Hitting Meets Packing: How Hard Can It Be?

Jacob Focke
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Fabian Frei
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Shaohua Li
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Dániel Marx
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Philipp Schepper
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Roohani Sharma
University of Bergen, Norway

Karol Węgrzycki
Saarland University, Saarbrücken, Germany
Max Planck Institute for Informatics, SIC, Saarbrücken, Germany

Abstract

We study a general family of problems that form a common generalization of classic hitting (also referred to as covering or transversal) and packing problems. An instance of $\mathcal{X}$-HitPack asks: Can removing k (deletable) vertices of a graph G prevent us from packing ℓ vertex-disjoint objects of type $\mathcal{X}$? This problem captures a spectrum of problems with standard hitting and packing on opposite ends. Our main motivating question is whether the combination $\mathcal{X}$-HitPack can be significantly harder than these two base problems. Already for one particular choice of $\mathcal{X}$, this question can be posed for many different complexity notions, leading to a large, so-far unexplored domain at the intersection of the areas of hitting and packing problems.

At a high level, we present two case studies: (1) $\mathcal{X}$ being all cycles, and (2) $\mathcal{X}$ being all copies of a fixed graph H. In each, we explore the classical complexity as well as the parameterized complexity with the natural parameters $k + \ell$ and treewidth. We observe that the combined problem can be drastically harder than the base problems: for cycles or for H being a connected graph on at least 3 vertices, the problem is Σ_2^{P}-complete and requires double-exponential dependence on the treewidth of the graph (assuming the Exponential-Time Hypothesis). In contrast, the combined problem admits qualitatively similar running times as the base problems in some cases, although significant novel ideas are required. For $\mathcal{X}$ being all cycles, we establish a $2^{\mathrm{poly}(k+\ell)} \cdot n^{\mathcal{O}(1)}$ algorithm using an involved branching method, for example. Also, for $\mathcal{X}$ being all edges (i.e., $H = K_2$; this combines Vertex Cover and Maximum Matching) the problem can be solved in time $2^{\mathrm{poly}(\mathsf{tw})} \cdot n^{\mathcal{O}(1)}$ on graphs of treewidth tw. The key step enabling this running time relies on a combinatorial bound obtained from an algebraic (linear delta-matroid) representation of possible matchings.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Algorithm design techniques

Keywords and phrases Hitting, Packing, Covering, Parameterized Algorithms, Lower Bounds, Treewidth

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.55

Related Version *Full Version*: `https://arxiv.org/abs/2402.14927` [22]

Funding *Philipp Schepper*: Part of Saarbrücken Graduate School of Computer Science, Germany.
Karol Węgrzycki: This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 850979).

1 Introduction

In the combinatorial optimization literature, many algorithmic problems can be classified into one of two dual classes: either as a packing problem or as a hitting problem. In *packing problems*, the goal is to find a large pairwise independent collection of objects of certain type. For example, one of the most-studied problems, Maximum Matching, can be described as finding a pairwise vertex-disjoint collection of at least ℓ edges. Network flow problems require finding a large collection of edge-disjoint paths from s to t. More generally, one can define the $\mathcal{X}$-Packing problem for any type $\mathcal{X}$ of objects. In *hitting problems* (sometimes referred to as *transversal* or *covering* problems), the task is to find a small set of elements that *hits* (i.e., intersects) every object of a certain type $\mathcal{X}$. For example, the Vertex Cover problem can be described as finding a set of at most k vertices that intersect every edge (i.e., contains at least one endpoint of each edge). The minimum s-t cut problem can be interpreted as finding a set of edges that intersects every s-t path. A quick note on terminology is in order here. The name of a covering problem typically refers to the type of objects used to cover, rather than the type of objects being covered. For example, Cycle Cover usually refers to the problem of covering the vertices of the graph with few (not necessarily disjoint) cycles, and *not* the problem of hitting every cycle with a small set of vertices (which is usually called Feedback Vertex Set). For this reason, we prefer to use $\mathcal{X}$-Hitting for the problem where the task is to find a set of elements or vertices that intersect every object of type $\mathcal{X}$.

There is a well-known duality phenomenon connecting hitting and packing problems. If there are ℓ disjoint objects of type $\mathcal{X}$, then clearly we need at least ℓ elements to hit every such object. In other words, the optimum of $\mathcal{X}$-Hitting is at least the optimum of $\mathcal{X}$-Packing. For some types of objects (such as edges in bipartite graphs and s-t paths), celebrated duality theorems demonstrate that there is always equality between the two optimum values. These duality results and their variants underlie many of the polynomial-time exact algorithms in combinatorial optimization. For problems where the two optimum values do not coincide, it is natural to ask how large the gap can be. Erdős and Pósa [20] showed that if ℓ is the maximum number of vertex-disjoint cycles, then all the cycles can be hit by a set of $k = \mathcal{O}(\ell \log \ell)$ vertices. More generally, we say that a type $\mathcal{X}$ of objects has the *Erdős–Pósa property* if the hitting optimum can be bounded by a function of the packing optimum. For example, it is known that the Erdős–Pósa property holds for undirected cycles passing through a set S [33, 58] or directed cycles [62], but it does not hold for cycles of odd length [61].

In this paper, we study a different, algorithmic question that connects hitting and packing problems. Let $\mathcal{X}$ be a type of graph objects, and consider the following problem. Given a graph G and integers k and ℓ, the task is to find a set S of at most k vertices such that $G - S$ does not contain ℓ disjoint copies of objects of type $\mathcal{X}$. This unified formulation captures both $\mathcal{X}$-Hitting and $\mathcal{X}$-Packing problems: for $\ell = 1$, it asks if every object can be hit with k vertices; for $k = 0$, it asks whether it is impossible to find ℓ disjoint objects. Therefore, $\mathcal{X}$-HitPack is at least as hard as $\mathcal{X}$-Hitting and the complement of $\mathcal{X}$-Packing. Note that deterministic algorithms, on which we focus in this paper, always work for the complement of a problem as well. The main meta-question that we explore is how hard such a combination of two problems may become:

> If some type of algorithm exists for both $\mathcal{X}$-Hitting and $\mathcal{X}$-Packing, then is there such an algorithm for $\mathcal{X}$-HitPack as well?

The main message of this paper is that the formulation of this question leads to a whole new unexplored continent of interesting and challenging questions. As we shall see, in some settings the combined problem is indeed strictly harder, while in other settings a qualitatively similar algorithm can be obtained for the combined problem, albeit only after developing significantly more involved techniques.

To make the problem statement more robust, we extend the problem by assuming that the input graph contains a set U of undeletable vertices and the solution S has to be disjoint from U. Such undeletable vertices may be needed to express problems where the objects $\mathcal{X}$ are, say, s-t paths or paths between terminals, and we do not want to allow the deletion of terminals. Note that this generalization makes our algorithmic results slightly stronger, while it makes the lower bound results slightly weaker. Formally, for a type $\mathcal{X}$ of objects, the problem is defined as follows.

$\mathcal{X}$-HitPack

Input: Graph G, set $U \subseteq V(G)$, integers k and ℓ.

Question: Is there a set $S \subseteq V(G) \setminus U$ of size $\leq k$ such that $G - S$ does not contain ℓ vertex-disjoint objects of type $\mathcal{X}$?

There are two ways of looking at the $\mathcal{X}$-HitPack problem. It can be considered as a *weaker* version of hitting: the solution does not have to destroy all sets in $\mathcal{X}$, but up to ℓ disjoint sets are allowed to survive in $G - S$. An alternative view is to interpret it as a more stable version of packing. We have to decide not only whether ℓ disjoint objects exist but whether the graph may lose this property if up to k arbitrary vertices are removed. Such a robust version of packability is clearly desirable in many situations, and the problem of detecting this property is precisely the complement of $\mathcal{X}$-HitPack. Frei et al. [29] recently initiated a systematic complexity study of a related stability notion, where a graph is *vertex-stable* if some parameter cannot change upon deletion of a single vertex. Specifically, an instance is in $\mathcal{X}$-HitPack if and only if it is not stable with respect to the deletion of up to k vertices and the property of containing ℓ vertex-disjoint objects of type $\mathcal{X}$.

Returning to the overarching question of how hard $\mathcal{X}$-HitPack may become compared to $\mathcal{X}$-Hitting, and $\mathcal{X}$-Packing, let us start with examples where all of them are polynomial-time solvable. If $\mathcal{X}$ is simply the set of edges, then the hitting problem is Vertex Cover and the packing problem is Maximum Matching. On bipartite graphs they are known to be polynomial-time solvable, and the size of a minimum vertex cover is equal to size of a maximum matching. Let d be this size. Deleting a set of k vertices can decrease the size of the maximum matching only by at most k, thus the answer is no if $d - k \geq \ell$. Otherwise, if $d - k < \ell$, then deleting any k vertices of a minimum vertex cover decreases the size of a maximum matching by k (as deleting $d - k$ further vertices of the vertex cover decreases this size to 0), showing that the answer is yes. This argument works for other objects where exact duality theorems are known, for example for internally vertex-disjoint s-t paths (since the maximum number of disjoint paths is equal to the minimum s-t separator).

In general, however, no such exact duality theorem is available. In such a case, $\mathcal{X}$-Hitting and $\mathcal{X}$-Packing are two very different problems that may require different techniques. Then, solving $\mathcal{X}$-HitPack would require combining the two solution techniques in a nontrivial way, and it very well may be the case that $\mathcal{X}$-HitPack is a qualitatively harder problem that both $\mathcal{X}$-Hitting and $\mathcal{X}$-Packing. Let us point out that one can explore different aspects of hardness (NP-hardness, exact running times, parameterized complexity, approximation, etc.), thus already for one particular choice of $\mathcal{X}$, one can ask many different questions. This means

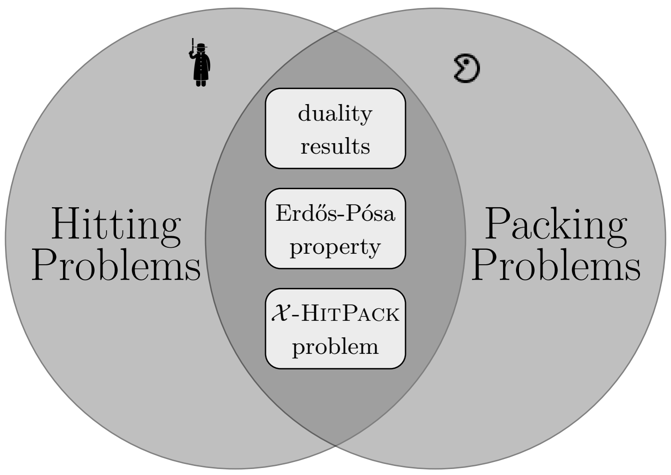

Figure 1 The areas of hitting and packing problems intersect in combinatorial duality and Erdős–Pósa property results, and in the algorithmic study of combined hitting and packing problems.

that understanding the $\mathcal{X}$-HitPack problem is a two-dimensional question: one dimension is the choice of $\mathcal{X}$ and the other dimension is the notion of complexity. We present two case studies (the objects $\mathcal{X}$ being cycles of arbitrary size or subgraphs isomorphic to a fixed graph H) and explore different aspects of the complexity of $\mathcal{X}$-HitPack. See Table 1 for an overview of our results.

Case Study 1: Cycles

Let us first consider the case when the type $\mathcal{X}$ of objects are the cycles in the given graph G. Thus, $\mathcal{X}$-Hitting is exactly Feedback Vertex Set (the problem of deciding whether there is a set S of up to k vertices such that $G - S$ has no cycle), $\mathcal{X}$-Packing is Cycle Packing (asking whether are there ℓ vertex-disjoint cycles), and $\mathcal{X}$-HitPack is the problem of asking if it is possible to remove a set S of k (deletable) vertices such that the remaining graph does not contain ℓ vertex-disjoint cycles.

Feedback Vertex Set and Cycle Packing are both known to be NP-complete. As we have seen, Cycle-HitPack generalizes both Feedback Vertex Set and the *complement* of Cycle Packing, thus it is unlikely to be in NP. Indeed, a solution S of size k is *not* a good certificate, as it is hard to verify due to the NP-hardness of Cycle Packing. We show that Cycle-HitPack is in fact located further up in the polynomial hierarchy.

▶ **Theorem 1.1.** *Cycle-HitPack is Σ_2^{P}-complete.*

Another jump in complexity can be observed if we consider how the problems behave on graphs of bounded treewidth. The study of parameterized algorithms and complexity on such graphs has been a fruitful area of research, as many NP-hard problems become tractable when restricted to graphs of small treewidth. In many cases, the problems are fixed-parameter tractable (FPT) parameterized by treewidth: there is an algorithm solving the problem in time $f(\mathsf{tw}) \cdot n^{\mathcal{O}(1)}$ for some function f. By now, there is a strong understanding on how the complexity of different problems depends on the treewidth of the graph, with a number of nontrivial algorithmic techniques and tight conditional lower bounds appearing in the literature [7, 8, 13, 14, 15, 19, 23, 24, 34, 41, 42, 45, 51, 52, 53, 55, 56]. The function f is typically of the form $2^{\mathrm{poly}(\mathsf{tw})}$, and we know of only a handful of problems where a double-

or even triple-exponential dependence on treewidth is necessary assuming the Exponential-Time Hypothesis (ETH) [28, 40, 42, 51, 57]. One may see a pattern that problems at higher level of the polynomial hierarchy may need more than exponential dependence on treewidth, but note that there are problems in NP that need double-exponential dependence (metric dimension [28]) and there are #P-hard counting problems[1] that can be solved with single-exponential dependence [14, 23, 24, 52]. Both Feedback Vertex Set and Cycle Packing can be solved in time $2^{\mathrm{poly}(\mathsf{tw})} \cdot n^{\mathcal{O}(1)}$ (it is known that, assuming ETH, the optimal dependence on treewidth is $2^{\mathcal{O}(\mathsf{tw})}$ and $2^{\mathcal{O}(\mathsf{tw} \log \mathsf{tw})}$ for the two problems, respectively). Does the common generalization Cycle-HitPack also admit an algorithm of this running time? We answer this question in the negative: the dependence becomes double-exponential on treewidth. We first present a new double-exponential algorithm.

▶ **Theorem 1.2.** *Cycle-HitPack can be solved in time $2^{2^{\mathcal{O}(\mathsf{tw} \log \mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$, where tw is the treewidth of the input graph.*

We show a matching lower bound that not only proves the double-exponential dependence for the larger parameter pathwidth but also shows that the additional logarithmic factor cannot be avoided.

▶ **Theorem 1.3.** *Assuming ETH, Cycle-HitPack has no $2^{2^{o(\mathsf{pw} \log \mathsf{pw})}} \cdot n^{\mathcal{O}(1)}$-time algorithm, where pw is the pathwidth of the input graph.*

Feedback Vertex Set is known to be fixed-parameter tractable (FPT) parameterized by k, in fact, it can be solved in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ [9, 16, 31, 38]. Cycle Packing is FPT parameterized by ℓ, but it is an open question whether there is a $2^{\mathcal{O}(\ell)} \cdot n^{\mathcal{O}(1)}$-time algorithm: the current best algorithm has running time $2^{\mathcal{O}(\ell \log^2 \ell / \log \log \ell)}$ [47]. As Cycle-HitPack is NP-hard for $k = 0$ and also for $\ell = 1$, a natural parameter for the problem is $p := k + \ell$.

We can try to use the following approach to show that Cycle-HitPack is FPT parameterized by p. Suppose that the input graph G has $p = k + \ell$ disjoint cycles. Then for every set S of k vertices, the graph $G - S$ has ℓ disjoint cycles, implying that G is a no-instance of Cycle-HitPack. Therefore, we can assume that G has at most p disjoint cycles. Then the Erdős–Pósa Theorem implies that G has a feedback vertex set of size $\mathcal{O}(p \log p)$, which also implies that G has treewidth $\mathcal{O}(p \log p)$. Now we can try to use an algorithm for Cycle-HitPack parameterized by treewidth. However, in light of Theorem 1.3, any such algorithm would give a running time with double-exponential dependence on p. Can we improve this running time to $2^{\mathrm{poly}(p)} \cdot n^{\mathcal{O}(1)}$, to qualitatively match the running times of the Feedback Vertex Set and Cycle Packing algorithms? We show that this is indeed possible, and hence we do not see such a drastic jump in complexity similar to the Σ_2^{P}-completeness of the problem and the double-exponential dependence on treewidth.

▶ **Theorem 1.4.** *Cycle-HitPack can be solved in time $2^{\mathrm{poly}(p)} \cdot n^{\mathcal{O}(1)}$ (where $p := k + \ell$).*

The proof exploits that G has a feedback vertex set F of size $\mathcal{O}(p \log p)$. By a simple branching argument, we assume that the solution S is disjoint from F. Then we interpret a packing of cycles as a collection of paths connecting some neighbors of F in the forest $G - F$. Our goal is to hit every such collection of paths that would lead to a collection of ℓ cycles. With a branching algorithm, we collect paths that have to be hit, until we can conclude that our collection of paths cannot be hit by k vertices.

[1] By Toda's theorem [65], it is known that #P contains the entire polynomial hierarchy.

Case Study 2: H-subgraphs

Next, let us consider the setting where the type $\mathcal{X}$ of objects are the (not necessarily induced) subgraphs isomorphic to a fixed graph H. Thus, H-Hitting is the problem of removing a set S of k vertices such that no subgraph isomorphic to H remains, H-Packing is finding ℓ disjoint copies of H as subgraphs, and H-HitPack is the problem of removing a set S of k vertices such that the remaining graph does not contain ℓ disjoint copies of H.

For every fixed connected graph H with at least 3 vertices, H-Hitting [37] and H-Packing [43] are NP-complete. Similarly to the case of Cycle-HitPack, the H-HitPack problem lies on the second level of the polynomial hierarchy.

▶ **Theorem 1.5.** *For any fixed connected graph H with at least three vertices, H-HitPack is Σ_2^{P}-complete.*

Similarly to Cycle-HitPack, we again see a jump to double-exponential dependency on treewidth and pathwidth. For the case of general graphs H, we provide an algorithm whose running time asymptotically matches the one of the algorithm for Cycle-HitPack.

▶ **Theorem 1.6.** *For any fixed connected graph H, H-HitPack can be solved in time $2^{2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$, where tw is the treewidth of the input graph.*

In the case when H is a clique, we exploit that we are only packing complete graphs which leads to an improvement where we remove the logarithmic factor from the exponent.

▶ **Theorem 1.7.** *For any fixed integer $q \geq 2$, q-Clique-HitPack can be solved in time $2^{2^{\mathcal{O}(\mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$, where tw is the treewidth of the input graph.*

By designing the matching lower bound for Cycle-HitPack in such a way that the construction already works for Square-HitPack, we obtain a matching lower bound for Square-HitPack.

▶ **Theorem 1.8.** *Assuming ETH, Square-HitPack has no $2^{2^{o(\mathsf{pw}\log\mathsf{pw})}} \cdot n^{\mathcal{O}(1)}$-time algorithm, where pw is the pathwidth of the input graph.*

For the case of general H we provide a separate reduction. In contrast to the matching lower bound for the case of Square-HitPack, we present a lower bound which is only matching for the case of cliques as for cliques we provide an improved algorithm. For the case when H is neither a clique nor a C_4, it remains open to remove the logarithmic factor from the running time or to improve the lower bound accordingly.

▶ **Theorem 1.9.** *Assuming ETH, for any fixed connected graph H with at least three vertices, H-HitPack has no $2^{2^{o(\mathsf{pw})}} \cdot n^{\mathcal{O}(1)}$-time algorithm, where pw is the pathwidth of the input graph.*

It is known that H-Hitting parameterized by k and H-Packing parameterized by ℓ are both fixed-parameter tractable (FPT), in fact, for fixed H, they can be solved in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ and $2^{\mathcal{O}(\ell)} \cdot n^{\mathcal{O}(1)}$, respectively. For H-Hitting, this follows from a simple bounded-depth search tree algorithm, while color coding [3] or representative set techniques [25] can be used for H-Packing. There is an easy bounded-depth search tree algorithm showing that H-HitPack is FPT parameterized by $p := k + \ell$.

▶ **Theorem 1.10.** *For any fixed graph H that might be unconnected, H-HitPack can be solved in time $2^{\mathcal{O}(p\log p)} \cdot n^{\mathcal{O}(1)}$ (where $p := k + \ell$).*

Proof. First, we use the $2^{\mathcal{O}(\ell)} \cdot n^{\mathcal{O}(1)}$ H-Packing algorithm to find ℓ copies of H if they exist [3, 25]. The solution S has to contain at least one of the $|V(H)| \cdot \ell = \mathcal{O}(\ell)$ vertices of this packing. Thus, we can branch on choosing one of these vertices, delete that vertex from the graph, and decrease our quota k of deletions by one. We repeat this process until either $k = 0$ or there are no ℓ disjoint copies of H in the graph. This results in a search tree of size $\ell^{\mathcal{O}(k)} = 2^{\mathcal{O}(p \log p)}$ and thus, concludes the proof. ◀

We leave open the potentially challenging question of whether this running time of the algorithm can be improved to $2^{\mathcal{O}(p)} \cdot n^{\mathcal{O}(1)}$: none of the techniques used for H-Hitting and H-Packing seem directly relevant for such improvement. In other words, it is easy to show that H-HitPack is also FPT, but whether quantitatively same FPT algorithms can be obtained for this more general problem is a far from trivial question.

The Curious Case of Edge-HitPack

Let us consider now the case of H-HitPack where H consists of only a single edge, i.e., $H = K_2$. Then Edge-Hitting is the NP-complete problem Vertex Cover, while Edge-Packing is the polynomial-time solvable Maximum Matching problem. This implies that, unlike in the cases where H has at least 3 vertices, Edge-HitPack is in NP: given a solution S, we can verify in polynomial time that $G - S$ has no matching of size ℓ.

While we do not have a $2^{\mathcal{O}(k+\ell)} \cdot n^{\mathcal{O}(1)}$-time algorithm for H-HitPack for general H, we present a very simple algorithm solving Edge-HitPack in time $3^{k+\ell} \cdot n^{\mathcal{O}(1)}$. The algorithm essentially relies on an augmenting-path argument, hence it gives no indication on how other H-HitPack problems could be solved with a similar running time.

▶ **Theorem 1.11.** *Edge-HitPack can be solved in time $3^{k+\ell} \cdot n^{\mathcal{O}(1)}$.*

Theorem 1.9 showed that double-exponential dependence on treewidth is needed to solve the H-HitPack problem when H is connected and has at least 3 vertices, that is, already for the Triangle-HitPack problem. However, Edge-HitPack can be solved with only exponential dependence on treewidth.

▶ **Theorem 1.12.** *Edge-HitPack can be solved in time $2^{\text{poly}(\mathsf{tw})} \cdot n^{\mathcal{O}(1)}$, where tw is the treewidth of the graph.*

Let us give an intuitive explanation for this difference in running time between Edge-HitPack and Triangle-HitPack. There is a well-understood methodology for designing algorithms on tree decompositions: for each rooted subtree of the tree decomposition, we define a certain number of subproblems, each asking for the existence of a certain class of partial solutions. The running time typically depends on how many equivalence classes of partial solutions we need to consider. For example, in the Triangle-Packing problem, the class of partial packings is described by the subset of the bag that is covered by the packing, so there are $2^{\mathsf{tw}+1}$ different classes of partial solutions. For the Triangle-Hitting problem, a partial solution is a set of vertices that destroys every triangle in a rooted subtree of the tree decomposition, and its class is described by its intersection with the bag.

For the combination, Triangle-HitPack, a partial solution is a set of vertices that does not necessarily destroy every triangle in the subtree of the tree decomposition, but may still leave some triangle packings of size $< \ell$ in the graph. Therefore, a partial solution S can be described by what kind of triangle packings survive after deleting S, that is, by describing which subsets of the bag can be covered/avoided by triangle packings of a certain size. This means that the class of a partial solution is described by a set system over a

Table 1 An overview of the main results for $\mathcal{X}$-HitPack. For ease of comparability, we omit the common factor $n^{\mathcal{O}(1)}$ from the FPT running times.

Object $\mathcal{X}$	UB $p = k + \ell$	UB Treewidth	LB Treewidth	Completeness
Edge	3^p T. 1.11	$2^{\text{poly}(\text{tw})}$ T. 1.12	no $2^{o(\text{tw})}$ [44, T. 1]	NP [30, T. 3.3]
Triangle	$2^{\mathcal{O}(p \log p)}$ T. 1.10	$2^{2^{\mathcal{O}(\text{tw})}}$ T. 1.7	no $2^{2^{o(\text{tw})}}$ T. 1.9	Σ_2^{P} T. 1.5
q-Clique				
Conn. H, 3+ vert.		$2^{2^{\mathcal{O}(\text{tw} \log \text{tw})}}$ T. 1.6		
Square	$2^{\mathcal{O}(p \log p)}$ T. 1.4		no $2^{2^{o(\text{tw} \log \text{tw})}}$ T. 1.8	
Cycles	$2^{\text{poly}(p)}$ T. 1.4	$2^{2^{\mathcal{O}(\text{tw} \log \text{tw})}}$ T. 1.2	no $2^{2^{o(\text{tw} \log \text{tw})}}$ T. 1.9	Σ_2^{P} T. 1.1

bag of the decomposition. As the set systems arising this way can be fairly arbitrary in the Triangle-HitPack problem, there are up to $2^{2^{\text{poly}(\text{tw})}}$ such set systems and hence up to that many different classes of partial solutions. This is the intuitive reason why a double-exponential dependence on treewidth is needed for the problem Triangle-HitPack.

In the case of Edge-HitPack, the set systems describing a partial solution show how the bag can be covered by matchings of a certain size. Such set systems have lots of structure and cannot be completely arbitrary; in particular, they are related to (delta)-matroids. Inspired by an argument of Wahlström [67], we give a combinatorial bound showing that such set systems can be represented algebraically with $\mathcal{O}(\text{tw}^3)$ bits, hence there are at most $2^{\mathcal{O}(\text{tw}^3)}$ different set systems that can arise. Interestingly, our proof is not algorithmic, but it is sufficient to bound the running time of our algorithm. In fact, we have to make no adjustment to the algorithm of Theorem 1.6 solving H-HitPack when H is a clique: the algorithm was designed in a way that a combinatorial proof on the relevant set systems immediately bounds the running time of the algorithm.

Discussion and Open Problems

We have initiated the study of a natural common generalization of hitting problems and packing problems. Certain basic techniques for hitting and packing problems can be lifted to this generalization, but we have seen that the generalization can be significantly harder and more challenging, requiring us to revisit classic problems from a new perspective. The familiar landscape of hitting and packing problems with their known properties and well-established techniques is replaced by a strange world where many of the known techniques are inapplicable, new techniques have to be brought in, and the problem has to be approached with a completely different mindset that takes into account the more complicated quantifier structure of the problem definition.

We have presented a selection of algorithmic results and lower bounds for $\mathcal{X}$-HitPack problems, but they probably just scratch the surface of a rich family of unexplored challenging problems. We list a few open questions and potential research directions to stimulate further work in this area.

- Is there a $2^{\mathcal{O}(k+\ell)} \cdot n^{\mathcal{O}(1)}$-time algorithm for H-HitPack for every fixed (connected) graph H?
- Is the $\mathcal{X}$-HitPack problem FPT in k and ℓ where $\mathcal{X}$ are the odd cycles in the graph? Note that the corresponding hitting problem Odd Cycle Transversal is well-known to be FPT by different techniques [48, 60, 63], and Odd Cycle Packing is also FPT using an extension of graph minor algorithms with parity conditions [35, 36].

- Is the $\mathcal{X}$-HitPack problem FPT in k and ℓ where $\mathcal{X}$ are the induced cycles of length at least 4 in the graph? The hitting problem Chordal Deletion [2, 10, 32, 49] and the packing problem Chordless Cycle Packing [50] are both FPT.
- In general, one could explore if induced versions of the H-HitPack problems are different compared to the case when we are considering not necessarily induced subgraphs isomorphic to H.
- We defined our framework in terms of removing vertices and vertex-disjoint packings, but one could analogously study a problem defined by removing edges and edge-disjoint packings. This setting may pose very different challenges compared to the problems studied in this paper.[2]
- A natural generalization of Cycle-HitPack is to consider the $\mathcal{X}$-HitPack problem where $\mathcal{X}$ is the set of all minor models of a fixed graph H (Cycle-HitPack is equivalent to the case when H is K_3). Observe that if we denote by $\ell \cdot H$ the graph consisting of ℓ disjoint copies of H, then the $\mathcal{X}$-HitPack problem defined for minor models of H is equivalent to removing k vertices such that the resulting graph does not contain $\ell \cdot H$ as a minor. Problems of this form where intensively studied [1, 21, 26]. Thus this gives a way of solving the problem for fixed k, ℓ, and H, but understanding the optimal form of the running time could be an interesting question. The same problem can be studied also in the context of topological minors.
- One could ask how the Erdős-Pósa Property relates to the complexity of the $\mathcal{X}$-HitPack problem, but it is not obvious how to formulate this question in a way that leads to meaningful results. Note first that for problems involving copies of a fixed graph H, the Erdős-Pósa Property trivially holds (in some sense, this was implicitly used by the simple algorithm of Theorem 1.10). The algorithm of Theorem 1.4 explicitly used the Erdős-Pósa Property for cycles as a starting step. This might be a useful starting step in other cases where H-minor models satisfy this property (which is known to be the case exactly when H is planar). However, note that the argument sketched in the previous item works irrespective of whether H-minor models satisfy the Erdős-Pósa Property, although it may affect the running time.
- Tournaments (i.e., directed graphs with exactly one directed edge between any pair of vertices) form a well-studied class of directed graphs where many hitting and packing problems are more tractable compared to general directed graphs [4, 5, 6, 11, 12, 17, 27, 39, 46, 54, 59, 68]. Which of these results generalize to the combined hitting and packing problem?
- Investigating the approximability of $\mathcal{X}$-HitPack problems is another area that is completely unexplored. The proper notion of approximation for these kind of problems seems to be the following: if there is a solution S of size k such that $G - S$ has no ℓ disjoint objects of type $\mathcal{X}$, can we find a set S' of size at most $c \cdot k$ such that $G - S'$ has no $c \cdot \ell$ disjoint objects of type $\mathcal{X}$. That is, in this approximate sense, it is ok to find a somewhat larger set S' that has the somewhat weaker property that it prevents only packings of $c \cdot \ell$ disjoint objects.

[2] Even though the edge version of the problem is also intriguing, let us point out that for a number of reasons the vertex version seems like the more natural starting point: 1) The base cases Vertex Cover and Maximum Matching are more natural than the corresponding basic edge problems. 2) The known dichotomy result for H-Packing is much simpler for the vertex version (compare [37] to [18]). 3) In a tree decomposition, each bag can intersect only a bounded number of vertex-disjoint copies of some object. This is no longer true for edge-disjoint packings. 4) More generally, the graph minor theory of Robertson and Seymour is inherently about vertex-disjointness. For edge-disjointness, the much less developed theory of immersions might become relevant.

2 Technical Overview

In this section, we give a brief overview of our results, highlighting the main technical ideas and putting them into context. The remaining sections of the paper prove these results in the order presented below. Note that, besides these individual technical contributions, it can be considered an equally important conceptual contribution that we demonstrate that the combination of hitting and packing can lead to a wide range of interesting and challenging problems.

2.1 Algorithmic Results

$2^{\mathrm{poly}(k+\ell)} \cdot n^{\mathcal{O}(1)}$-Time Algorithm for Cycle-HitPack. As noted earlier, we may assume in this problem that the graph G has a feedback vertex set F of size $\mathcal{O}((k+\ell)\log(k+\ell))$, otherwise the Erdős–Pósa Theorem implies that the answer is no. Instead of using the fact that this gives a bound on the treewidth and trying to use a general algorithm parameterized by treewidth, we present an algorithm with running time $2^{\mathrm{poly}(k+|F|)} \cdot n^{\mathcal{O}(1)}$ where F is a feedback vertex set.

With a standard branching step, we can guess which vertices of F are in the solution, remove these vertices from G, adjust k appropriately, and then assume that the feedback vertex set F is undeletable. To sketch the main ideas of the proof, let us assume that $G - F$ is not only a forest, but every component of $G - F$ is a path. If C is a cycle in G, then it contains at least one vertex of F, and $C - F$ consists of one or more (sub-)paths in $G - F$. If a graph contains a packing of ℓ cycles, then we may assume that each cycle is an induced cycle (this can be achieved by possibly shortening some cycles). If C is an induced cycle, then every path P of $C - F$ is of the following form: P goes from a neighbor of some $f_1 \in F$ to a neighbor of some $f_2 \in F$ (possibly $f_1 = f_2$) such that the internal vertices of P are adjacent to neither f_1 nor f_2. Let us call this a *usable path.*

Suppose that we have a packing of ℓ (induced) cycles in G. The solution has to contain a vertex of a cycle C of this packing, that is, a vertex of one of the paths $C - F$ (as F is undeletable). We branch on choosing a cycle C of the packing and choosing a path P of $C - F$ that is broken by the solution, but we *do not* choose a vertex of P. Instead, we put P into a collection $\mathcal{P}$ of forbidden paths that need to be broken by the solution. Then we find a packing of ℓ cycles that does not use any of the forbidden paths. Such a collection can be found by branching on the number and type of paths in the packing and then by a dynamic programming algorithm that scans paths in $G - F$ in a left-to-right order and tries to find disjoint paths of these types that are not on the forbidden list $\mathcal{P}$. Once we have such a collection, we once again branch on a path that has to be broken by the solution and put it into the collection $\mathcal{P}$. We repeat this procedure as long as we are able to find an appropriate packing.

If the algorithm is not able to find a packing of ℓ cycles that does not use any forbidden path, then we need to check if there is a set of k vertices that can break every forbidden path in $\mathcal{P}$. This can be done by a simple polynomial-time algorithm (find a minimum number of points covering a set of intervals). If there is such a set S, then it forms a solution; if there is no such set, then this is an incorrect branch of the algorithm.

To bound the running time, we need to bound the depth of the search tree, that is, the number of paths we put into the solution. The key observation is that a vertex v can cover at most $|F|^2$ different usable paths. If v covers a useful path P from u_1 to u_2, then u_1 should be the last vertex before v that is the neighbor of some $f_1 \in F$ and u_2 is the first vertex after v that is the neighbor of some $f_2 \in F$. Therefore, if $|\mathcal{P}| > k|F|^2$, then surely there is no set

S of k vertices intersecting all these paths. This observation gives a $\text{poly}(k + |F|)$ bound on the height of the search tree. As we branch into $\text{poly}(|F|)$ cases in each step, the claimed running time follows.

With additional work, this algorithmic idea can be extended to the case when $G - F$ is not a collection of paths, but a general forest. The situation becomes significantly more complicated due to high degree vertices in the forest, paths with many branch nodes, and other issues, but the difficulties can be overcome by additional layers of arguments.

$3^{k+\ell} \cdot n^{\mathcal{O}(1)}$-Time Algorithm for Edge-HitPack. Let us sketch a very simple branching algorithm. We measure our progress by $k + \ell + 1 - \nu(G[U])$, where $\nu(G[U])$ is the size of the maximum matching in the graph induced by a set U of undeletable vertices that is maintained throughout the algorithm. Let us find a maximum matching M in $G[U]$. For non-trivial instances, we have $\nu(G[U]) < \ell \leq \nu(G)$. Hence, there is an augmenting path increasing the size of M. Let u and v be the two endpoints of the augmenting path. We branch into three directions:

- u is in the solution: remove u, decrease k by one.
- v is in the solution: remove v, decrease k by one.
- neither u nor v is in the solution: put u and v into U.

As the last branch strictly increases the size of the maximum matching in U, we can conclude that the measure $k + \ell + 1 - \nu(G[U])$ strictly decreases in each branch, giving a bound of $k + \ell$ on the depth of the search tree.

$2^{2^{\mathcal{O}(\mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$-Time Algorithm for H-HitPack When H Is a Clique. A typical way of designing algorithms for bounded-treewidth graphs is the following. Let us recall the definition of tree decompositions. A *tree decomposition* of graph G is a rooted tree T with a collection $\{X_t \subseteq V(G) \mid t \in V(T)\}$ of sets called *bags*. The conditions for a tree decomposition are:

- For any vertex u in G, the nodes in T with bags containing u form a connected subtree of T.
- For any edge uv in G, there exists a node in T with a bag containing both u and v.

The *width* of a tree decomposition (T, B) is $\max_{t \in V(T)} |X_t| - 1$. The *treewidth* tw of G is the minimum possible width of a tree decomposition. *Pathwidth* is defined similarly, with T restricted to be a path. We denote by V_t the set of vertices appearing in the bags of the nodes in the subtree rooted at some node t. Similarly, we define G_t as the graph induced by the vertices or rather (in case of a tree decomposition with edge introduce nodes) given by the edges introduced in the subtree rooted at node t.

For H-HitPack, the solution S has a part $S \cap V_t$ that somehow influences packings of cliques that intersect V_t. A key observation is that a clique K is either fully contained in V_t, or intersects only the root bag X_t, i.e., $K \cap V_t = K \cap X_t$. Based on this observation, we can argue that the effect of $S \cap V_t$ can be described by the following information:

- The intersection $S \cap X_t$.
- For every $D \subseteq X_t$, the maximum size of a packing in $G[V_t \setminus S] - D$.

Note that if the maximum packing size in $G[V_t \setminus S]$ is m, then the maximum packing size in $G[V_t \setminus S] - D$ is between $m - |D|$ and m. Thus all the relevant information about $S \cap V_t$ can be described by the set $S \cap X_t$ ($2^{\mathsf{tw}+1}$ possibilities) and by a sequence of $2^{\mathsf{tw}+1}$ integers between 0 and $\mathsf{tw}+1$ ($(\mathsf{tw}+2)^{2^{\mathsf{tw}+1}}$ possibilities), leading to a bound of $2^{2^{\mathcal{O}(\mathsf{tw})}} \cdot n$ different ways $S \cap X_t$ can behave. With this bound at hand, we can follow the standard methodology of designing algorithms on tree decompositions: we define subproblems at each node t corresponding to the different behaviors of S and solve these subproblems in a bottom-up manner. The dominating factor of the running time is the number of subproblems at each node of the tree decomposition, leading to a $2^{2^{\mathcal{O}(\mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$-time algorithm.

$2^{\text{poly}(\mathsf{tw})} \cdot n^{\mathcal{O}(1)}$-Time Algorithm for Edge-HitPack. When H is a single edge, we can improve the running time the following way. As noted above, the behavior of the set $S \cap V_t$ can be described by the set $S \cap X_t$ and by a function showing how the size of the maximum matching decreases if we remove a subset $D \subseteq X_t$, that is, by the function $g(D) = \nu(G[V_t \setminus S]) - \nu(G[V_t \setminus S] - D)$. Because of the highly structured nature of the matching problem, this function cannot be arbitrary and can be compactly described, hence the number of possibilities is much smaller than $2^{2^{\mathcal{O}(\mathsf{tw})}}$. Let us first consider the related function that depends on whether $G[V_t \setminus S] - D$ has a perfect matching or not. This function describes a so-called *delta-matroid* and has an algebraic representation as a skew-symmetric matrix. Following a proof sketch[3] of Wahlström [67], this matrix can be turned into a representation with $\mathcal{O}(\mathsf{tw}^3)$ bits. Formally, in the full version [22] we prove the following lemma.

▶ **Lemma 2.1.** *Let G be an n-vertex graph over a vertex set $V \supseteq [k]$ for some integer k. Let $f_{G,k} \colon 2^{[k]} \to \mathbb{Z}^+$ be the function defined by $f_{G,k}(S) = \nu(G - S)$. For each k and n, there are $n \cdot 2^{\mathcal{O}(k^3)}$ functions $f_{G,k}$ that can arise this way.*

Then a simple graph-theoretic construction can be used to compactly describe the function g using the compact representation of the aforementioned function. Interestingly, our proof of obtaining the algebraic representation is not algorithmic, but it is sufficient for our purposes: the dynamic-programming algorithm on the tree decomposition can be designed in a way that it needs only a combinatorial bound on the number of different subproblems that has a solution in the current graph, but does not need to be able to compute which subproblems have no solution in any graph. Therefore, the algorithm for H-HitPack with H being a clique does not need any modification at all to achieve this running time.

The following statement is the crucial combinatorial insight that allows us to achieve $\mathcal{O}(\mathsf{tw}^3)$-bits and prove Lemma 2.1.

▶ **Lemma 2.2.** *Let G be a graph over a vertex set $V \supseteq [k]$ for some integer k. Let us define the function $h_{G,k} \colon 2^{[k]} \to \{0,1\}$ the following way:*

$$h_{G,k}(S) = \begin{cases} 1 & \textit{if } G - S \textit{ has a perfect matching,} \\ 0 & \textit{otherwise.} \end{cases}$$

For each integer k, the number of distinct functions $h_{G,k}$ is $2^{\mathcal{O}(k^3)}$.

Note, that naively the number of such functions is double-exponential in k. Now, we present the proof of Lemma 2.2 and use algebraic tools, in particular, representation of linear delta-matroids and multivariate polynomials over finite fields. For a field $\mathbb{F}$, we denote by $\mathbb{F}[x_1, \dots, x_n]$ the ring of n-variable polynomials with coefficients from $\mathbb{F}$. We have to be careful to make a distinction between the *zero polynomial* of $\mathbb{F}[x_1, \dots, x_n]$, which is the polynomial where every coefficient is zero, and a *vanishing polynomial* over $\mathbb{F}$, which is a polynomial that is 0 for every substitution of values from $\mathbb{F}$ to the variables. For example, $x^{|\mathbb{F}|} - x$ is a nonzero vanishing polynomial over the field $\mathbb{F}$. The following observation can be used to argue that some nonzero polynomial is not vanishing.

[3] The given proof sketch does not treat the question of what field to choose and the obvious way of handling this issue does not lead to the claimed bound (as confirmed by the author). Our proof needs additional arguments to ensure the existence of a representation over a suitable field.

▶ **Observation 2.3.** *If $P \in \mathbb{F}[x_1, \dots, x_n]$ is an n-variable polynomial over the field $\mathbb{F}$ and every variable x_i has degree less than $|\mathbb{F}|$ in P, then P is a vanishing polynomial if and only if it is the zero polynomial.*

The proof of Lemma 2.2 follows from the fact that polynomials $P \in \mathbb{F}[x_1, \dots, x_n]$ where every variable has degree less than $|\mathbb{F}|$ are in one-to-one correspondence with functions $f : \mathbb{F}^n \to \mathbb{F}$. Indeed, both sets have size exactly $|\mathbb{F}|^{|\mathbb{F}|^n}$ and Lagrange interpolation shows that for any function $f : \mathbb{F}^n \to \mathbb{F}$, there is a corresponding polynomial where the degree of every variable is less than $|\mathbb{F}|$.

Proof of Lemma 2.2. For notational convenience, let us assume that $V = [n]$. Let $\mathbb{F}$ be a field of size 2^{k+2}. Let A be the Tutte matrix corresponding to G, that is, an element $a_{i,j}$ is defined as

$$a_{i,j} := \begin{cases} x_{i,j} & \text{if } i \text{ and } j \text{ are adjacent and } i < j, \\ -x_{i,j} & \text{if } i \text{ and } j \text{ are adjacent and } i > j, \\ 0 & \text{if } i \text{ and } j \text{ are not adjacent.} \end{cases}$$

We consider each entry of A as an n^2-variate polynomial from $\mathbb{F}[x_{1,1}, \dots, x_{n,n}]$ (note that we use the variables $x_{i,i}$ for $1 \le i \le n$ in there just for notational convenience). For $X \subseteq [n]$, we denote by $A[X]$ the principal submatrix of A corresponding to the rows and columns described by X. It is well known that $G[X]$ has a perfect matching if and only if $\det(A[X])$ is a nonzero polynomial (similar to Observation 2.3). Consider $S \subseteq [k]$ and let us denote by S^c the complement of S, i.e., the set of rows/columns *not* indexed by S. Thus, for every $S \subseteq [k]$, we have

$$G - S \text{ has a perfect matching } \Leftrightarrow \det(A[S^c]) \text{ is nonzero.} \qquad (\star)$$

We would like to obtain a matrix A' with the same property $(\star)$, but over $\mathbb{F}$ (so the elements of A' are *not* polynomials). Let $S_1, \dots, S_t$ be the subsets of $[k]$ such that $G - S$ has a perfect matching. For $\ell \in [t]$, the polynomial $P_\ell = \det(A[S_\ell^c])$ is a nonzero polynomial where every variable $x_{i,j}$ has degree at most 2 (as every variable appears at most twice in the Tutte matrix). The product $P = \prod_{\ell=1}^{t} P_\ell$ is also a nonzero polynomial (as $\mathbb{F}$ is a field, $\mathbb{F}[x_{1,1}, \dots, x_{n,n}]$ is an integral domain) where every variable has degree at most $2t \le 2 \cdot 2^k < |\mathbb{F}|$. Therefore, Observation 2.3 implies that P is not vanishing. This means that we can substitute values to the variables such that P evaluates to a nonzero value, which also means that every P_ℓ is nonzero under this substitution. Let A' be obtained from A by this substitution. It is easy to see that this matrix A' over $\mathbb{F}$ has the desired property $(\star)$: if S has no perfect matchings, then $\det(A'[S^c]) = 0$ (as this already holds for A), while if S has a perfect matching, then the determinant is one of the polynomials P_ℓ, and hence it evaluates to a nonzero value under the substitution.

Now we argue that we can obtain a $k \times k$ matrix A'' that also has the property $(\star)$. Matrix A' represents the set system $\mathcal{X} = \{X \mid \det(A'[X]) \neq 0\}$ over $[n]$, which is known to be a delta-matroid. The set system $\mathcal{X}' = \{X \subseteq [k] \mid X \cup ([n] \setminus [k]) \in \mathcal{X}\}$ is a set system over $[k]$, which is called the *contraction* of $\mathcal{X}$. It is known (see, e.g., Wahlström [67, Section 5.2, Theorem 11]) that given some $[n] \times [n]$ matrix A' over $\mathbb{F}$ representing $\mathcal{X}$, we can use algebraic operations to compute a $k \times k$ matrix A'' representing $\mathcal{X}'$. Now we can verify that this matrix A'' indeed satisfies property $(\star)$.

As A'' has property $(\star)$ we can deduce from A'' the value of the function f for any $S \subseteq [k]$. As A'' is an $k \times k$ matrix over a field of size 2^{k+2}, it can be described by $k^2(k+2)$ bits. Therefore, $\mathcal{O}(k^3)$ bits are sufficient to describe the function $h_{G,k}$, that is, there are at most $2^{\mathcal{O}(k^3)}$ such functions. ◀

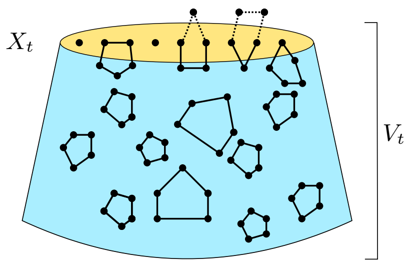

Figure 2 If H is not a clique (e.g., H is a cycle on 5 vertices), then restricting a packing to V_t may result in partial copies of H that contain vertices from $V_t \setminus X_t$. Therefore, the description of a partial packing needs to include how these partial copies interact with X_t.

$2^{2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$-Time Algorithm for H-HitPack for Arbitrary Connected H. The dynamic programming approach becomes significantly more complicated if we generalize it to H-HitPack where H is not a clique. The main issue is that now it is no longer true that in every packing every copy of H intersecting V_t is either fully contained in V_t or intersects V_t only in X_t. As H is not a clique, it can be the case that a copy is split by X_t (see Figure 2). Therefore, we need to argue about *partial packings* in V_t that may contain some partial copies of H split by X_t. When reasoning about such a partial packing, we need to describe not only which vertices of X_t the partial packing covers, but also how the partial copies partition X_t. We formalize this intuitive idea by the notion of *types*. The type of a (partial) packing contains the following information:

- A set of those vertices in X_t that are not covered by the packing including the vertices which are deleted.
- A partition of the vertices in X_t describing which vertices contribute to the same copy of H.
- For each part of the partition, a mapping between the vertices of X_t and the vertices of the partial copy of H, so we can determine which vertices of H have been packed and which vertices of H still need to be packed.

Clearly there are at most $2^{\mathsf{tw}+1}$ choices for the set of uncovered vertices. Since we consider a fixed graph H, the precise mapping for each of the at most $\mathsf{tw}+1$ parts can be described by $|V(H)|^{\mathsf{tw}+1}$ possible functions although involving a significant notational overhead in the formal description. However, the partition of the vertices into the different parts dominates the number of possible types for which there are $\mathsf{tw}^{\mathcal{O}(\mathsf{tw})}$ possibilities for each node. Due to this larger number of types, the running time of the algorithm involves an additional logarithmic factor which we avoid for the case when H is a clique.

Similar to the algorithm when packing cliques, we also have to remember the size of the largest packing for each type. As this number could potentially range from 0 to ℓ, a naive bound for the number of possible states is $\ell^{\mathsf{tw}^{\mathcal{O}(\mathsf{tw})}}$ which does not depend on the treewidth only. However, since each graph H has only a fixed size, the packing number for two different types cannot differ by too much. Indeed, the size of two optimal partial packings with different types can differ by at most $\mathcal{O}(|H| \cdot \mathsf{tw})$ as each partial packing of H can "block" at most $|H|$ vertices from being packed in the other packing. This observation drastically reduces the number of states for each node to $\mathsf{tw}^{\mathsf{tw}^{\mathcal{O}(\mathsf{tw})}}$ which then determines the running time of the algorithm.

$2^{2^{\mathcal{O}(\mathrm{tw}\log \mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$-Time Algorithm for Cycle-HitPack. For Cycle-HitPack the situation changes slightly. Since we are not dealing with a fixed graph H but an infinitely sized family of graphs, we cannot decide a priori which graphs are packed where; meaning we cannot determine the length of the cycle. Instead, we exploit that the family of graphs comprises *all* cycles and therefore, it does not matter in which cycle a vertex appears. Hence, the type of a partial cycle-packing can be described by the following four-tuple:

- A set of vertices that are not covered by the packing which includes the deleted vertices.
- A set of vertices that have one incident edge in the packing. These vertices are the endpoints of a path (i.e., incomplete cycle) in the partial cycle-packing.
- A set with the (remaining) vertices which have two incident edges in the packing, that is, these vertices are already fully covered (although the corresponding cycle might not be closed yet).
- A perfect matching for the vertices with one incident edge. This perfect matching describes which vertices form the endpoints of the same path. Phrased differently, for each path in the partial cycle-packing (which corresponds to a cycle that is not closed yet), we just remember the tuple consisting of the two endpoints. This information is already sufficient to combine two paths to form a new (longer) path or to close a cycle.

When additionally also discarding those packings with too few cycles inside, the total number of subproblems at each node can be bounded by $2^{2^{\mathcal{O}(\mathrm{tw}\log \mathrm{tw})}} \cdot n$.

2.2 Lower Bounds

Σ_2^{P}-completeness of Triangle-HitPack. The containment in Σ_2^{P} for Triangle-HitPack is clear, as we can guess the set of deleted vertices and use an NP-oracle to determine the size of the maximum packing. To establish the Σ_2^{P}-hardness, we reduce from a special Σ_2^{P}-complete satisfiability problem, the Smallest Unsatisfiable Subformula problem (SUS) [64, 66]. For SUS the input is a CNF-formula φ together with a parameter k and the task is to decide if there *exists* a collection of at most k clauses of φ (we refer to this as subformula) such that, *for all* assignments, the subformula is not satisfied, i.e., the subformula is unsatisfiable. This "exists, for all" formulation of SUS already gives a hint for the reduction to Triangle-HitPack.

We construct a graph consisting of clause gadgets, variable gadgets, and literal edges. Each of the clause gadgets consists of a single triangle with a distinguished deletable vertex. The interpretation of this vertex is as follows: a clause is selected as part of the subformula if the corresponding vertex is deleted, and otherwise, the clause is in the default state of not being selected. Each variable gadget consists of a cycle of triangles that has exactly two maximum triangle packings, which leave two disjoint sets of vertices uncovered. One option corresponds to setting the variable to true and the other to setting the variable to false. As a last component the graph contains literal edges. There is a literal edge between a variable gadget and a clause gadget if a corresponding literal occurs in the corresponding clauses. The idea is that the deletion of deletable vertices does not decrease the size of a maximum packing of the entire graph if and only the subformula consisting of the clauses selected by these deletions is still satisfiable.

To strengthen the result, we prove the hardness for tripartite graphs by reducing from 3CNF-SUS (the restriction of SUS to 3CNF-formulas), for which we also provide the Σ_2^{P}-completeness as, to our knowledge, no proof appeared in the literature, although the completeness was claimed.

Σ_2^{P}-completeness of H-HitPack. For connected graphs H other than the triangle, the Σ_2^{P}-completeness result can be obtained from Triangle-HitPack by a clean reduction. Kirkpatrick and Hell [37] showed how to reduce a triangle packing problem to arbitrary H-packing problems for connected H with at least 3 vertices. However, they considered the problem of finding a packing that covers every vertex of the graph and the arguments do not readily work for problems where not every vertex needs to be covered. Nevertheless, we show that with additional arguments and by extending their construction, a reduction can be obtained from Triangle-HitPack to H-HitPack, showing the Σ_2^{P}-completeness of the latter problem.

Σ_2^{P}-completeness of Cycle-HitPack. This hardness proof is obtained by observing that in the Σ_2^{P}-completeness proof of Triangle-HitPack, every cycle relevant for a packing is a triangle.

Double-exponential Lower Bounds for H-HitPack Parameterized by Treewidth. We reduce from an instance of 3-SAT with n variables and m clauses to achieve a double-exponential lower bound by constructing a graph with pathwidth $\mathcal{O}(\log m)$. Although the starting point is again a 3CNF-formula as for the Σ_2^{P}-hardness, the interpretation and the basic ideas differ because of the change in the quantification. For SUS the task is to check if there *exists* a set of *clauses* such that, *for all assignments* to the variables, the formula is not satisfiable. For 3-SAT the task is to check if there *exists an assignment* for the variables such that *for all clauses* at least one literal of the clause is satisfied, i.e., for each clause not all three literals are false. While as before, we use gadgets consisting of a long cycle with attached triangles, the overall construction of how these gadgets interact is substantially different. The critical difference to the Σ_2^{P}-hardness proof lies in the way we verify the satisfying assignment. Here, we do not explicitly construct a different gadget for every clause as this would result in a construction with treewidth linear in the number of clauses.

Our constructed instance comprises three parts, which we refer to as left, middle, and right. The deleted vertices on the right correspond to a choice of the satisfying assignment. The left part consists of the so-called *selector gadgets*. Each selector gadget models one bit of the binary encoding of some clause. So, if there are m clauses, we have roughly $\log m$ selector gadgets. For each selector gadget we introduce three pairs of vertices in the middle. Intuitively, there are three pairs since each clause contains three literals, and there are pairs to encode whether the bit corresponding to this selector gadget is 0 or 1. Then a packing of the vertices on the left, i.e., a packing for the selector gadgets, can interact in m different ways with the $\mathcal{O}(\log(m))$ vertices in the middle. Each of these possibilities corresponds to a different selection of (the encoding of) a clause.

We verify the satisfying assignment by ensuring that no matter how we might choose a maximum packing on the left (i.e., no matter which clause we look at), the maximum packing on the right is small. This corresponds to verifying that each clause is satisfiable. A crucial difference to the Σ_2^{P}-hardness proof lies in the fact that, here, the small packing is ensured by the variable gadgets (if a solution exists) and not by the clause gadget. The treewidth of the construction is $\mathcal{O}(\log(m))$ since this is the number of vertices with which the gadgets interact (and each gadget separately has constant treewidth).

Double-exponential Lower Bound for Square-HitPack and Cycle-HitPack Parameterized by Treewidth. We first prove the lower bound for Square-HitPack and then extend it to Cycle-HitPack as follows. We define the reduction for Square-HitPack in a way such that whenever a cycle packing $\mathcal{P}$ contains a *large* cycle of length at least five, then we can repack $\mathcal{P}$ to obtain a packing $\mathcal{P}'$ by replacing this large cycle with a C_4. This keeps the number of cycles the same (or might actually increase it) while reducing the number of large cycles. Hence, the maximum possible cycle packing only contains cycles of length four.

The basic high-level idea for the lower bound for Square-HitPack is similar to the previous one for the general case. However, instead of having a middle part with $\mathcal{O}(\log m)$ vertices (which dominates the pathwidth), we have to find a different way of encoding the clause numbers by only using $\mathcal{O}(\log m/\log\log m)$ vertices.

To make this possible, we have to change the way the gadgets on the left and right side interact with the vertices in the middle. For the previous construction the gadgets only cover a single vertex while the remaining part of the triangle is entirely contained either in the left or the right half. When packing four-cycles we can change this and allow that those cycles which contain vertices in the middle have exactly one vertex on the left side and exactly one vertex on the right side. With this method, the position of the cycle can be described by a matching for the vertices in the middle. The idea is to introduce two groups of t vertices each where $t! \approx m$. Then there are $t!$ possible perfect matchings between the vertices from the first group and the vertices from the second group. With this it is possible to associate with each clause a unique perfect matching on these vertices. The gadgets on the left and right side are then adjusted such that they connect each pair of vertices from the matching by a path of length two.

Since choosing $t \approx \log m/\log\log m$ satisfies the above property, the number of vertices in the middle part is bounded by $\mathcal{O}(\log m/\log\log m)$ which then determines the pathwidth of the graph.

References

1 Isolde Adler, Martin Grohe, and Stephan Kreutzer. Computing excluded minors. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008*, pages 641–650. SIAM, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347153.

2 Akanksha Agrawal, Daniel Lokshtanov, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Feedback Vertex Set Inspired Kernel for Chordal Vertex Deletion. *ACM Trans. Algorithms*, 15(1):11:1–11:28, 2019. doi:10.1145/3284356.

3 Noga Alon, Raphael Yuster, and Uri Zwick. Color-Coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

4 Jørgen Bang-Jensen, Alessandro Maddaloni, and Saket Saurabh. Algorithms and Kernels for Feedback Set Problems in Generalizations of Tournaments. *Algorithmica*, 76(2):320–343, 2016. doi:10.1007/s00453-015-0038-2.

5 Stéphane Bessy, Marin Bougeret, R. Krithika, Abhishek Sahu, Saket Saurabh, Jocelyn Thiebaut, and Meirav Zehavi. Packing Arc-Disjoint Cycles in Tournaments. *Algorithmica*, 83(5):1393–1420, 2021. doi:10.1007/s00453-020-00788-2.

6 Stéphane Bessy, Fedor V. Fomin, Serge Gaspers, Christophe Paul, Anthony Perez, Saket Saurabh, and Stéphan Thomassé. Kernels for feedback arc set in tournaments. *J. Comput. Syst. Sci.*, 77(6):1071–1078, 2011. doi:10.1016/j.jcss.2010.10.001.

7 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.

8 Glencora Borradaile and Hung Le. Optimal Dynamic Program for r-Domination Problems over Tree Decompositions. In *11th International Symposium on Parameterized and Exact Computation, IPEC 2016*, volume 63 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.8.

9 Yixin Cao. A Naive Algorithm for Feedback Vertex Set. In *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018*, volume 61 of *OASIcs*, pages 1:1–1:9. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/OASIcs.SOSA.2018.1.

10 Yixin Cao and Dániel Marx. Chordal Editing is Fixed-Parameter Tractable. *Algorithmica*, 75(1):118–137, 2016. doi:10.1007/s00453-015-0014-x.

11 Maria Chudnovsky, Alexandra Ovetsky Fradkin, and Paul D. Seymour. Tournament immersion and cutwidth. *J. Comb. Theory, Ser. B*, 102(1):93–101, 2012. doi:10.1016/j.jctb.2011.05.001.

12 Maria Chudnovsky, Alex Scott, and Paul D. Seymour. Disjoint paths in unions of tournaments. *J. Comb. Theory, Ser. B*, 135:238–255, 2019. doi:10.1016/j.jctb.2018.08.007.

13 Radu Curticapean, Nathan Lindzey, and Jesper Nederlof. A Tight Lower Bound for Counting Hamiltonian Cycles via Matrix Rank. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1080–1099. SIAM, 2018. doi:10.1137/1.9781611975031.70.

14 Radu Curticapean and Dániel Marx. Tight conditional lower bounds for counting perfect matchings on graphs of bounded treewidth, cliquewidth, and genus. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 1650–1669. SIAM, 2016. doi:10.1137/1.9781611974331.ch113.

15 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving Connectivity Problems Parameterized by Treewidth in Single Exponential Time. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

16 Frank K. H. A. Dehne, Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Kim Stevens. An $2^{O(k)}n^3$ FPT Algorithm for the Undirected Feedback Vertex Set Problem. *Theory Comput. Syst.*, 41(3):479–492, 2007. doi:10.1007/s00224-007-1345-z.

17 Michael Dom, Jiong Guo, Falk Hüffner, Rolf Niedermeier, and Anke Truß. Fixed-parameter tractability results for feedback set problems in tournaments. *J. Discrete Algorithms*, 8(1):76–86, 2010. doi:10.1016/j.jda.2009.08.001.

18 Dorit Dor and Michael Tarsi. Graph decomposition is np-complete: A complete proof of holyer's conjecture. *SIAM J. Comput.*, 26(4):1166–1187, 1997. doi:10.1137/S0097539792229507.

19 László Egri, Dániel Marx, and Paweł Rzążewski. Finding List Homomorphisms from Bounded-treewidth Graphs to Reflexive Graphs: a Complete Complexity Characterization. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018*, volume 96 of *LIPIcs*, pages 27:1–27:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.27.

20 P. Erdős and L. Pósa. On independent circuits contained in a graph. *Canad. J. Math.*, 17:347–352, 1965.

21 Michael R. Fellows and Michael A. Langston. On search, decision and the efficiency of polynomial-time algorithms (extended abstract). In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989*, pages 501–512. ACM, 1989. doi:10.1145/73007.73055.

22 Jacob Focke, Fabian Frei, Shaohua Li, Dániel Marx, Philipp Schepper, Roohani Sharma, and Karol Węgrzycki. Hitting meets packing: How hard can it be? *CoRR*, abs/2402.14927, 2024. doi:10.48550/arXiv.2402.14927.

23 Jacob Focke, Dániel Marx, Fionn Mc Inerney, Daniel Neuen, Govind S. Sankar, Philipp Schepper, and Philip Wellnitz. Tight Complexity Bounds for Counting Generalized Dominating Sets in Bounded-Treewidth Graphs. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 3664–3683. SIAM, 2023. doi:10.1137/1.9781611977554.ch140.

24 Jacob Focke, Dániel Marx, and Paweł Rzążewski. Counting list homomorphisms from graphs of bounded treewidth: tight complexity bounds. In *Proceedings of the 2022 ACM-SIAM*

Symposium on Discrete Algorithms, SODA 2022, pages 431–458. SIAM, 2022. doi:10.1137/1.9781611977073.22.

25 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient Computation of Representative Families with Applications in Parameterized and Exact Algorithms. *J. ACM*, 63(4):29:1–29:60, 2016. doi:10.1145/2886094.

26 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Hitting topological minors is FPT. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 1317–1326. ACM, 2020. doi:10.1145/3357713.3384318.

27 Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Fast Local Search Algorithm for Weighted Feedback Arc Set in Tournaments. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*. AAAI Press, 2010. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1829, doi:10.1609/AAAI.V24I1.7557.

28 Florent Foucaud, Esther Galby, Liana Khazaliya, Shaohua Li, Fionn Mc Inerney, Roohani Sharma, and Prafullkumar Tale. Problems in np can admit double-exponential lower bounds when parameterized by treewidth or vertex cover, 2024. To appear at ICALP 2024. arXiv:2307.08149.

29 Fabian Frei, Edith Hemaspaandra, and Jörg Rothe. Complexity of stability. *J. Comput. Syst. Sci.*, 123:103–121, 2022. doi:10.1016/J.JCSS.2021.07.001.

30 M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

31 Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *J. Comput. Syst. Sci.*, 72(8):1386–1396, 2006. doi:10.1016/j.jcss.2006.02.001.

32 Bart M. P. Jansen and Marcin Pilipczuk. Approximation and Kernelization for Chordal Vertex Deletion. *SIAM J. Discret. Math.*, 32(3):2258–2301, 2018. doi:10.1137/17M112035X.

33 Naonori Kakimura, Ken-ichi Kawarabayashi, and Dániel Marx. Packing cycles through prescribed vertices. *J. Comb. Theory, Ser. B*, 101(5):378–381, 2011. doi:10.1016/j.jctb.2011.03.004.

34 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discret. Appl. Math.*, 264:90–117, 2019. doi:10.1016/j.dam.2018.11.002.

35 Ken-ichi Kawarabayashi and Bruce A. Reed. Odd cycle packing. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 695–704. ACM, 2010. doi:10.1145/1806689.1806785.

36 Ken-ichi Kawarabayashi, Bruce A. Reed, and Paul Wollan. The Graph Minor Algorithm with Parity Conditions. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 27–36. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.52.

37 David G. Kirkpatrick and Pavol Hell. On the Complexity of General Graph Factor Problems. *SIAM J. Comput.*, 12(3):601–609, 1983. doi:10.1137/0212040.

38 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic Feedback Vertex Set. *Inf. Process. Lett.*, 114(10):556–560, 2014. doi:10.1016/j.ipl.2014.05.001.

39 Mithilesh Kumar and Daniel Lokshtanov. Faster Exact and Parameterized Algorithm for Feedback Vertex Set in Tournaments. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016*, volume 47 of *LIPIcs*, pages 49:1–49:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.49.

40 Michael Lampis. Model Checking Lower Bounds for Simple Graphs. *Log. Methods Comput. Sci.*, 10(1), 2014. doi:10.2168/LMCS-10(1:18)2014.

41 Michael Lampis, Stefan Mengel, and Valia Mitsou. QBF as an alternative to courcelle's theorem. In *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018*, volume 10929 of *Lecture Notes in Computer Science*, pages 235–252. Springer, 2018. doi:10.1007/978-3-319-94144-8_15.

42 Michael Lampis and Valia Mitsou. Treewidth with a Quantifier Alternation Revisited. In *12th International Symposium on Parameterized and Exact Computation, IPEC 2017*, volume 89 of *LIPIcs*, pages 26:1–26:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.IPEC.2017.26.

43 John M. Lewis and Mihalis Yannakakis. The Node-Deletion Problem for Hereditary Properties is NP-Complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

44 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

45 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly Superexponential Parameterized Problems. *SIAM J. Comput.*, 47(3):675–702, 2018. doi:10.1137/16M1104834.

46 Daniel Lokshtanov, Pranabendu Misra, Joydeep Mukherjee, Fahad Panolan, Geevarghese Philip, and Saket Saurabh. 2-Approximating Feedback Vertex Set in Tournaments. *ACM Trans. Algorithms*, 17(2):11:1–11:14, 2021. doi:10.1145/3446969.

47 Daniel Lokshtanov, Amer E. Mouawad, Saket Saurabh, and Meirav Zehavi. Packing Cycles Faster Than Erdős Pósa. *SIAM J. Discret. Math.*, 33(3):1194–1215, 2019. doi:10.1137/17M1150037.

48 Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster Parameterized Algorithms Using Linear Programming. *ACM Trans. Algorithms*, 11(2):15:1–15:31, 2014. doi:10.1145/2566616.

49 Dániel Marx. Chordal Deletion is Fixed-Parameter Tractable. *Algorithmica*, 57(4):747–768, 2010. doi:10.1007/s00453-008-9233-8.

50 Dániel Marx. Chordless Cycle Packing Is Fixed-Parameter Tractable. In *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020*, volume 173 of *LIPIcs*, pages 71:1–71:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.71.

51 Dániel Marx and Valia Mitsou. Double-Exponential and Triple-Exponential Bounds for Choosability Problems Parameterized by Treewidth. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 28:1–28:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.28.

52 Dániel Marx, Govind S. Sankar, and Philipp Schepper. Degrees and Gaps: Tight Complexity Results of General Factor Problems Parameterized by Treewidth and Cutwidth. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 95:1–95:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.95.

53 Dániel Marx, Govind S. Sankar, and Philipp Schepper. Anti-Factor Is FPT Parameterized by Treewidth and List Size (But Counting Is Hard). In *17th International Symposium on Parameterized and Exact Computation, IPEC 2022*, volume 249 of *LIPIcs*, pages 22:1–22:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.IPEC.2022.22.

54 Pranabendu Misra, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. A Polynomial Kernel for Feedback Arc Set on Bipartite Tournaments. *Theory Comput. Syst.*, 53(4):609–620, 2013. doi:10.1007/s00224-013-9453-4.

55 Karolina Okrasa, Marta Piecyk, and Paweł Rzążewski. Full Complexity Classification of the List Homomorphism Problem for Bounded-Treewidth Graphs. In *28th Annual European Symposium on Algorithms, ESA 2020*, volume 173 of *LIPIcs*, pages 74:1–74:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.74.

56 Karolina Okrasa and Paweł Rzążewski. Fine-Grained Complexity of the Graph Homomorphism Problem for Bounded-Treewidth Graphs. *SIAM J. Comput.*, 50(2):487–508, 2021. doi:10.1137/20M1320146.

57 Guoqiang Pan and Moshe Y. Vardi. Fixed-Parameter Hierarchies inside PSPACE. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 27–36. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.25.

58 M. Pontecorvi and Paul Wollan. Disjoint cycles intersecting a set of vertices. *J. Comb. Theory, Ser. B*, 102(5):1134–1141, 2012. doi:10.1016/j.jctb.2012.05.004.

59 Venkatesh Raman and Saket Saurabh. Parameterized algorithms for feedback set problems and their duals in tournaments. *Theor. Comput. Sci.*, 351(3):446–458, 2006. doi:10.1016/j.tcs.2005.10.010.

60 M. S. Ramanujan and Saket Saurabh. Linear-Time Parameterized Algorithms via Skew-Symmetric Multicuts. *ACM Trans. Algorithms*, 13(4):46:1–46:25, 2017. doi:10.1145/3128600.

61 Bruce A. Reed. Mangoes and Blueberries. *Combinatorica*, 19(2):267–296, 1999. doi:10.1007/s004930050056.

62 Bruce A. Reed, Neil Robertson, Paul D. Seymour, and Robin Thomas. Packing Directed Circuits. *Combinatorica*, 16(4):535–554, 1996. doi:10.1007/BF01271272.

63 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004. doi:10.1016/j.orl.2003.10.009.

64 Marcus Schaefer and Christopher Umans. SIGACT news complexity theory column 37. Guest column: Completeness in the polynomial-time hierarchy: Part I: A compendium. *SIGACT News*, 33(3):32–49, 2002. doi:10.1145/582475.582484.

65 S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

66 Christopher Umans. Hardness of Approximating Sigma2P Minimization Problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS 99*, pages 465–474. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814619.

67 Magnus Wahlström. Representative set statements for delta-matroids and the mader delta-matroid. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 780–810. SIAM, 2024.

68 Mingyu Xiao and Jiong Guo. A Quadratic Vertex Kernel for Feedback Arc Set in Bipartite Tournaments. *Algorithmica*, 71(1):87–97, 2015. doi:10.1007/s00453-013-9783-2.