using the dynamic programming core model, one can easily compose dynamic programming tree-decomposition-based algorithms with several added constraints. Moreover we also believe that it is adapted to quickly study exotic problems parameterized by treewidth.

References

K. Andreev and H. Racke. Balanced Graph Partitioning. *Theory of Computing Systems*, 39:929–939, 2006.

J. Baste, C. Paul, I. Sau, and C. Scornavacca. Efficient FPT algorithms for (strict) compatibility of unrooted phylogenetic trees. *Bulletin of Mathematical Biology*, 79(4):920–938, 2017.

J. Baste, M. Fellows, L. Jaffke, T. Masarik, M. de Oliveira Oliveira, G. Philip, and F. Rosamond. Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory. *Artificial Intelligence*, page 103644, 2022.

H. Bodlaender, M. Cygan, S. Kratsch, and J. Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243(C):86–111, 2015.

D. Bryant and J. Lagergren. Compatibility of unrooted phylogenetic trees is FPT. *Theoretical Computer Science*, 351:296–302, 2006.

G. Chang, C. Chen, and Y. Chen. Vertex and Tree Arboricities of Graphs. *Journal of Combinatorial Optimization*, 8(3):295–306, 2004.

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

M. Cygan, F. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

O. Goldschmidt and D. Hochbaum. A Polynomial Algorithm for the k-cut Problem for Fixed k. *Mathematics of Operations Research*, 19(1):24–37, 1994. doi: 10.1287/moor.19.1.24.

L. Grady and E. Schwartz. Isoperimetric graph partitioning for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(3):469–475, 2006.

R. Impagliazzo, R. Paturi, and F. Zane. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

R. Janssen, M. Jones, S. Kelk, G. Stamoulis, and T. Wu. Treewidth of display graphs: bounds, brambles and applications. *Journal of Graph Algorithms and Applications*, 23(4):715–743, 2019. doi: 10.7155/jgaa.00508.

R. Karp. Reducibility Among Combinatorial Problems. *Complexity of Computer Computations*, pages 85–103, 1972.

T. Kloks. *Treewidth, Computations and Approximations*. Springer, 1994.

E. Mezuman and Y. Weiss. Globally Optimizing Graph Partitioning Problems Using Message Passing. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 22 of *Proceedings of Machine Learning Research*, pages 770–778, 2012.

T. Qian, Y. Yang, and S. Wang. Refining Graph Partitioning for Social Network Clustering. In *Proceedings of the 11th International Conference on Web Information System Engineering (WISE)*, volume 6488 of *LNCS*, pages 77–90, 2010.

I. Rajasingh and A. Shanthi. Induced Matching Partition of Petersen and Circulant Graphs. *Procedia Engineering*, 64:395–400, 2013.

M. Rao. MSOL partitioning problems on graphs of bounded treewidth and clique-width. *Theoretical Computer Science*, 377(1):260–267, 2007.

S. Schaeffer. Graph clustering. *Computer Science Review*, 1:27–64, 2007.

C. Scornavacca, L. van Iersel, S. Kelk, and D. Bryant. The agreement problem for unrooted phylogenetic trees is FPT. *Journal of Graph Algorithms and Applications*, 18:385–392, 2014.

R. Sharan, A. Maron-Katz, and R. Shamir. CLICK and EXPANDER: a system for clustering and visualizing gene expression data. *Bioinformatics*, 19:1787–1799, 2003.

J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 2000.

A. Torres and F. Monteiro. Image Segmentation by Graph Partitioning. *AIP Conference Proceedings*, 1479(1):802–805, 2012.

Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:1101–1113, 1993.

A. Yang and J. Yuan. Partition a graph with small diameter into two induced matchings. *Applied Mathematics-A Journal of Chinese Universities*, 19(3):245–251, 2004.

A. Yang and J. Yuan. On the vertex arboricity of planar graphs of diameter two. *Discrete Mathematics*, 307(19):2438–2447, 2007.

J. Yuan and Q. Wang. Partition the vertices of a graph into induced matchings. *Discrete Mathematics*, 263(1):323–329, 2003.