Dynamic programming on bipartite tree decompositions

Lars Jaffke, Laure Morelle, Ignasi Sau, Dimitrios M. Thilikos

Dynamic programming on bipartite tree decompositions

Lars Jaffke ✉
Department of Informatics, University of Bergen, Norway

Laure Morelle ✉
LIRMM, Université de Montpellier, CNRS, France

Ignasi Sau ✉
LIRMM, Université de Montpellier, CNRS, France

Dimitrios M. Thilikos ✉
LIRMM, Université de Montpellier, CNRS, France

Abstract

We revisit a graph width parameter that we dub *bipartite treewidth*, along with its associated graph decomposition that we call *bipartite tree decomposition*. Bipartite treewidth can be seen as a common generalization of treewidth and the odd cycle transversal number. Intuitively, a bipartite tree decomposition is a tree decomposition whose bags induce almost bipartite graphs and whose adhesions contain at most one vertex from the bipartite part of any other bag, while the width of such decomposition measures how far the bags are from being bipartite. Adapted from a tree decomposition originally defined by Demaine, Hajiaghayi, and Kawarabayashi [SODA 2010] and explicitly defined by Tazari [Theor. Comput. Sci. 2012], bipartite treewidth appears to play a crucial role for solving problems related to odd-minors, which have recently attracted considerable attention. As a first step toward a theory for solving these problems efficiently, the main goal of this paper is to develop dynamic programming techniques to solve problems on graphs of small bipartite treewidth. For such graphs, we provide a number of para-NP-completeness results, FPT-algorithms, and XP-algorithms, as well as several open problems. In particular, we show that K_t-Subgraph-Cover, Weighted Vertex Cover/Independent Set, Odd Cycle Transversal, and Maximum Weighted Cut are FPT parameterized by bipartite treewidth. We also provide the following complexity dichotomy when H is a 2-connected graph, for each of the H-Subgraph-Packing, H-Induced-Packing, H-Scattered-Packing, and H-Odd-Minor-Packing problems: if H is bipartite, then the problem is para-NP-complete parameterized by bipartite treewidth while, if H is non-bipartite, then the problem is solvable in XP-time. Beyond bipartite treewidth, we define 1-$\mathcal{H}$-treewidth by replacing the bipartite graph class by any graph class $\mathcal{H}$. Most of the technology developed here also works for this more general parameter.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms.

Keywords and phrases tree decomposition, bipartite graphs, dynamic programming, odd-minors, packing, maximum cut, vertex cover, independent set, odd cycle transversal.

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.16

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2309.07754

Funding The second and the third authors were supported by the ANR project ELIT (ANR-20-CE48-0008-01), the three last authors were supported by the French-German Collaboration ANR/DFG Project UTMA (ANR-20-CE92-0027), and the first author was supported by the Research Council of Norway (No 274526).

1 Introduction

A graph H is said to be an *odd-minor* of a graph G if it can be obtained from G by iteratively removing vertices, edges, and contracting edge cuts. Hadwiger's conjecture [15], which is open since 1943, states that if a graph excludes K_t as a minor, then its chromatic number is at most $t-1$. In 1993, Gerards and Seymour [19] generalized this conjecture to odd-minors, hence drawing attention to odd-minors: the Odd Hadwiger's conjecture states that if a graph excludes K_t as an odd-minor, then its chromatic number is at most $t-1$. Since then, a number of papers regarding odd-minors appeared. Most of them focused to the resolution of the Odd Hadwiger's conjecture (see for instance [11], and [30] for a nice overview of the results), while some others aimed at extending the results of graph minor theory to odd-minors (see for instance [6, 16, 22]). In particular, Demaine, Hajiaghayi, and Kawarabayashi [6] provided a structure theorem which essentially states that graphs excluding an odd-minor can be obtained by clique-sums of almost-embeddable graphs and almost bipartite graphs. To prove this, they implicitly proved the following, which is described more explicitly by Tazari [31].

▶ **Proposition 1** ([31], adapted from [6]). *Let H be a fixed graph and let G be a given H-odd-minor-free graph. There exists a fixed graph H', $\kappa, \mu \in \mathbb{N}$ depending only on H, and an explicit uniform algorithm that computes a rooted tree decomposition of G such that:*

- *the adhesion of two nodes has size at most κ, and*
- *the torso of each bag B either consists of a bipartite graph W_B together with μ additional vertices (bags of Type 1) or is H'-minor-free (bags of Type 2).*

Furthermore, the following properties hold:

1. *Bags of Type 2 appear only in the leaves of the tree decomposition,*
2. *if B_2 is a bag that is a child of a bag B_1 in the tree decomposition, then $|B_2 \cap V(W_{B_1})| \leq 1$; and if B_2 is of Type 1, then $|B_1 \cap V(W_{B_2})| \leq 1$ as well,*
3. *the algorithm runs in time $\mathcal{O}_H(|V(G)|^4)$, and*
4. *the μ additional vertices of the bags of Type 1, called* apex *vertices, can be computed within the same running time.*

It is worth mentioning that Condition 2 of Proposition 1 is slightly stronger than what is stated in [31], but it follows from the proof of [6, Theorem 4.1].

The tree decomposition described in Proposition 1 seems hence adapted to study problems related to odd-minors. As a first step toward building a theory for solving such problems, we study in this paper a new type of tree decomposition, which we call *bipartite tree decomposition*, corresponding to the tree decompositions of Proposition 1, but where all bags are only of Type 1. We also stress that this decomposition has also been implicitly used in [21] and is also introduced, under the same name, in [4].

Bipartite treewidth. Let $\mathcal{B}$ denotes the class of bipartite graphs. A *bipartite tree decomposition* of a graph G is a triple (T, α, β), where T is a tree and $\alpha, \beta : V(T) \to 2^{V(G)}$, such that

- $(T, \alpha \cup \beta)$ is a tree decomposition of G,
- for every $t \in V(T)$, $\alpha(t) \cap \beta(t) = \emptyset$,
- for every $t \in V(T)$, $G[\beta(t)] \in \mathcal{B}$, and
- for every $tt' \in E(T)$, $|(\alpha \cup \beta)(t') \cap \beta(t)| \leq 1$.

The *width* of (T, α, β) is equal to $\max\left\{ |\alpha(t)| \;\middle|\; t \in V(T) \right\}$. The *bipartite treewidth* of G, denoted by $\mathsf{btw}(G)$, is the minimum width over all bipartite tree decompositions of G.

It follows easily from the definition that $\mathsf{btw}(G) = 0$ if and only if G is bipartite (indeed, to prove the sufficiency, just take a single bag containing the whole bipartite graph, with no apex vertices). More generally, for every graph G it holds that $\mathsf{btw}(G) \leq \mathsf{oct}(G)$, where oct denotes the size of a minimum *odd cycle transversal*, that is, a vertex set intersecting every odd cycle. On the other hand, since a bipartite tree decomposition is a tree decomposition whose width is not larger than the maximum size of a bag (in each bag, just declare all vertices as apices), for every graph G it holds that $\mathsf{btw}(G) \leq \mathsf{tw}(G) + 1$, where tw denotes treewidth. Thus, bipartite treewidth can be seen as a common generalization of treewidth and the odd cycle transversal number. Hence, an FPT-algorithm parameterized by btw should generalize *both* FPT-algorithms parameterized by tw and by oct. Since our goal is to develop a theory for solving problems related to odd-minors, the first prerequisite is that bipartite treewidth is closed under odd-minors. Fortunately, this is indeed the case (cf. [17, Lemma 3.2]). Interestingly, this would *not* be true anymore if, in Condition **2** of Proposition 1, the considered intersections were required to be upper-bounded by some integer larger than one (cf. [17, Lemma 3.3]).

This type of tree decomposition has been already used implicitly by Kawarabayashi and Reed [21] in order to solve Odd Cycle Transversal parameterized by the solution size. Independently of our work, Campbell, Gollin, Hendrey, and Wiederrecht [4] are also currently studying bipartite tree decompositions. In particular, they provide universal obstructions characterizing bounded btw in the form of a "grid theorem" (actually the result of [4] apply in the much more general setting of undirected group labeled graphs). They also designed an FPT-approximation algorithm that can construct a bipartite tree decomposition in time $g(k) \cdot n^4 \log n$. This FPT-approximation is an important prerequisite for our algorithmic results as it permits us to assume that, for the implementation of our algorithms, some (approximate) bipartite tree decomposition is provided in advance.

Our aim is to provide a general framework for the design of dynamic programming algorithms on bipartite tree decompositions and, more generally, on a broader type of decompositions that we call *1-$\mathcal{H}$-tree decompositions*. These decompositions generalize bipartite tree decompositions, in the sense that the role of bipartite graphs is replaced by a general graph class $\mathcal{H}$.

Our results. In this article we formally introduce bipartite treewidth and bipartite tree decompositions (noticing that they were implicitly already used before, as discussed above). We then focus on the complexity of various problems when the bipartite treewidth of the input graph is taken as a parameter. In particular, we show the following (cf. Table 1):

- While a graph with btw at most k is $(k+2)$-colorable (cf. [17, Lemma 6.1]), 3-Coloring is NP-complete even on graphs of oct of size three (cf. [17, Lemma 6.2]), and thus btw at most three.
- K_t-Subgraph-Cover, Weighted Vertex Cover/Independent Set, Odd Cycle Transversal, and Maximum Weighted Cut are FPT parameterized by btw (cf. [17, Corollaries 4.3, 4.5, 4.6, 4.7]). In particular, our FPT-algorithms extend the domain where these well-studied problems can be solved in polynomial time to graphs that are "locally close to being bipartite". Furthermore, as $\mathsf{btw}(G) \leq \mathsf{oct}(G)$ for any graph G, we think that the fact that Odd Cycle Transversal is FPT parameterized by btw is relevant by itself, as it generalizes the well-known FPT-algorithms parameterized by the solution size [25, 28]. We would like to mention that combining in a win-win manner our dynamic programming algorithm with the FPT-approximation and the Grid Exclusion Theorem of [4] we may derive an FPT-algorithm for Odd Cycle Transversal parameterized by

the solution size, whose running time is considerably better than the one in [4], which has been obtained independently by using the irrelevant vertex technique (see also [21]).

- Let H be a 2-connected graph. We prove that H-Minor-Packing is para-NP-complete parameterized by btw. For each of the H-Subgraph-Packing, H-Induced-Subgraph-Packing, H-Scattered-Packing, and H-Odd-Minor-Packing problems (cf. Appendix B for the definitions), we obtain the following complexity dichotomy: if H is bipartite, then the problem is para-NP-complete parameterized by btw (in fact, even for $\mathsf{btw} = 0$), and if H is non-bipartite, then the problem is solvable in XP-time. The definition of the problems and the XP-algorithms are presented in [17, Section 5] and the hardness results in [17, Lemma 6].
- In view of the definition of bipartite tree decompositions, it seems natural to consider, instead of bipartite graphs as the "free part" of the bags, any graph class $\mathcal{H}$. This leads to the more general definition of *1-$\mathcal{H}$-tree decomposition* and *1-$\mathcal{H}$-treewidth* (cf. [17, Section 3]), with 1-$\{\emptyset\}$-treewidth being equivalent to the usual treewidth and 1-$\mathcal{B}$-treewidth being the bipartite treewidth if $\mathcal{B}$ is the class of bipartite graphs. We introduce these more general definitions because our dynamic programming technologies easily extend to 1-$\mathcal{H}$-treewidth. It also seems natural to consider, instead of allowing at most one "bipartite vertex" in each adhesion, allowing any number q of them. For $q = 0$, this corresponds to the $\mathcal{H}$-treewidth defined in [8] (see also [1,18] on the study of $\mathcal{H}$-treewidth for several instantiations of $\mathcal{H}$). However, as mentioned above, while 1-$\mathcal{B}$-treewidth is closed under odd-minors (cf. [17, Lemma 3.2]), this is not the case anymore for $q \geq 2$ (cf. [17, Lemma 3.3]). For $q \geq 2$, some problems remain intractable even when H is not bipartite. As an illustration of this phenomenon, we prove that H-Scattered-Packing (where there cannot be an edge in G among the copies of H to be packed) is para-NP-complete parameterized by q-$\mathcal{B}$-treewidth for $q \geq 2$ even if H is *not* bipartite (cf. [17, Lemma 6.7]).

In the statements of the running time of our algorithms, we always let n (resp. m) be the number of vertices (resp. edges) of the input graph of the considered problem.

Problem	Complexity	Constraints on H/Running time
H(-Induced)-Subgraph/Odd-Minor-Cover [32]	para-NP-complete, $k = 0$	H bipartite containing P_3 as a subgraph
H-Minor-Cover [32]		H containing P_3 as a subgraph
H(-Induced)-Subgraph-Packing		H bipartite containing P_3 as a subgraph
H-Minor-Packing		H 2-connected with $\lvert V(H)\rvert \geq 3$
H-Odd-Minor-Packing		H 2-connected bipartite with $\lvert V(H)\rvert \geq 3$
H-Scattered-Packing		H 2-connected bipartite with $\lvert V(H)\rvert \geq 2$
3-Coloring	para-NP-complete, $k = 3$	
K_t-Subgraph-Cover	FPT	$\mathcal{O}(2^k \cdot (k^t \cdot (n+m) + m\sqrt{n}))$
Independent Set		$\mathcal{O}(2^k \cdot (k \cdot (k+n) + m\sqrt{n}))$
Weighted Independent Set		$\mathcal{O}(2^k \cdot (k \cdot (k+n) + n \cdot m))$
Odd Cycle Transversal		$\mathcal{O}(3^k \cdot k \cdot n \cdot (m + k^2))$
Maximum Weighted Cut		$\mathcal{O}(2^k \cdot (k \cdot (k+n) + n^{\mathcal{O}(1)}))$
H-Subgraph-Packing	XP	H non-bipartite 2-connected
H-Induced-Subgraph-Packing		
H-Scattered-Packing		$n^{\mathcal{O}(k)}$
H-Odd-Minor-Packing		

Table 1 Summary of the results obtained in this article.

Related results. Other types of tree decompositions integrating some "free bipartite parts" have been defined recently. As we already mentioned, Eiben, Ganian, Hamm, and Kwon [8] defined $\mathcal{H}$-*treewidth* for a fixed graph class $\mathcal{H}$. The $\mathcal{H}$-treewidth of a graph G is essentially the minimum treewidth of the graph induced by some set $X \subseteq V(G)$ such that the connected components of $G \setminus X$ belong to $\mathcal{H}$, and is equal to 0-$\mathcal{H}$-treewidth minus one (cf. [17, Section 3]). In particular, when $\mathcal{H}$ is the class of bipartite graphs $\mathcal{B}$, Jansen and de Kroon [18] provided an FPT-algorithm to test whether the $\mathcal{B}$-treewidth of a graph is at most k.

Recently, as a first step to provide a systematic theory for odd-minors, Gollin and Wiederrecht [12] defined the $\mathcal{H}$-*blind-treewidth* of a graph G, where $\mathcal{H}$ is a property of annotated graphs. Then the $\mathcal{H}$-blind-treewidth is the smallest k such that G has a tree decomposition where every bag $\beta(t)$ such that $(G, \beta(t)) \notin \mathcal{H}$ has size at most k. For the case where $\mathcal{C}$ consists of every (G, X) where every odd cycle in H as *at most* one vertex in X, we obtain the $\mathcal{C}$-blind-treewidth, for which [12] gives an analogue of the Grid Exclusion Theorem [5, 29] under the odd-minor relation. Moreover, [12] provides an FPT-algorithm for Independent Set parameterized by $\mathcal{C}$-blind-treewidth. According to [12], the *bipartite-blind treewidth* of a graph G is lower-bounded by a function of the maximum treewidth over all non-bipartite blocks of G. This immediately implies that bipartite-blind treewidth is lower-bounded by bipartite treewidth. Hence, our FPT-algorithm for Independent Set is more general than the one of [12]. Independently of our work, [4] presents an FPT-algorithm to solve Odd Cycle Transversal parameterized by btw in time $f(\mathsf{btw}) \cdot n^4 \log n$ (in fact, they solve a more general group labeled problem). Our algorithm for Odd Cycle Transversal (cf. [17, Corollary 4.6]) is considerably faster.

Organization of the paper. Due to space restrictions, many definitions, results and proofs cannot be provided here, but are available in the full version of the paper [17]. In Section 2 we provide an overview of our techniques. In Section 3 we give a general dynamic programming algorithm to obtain FPT-algorithms, and apply it to Maximum Weighted Cut. Finally, we present several questions for further research in Section 4. Additional necessary definitions are provided in Appendix A.

2 Overview of our techniques

In this section we present an overview of the techniques that we use to obtain our results.

2.1 Dynamic programming algorithms

Compared to dynamic programming on classical tree decompositions, there are two main difficulties for doing dynamic programming on (rooted) *bipartite* tree decompositions. The first one is that the bags in a bipartite tree decomposition may be arbitrarily large, which prevents us from applying typical brute-force approaches to define table entries. The second one, and apparently more important, is the lack of an upper bound on the number of children of each node of the decomposition. Indeed, unfortunately, a notion of "nice bipartite tree decomposition" preserving the width (even approximately) does not exist (cf. [17, Lemma 3.4]). We discuss separately the main challenges involved in our FPT-algorithms and in our XP-algorithms.

2.1.1 FPT-algorithms

In fact, for most of the considered problems, in order to obtain FPT-algorithms parameterized by btw, it would be enough to bound the number of children as a function of btw, but we were not able to come up with a general technique that achieves this property (cf. [17, Lemma 3.4]). For particular problems, however, we can devise ad-hoc solutions. Namely, for K_t-Subgraph-Cover, Weighted Vertex Cover/Independent Set, Odd Cycle Transversal, and Maximum Weighted Cut parameterized by btw, we overcome the above issue by managing to replace the children by constant-sized bipartite gadgets. More specifically, we guess an annotation of the "apex" vertices of each bag t, whose number is bounded by btw, that essentially tells which of these vertices go to the solution or not (with some extra information depending on each particular problem; for instance, for Odd Cycle Transversal, we also guess the side of the bipartition of the non-solution vertices). Having this annotation, each adhesion of the considered node t with a child contains, by the definition of bipartite tree decompositions, at most one vertex v that is not annotated. At this point, we crucially observe that, for the considered problems, we can make local computation for each child, independent from the computations at other children, depending only on the values of the optimum solutions at that child that are required to *contain* or to *exclude* v (note that we need to be able to keep this extra information at the tables of the children). Using the information given by these local computations, we can replace the children of t by constant-sized bipartite gadgets (sometimes empty) so that the newly built graph, which we call a *nice reduction*, is an equivalent instance modulo some constant. If a nice reduction can be efficiently computed for a problem Π, then we say that Π is a *nice problem* (cf. Appendix A, and [17, Section 4] for additional intuition). The newly modified bag has bounded oct, so we can then use an FPT-algorithm parameterized by oct to find the optimal solution with respect to the guessed annotation.

An illustrative example. Before entering into some more technical details and general definitions, let us illustrate this idea with the Weighted Vertex Cover problem. We want to compute the dynamic programming tables at a bag associated with a node t of the rooted tree given by the bipartite tree decomposition. Remember that the vertices of the bag at t are partitioned into two sets: $\beta(t)$ induces a bipartite graph and its complement, denoted by $\alpha(t)$, corresponds to the apex vertices, whose size is bounded by the parameter, namely btw. The first step is to guess, in time at most 2^{btw}, which vertices in $\alpha(t)$ belong to the desired minimum vertex cover. After such a guess, all the vertices in $\alpha(t)$ can be removed from the graph, by also removing the neighborhood of those that were *not* taken into the solution. The definition of bipartite tree decomposition implies that, in each adhesion with a child of the current bag, there is at most one "surviving" vertex. Let v be such a vertex belonging to the adhesion with a child t' of t. Suppose that, inductively, we have computed in the tables for t' the following two values, subject to the choice that we made for $\alpha(t)$: the minimum weight w_v of a vertex cover in the graph below t' that contains v, and the minimum weight $w_{\bar{v}}$ of a vertex cover in the graph below t' that does *not* contain v. Then, the trick is to observe that, having these two values at hand, we can totally forget the graph below t': it is enough to delete this whole graph, except for v, and attach a new pendant edge vu, where u is a new vertex, such that v is given weight w_v and u is given weight $w_{\bar{v}}$. It is easy to verify that this gadget mimics, with respect to the current bag, the behavior of including vertex v or not in the solution for the child t'. Adding this gadget for every child results in a bipartite graph, where we can just solve Weighted Vertex Cover in polynomial time using a classic algorithm [23, 27], and add the returned weight to our tables.

The running time of this whole procedure, from the leaves to the root of the decomposition, is clearly FPT parameterized by the bipartite treewidth of the input graph.

Extensions and limitations. Note that the algorithm sketched above for Weighted Vertex Cover is problem-dependent, in particular the choice of the gadgets for the children, and the fact of deleting the neighborhood of the vertices chosen in the solution. Which type of replacements and reductions can be afforded in order to obtain an FPT-algorithm for bipartite treewidth? For instance, concerning the gadgets for the children, as far as the considered problem can be solved in polynomial time on bipartite graphs, we could attach to the "surviving" vertices an arbitrary bipartite graph instead of just an edge. If we assume that the considered problem is FPT parameterized by oct (which is a reasonable assumption, as btw generalizes oct), then one could think that it may be sufficient to devise gadgets with bounded oct. Unfortunately, this will not work in general: even if each of the gadgets has bounded oct (take, for instance, a triangle), since we do not have any upper bound, in terms of btw, on the number of children (hence, the number of different adhesions), the resulting graph after the gadget replacement may have unbounded oct. In order to formalize the type of replacements and reductions that can be allowed, we introduce in Appendix A the notions of *nice reduction* and *nice problem*, along with an illustration (cf. Figure 1). Additional insights into these definitions, which are quite lengthy, are provided in [17, Section 4.1].

Another sensitive issue is that of "guessing the vertices into the solution". While this is quite simple for Weighted Vertex Cover (either a vertex is in the solution, or it is not), for some other problems we may have to guess a richer structure in order to have enough information to combine the tables of the children into the tables of the current bag. This is the reason for which, in the general dynamic programming scheme that we present in Section 3, we deal with *annotated problems*, i.e., problems that receive as input, apart from a graph, a collection of annotated sets in the form of a partition $\mathcal{X}$ of some $X \subseteq V(G)$. For instance, for Weighted Vertex Cover, we define its *annotated extension*, which we call Annotated Weighted Vertex Cover, that takes as an input a graph G and two disjoint sets R and S of vertices of G, and asks for a minimum vertex cover $S^\star$ such that $S \subseteq S^\star$ and $S^\star \cap R = \emptyset$.

General dynamic programming scheme. Our general scheme essentially says that if a problem Π has an annotated extension Π' that is

- a nice problem and
- solvable in FPT-time parameterized by oct,

then Π is solvable in FPT-time parameterized by btw. More specifically, it is enough to prove that Π' is solvable in time $f(|X|) \cdot n^{\mathcal{O}(1)}$ on an instance $(G, \mathcal{X})$ such that $G \setminus X$ is bipartite, where $\mathcal{X}$ is a partition of X corresponding to the annotation. This general dynamic programming algorithm works in a wider setting, namely for a general graph class $\mathcal{H}$ that plays the role of bipartite graphs, as far as the annotated extension Π' is what we call $\mathcal{H}$-*nice*; cf. Lemma 2 for the details.

Applications. We then apply this general framework to give FPT-algorithms for several problems parameterized by bipartite treewidth. For each of Maximum Weighted Cut (Subsection 3.4), K_t-Subgraph-Cover (cf. [17, Section 4.4.1]), Weighted Vertex Cover/Independent Set (cf. [17, Section 4.4.2]), and Odd Cycle Transversal (cf. [17, Section 4.4.3]), we prove that the problem has an annotated extension that is 1) nice and 2) solvable in FPT-time parameterized by oct, as discussed above.

To prove that an annotated problem has a nice reduction, we essentially use two ingredients. Given two compatible boundaried graphs $\mathbf{F}$ and $\mathbf{G}$ with boundary X (a boundaried graph is essentially a graph along with some labeled vertices that form a boundary, see the formal definition in Appendix A), an annotated problem is usually nice if the following hold:

- (*Gluing property*) Given that we have guessed the annotation $\mathcal{X}$ in the boundary X, a solution compatible with the annotation is optimal in the graph $\mathbf{F} \oplus \mathbf{G}$ obtained by gluing $\mathbf{F}$ and $\mathbf{G}$ if and only if it is optimal in each of the two glued graphs. In this case, it means that the optimum on $(\mathbf{F} \oplus \mathbf{G}, \mathcal{X})$ is equal to the optimum on $(F, \mathcal{X})$ modulo some constant depending only on G and $\mathcal{X}$.
- (*Gadgetization*) Given that we have guessed the annotation in the boundary $X \setminus \{v\}$ for some vertex v in X, there is a small boundaried graph G', that is bipartite (maybe empty), such that the optimum on $(\mathbf{F} \oplus \mathbf{G}, \mathcal{X})$ is equal to the optimum on $(\mathbf{F} \oplus \mathbf{G}', \mathcal{X})$ modulo some constant depending only on G and $\mathcal{X}$.

The gluing property seems critical to show that a problem is nice. This explains why we solve H-Subgraph-Cover only when H is a clique. For any graph H, Annotated H-Subgraph-Cover is defined similarly to Annotated Weighted Vertex Cover by specifying vertices that must or must not be taken in the solution. If H is a clique, then we crucially use the fact that H is a subgraph of $\mathbf{F} \oplus \mathbf{G}$ if and only if it is a subgraph of either F or G to prove that Annotated H-Subgraph-Cover has the gluing property. However, we observe that if H is not a clique, then Annotated H-Subgraph-Cover does not have the gluing property (cf. [17, Lemma 4.3]). This is the main difficulty that we face to solve H-Subgraph-Cover in the general case.

Note also that if we define the annotated extension of Odd Cycle Transversal in a similar fashion (that is, a set S of vertices contained in the solution and a set R of vertices that do not belong to the solution), then we can prove that this annotated extension does not have the gluing property. However, if we define Annotated Odd Cycle Transversal as the problem that takes as an input a graph G and three disjoint sets S, X_1, X_2 of vertices of G and aims at finding an odd cycle transversal $S^\star$ of minimum size such that $S \subseteq S^\star$ and X_1 and X_2 are on different sides of the bipartition obtained after removing $S^\star$, then Annotated Odd Cycle Transversal does have the gluing property (cf. [17, Lemma 4.9]).

For Maximum Weighted Cut, the annotation is pretty straightforward: we use two annotation sets X_1 and X_2, corresponding to the vertices that will be on each side of the cut. It is easy to see that this annotated extension has the gluing property (cf. Lemma 3).

Finding the right gadgets is the main difficulty to prove that a problem is nice. As explained above, for Annotated Weighted Vertex Cover, we replace the boundaried graph $\mathbf{G}$ by an edge that simulates the behavior of G with respect to v, which is the only vertex that interest us (cf. [17, Lemma 4.7]). For Annotated Maximum Weighted Cut, if $\mathcal{X} = (X_1, X_2)$, the behavior of G can be simulated by an edge between v and a vertex in X_1 of weight equal to the optimum on $(G, (X_1, X_2 \cup \{v\}))$ and an edge between v and a vertex in X_2 of weight equal to the optimum on $(G, (X_1 \cup \{v\}, X_2))$ (see Lemma 4). For Annotated K_t-Subgraph-Cover, if $\mathcal{X} = (R, S)$, depending on the optimum on $(G, (R \cup \{v\}, S))$ and the one on $(G, (R, S \cup \{v\}))$, we can show that the optimum on $(\mathbf{F} \oplus \mathbf{G}, \mathcal{X})$ is equal to the optimum on $(F, \mathcal{X})$ or $(F \setminus \{v\}, \mathcal{X})$ modulo some constant (cf. [17, Lemma 4.4]). For Annotated Odd Cycle Transversal, if $\mathcal{X} = (S, X_1, X_2)$, we can show that the optimum on $(\mathbf{F} \oplus \mathbf{G}, \mathcal{X})$ is equal modulo some constant to the optimum on either $(F, \mathcal{X})$, or $(F \setminus \{v\}, \mathcal{X})$, or $(F', \mathcal{X})$, where F' is obtained from F by adding an edge between v and either a vertex of X_1 or a vertex of X_2 (cf. [17, Lemma 4.10]).

Finally, let us now mention some particular ingredients used to prove that the considered

annotated problems are solvable in time $f(|X|) \cdot n^{\mathcal{O}(1)}$ on an instance $(G, \mathcal{X})$ such that $G \setminus X$ is bipartite, where $\mathcal{X}$ is a partition of a vertex set X corresponding to the annotation. For Annotated K_t-Subgraph-Cover and Annotated Weighted Vertex Cover, this is simply a reduction to (Weighted Vertex) Cover on bipartite graphs. For Odd Cycle Transversal, we adapt the algorithm of Reed, Smith, and Vetta [28] that uses iterative compression to solve Annotated Odd Cycle Transversal in FPT-time parameterized by oct, so that it takes annotations into account (cf. [17, Lemma 4.12]). As for Maximum Weighted Cut parameterized by oct, the most important trick is to reduce to a K_5-odd-minor-free graph, and then use known results of Grötschel and Pulleyblank [13] and Guenin [14] to solve the problem in polynomial time (Proposition 6).

2.1.2 XP-algorithms

We now sketch some of the basic ingredients of the XP-algorithms that we present in [17, Section 5] for H(-Induced)-Subgraph/Scattered/Odd-Minor-Packing. The main observation is that, if H is 2-connected and non-bipartite, since the "non-apex" part of each bag is bipartite and H is non-bipartite, in any H-subgraph/induced/scattered/odd-minor-packing and every bag of the decomposition, there are at most btw occurrences of H that intersect that bag. We thus guess these occurrences, and how they intersect the children, which allow us to reduce the number of children by just deleting those not involved in the packing. The guess of these occurrences is the dominant term in the running time of the resulting XP-algorithm using this method. Note that for H(-Induced)-Subgraph/Scattered-Packing, we can indeed easily guess those occurrences in XP-time parameterized by btw, as the total size of the elements of the packing intersecting a given bag is bounded by a function of btw and H. However, for H-Odd-Minor-Packing, this is not the case anymore, as an element of the packing may contain an arbitrary number of vertices in the bipartite part of a bag. We overcome this issue as follows. As stated in [17, Lemma 3.1], the existence of an H-odd-minor is equivalent to the existence of a so-called *odd H-expansion*, which is essentially a collection of trees connected by edges preserving the appropriate parities of the resulting cycles. In an odd H-expansion, the *branch vertices* are those that have degree at least three, or that are incident to edges among different trees. Note that, in an odd H-expansion, the number of branch vertices depends only on H (cf. [17, Lemma 5.2]). Equipped with this property, we first guess, at a given bag, the branch vertices of the packing that intersect that bag. Note that this indeed yields an XP number of choices, as required. Finally, for each such a choice, we use an FPT-algorithm of Kawarabayashi, Reed, and Wollan [22] solving the Parity k-Disjoint Paths to check whether the guessed packing exists or not. This approach is formalized in [17, Lemma 5.3].

It is worth mentioning that, as discussed in Section 4, we leave as an open problem the existence of FPT-algorithms for the above packing problems parameterized by btw.

2.2 Hardness results

Finally, we discuss some of the tools that we use to obtain the para-NP-completeness results summarized in Table 1, which can be found in [17, Section 6]. We present a number of different reductions, some of them consisting of direct simple reductions, such as the one we provide for 3-Coloring in [17, Lemma 6.2].

Except for 3-Coloring, all the considered problems fall into two categories: covering or packing problems. For the first family (cf. [17, Section 6.2]), the para-NP-completeness is an immediate consequence of a result of Yannakakis [32] that characterizes hereditary

graph classes $\mathcal{G}$ for which Vertex Deletion to $\mathcal{G}$ on bipartite graphs is polynomial-time solvable and those for which Vertex Deletion to $\mathcal{G}$ remains NP-complete.

For the packing problems (cf. [17, Section 6.2]), we do not have such a general result as for the covering problems, and we provide several reductions for different problems. For instance, we prove in [17, Lemma 6.3] that if H is a bipartite graph containing P_3 as a subgraph, then H-Subgraph-Packing and H-Induced-Subgraph-Packing are NP-complete on bipartite graphs. The proof consists in a careful analysis and a slight modification of a reduction of Kirkpatrick and Hell [24] for the problem of partitioning the vertex set of an input graph G into subgraphs isomorphic to a fixed graph H. The hypothesis about containing P_3 is easily seen to be tight.

For the minor version, we prove in [17, Lemma 6.4] that if H is a 2-connected graph with at least three vertices, then H-Minor-Packing is NP-complete on bipartite graphs. The proof uses a reduction from P_3-Subgraph-Packing on bipartite graphs, which was proved to be NP-complete by Monnot and Toulouse [26]. The 2-connectivity of H is crucially used in the proof. Given that odd-minors preserve cycle parity (cf. [17, Lemma 3.1]), when H is bipartite, H-Odd-Minor-Packing and H-Minor-Packing are the same problem on bipartite graphs. Hence, the same hardness result holds for H-Odd-Minor-Packing when H is 2-connected and bipartite (cf. [17, Lemma 6.5]).

In [17, Lemma 6.6] we prove that, if H is a 2-connected bipartite graph with at least one edge, then H-Scattered-Packing is NP-complete on bipartite graphs, by a simple reduction from the Induced Matching on bipartite graphs, which is known to be NP-complete [3].

Finally, in [17, Lemma 6.7] we prove that if H is a (non-necessarily bipartite) 2-connected graph containing an edge and $q \in \mathbb{N}_{\geq 2}$, then H-Scattered-Packing is para-NP-complete parameterized by q-$\mathcal{B}$-treewidth. In fact, this reduction is exactly the same as the one when $q = 1$, with the extra observation that, if G' is the graph constructed in the reduction, then the "bipartite" treewidth of G' is at most the one of H for $q \geq 2$.

3 General dynamic programming to obtain FPT-algorithms

In this section, we give introduce a framework for giving FPT-algorithms for problems parameterized by the width of a given bipartite tree decomposition of the input graph. In Subsection 3.1 we provide some preliminary definitions and notations, especially concerning *annotated problems*. Due to space constraints *treewidth*, *boundaried graphs*, and *nice problems* are defined in Appendix A. In Subsection 3.2 we provide a dynamic programming scheme for nice problems, along with some generalizations of this scheme in Subsection 3.3. Finally, we give an application to Maximum Weighted Cut in Subsection 3.4. Applications to K_t-Vertex Cover, Weighted Vertex Cover, and Odd Cycle Transversal are additionally given in [17].

3.1 Preliminaries

Partitions. Given $p \in \mathbb{N}$, a *p-partition* of a set X is a tuple $(X_1, \ldots, X_p)$ of pairwise disjoint subsets of X such that $X = \bigcup_{i \in [p]} X_i$. We denote by $\mathcal{P}_p(X)$ the set of all p-partitions of X. Given a partition $\mathcal{X} \in P_p(X)$, its domain X is also denoted as $\cup\mathcal{X}$. A *partition* is a p-partition for some $p \in \mathbb{N}$. Note that this corresponds to the usual definition of an ordered near-partition, since we allow empty sets in a p-partition and since the order matters. Given $Y \subseteq X$, $\mathcal{X} = (X_1, \ldots, X_p) \in \mathcal{P}_p(X)$, and $\mathcal{Y} = (Y_1, \ldots, Y_p) \in \mathcal{P}_p(Y)$, we say that $\mathcal{Y} \subseteq \mathcal{X}$ if $Y_i \subseteq X_i$ for each $i \in [p]$. Given a set U, two subsets $X, A \subseteq U$, and $\mathcal{X} = (X_1, \ldots, X_p) \in \mathcal{P}_p(X)$, $\mathcal{X} \cap A$ denotes the partition $(X_1 \cap A, \ldots, X_p \cap A)$ of $X \cap A$.

Optimization problems. A *p-partition-evaluation function on graphs* is a function f that receives as input a graph G along with a p-partition $\mathcal{P}$ of its vertices and outputs a non-negative integer. Given such a function f and some choice $\mathsf{opt} \in \{\max, \min\}$ we define the associated graph parameter $\mathsf{p}_{f,\mathsf{opt}}$ where, for every graph G,

$$\mathsf{p}_{f,\mathsf{opt}}(G) = \mathsf{opt}\{f(G,\mathcal{P}) \mid \mathcal{P} \text{ is a } p\text{-partition of } V(G)\}.$$

An *optimization problem* is a problem that can be expressed as follows.

Input: A graph G.
Objective: Compute $\mathsf{p}_{f,\mathsf{opt}}(G)$.

The *annotated extension* of $\mathsf{p}_{f,\mathsf{opt}}$ is the parameter $\hat{\mathsf{p}}_{f,\mathsf{opt}}$ such that

$$\hat{\mathsf{p}}_{f,\mathsf{opt}}(G,\mathcal{X}) = \mathsf{opt}\{f(G,\mathcal{P}) \mid \mathcal{P} \text{ is a } p\text{-partition of } V(G) \text{ with } \mathcal{X} \subseteq \mathcal{P}\}.$$

Observe that $\mathsf{p}_{f,\mathsf{opt}}(G) = \hat{\mathsf{p}}_{f,\mathsf{opt}}(G, \emptyset^p)$, for every graph G. The problem Π' is a *p-annotated extension* of the optimization problem Π if Π can be expressed by some p-partition-evaluation function f and some choice $\mathsf{opt} \in \{\max, \min\}$, and that Π' can be expressed as follows.

Input: A graph G and $\mathcal{X} \in \mathcal{P}_p(X)$ for some $X \subseteq V(G)$.
Objective: Compute $\hat{\mathsf{p}}_{f,\mathsf{opt}}(G,\mathcal{X})$.

We also say that Π' is a *p-annotated problem.*

While our goal in this article is to study bipartite treewidth, defined below, we define a more general parameter, namely 1-$\mathcal{H}$-treewidth, with the hope of it finding some application in future work. We use the term 1-$\mathcal{H}$-treewidth to signify that the "$\mathcal{H}$-part" of each bag intersects each neighboring bag in at most one vertex. This also has the benefit of avoiding confusion with $\mathcal{H}$-treewidth defined in [8], which would be another natural name for this class of parameters.

1-$\mathcal{H}$-tree decompositions. Let $\mathcal{H}$ be a graph class. A *1-$\mathcal{H}$-tree decomposition* is defined exactly like a tree decomposition, but by replacing the class $\mathcal{B}$ of bipartite graphs by $\mathcal{H}$.

3.2 General dynamic programming scheme

We now have all the ingredients for our general scheme dynamic programming algorithm on bipartite tree decompositions. We essentially prove that if a problem Π has an annotated extension that is $\mathcal{B}$-nice and solvable in FPT-time parameterized by oct, then Π is solvable in FPT-time parameterized by btw. This actually holds for more general $\mathcal{H}$.

▶ **Lemma 2.** *Let $p \in \mathbb{N}$. Let $\mathcal{H}$ be a graph class. Let Π be an optimization problem. Let Π' be a problem that is:*

- *a p-annotated extension of Π corresponding to some choice of p-partition-evaluation function g and some $\mathsf{opt} \in \{\max, \min\}$,*
- *$\mathcal{H}$-nice, and*
- *solvable on instances $(G,\mathcal{X})$ such that $G \setminus \cup\mathcal{X} \in \mathcal{H}$ in time $f(|\cup\mathcal{X}|) \cdot n^c \cdot m^d$, for some $c, d \in \mathbb{N}$.*

Then, there is an algorithm that, given a graph G and a 1-$\mathcal{H}$-tree decomposition of G of width k, computes $\mathsf{p}_{f,\mathsf{opt}}(G)$ in time $\mathcal{O}(p^k \cdot f(k+\mathcal{O}(1)) \cdot (k \cdot n)^c \cdot m^d)$ (or $\mathcal{O}(p^k \cdot f(k+\mathcal{O}(1)) \cdot (m+k^2 \cdot n)^d)$ if $c = 0$).

Proof. Let Alg be the algorithm that solves instances $(G, \mathcal{X})$ such that $G \setminus \cup\mathcal{X} \in \mathcal{H}$ in time $f(|\cup\mathcal{X}|) \cdot n^c \cdot m^d$.

Let (T, α, β, r) be a rooted 1-$\mathcal{H}$-tree decomposition of G of width at most k. Let $\sigma : V(G) \to \mathbb{N}$ be an injection. For $t \in V(T)$, let $\mathbf{G}_t = (G_t, \delta_t, \sigma_{|\delta_t})$, let $X_t = \alpha(t) \cup \delta_t \cup \bigcup_{t' \in \mathsf{ch}_r(t)} \delta_{t'}$, let $\mathbf{X}_t = (G[X_t], X_t, \sigma_{|X_t})$, let $\mathbf{H}_t = \mathbf{X}_t \boxplus (\boxplus_{t' \in \mathsf{ch}_r(t)} \mathbf{G}_{t'})$, let $\mathbf{F}_t$ be such that $G_t = \mathbf{F}_t \oplus \mathbf{H}_t$. let $A_t = \alpha(t) \cup \delta_t$, and let $B_t = X_t \setminus A_t = X_t \cap \beta(t) \setminus \delta_t$. Note that $|\mathsf{bd}(\mathbf{G}_{t'}) \setminus A_t| \leq 1$ for $t' \in \mathsf{ch}_r(t)$.

We proceed in a bottom-up manner to compute $s_t^{\mathcal{X}} := \hat{\mathsf{p}}_{g,\mathsf{opt}}(G_t, \mathcal{X})$, for each $t \in V(T)$, for each $\mathcal{X} \in \mathcal{P}_p(\delta_t)$. Hence, given that $\delta_r = \emptyset$, $s_r^{\emptyset} = \mathsf{p}_{g,\mathsf{opt}}(G)$.

Let $t \in V(T)$. By induction, for each $t' \in \mathsf{ch}_r(t)$ and for each $\mathcal{X}_{t'} \in \mathcal{P}_p(\delta_{t'})$, we compute the value $s_{t'}^{\mathcal{X}_{t'}}$. Let $\mathcal{X} \in \mathcal{P}_p(\delta_t)$. Let $\mathcal{Q}$ be the set of all $\mathcal{A} \in \mathcal{P}_p(A_t)$ such that $\mathcal{A} \cap \delta_t = \mathcal{X}$. Let $\mathcal{A} \in \mathcal{Q}$. Since Π' is $\mathcal{H}$-nice, there is an $\mathcal{H}$-nice reduction $(\mathbf{H}_{\mathcal{A}}, \mathcal{A}', s_{\mathcal{A}})$ of $(\mathbf{H}_t, \mathcal{A})$ with respect to Π'. Hence, $\hat{\mathsf{p}}_{g,\mathsf{opt}}(G_t, \mathcal{A}) = \hat{\mathsf{p}}_{g,\mathsf{opt}}(\mathbf{H}_{\mathcal{A}} \triangleright \mathbf{F}_t, \mathcal{A}') + s_{\mathcal{A}}$. Let us compute $\hat{\mathsf{p}}_{g,\mathsf{opt}}(\mathbf{H}_{\mathcal{A}} \triangleright \mathbf{F}_t, \mathcal{A}')$.

By definition of a $\mathcal{H}$-reduction, $(\mathbf{H}_{\mathcal{A}} \triangleright \mathbf{F}_t) \setminus (\cup\mathcal{A}') \in \mathcal{H}$. Hence, we can compute $\hat{\mathsf{p}}_{g,\mathsf{opt}}(\mathbf{H}_{\mathcal{A}} \triangleright \mathbf{F}_t, \mathcal{A}')$, and thus $\hat{\mathsf{p}}_{g,\mathsf{opt}}(G_t, \mathcal{A})$, using Alg on the instance $(\mathbf{H}_{\mathcal{A}} \triangleright \mathbf{F}_t, \mathcal{A}')$. Finally, $s_t^{\mathcal{X}} = \mathsf{opt}_{\mathcal{A} \in \mathcal{Q}} \hat{\mathsf{p}}_{g,\mathsf{opt}}(G_t, \mathcal{A})$.

It remains to calculate the complexity. Throughout, we make use of the fact that p is a fixed constant. We can assume that T has at most n nodes: for any pair of nodes t, t' with $(\alpha \cup \beta)(t) \subseteq (\alpha \cup \beta)(t')$, we can contract the edge tt' of T to a new vertex t'' with $\alpha(t'') = \alpha(t')$ and $\beta(t'') = \beta(t')$. This defines a valid 1-$\mathcal{H}$-tree decomposition of same width. For any leaf t of T, there is a vertex $u \in V(G)$ that only belongs to the bag of t. From this observation, we can inductively associate each node of T to a distinct vertex of G. So this $\mathcal{H}$-tree decomposition has at most n bags. Hence, if $c_t = |\mathsf{ch}_r(t)|$, then we have $\sum_{t \in V(T)} c_t \leq n$. Let also $n_t = |(\alpha \cup \beta)(t)|$ and $m_t = |E(G[(\alpha \cup \beta)(t)])|$. Note that $|A_t| = |\alpha(t)| + |\delta_t \cap \beta(t)| \leq k+1$ and that $|B_t| = |\bigcup_{t' \in V(T)} \delta_{t'} \cap \beta(t)| \leq c_t$, so $|X_t| \leq k + 1 + c_t$. Moreover, the properties of the tree decompositions imply that the vertices in $\beta(t) \setminus X_t$ are only present in node t. Then, $\sum_{t \in V(T)} n_t = \sum_{t \in V(T)} (|X_t| + |\beta(t) \setminus X_t|) = \mathcal{O}(k \cdot n)$. Also, let $\bar{m}_t$ be the number of edges only present in the bag of node t. The edges that are present in several bags are those in the adhesion of t and its neighbors. t is adjacent to its $|c_t|$ children and its parent, and an adhesion has size at most $k+1$. Thus, $\sum_{t \in V(T)} m_t \leq \sum_{t \in V(T)} (\bar{m}_t + k^2(1 + c_t)) = \mathcal{O}(m + k^2 \cdot n)$.

There are $p^{|A_t|} \leq p^{k+1} = \mathcal{O}(p^k)$ partitions of $\mathcal{P}_p(A_t)$. For each of them, we compute in time $\mathcal{O}(k \cdot c_t)$ a $\mathcal{H}$-nice reduction $(\mathbf{H}_{\mathcal{A}}, \mathcal{A}', s_{\mathcal{A}})$ with $|\cup\mathcal{A}'| = |A_t| + \mathcal{O}(1) = k + \mathcal{O}(1)$ and with $\mathcal{O}(|B_t|) = \mathcal{O}(c_t)$ additional vertices and edges. We thus solve Π' on $(\mathbf{H}_{\mathcal{A}} \triangleright \mathbf{F}_t, \mathcal{A}')$ in time $f(k+\mathcal{O}(1)) \cdot \mathcal{O}((n_t + c_t)^c \cdot (m_t + c_t)^d)$. Hence, the running time is $\mathcal{O}(p^k \cdot f(k+\mathcal{O}(1)) \cdot (k \cdot n)^c \cdot m^d)$ (or $\mathcal{O}(p^k \cdot f(k + \mathcal{O}(1)) \cdot (m + k^2 \cdot n)^d)$ if $c = 0$). ◀

3.3 Generalizations

For the sake of simplicity, we assumed in Lemma 2 that the problem Π under consideration takes as input just a graph. However, a similar statement still holds if we add labels/weights on the vertices/edges of the input graph. This is in particular the case for Maximum Weighted Cut (Subsection 3.4) and Weighted Independent Set where the vertices or edges are weighted.

Moreover, again for the sake of simplicity, we assumed that Π' is solvable in FPT-time, while other complexities such as XP-time could be considered. Similarly, in the definition of the nice reduction, the contraints $|A'| = |A| + \mathcal{O}(1)$, $|V(G')| \leq |X| + \mathcal{O}(|B|)$, $|E(G')| \leq |E(G[X])| + \mathcal{O}(|B|)$ can be modified. In both cases, the dynamic programming algorithm still holds, but the running time of Lemma 2 changes.

To give a precise running time for Maximum Weighted Cut (Subsection 3.4), K_t-Subgraph-Cover, and Weighted Independent Set, let us observe that, if Π' is solvable in time $f(|\cup\mathcal{X}|)\cdot n'^c\cdot m'^d$, where $G'=G\setminus\cup\mathcal{X}$, $n'=|V(G')|$, and $m'=|E(G')|$, then the running time of Lemma 2 is better. Indeed, in the proof of the complexity of Lemma 2, we now solve Π' on $(\mathbf{H}_{\mathcal{A}}\triangleright\mathbf{F},\mathcal{A}')$ in time $f(k+\mathcal{O}(1))\cdot\mathcal{O}((n'_t+c_t)^c\cdot(m'_t+c_t)^d)$, where $n'_t=|\beta(t)|$ and $m'_t=|E(G[\beta(t)])|$. We have $\sum_{t\in V(T)}n'_t=\sum_{t\in V(T)}(|B|+|\beta(t)\cap\delta_t|+|\beta(t)\setminus X|)=\mathcal{O}(n)$ and $\sum_{t\in V(T)}m'_t\leq m$. Hence, the total running time is $\mathcal{O}(p^k\cdot(k\cdot n+f(k+\mathcal{O}(1))\cdot n^c\cdot m^d))$.

3.4 Application to Maximum Cut

We now apply the above framework to give an FPT-algorithm for Maximum Weighted Cut parameterized by bipartite treewidth. Thanks to Lemma 2, this now reverts to showing that the problem under consideration has an $\mathcal{B}$-nice annotated extension that is solvable in FPT time when parameterized by oct, where $\mathcal{B}$ is the class of bipartite graphs.

The Maximum Weighted Cut problem is defined as follows.

Maximum Weighted Cut
Input: A graph G and a weight function $w:E(G)\to\mathbb{N}$.
Objective: Find an edge cut of maximum weight.

Let H be a graph. We define f_{cut} as the 2-partition-evaluation function where, for every graph G with edge weight w and for every $\mathcal{P}=(X_1,X_2)\in\mathcal{P}_2(V(G))$,

$$f_{\mathsf{cut}}(G,\mathcal{P})=w(\mathcal{P})=w(E(X_1,X_2)).$$

Hence, Maximum Weighted Cut is the problem of computing $\mathsf{p}_{f_{\mathsf{cut}},\max}(G)$. We call its annotated extension Annotated Maximum Weighted Cut. In other words, Annotated Maximum Weighted Cut is defined as follows.

Annotated Maximum Weighted Cut
Input: A graph G, a weight function $w:E(G)\to\mathbb{N}$, and two disjoint sets $X_1,X_2\subseteq V(G)$.
Objective: Find an edge cut of maximum weight such that the vertices in X_1 belongs to one side of the cut, and the vertices in X_2 belong to the other side.

The following property seems critical to show that a problem is $\mathcal{H}$-nice.

Gluing property. Let Π be a p-annotated problem corresponding to some choice of p-partition-evaluation function f and some $\mathsf{opt}\in\{\max,\min\}$. We say that Π has the *gluing property* if, given two compatible boundaried graphs $\mathbf{F}$ and $\mathbf{G}$ with boundary X, $\mathcal{X}\in\mathcal{P}_p(X)$, and $\mathcal{P}\in\mathcal{P}_p(V(\mathbf{F}\oplus\mathbf{G}))$ such that $\mathcal{X}\subseteq\mathcal{P}$, then $\hat{\mathsf{p}}_{f,\mathsf{opt}}(\mathbf{F}\oplus\mathbf{G},\mathcal{X})=f(\mathbf{F}\oplus\mathbf{G},\mathcal{P})$ if and only if $\hat{\mathsf{p}}_{f,\mathsf{opt}}(F,\mathcal{X})=f(F,\mathcal{P}\cap V(F))$ and $\hat{\mathsf{p}}_{f,\mathsf{opt}}(G,\mathcal{X})=f(G,\mathcal{P}\cap V(G))$.

We first prove that Annotated Maximum Weighted Cut has the gluing property.

▶ **Lemma 3** (Gluing property). *Annotated Maximum Weighted Cut has the gluing property. More precisely, given two boundaried graphs $\mathbf{F}=(F,B_F,\rho_F)$ and $\mathbf{G}=(G,B_G,\rho_G)$, a weight function $w:E(\mathbf{F}\oplus\mathbf{G})\to\mathbb{N}$, a set $X\subseteq V(\mathbf{F}\oplus\mathbf{G})$ such that $B_F\cap B_G\subseteq X$, and $\mathcal{X}=(X_1,X_2)\in\mathcal{P}_2(X)$, if we set $\bar{w}=w(\mathcal{X}\cap B_F\cap B_G)$, then we have*

$$\hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(\mathbf{F}\oplus\mathbf{G},\mathcal{X},w)=\hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(F,\mathcal{X}\cap V(F),w)+\hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(G,\mathcal{X}\cap V(G),w)-\bar{w}.$$

Proof. Let $\mathcal{P} \in \mathcal{P}_2(V(\mathbf{F} \oplus \mathbf{G}))$ be such that $\mathcal{X} \subseteq \mathcal{P}$ and $\hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(\mathbf{F} \oplus \mathbf{G}, \mathcal{X}, w) = f_{\mathsf{cut}}(\mathbf{F} \oplus \mathbf{G}, \mathcal{P}, w)$. Then,

$$\begin{aligned}\hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(\mathbf{F} \oplus \mathbf{G}, \mathcal{X}, w) &= w(\mathcal{P}) \\ &= w(\mathcal{P} \cap V(F)) + w(\mathcal{P} \cap V(G)) - \bar{w} \\ &\leq \hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(F, \mathcal{X} \cap V(F), w) + \hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(G, \mathcal{X} \cap V(G), w) - \bar{w}.\end{aligned}$$

Reciprocally, for $H \in \{F, G\}$, let $\mathcal{P}_H = (X_1^H, X_2^H) \in \mathcal{P}_2(V(H))$ be such that $\mathcal{X} \cap V(H) \subseteq \mathcal{P}_H$ and $\hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(H, \mathcal{X} \cap V(H), w) = f_{\mathsf{oct}}(H, \mathcal{P}_H, w)$. Then, since $\mathcal{P}_H \cap B_F \cap B_G = \mathcal{X} \cap B_F \cap B_G$ for $H \in \{F, G\}$, we have

$$\begin{aligned}\hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(\mathbf{F} \oplus \mathbf{G}, \mathcal{X}, w) &\geq w(E(X_1^F \cup X_1^G, X_2^F \cup X_2^G)) \\ &= w(E(X_1^F, X_2^F)) + w(E(X_1^G, X_2^G)) - \bar{w} \\ &= \hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(F, \mathcal{X} \cap V(F), w) + \hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(G, \mathcal{X} \cap V(G), w) - \bar{w}.\end{aligned}$$

◀

We now show how to reduce a graph $\mathbf{F} \oplus \mathbf{G}$ to a graph F' when the boundary of $\mathbf{F}$ and $\mathbf{G}$ has a single vertex v that is not annotated.

▶ **Lemma 4** (Gadgetization). *Let $\mathbf{F} = (F, B_F, \rho_F)$ and $\mathbf{G} = (G, B_G, \rho_G)$ be two boundaried graphs, let $w : E(\mathbf{F} \oplus \mathbf{G}) \to \mathbb{N}$ be a weight function, let $X \subseteq V(\mathbf{F} \oplus \mathbf{G})$ be such that $B_F \cap B_G \subseteq X$, let $v \in B_F \cap B_G$, and let $\mathcal{X} = (X_1, X_2) \in \mathcal{P}_2(X \setminus \{v\})$. Suppose that there is $v_1 \in X_1$ and $v_2 \in X_2$ adjacent to v with $w(vv_1) = w(vv_2) = 0$. Let $\mathcal{X}^1 = (X_1 \cup \{v\}, X_2)$ and $\mathcal{X}^2 = (X_1, X_2 \cup \{v\})$. For $a \in [2]$, let $g_a = \hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(G, \mathcal{X}^a \cap V(G), w)$. Let $\bar{w} = w(\mathcal{X} \cap B_F \cap B_G)$. Let $w' : E(F) \to \mathbb{N}$ be such that $w'(vv_1) = g_2 - \bar{w}$, $w'(vv_2) = g_1 - \bar{w}$, and $w'(e) = w(e)$ otherwise. Then*

$$\hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(\mathbf{F} \oplus \mathbf{G}, \mathcal{X}, w) = \hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(F, \mathcal{X}, w').$$

Proof. For $a \in [2]$, let $f_a = \hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(F, \mathcal{X}^a \cap V(F), w)$. Note that in F with partition $\mathcal{X}$, if v is on the same side as X_1, then we must count the weight of the edge vv_2, but not the weight of vv_1, and vice versa when exchanging 1 and 2. Thus, using Lemma 3, we have

$$\begin{aligned}\hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(\mathbf{F} \oplus \mathbf{G}, \mathcal{X}, w) &= \max\{\hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(\mathbf{F} \oplus \mathbf{G}, \mathcal{X}^1, w), \hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(\mathbf{F} \oplus \mathbf{G}, \mathcal{X}^2, w)\} \\ &= \max\{f_1 + g_1 - \bar{w}, f_2 + g_2 - \bar{w}\} \\ &= \max\{f_1 + w'(vv_2), f_2 + w'(vv_1)\} \\ &= \max\{\hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(F', \mathcal{X}^1, w'), \hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(F, \mathcal{X}^2, w')\} \\ &= \hat{\mathsf{p}}_{f_{\mathsf{cut}},\max}(F, \mathcal{X}, w').\end{aligned}$$

◀

Using Lemma 3 and Lemma 4, we can prove that Annotated Maximum Weighted Cut is $\mathcal{H}$-nice. Essentially, given an instance $(\mathbf{G} = \mathbf{X} \boxplus (\boxplus_{i \in [d]} \mathbf{G}_i), (A, B), \mathcal{A}, w)$, we reduce $\mathbf{G}$ to $\mathbf{X}$ where we add two new vertices in $\mathcal{A}$ and add every edges between this new vertices and the vertices in B. We then show that if the appropriate weight is given to each new edge, then the resulting boundaried graph is equivalent to $\mathbf{G}$ modulo some constant s.

▶ **Lemma 5** (Nice problem). *Let $\mathcal{H}$ be a graph class.* Annotated Maximum Weighted Cut *is $\mathcal{H}$-nice.*

Maximum Weighted Cut is a NP-hard problem [20]. However, there exists a polynomial-time algorithm when restricted to some graph classes. In particular, Grötschel and Pulleyblank [13] proved that Maximum Weighted Cut is solvable in polynomial-time on weakly bipartite graphs, and Guenin [14] proved that weakly bipartite graphs are exactly K_5-odd-minor-free graphs, which gives the following result.

▶ **Proposition 6** ([13,14]). *There is a constant $c \in \mathbb{N}$ and an algorithm that solves* Maximum Weighted Cut *on K_5-odd-minor-free graphs in time $\mathcal{O}(n^c)$.*

Moreover, we observe the following.

▶ **Lemma 7.** *A graph G such that $\mathsf{oct}(G) \leq 2$ does not contain K_5 as an odd-minor.*

Proof. Let $u, v \in V(G)$ be such that $G' = G \setminus \{u, v\}$ is bipartite. G' does not contain K_3 as an odd-minor, so G does not contain K_5 as an odd-minor. ◀

Combining Proposition 6 and Lemma 7, we have that Annotated Maximum Weighted Cut is FPT parameterized by oct.

▶ **Lemma 8.** *There is an algorithm that, given a graph G, a weight function $w : E(G) \to \mathbb{N}$, and two disjoint sets $X_1, X_2 \subseteq V(G)$, such that $G' = G \setminus (X_1 \cup X_2)$ is bipartite, solves* Annotated Maximum Weighted Cut *on (G, X_1, X_2, w) in time $\mathcal{O}(k \cdot n' + n'^c)$, where $k = |X_1 \cup X_2|$ and $n' = |V(G')|$.*

Proof. Let G'' be the graph obtained from G by identifying all vertices in X_1 (resp. X_2) to a new vertex x_1 (resp. x_2). Let $w' : V(G'') \to \mathbb{N}$ be such that $w'(x_1x_2) = \sum_{e \in E(G)} w(e) + 1$, $w'(x_iu) = \sum_{x \in X_i} w(xu)$ for $i \in [2]$ and $u \in N_G(X_i)$, and $w'(e) = w(e)$ otherwise. Let $(X_1^\star, X_2^\star) \in \mathcal{P}_2(V(G))$ be such that $(X_1, X_2) \subseteq (X_1^\star, X_2^\star)$. For $i \in [2]$, let $X_i' = X_i^\star \setminus X_i$. Then

$$\begin{aligned} w(X_1^\star, X_2^\star) &= w(X_1, X_2) + w(X_1', X_2') + \sum_{xy \in E(X_1, X_2')} w(xy) + \sum_{xy \in E(X_1', X_2)} w(xy) \\ &= w(X_1, X_2) + w'(X_1', X_2') + \sum_{u \in X_2 \cap N_G(X_1)} w'(x_1u) + \sum_{u \in X_1 \cap N_G(X_2)} w'(x_2u) \\ &= w'(X_1' \cup \{x_1\}, X_2' \cup \{x_2\}) + w(X_1, X_2) - w'(x_1x_2) \end{aligned}$$

Let $\bar{w}$ be the contant $w(X_1, X_2) - w'(x_1x_2)$. Hence, $f_{\mathsf{cut}}(G, (X_1^\star, X_2^\star)) = f_{\mathsf{cut}}(G'', (X_1' \cup \{x_1\}, X_2' \cup \{x_2\})) + \bar{w}$, and so $\hat{p}_{f_{\mathsf{cut}},\max}(G, (X_1, X_2)) = \hat{p}_{f_{\mathsf{cut}},\max}(G'', (\{x_1\}, \{x_2\})) + \bar{w}$. Furthermore, given that the weight of the edge x_1x_2 is larger than the sum of all other weights, x_1 and x_2 are never on the same side of a maximum cut in G''. Hence, $\hat{p}_{f_{\mathsf{cut}},\max}(G'', (\{x_1\}, \{x_2\})) = \mathsf{p}_{f_{\mathsf{cut}},\max}(G'')$, and therefore, $\hat{p}_{f_{\mathsf{cut}},\max}(G, (X_1, X_2)) = p_{f_{\mathsf{cut}},\max}(G'') + \bar{w}$.

Constructing G'' takes time $\mathcal{O}(k \cdot n)$ and computing $\bar{w}$ takes time $\mathcal{O}(k^2)$. Since $\mathsf{oct}(G'') = 2$, according to Proposition 6 and Lemma 7, an optimal solution to Maximum Weighted Cut on G'' can be found in time $\mathcal{O}(n'^c)$, and thus, an optimal solution to Annotated Maximum Weighted Cut on (G, X_1, X_2) can be found in time $\mathcal{O}(k \cdot (k + n') + n'^c)$. ◀

We apply Lemma 5 and Lemma 8 to the dynamic programming algorithm of Lemma 2 to obtain the following result.

▶ **Corollary 9.** *Given a graph G and a bipartite tree decomposition of G of width k, there is an algorithm that solves* Maximum Weighted Cut *on G in time $\mathcal{O}(2^k \cdot (k \cdot (k + n) + n^c))$.*

4 Further research

In this paper we study the complexity of several problems parameterized by bipartite treewidth, denoted by btw. In particular, our results extend the graph classes for which Vertex Cover/Independent Set, Maximum Weighted Cut, and Odd Cycle Transversal are polynomial-time solvable. A number of interesting questions remain open.

Except for 3-Coloring, all the problems we consider are covering and packing problems. We are still far from a full classification of the variants that are para-NP-complete, and those that are not (FPT or XP). For instance, concerning H-Subgraph-Cover, we provided an FPT-algorithms when H is a clique. This case is particularly well-behaved because we know that in a tree decomposition every clique appears in some bag. On the other hand, as an immediate consequence of the result of Yannakakis [32], we know that H-Subgraph-Cover is para-NP-complete for every bipartite graph H containing P_3. We do not know what happens when H is not bipartite nor a clique. An apparently simple but challenging case is C_5-Subgraph-Cover. The main difficulty seems to be that C_5-Subgraph-Cover does not have the gluing property, which is the main ingredient in this paper to show that a problem is nice, and therefore to obtain an FPT-algorithm. We do not exclude the possibility that the problem is para-NP-complete, as we were not even able to obtain even an XP algorithm.

Concerning the packing problems, namely H-Subgraph/Induced/Scattered/Odd-Minor-Packing, we provide XP-algorithms for them when H is non-bipartite. Unfortunately, we do not know whether any of them admits an FPT-algorithm, although we suspect that it is indeed the case. We would like to mention that it is possible to apply the framework of equivalence relations and representatives (see for instance [2, 9, 10]) to obtain an FPT-algorithm for K_t-Subgraph-Packing parameterized by btw. However, since a number of definitions and technical details are required to present this algorithm, we decided not to include it in this paper (which is already quite long). However, when H is not a clique, we do not know whether H-Subgraph-Packing admits an FPT-algorithm. A concrete case that we do not know how to solve is when H is the *paw*, i.e., the 4-vertex graph consisting of one triangle and one pendent edge.

Beyond bipartite tree decompositions, we introduce a more general type of decompositions that we call q(-torso)-$\mathcal{H}$-tree decompositions. For $\mathcal{B}$ being the class of bipartite graphs, we prove that for every $q \geq 2$ and every 2-connected graph H with an edge, H-Scattered-Packing is para-NP-complete parameterized by q(-torso)-$\mathcal{B}$-treewidth. It should be possible to prove similar results for other covering and packing problems considered in this article.

Most of our para-NP-completeness results consist just in proving NP-completeness on bipartite graph. There are two exceptions. On the one hand, the NP-completeness of 3-Coloring on graphs with odd cycle transversal at most three and H-Scattered-Packing parameterized by q-$\mathcal{B}$-treewidth for every integer $q \geq 2$. Interestingly, none of our hardness results really exploits the structure of bipartite tree decompositions (i.e., for $q = 1$), beyond being bipartite or having bounded odd cycle transversal.

Finally, as mentioned in the introduction, the goal of this article is to make a first step toward efficient algorithms to solve problems related to odd-minors. We already show in this paper that bipartite treewidth can be useful in this direction, by providing an XP-algorithm for H-Odd-Minor-Packing. Bipartite treewidth, or strongly related notions, also plays a strong role in the recent series of papers about odd-minors by Campbell, Gollin, Hendrey, and Wiederrecht [4, 12]. This looks like an emerging topic that is worth investigating.

Acknowledgments. We thank Sebastian Wiederrecht and the reviewers for helpful remarks.

References

1 Akanksha Agrawal, Lawqueen Kanesh, Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Deleting, eliminating and decomposing to hereditary classes are all fpt-equivalent. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 1976–2004. SIAM, 2022. URL: https://doi.org/10.1137/1.9781611977073.79, doi:10.1137/1.9781611977073.79.

2 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. A complexity dichotomy for hitting connected minors on bounded treewidth graphs: the chair and the banner draw the boundary. In *Proc. of the 31st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 951–970, 2020. doi:10.1137/1.9781611975994.57.

3 Kathie Cameron. Induced matchings. *Discrete Applied Mathematics*, 24(1-3):97–102, 1989. doi:10.1016/0166-218X(92)90275-F.

4 Rutger Campbell, J. Pascal Gollin, Kevin Hendrey, and Sebastian Wiederrecht. Odd-Minors II: Bipartite treewidth. Manuscript under preparation (private communication), 2023.

5 Julia Chuzhoy and Zihan Tan. Towards tight(er) bounds for the Excluded Grid Theorem. *Journal of Combinatorial Theory, Series B*, 146:219–265, 2021. doi:10.1016/j.jctb.2020.09.010.

6 Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Decomposition, approximation, and coloring of odd-minor-free graphs. In *Proc. of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 329–344. SIAM, 2010. doi:10.1137/1.9781611973075.28.

7 Reinhard Diestel. *Graph Theory*, volume 173. Springer-Verlag, 5th edition, 2017. doi:10.1007/978-3-662-53622-3.

8 Eduard Eiben, Robert Ganian, Thekla Hamm, and O-joung Kwon. Measuring what matters: A hybrid approach to dynamic programming with treewidth. *Journal of Computer and System Sciences*, 121:57–75, 2021. doi:10.1016/j.jcss.2021.04.005.

9 Valentin Garnero, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. Explicit linear kernels via dynamic programming. *SIAM Journal on Discrete Mathematics*, 29(4):1864–1894, 2015. doi:10.1137/140968975.

10 Valentin Garnero, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. Explicit linear kernels for packing problems. *Algorithmica*, 81(4):1615–1656, 2019. doi:10.1007/s00453-018-0495-5.

11 Jim Geelen, Bert Gerards, Bruce A. Reed, Paul D. Seymour, and Adrian Vetta. On the odd-minor variant of Hadwiger's conjecture. *Journal of Combinatorial Theory, Series B*, 99(1):20–29, 2009. doi:10.1016/j.jctb.2008.03.006.

12 J. Pascal Gollin and Sebastian Wiederrecht. Odd-Minors I: Excluding small parity breaks. *CoRR*, abs/2304.04504, 2023. arXiv:2304.04504.

13 Martin Grötschel and William R. Pulleyblank. Weakly bipartite graphs and the max-cut problem. *Operations Research Letters*, 1(1):23–27, 1981. doi:10.1016/0167-6377(81)90020-1.

14 Bertrand Guenin. A characterization of weakly bipartite graphs. *Journal of Combinatorial Theory, Series B*, 83(1):112–168, 2001. doi:10.1006/jctb.2001.2051.

15 Hugo Hadwiger. Über eine klassifikation der streckenkomplexe. *Vierteljschr. Naturforsch. Ges. Zürich*, 88(2):133–142, 1943. URL: https://www.ngzh.ch/archiv/1943_88/88_2/88_17.pdf.

16 Huynh, Tony. *The Linkage Problem for Group-labelled Graphs*. PhD thesis, University of Waterloo, 2009. URL: http://hdl.handle.net/10012/4716.

17 Lars Jaffke, Laure Morelle, Ignasi Sau, and Dimitrios M. Thilikos. Dynamic programming on bipartite tree decompositions. *CoRR*, abs/2309.07754, 2023. arXiv:2309.07754, doi:10.48550/arXiv.2309.07754.

18 Bart M. P. Jansen and Jari J. H. de Kroon. FPT algorithms to compute the elimination distance to bipartite graphs and more. In *Proc. of the 47th International Workshop on Graph-*

Theoretic Concepts in Computer Science (WG), volume 12911 of *LNCS*, pages 80–93, 2021. doi:10.1007/978-3-030-86838-3_6.

19 Tommy R Jensen and Bjarne Toft. *Graph coloring problems*. Wiley, 2011. doi:10.1002/9781118032497.

20 Richard M. Karp. Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 219–241. Springer, 2010. doi:10.1007/978-3-540-68279-0_8.

21 Ken-ichi Kawarabayashi and Bruce A. Reed. An (almost) linear time algorithm for odd cyles transversal. In *Proc. of the21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 365–378. SIAM, 2010. doi:10.1137/1.9781611973075.31.

22 Ken-ichi Kawarabayashi, Bruce A. Reed, and Paul Wollan. The graph minor algorithm with parity conditions. In *Proc. of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 27–36. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.52.

23 Valerie King, S. Rao, and Robert Endre Tarjan. A faster deterministic maximum flow algorithm. *Journal of Algorithms*, 17(3):447–474, 1994. doi:10.1006/jagm.1994.1044.

24 David G. Kirkpatrick and Pavol Hell. On the complexity of general graph factor problems. *SIAM Journal on Computing*, 12(3):601–609, 1983. doi:10.1137/0212040.

25 Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Transactions on Algorithms*, 11(2):15:1–15:31, 2014. doi:10.1145/2566616.

26 Jérôme Monnot and Sophie Toulouse. The path partition problem and related problems in bipartite graphs. *Operations Research Letter*, 35(5):677–684, 2007. doi:10.1016/j.orl.2006.12.004.

27 James B. Orlin. Max flows in $O(nm)$ time, or better. In *Proc. of the 45th annual ACM Symposium on Theory of Computing Conference (STOC)*, pages 765–774. ACM, 2013. doi:10.1145/2488608.2488705.

28 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004. doi:10.1016/j.orl.2003.10.009.

29 Neil Robertson, Paul D. Seymour, and Robin Thomas. Quickly excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 62(2):323–348, 1994. doi:10.1006/jctb.1994.1073.

30 Raphael Steiner. Improved bound for improper colourings of graphs with no odd clique minor. *Combinatorics, Probability and Computing*, 32(2):326–333, 2023. doi:10.1017/S0963548322000268.

31 Siamak Tazari. Faster approximation schemes and parameterized algorithms on (odd-)h-minor-free graphs. *Theoretical Computer Science*, 417:95–107, 2012. doi:10.1016/j.tcs.2011.09.014.

32 Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM Journal on Computing*, 10(2):310–327, 1981. doi:10.1137/0210022.

A Graphs, treewidth, boundaried graphs, and nice problems

Functions. Given two sets A and B, and two functions $f, g : A \to 2^B$, we denote by $f \cup g$ the function that maps $x \in A$ to $f(x) \cup g(x) \in 2^B$. Let $f : A \to B$ be an injection. Let $K \subseteq B$ be the image of f. By convention, if f is referred to as a bijection, it means that we consider that f maps A to K. Given a function $w : A \to \mathbb{N}$, and $A' \subseteq A$, $w(A') = \sum_{x \in A'} w(x)$.

Basic concepts on graphs. All graphs considered in this paper are undirected, finite, and without loops or multiple edges. We use standard graph-theoretic notation and we refer the reader to [7] for any undefined terminology. For convenience, we use uv instead of $\{u, v\}$ to denote an edge of a graph. Let G be a graph. In the rest of this paper we always use

n for the cardinality of $V(G)$, and m for the cardinality of $E(G)$, where G is the input graph of the problem under consideration. For $S \subseteq V(G)$, we set $G[S] = (S, E \cap \binom{S}{2})$ and use the shortcut $G \setminus S$ to denote $G[V(G) \setminus S]$. Given a vertex $v \in V(G)$, we denote by $N_G(v)$ the set of vertices of G that are adjacent to v in G. Moreover, given a set $A \subseteq V(G)$, $N_G(A) = \bigcup_{v \in A} N_G(v) \setminus A$. For $k \in \mathbb{N}$, we denote by P_k the path with k vertices, and we say that P_k has length $k-1$ (i.e., the *length* of a path is its number of edges). We denote by $\mathsf{cc}(G)$ the set of connected components of a graph G. For $A, B \subseteq V(G)$, $E(A,B)$ denotes the set of edges of G with one endpoint in A and the other in B. We say that $E' \subseteq E(G)$ is an *edge cut* of G if there is a partition (A,B) of $V(G)$ such that $E' = E(A,B)$. We say that a pair $(L,R) \in 2^{V(G)} \times 2^{V(G)}$ is a *separation* of G if $L \cup R = V(G)$ and $E(L \setminus R, R \setminus L) = \emptyset$. The *order* of (L,R) is $|L \cap R|$. $L \cap G$ is called a $|L \cap R|$-*separator* of G. A graph G is k-*connected* if, for any separation (L,R) of G of order at most $k-1$, either $L \subseteq R$ or $R \subseteq L$. A graph class $\mathcal{H}$ is *hereditary* if for any $G \in \mathcal{H}$ and $v \in V(G)$, $G \setminus \{v\} \in \mathcal{H}$.

Treewidth. A *tree decomposition* of a graph G is a pair (T, χ) where T is a tree and $\chi : V(T) \to 2^{V(G)}$ such that

- $\bigcup_{t \in V(T)} \chi(t) = V(G)$,
- for every $e \in E(G)$, there is a $t \in V(T)$ such that $\chi(t)$ contains both endpoints of e, and
- for every $v \in V(G)$, the subgraph of T induced by $\{t \in V(T) \mid v \in \chi(t)\}$ is connected.

The *width* of (T, χ) is equal to $\max\left\{ |\chi(t)| - 1 \;\middle|\; t \in V(T) \right\}$ and the *treewidth* of G, denoted by $\mathsf{tw}(G)$, is the minimum width over all tree decompositions of G.

For every node $t \in V(T)$, $\chi(t)$ is called *bag* of t. Given $tt' \in E(T)$, the *adhesion* of t and t', denoted by $\mathsf{adh}(t,t')$, is the set $\chi(t) \cap \chi(t')$.

A *rooted tree decomposition* is a triple (T, χ, r) where (T, χ) is a tree decomposition and (T, r) is a *rooted tree* (i.e., T is a tree and $r \in V(T)$). Given $t \in V(T)$, we denote by $\mathsf{ch}_r(t)$ the set of children of t and by $\mathsf{par}_r(t)$ the parent of t (if $t \neq r$). We set $\delta_t^r = \mathsf{adh}(t, \mathsf{par}_r(t))$, with the convention that $\delta_r^r = \emptyset$. Moreover, we denote by G_t^r the graph induced by $\bigcup_{t' \in V(T_t)} \chi(t')$ where (T_t, t) is the rooted subtree of (T, r). We may use δ_t and G_t instead of δ_t^r and G_t^r when there is no risk of confusion.

Boundaried graphs. Let $t \in \mathbb{N}$. A t-*boundaried graph* is a triple $\mathbf{G} = (G, B, \rho)$ where G is a graph, $B \subseteq V(G)$, $|B| = t$, and $\rho : B \to \mathbb{N}$ is an injection. We say that B is the *boundary* of $\mathbf{G}$ and we write $B = \mathsf{bd}(G)$. We call $\mathbf{G}$ *trivial* if all its vertices belong to the boundary. We say that two t-boundaried graphs $\mathbf{G}_1 = (G_1, B_1, \rho_1)$ and $\mathbf{G}_2 = (G_2, B_2, \rho_2)$ are *isomorphic* if $\rho_1(B_1) = \rho_2(B_2)$ and there is an isomorphism from G_1 to G_2 that extends the bijection $\rho_2^{-1} \circ \rho_1$. A triple (G, B, ρ) is a *boundaried graph* if it is a t-boundaried graph for some $t \in \mathbb{N}$. We denote by $\mathcal{B}^t$ the set of all (pairwise non-isomorphic) t-boundaried graphs. A boundaried graph $\mathbf{F}$ is a *boundaried induced subgraph* (resp. *boundaried subgraph*) of $\mathbf{G}$ if $\mathbf{F}$ can be obtained from $\mathbf{G}$ by removing vertices (resp. and edges). A boundaried graph $\mathbf{F}$ is a *boundaried odd-minor* of $\mathbf{G}$ if $\mathbf{F}$ can be obtained from a bounderied subgraph $\mathbf{G}'$ of $\mathbf{G}$ by contracting an edge cut such that every vertex in $\mathsf{bd}(\mathbf{G}')$ is on the same side of the cut. We say that two boundaried graphs $\mathbf{G}_1 = (G_1, B_1, \rho_1)$ and $\mathbf{G}_2 = (G_2, B_2, \rho_2)$ are *compatible* if $\rho_1(B_1) = \rho_2(B_2)$ and $\rho_2^{-1} \circ \rho_1$ is an isomorphism from $G_1[B_1]$ to $G_2[B_2]$. Given two boundaried graphs $\mathbf{G}_1 = (G_1, B_1, \rho_1)$ and $\mathbf{G}_2 = (G_2, B_2, \rho_2)$, we define $\mathbf{G}_1 \oplus \mathbf{G}_2$ as the unboundaried graph obtained if we take the disjoint union of G_1 and G_2 and, for every $i \in \rho_1(B_1) \cap \rho_2(B_2)$, we identify vertices $\rho_1^{-1}(i)$ and $\rho_2^{-1}(i)$. If v is the result of the identification of $v_1 := \rho_1^{-1}(i)$ and $v_2 := \rho_2^{-1}(i)$ then we say that v is the *heir* of v_i from $\mathbf{G}_i, i \in [2]$. If v is either a vertex of G_1 where $\rho_1(v) \notin \rho_1(B_1) \cap \rho_2(B_2)$ (if $v \in B_1$) or a vertex

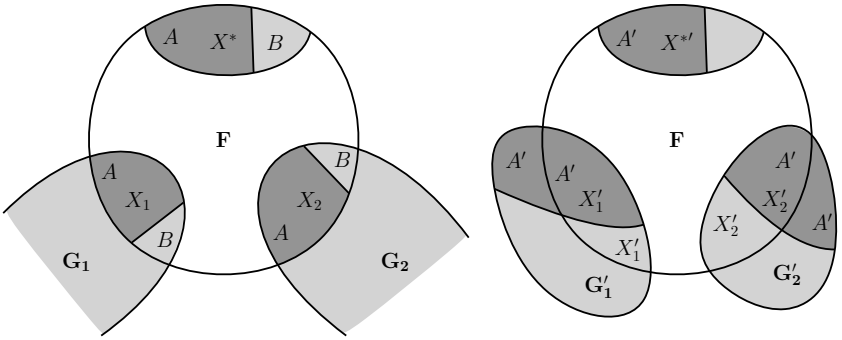

Figure 1 Illustration of the setting of the nice problem and reduction. The shaded area on the left is $\mathbf{G}$ where $X = X_1 \cup X_2 \cup X^\star$, and the shaded area on the right is $\mathbf{G}'$ where $X' = X'_1 \cup X'_2 \cup X^{\star\prime}$.

of G_2 where $\rho_2(v) \notin \rho_1(B_1) \cap \rho_2(B_2)$ (if $v \in B_2$), then v is also a (non-identified) vertex of $\mathbf{G}_1 \oplus \mathbf{G}_2$ and is a *heir* of itself (from $\mathbf{G}_1$ or $\mathbf{G}_2$ respectively). For $i \in [2]$, and given an edge vu in $\mathbf{G}_1 \oplus \mathbf{G}_2$, we say that vu is the *heir* of an edge $v'u'$ from $\mathbf{G}_i$ if v' (resp. u') is the heir of v (resp. u) from $\mathbf{G_i}$ and $v'u'$ is an edge of G_i. If x' is an heir of x from $\mathbf{G} = (G, B, \rho)$ in $\mathbf{G}'$, then we write $x = \mathsf{heir}_{\mathbf{G},\mathbf{G}'}(x')$. If $B' \subseteq B$, then $\mathsf{heir}_{\mathbf{G},\mathbf{G}'}(B) = \bigcup_{v \in B'} \mathsf{heir}_{\mathbf{G},\mathbf{G}'}(x')$. We also define $\mathbf{G}_1 \boxplus \mathbf{G}_2$ as the *boundaried* graph $(\mathbf{G}_1 \oplus \mathbf{G}_2, B, \rho)$, where B is the sets of all heirs from $\mathbf{G_1}$ and $\mathbf{G_2}$ and $\rho : B \to \mathbb{N}$ is the union of ρ_1 and ρ_2 after identification. Note that in circumstances where $\boxplus$ is repetitively applied, the heir relation is maintained due to its transitivity. Moreover, we define $\mathbf{G}_1 \triangleright \mathbf{G}_2$ as the unboundaried graph G obtained from $\mathbf{G}_1 \oplus \mathbf{G}_2$ by removing all heirs from $\mathbf{G}_2$ that are not heirs from $\mathbf{G}_1$ and all heirs of edges from $\mathbf{G}_2$ that are not heirs of edges from $\mathbf{G}_1$. Note that $\triangleright$ is not commutative. For the sake of simplicity, with a slight abuse of notation, we sometimes identify a vertex with its heir.

Nice problem and nice reduction. Let $p \in \mathbb{N}$, let $\mathcal{H}$ be a graph class, and let Π be a p-annotated problem corresponding to some choice of p-partition-evaluation function f and some $\mathsf{opt} \in \{\max, \min\}$. We say that Π is a *$\mathcal{H}$-nice problem* if there exists an algorithm that receives as input

- a boundaried graph $\mathbf{G} = (G, X, \rho)$,
- a trivial boundaried graph $\mathbf{X} = (G[X], X, \rho_X)$ and a collection $\{\mathbf{G}_i = (G_i, X_i, \rho_i) \mid i \in [d]\}$ of boundaried graphs, such that $d \in \mathbb{N}$ and $\mathbf{G} = \mathbf{X} \boxplus (\boxplus_{i \in [d]} \mathbf{G}_i)$,
- a partition (A, B) of X such that for all $i \in [d]$, $|\mathsf{heir}_{\mathbf{G_i},\mathbf{G}}(X_i) \setminus A| \leq 1$,
- some $\mathcal{A} \in \mathcal{P}_p(A)$, and
- for every $i \in [d]$ and each $\mathcal{X}_i \in \mathcal{P}_p(X_i)$, the value $\hat{\mathsf{p}}_{f,\mathsf{opt}}(G_i, \mathcal{X}_i)$,

and outputs, in time $\mathcal{O}(|A| \cdot d)$, a tuple $(\mathbf{G}' = (G', X', \rho'), \mathcal{A}', s')$, called *$\mathcal{H}$-nice reduction of the pair $(\mathbf{G}, \mathcal{A})$ with respect to* Π, such that the following hold.

- There is a set $A' \subseteq V(G')$ such that $|A'| = |A| + \mathcal{O}(1)$, and $\mathcal{A}' \in \mathcal{P}_p(A')$.
- There is a trivial boundaried graph $\mathbf{X}' = (G[X'], X', \rho_{X'})$ and a collection $\{\mathbf{G}'_i = (G'_i, X'_i, \rho'_i) \mid i \in [d']\}$, where $d' \in \mathbb{N}$, of boundaried graphs such that $\mathbf{G}' = \mathbf{X}' \boxplus (\boxplus_{i \in [d']} \mathbf{G}'_i)$ and $|V(G')| \leq |X| + \mathcal{O}(|B|)$, $|E(G')| \leq |E(G[X])| + \mathcal{O}(|B|)$.
- For any boundaried graph $\mathbf{F}$ compatible with $\mathbf{G}$, it holds that
$$\hat{\mathsf{p}}_{f,\mathsf{opt}}(\mathbf{G} \oplus \mathbf{F}, \mathcal{A}) \quad = \quad \hat{\mathsf{p}}_{f,\mathsf{opt}}(\mathbf{G}' \triangleright \mathbf{F}, \mathcal{A}') + s'.$$
- For any boundaried graph $\mathbf{F} = (F, X_F, \rho_F)$ compatible with $\mathbf{G}$, if $\bar{F} \setminus A_F \in \mathcal{H}$, where $\bar{F} = (\mathbf{F} \oplus \mathbf{G})[\mathsf{heir}_{\mathbf{F},\mathbf{G}\oplus\mathbf{F}}(V(F))]$ and $A_F = \mathsf{heir}_{\mathbf{G},\mathbf{G}\oplus\mathbf{F}}(A)$, then $(\mathbf{G}' \triangleright \mathbf{F}) \setminus A' \in \mathcal{H}$.

See Figure 1 for an illustration.

B Definition of the problems and their annotated extensions

K_t-Subgraph-Cover. Let $\mathcal{G}$ be a graph class. We define the problem Vertex Deletion to $\mathcal{G}$ as follows.

(Weighted) Vertex Deletion to $\mathcal{G}$
Input: A graph G (and a weight function $w : V(G) \to \mathbb{N}$).
Objective: Find the set $S \subseteq V(G)$ of minimum size (resp. weight) such that $G \setminus S \in \mathcal{G}$.

If $\mathcal{G}$ is the class of edgeless (resp. acyclic, planar, bipartite, (proper) interval, chordal) graphs, then we obtain the Vertex Cover (resp. Feedback Vertex Set, Vertex Planarization, Odd Cycle Transversal, (proper) Interval Vertex Deletion, Chordal Vertex Deletion) problem. Also, given a graph H, if $\mathcal{G}$ is the class of graphs that do not contain H as a subgraph (resp. a minor/odd-minor/induced subgraph), then the corresponding problem is called H-Subgraph-Cover (resp. H-Minor-Cover/H-Odd-Minor-Cover/H-Induced-Subgraph-Cover).

Let H be a graph and $w : V(G) \to \mathbb{N}$ be a weight function (constant equal to one in the unweighted case). We define f_H as the 2-partition-evaluation function where, for every graph G, for every $(R, S) \in \mathcal{P}_2(V(G))$,

$$f_H(G, (R, S)) = \begin{cases} +\infty & \text{if } H \text{ is a subgraph of } G \setminus S, \\ w(S) & \text{otherwise.} \end{cases}$$

Seen as an optimization problem, (Weighted) H-Subgraph-Cover is the problem of computing $\mathsf{p}_{f_H,\min}(G)$. We call its annotated extension (Weighted) Annotated H-Subgraph-Cover. In other words, (Weighted) Annotated H-Subgraph-Cover is defined as follows.

(Weighted) Annotated H-Subgraph-Cover
Input: A graph G, two disjoint sets $R, S \subseteq V(G)$ (and a weight function $w : V(G) \to \mathbb{N}$).
Objective: Find, if it exists, the minimum size (resp. weight) of a set $S^\star \subseteq V(G)$ such that $R \cap S^\star = \emptyset$, $S \subseteq S^\star$, and $G \setminus S^\star$ does not contain H as a subgraph.

Odd Cycle Transversal. Let H be a graph. We define f_{oct} as the 3-partition-evaluation function where, for every graph G and for every $(S, X_1, X_2) \in \mathcal{P}_3(V(G))$,

$$f_{\mathsf{oct}}(G, (S, X_1, X_2)) = \begin{cases} |S| & \text{if } G \setminus S \in \mathcal{B}, \text{witnessed by the bipartition } (X_1, X_2), \\ +\infty & \text{otherwise.} \end{cases}$$

Hence, seen as an optimization problem, Odd Cycle Transversal is the problem of computing $\mathsf{p}_{f_{\mathsf{oct}},\min}(G)$. We call its annotated extension Annotated Odd Cycle Transversal. In other words, Annotated Odd Cycle Transversal is defined as follows.

(Weighted) Annotated Odd Cycle Transversal
Input: A graph G, three disjoint sets $S, X_1, X_2 \subseteq V(G)$ (and a weight function $w : V(G) \to \mathbb{N}$).
Objective: Find, if it exists, a set $S^\star$ of minimum size (resp. weight) such that $S \subseteq S^\star$, $(X_1 \cup X_2) \cap S^\star = \emptyset$, and $G \setminus S^\star$ is bipartite with X_1 and X_2 on different sides of the bipartition.

Packing. Let $\mathcal{G}$ be a graph class. We define the $\mathcal{G}$-Packing problem as follows.

$\mathcal{G}$-Packing
Input: A graph G.
Objective: Find the maximum number k of pairwise-disjoint subgraphs $H_1, \ldots, H_k$ such that, for each $i \in [k]$, $H_i \in \mathcal{G}$.

Let H be a graph. If $\mathcal{G} = \{H\}$ (resp. $\mathcal{G}$ is the class of all graphs containing H as a minor/odd-minor/induced subgraph), then we refer to the corresponding problem as H-Subgraph-Packing (resp. H-Minor-Packing/H-Odd-Minor-Packing/H-Induced-Subgraph-Packing). Note, in particular, that K_3-Odd-Minor-Packing is exactly Odd Cycle Packing.

If in the definition of $\mathcal{G}$-Packing we add the condition that there is no edge in the input graph between vertices of different H_i's, then we refer to the corresponding problem as H-Scattered-Packing, where we implicitly assume that we refer to the subgraph relation, and where we do *not* specify a degree of "scatteredness", as it is usual in the literature when dealing, for instance, with the scattered version of Independent Set. For instance, K_2-Scattered-Packing is exactly Induced Matching.