Algorithms and Lower Bounds for Finding (Exact-Weight) Subgraphs of Bounded Treewidth

Karl Bringmann
Saarland University, Germany
Max Planck Institute for Informatics, Germany

Jasper C. P. Slusallek
Saarland University, Germany

Abstract

The Subgraph Isomorphism problem is of considerable importance in computer science. We examine the problem when the pattern graph H is of bounded treewidth, as occurs in a variety of applications. This problem has a known algorithm running in time $O(n^{tw(H)+1})$. By studying a colored variant of the problem, we show that assuming the hyperclique hypothesis, no algorithm with running time $O(n^{tw(H)+1-\varepsilon})$ can exist for any $\varepsilon > 0$. This is the first tight conditional lower bound for this problem. Moreover, we will explore a weighted variant where the solution subgraph must be of total weight zero. Allowing the largest absolute weight to appear in the running time, we show similar tight conditional lower bounds in almost all cases. In the process, we also immediately get a reduction that shows conditional lower bounds for Subset Sum.

On the algorithmic side, we first analyze the unweighted variant and use so-called k-wise matrix products (a generalization of the standard matrix product to tensors) to unify existing algorithms with a single technique, while still matching their running times. We then expand the algorithms to the weighted variant, improving on the best-known naive dynamic programming algorithm.

We also show similar results for the case of bounded pathwidth, with slight algorithmic improvements for both the weighted and the unweighted case using rectangular matrix multiplication. In this setting, we also show still further improvements for the node-weighted case.

Acknowledgements When it comes to this Bachelor thesis, my thanks goes first and foremost to Karl for his excellent guidance in writing this thesis, for his incredibly detailed and helpful feedback on a draft, for his patient answers to all my questions, and for co-teaching the two courses “Multivariate Algorithmics” and “Fine-Grained Complexity Theory” that started my fascination with the subject matter. Furthermore, I would like to thank Markus Bläser, for answering my questions about matrix multiplication, and Dániel Marx, for agreeing to review the thesis.
I also want to thank Alejandro, for insightful discussions on all things Computer Science, including some about possible extensions of this thesis; Simon, for introducing me to the one and only “Silkesport”, which kept me sane; Moritz, for introducing me to the work of Alastair Reynolds, which helped me procrastinate more than I ever wanted to; and Alex and Karl S., for not getting fed up with me for not helping with exercise sheets during the completion of my thesis.

Paper Forthcoming This thesis will be turned into a paper very shortly, and we are planning significant simplifications of the algorithms, as well as the addition of some new and related results. It will also contain more figures, since I still need to port existing figures from an earlier version of the lower bounds. If for some reason you are reading this document, and you are neither a reviewer of the thesis nor really into making your life more complicated than it needs to be, you almost certainly want to be reading that paper instead.

Contents

1 Introduction

The Subgraph Isomorphism problem is commonly defined as follows: Given a graph H on k vertices, and a graph G on n vertices, is there a (not necessarily induced) subgraph of G which is isomorphic to H?

Subgraph Isomorphism generalizes many problems of independent interest, such as the k-path and k-clique problems. The problem is also of considerable interest when H is less structured, with applications to discovering patterns in graphs that, for example, arise from biological processes such as gene transcription or food networks, from social interaction, from electronic circuits, from neural networks [73], from chemical compounds [82] or from control flow in programs [33]. In some fields, the problem is sometimes referred to as the search for "network motifs", i.e. subgraphs that appear more often than would normally be expected.

In its general form, the problem is NP-hard. We are interested in solving the problem when the pattern graph H is "tree-like" or "path-like", i.e. when the treewidth $\mathrm{tw}(H)$ or the pathwidth $\mathrm{pw}(H)$ of H is bounded. Such pattern graphs of low treewidth or pathwidth often arise in practice when considering the structure of chemical compounds, the control flow of programs, syntactic relations in natural language, or many other graphs from practical applications (see e.g. [22, 25]). On the theoretical side, many restricted classes of graphs have bounded treewidth, see also [24]. Restricting NP-hard problems to graphs of bounded tree- and pathwidth often yields polynomial-time algorithms, and Subgraph Isomorphism is no exception. Most notably, there is the classic Color-Coding algorithm by Alon, Yuster and Zwick [15] which shows that the problem can be solved by a Las Vegas algorithm with expected running time $O(n^{\mathrm{tw}(H)+1}g(k))$, or by a deterministic algorithm with running time $O(n^{\mathrm{tw}(H)+1}\log(n)g(k))$, where g is a computable function. In other words, if the input is restricted to pattern graphs H with treewidth at most some constant, the problem is fixed-parameter tractable when parameterized by k. The Color-Coding algorithm is also relevant for practical purposes: Recently, it has received an efficient implementation, which tested rather competitively against state-of-the-art programs for Subgraph Isomorphism on public data sets [70].

In one of our results, we explore whether the polynomial factor $n^{\mathrm{tw}(H)+1}\log(n)$ of the Color-Coding algorithm can be improved. Marx [71] already showed that for any class of graphs of unbounded treewidth, there cannot be an algorithm solving Subgraph Isomorphism for instances where the pattern graph is from that class in time $O(n^{o(\mathrm{tw}(H)/\log(\mathrm{tw}(H)))}g(k))$, unless the Exponential Time hypothesis (ETH) fails. One of our contributions is that there exists a class of graphs of unbounded treewidth such that for $\mathrm{tw}(H) \geq 3$, there cannot be an algorithm with running time $O(n^{\mathrm{tw}(H)+1-\varepsilon}g(k))$ for any $\varepsilon > 0$ for graphs from that class, unless the hyperclique problem can be solved significantly faster than currently possible. Hence, under the assumption that the algorithms for hyperclique cannot be significantly improved, the algorithm by Alon, Yuster and Zwick also cannot be significantly improved and still work on all graph classes.

For the case of $\mathrm{tw}(H) = 2$, an algorithm of Curticapean, Dell and Marx [43] can be adapted such that it solves the problem in time $O(n^{\omega}\log(n)g(k))$. We unify this with the algorithm of Alon, Yuster and Zwick by showing that both time bounds can be achieved with a simple framework. In particular, we use so-called k-wise matrix products, an operation which was introduced in its general form in [56] and studied further in [68]. Indeed, that operation is the bottleneck for both of our main algorithms. Unfortunately, for $k \geq 3$, it is known that the usual Strassen-like techniques that are used for fast matrix multiplication cannot be applied, and hence improvements over the naive running time of $O(n^{k+1})$ seem far out of reach [68].

We also look at the problem when the pathwidth of H is bounded, and specialize the framework to show slight improvements in running time with respect to the case of bounded treewidth. Here, we use rectangular matrix products, for which faster-than-naive algorithms are known [54].

In all further results, our focus is on a weighted variant of the problem, where the subgraph must also have total weight equal to zero. In this work, we consider both the node-weighted and the edge-weighted variant, for both bounded treewidth and bounded pathwidth, allowing the maximum absolute weight W to appear in the running time (i.e. we analyze pseudopolynomial-time algorithms for the problem). We show that our algorithms for the unweighted case can be adapted to the weighted case. We also speed up the weighted algorithms by using the fact that fast convolution (or rather, sumset computation), a folklore technique that lies at the core of many fast algorithms for problems with weights (e.g. [34, 28, 64, 62, 26, 20] and [41, exercise 30.1.7]), can be adapted to work with rectangular matrices and tensors. We furthermore show corresponding conditional lower bounds. Last but not least, we show that our algorithms can be slightly improved for the case of node-weighted instances for which either the pathwidth of H is bounded, or H is a tree. These algorithms also rely on fast rectangular matrix products.

All mentioned algorithms and conditional lower bounds are shown for the restricted problems of Colored Subgraph Isomorphism, where the nodes of G and H are colored with $|V(H)|$ colors and the isomorphism must preserve colors, as also studied in [71]. One of our first results is that for the type of algorithms we are considering, this problem is essentially equivalent to the general Subgraph Isomorphism problem.

2 Related Work

Consider algorithms for the unweighted problem. Next to Marx's [71] result from the introduction that a $O(n^{o(\mathrm{tw}(H)/\log(\mathrm{tw}(H)))}g(k))$ time algorithm is impossible for unbounded-treewidth classes under ETH, there is also an unconditional lower bound of $O(n^{\kappa(H)})$ for the size of a AC^0-circuit, for a graph parameter $\kappa(H) = \Omega(\mathrm{tw}(H)/\log(\mathrm{tw}(H)))$, which holds even when considering the average case [66]. Interestingly, the factor of $\frac{1}{\log(\mathrm{tw}(H))}$ does not seem to be an artefact of the proof: There is an AC^0-circuit of size $O(n^{o(\mathrm{tw}(H))}g(k))$ that solves the problem on certain unbounded-treewidth classes in the average case [79].

There have also been efforts to optimize the factor of $g(k)$ in the running-time of the Color-Coding algorithm instead of the polyonmial factor, see e.g. the $O(4.32^k \cdot k \cdot n^{\mathrm{tw}(H)+1})$ algorithm in [16]. When optimizing only $g(k)$, and permitting a worse polynomial factor, algorithms of running time $O^*(2^k n^{2\,\mathrm{tw}(H)})$ are known [53]. There is also a randomized algorithm that approximates the number of isomorphic subgraphs arbitrarily closely, with running time $O(4.075^k n^{\mathrm{tw}(H)+O(1)})$ [75].

Of particular interest is the problem of finding a tree pattern, sometimes called Subtree Isomorphism. For constant-size H, this problem has trivial upper and unconditional lower bounds of $\Theta(n^2)$. The situation is less clear when $k = O(\log n)$ or even $k = O(n)$. For an introduction and many pointers to relevant literature, see [1]. Some of their lower bounds are superseded by recent improvements in [36].

Our lower bound of no $O(n^{\mathrm{tw}(H)+1-\varepsilon})$ algorithm for the unweighted problem does not, of course, rule out faster algorithms for specific classes of graphs: Famously, finding a k-clique[1]

[1] i.e. a set of k vertices such that each pair is connected by an edge

– whose treewidth is $(k-1)$ – can be done in $O(n^{k\omega/3}g(k))$ for k divisible by three [74], with similar results for k not divisible by three [48]. The same paper shows that the same running time is possible for all pattern graphs H with k vertices.

Now consider the weighted problem. For stars, paths, cycles and other H, conditional lower bounds under the k-Sum hypothesis are presented in [8]. The nowadays well-known conditional lower bound of $O(n^{3-\varepsilon})$ for edge-weighted triangle finding under both the the 3Sum hypothesis and the APSP hypothesis[2] are proven in [12]. On the other hand, in [9], it is proven that finding node-weighted k-cliques can be done almost as quickly as finding unweighted k-cliques. So far, there seem to be no results on the Exact Weight Subgraph Isomorphism problem when W may appear in the running time (i.e. a pseudopolynomial-time algorithm), which is what we focus on in our analysis of the weighted problem.

For a large survey of Subgraph Isomorphism under various parameters including the treewidth of H, see Marx and Pilipczuk [72]. In our work, we pose no restrictions on the host graph G, but it should be noted that for some such restrictions, the problem becomes considerably easier. For example, by a simple application of Courcelle's theorem [42], when the treewidth of G is bounded and H is of constant size, the problem is in linear time. Furthermore, when G is planar and H is again of constant size, the problem can also be solved in linear time [50].

3 Results

We begin in Section 5 by showing that, when considering treewidth, Subgraph Isomorphism and Colored Subgraph Isomorphism are, both for the weighted and the unweighted version, equivalent. For the weighted variant, they are also equivalent when considering pathwidth. All subsequent results are then proven for Colored Subgraph Isomorphism, leading to algorithms and conditional lower bounds for Subgraph Isomorphism.

In Section 6, we present algorithms for unweighted and weighted Subgraph Isomorphism. We start with the unweighted variant for bounded treewidth and prove the upper bounds in Theorem 1 below. These results themselves are not new. Part 1 was shown in [15] and part 2 follows from techniques in [43]. We unify these two results by providing a single, relatively simple algorithmic technique achieving both, based on k-wise matrix products. In Theorem 2, we then expand the technique to also work for the weighted version, hence obtaining new results.

We say that an algorithm $\mathcal{A}$ runs in $T(n)$ **expected time** if $\mathcal{A}$ is a Las Vegas algorithm with expected running time $T(n)$. Furthermore, $\omega < 2.373$ [65] is the exponent of matrix multiplication. When we speak of the Exact Weight Subgraph Isomorphism problem, the instances may be either node- or edge-weighted, unless stated otherwise. The weight function is always denoted by w. For exact definitions of the problems and the hypotheses used in the theorem statements, we refer the reader to the preliminaries (Sections 4.3 and 4.4).

▶ **Theorem 1.** *There are algorithms which, given an arbitrary instance $\phi = (H, G)$ of Subgraph Isomorphism where H has treewidth $\mathrm{tw}(H)$, solve ϕ in*

1. *expected time $O(n^{\mathrm{tw}(H)+1}g(k))$ and time $O(n^{\mathrm{tw}(H)+1}\log(n)g(k))$ when $\mathrm{tw}(H) \geq 3$ ([15]),*
2. *expected time $O(n^{\omega}g(k))$ and time $O(n^{\omega}\log(n)g(k))$ when $\mathrm{tw}(H) = 2$ ([43]), and*
3. *expected time $O(n^2 g(k))$ and time $O(n^2\log(n)g(k))$ when $\mathrm{tw}(H) = 1$.*

where $k := |V(H)|, n := |V(G)|$ and g is a computable function.

[2] i.e. that the all-pairs shortest path problem cannot be solved in time $O(n^{3-\varepsilon})$.

▶ **Theorem 2.** *There are algorithms which, given an arbitrary instance $\phi = (H, G, w)$ of the* Exact Weight Subgraph Isomorphism *problem where H has treewidth $\mathrm{tw}(H)$, solve ϕ in*

1. *expected running time $O((n^{\mathrm{tw}(H)+1}W + n^{\mathrm{tw}(H)}W \log W)g(k))$ and running time $O((n^{\mathrm{tw}(H)+1}W + n^{\mathrm{tw}(H)}W \log W)\log(n)g(k))$ when $\mathrm{tw}(H) \geq 3$,*
2. *expected time $O((n^{\omega}W + n^2W \log W)g(k))$ and time $O((n^{\omega}W + n^2W \log W)\log(n)g(k))$ when $\mathrm{tw}(H) = 2$, or*
3. *expected time $O((n^2W + nW \log W)g(k))$ and time $O((n^2W + nW \log W)\log(n)g(k))$ when $\mathrm{tw}(H) = 1$,*

where $n := |V(G)|, k := |V(H)|$, g is a computable function, and W is the maximum absolute weight in the image of w.

We show later that these upper bounds are essentially tight. However, when instead of the treewidth, the pathwidth is bounded, we can use rectangular matrix multiplication to speed things up slightly. For $z \in \mathbb{R}^+$, let $\omega(z)$ be the smallest real number such that multiplying a $n \times n$ matrix with a $n \times n^z$ matrix can be done in time $O(n^{\omega(z)})$; see Section 4.7 for discussion of this value. We prove the following upper bounds.

▶ **Theorem 3.** *There are algorithms which, given an arbitrary instance $\phi = (H, G)$ of* Subgraph Isomorphism *where H has pathwidth p, solve ϕ in*

1. *expected time $O(n^{\omega(p-1)}g(k))$ and time $O(n^{\omega(p-1)}\log(n)g(k))$ when $p \geq 2$, and*
2. *expected time $O(n^2g(k))$ and time $O(n^2\log(n)g(k))$ when $p = 1$*

where $k := |H|$ and $n := |V(G)|$.

▶ **Theorem 4.** *There are algorithms which, given an arbitrary instance $\phi = (H, G, w)$ of the* Exact Weight Subgraph Isomorphism *problem, solve ϕ in*

1. *expected time $O((n^{\omega(\mathrm{pw}(H)-1)}W + n^{\mathrm{pw}(H)}W \log W)g(k))$ and time $O((n^{\omega(\mathrm{pw}(H)-1)}W + n^{\mathrm{pw}(H)}W \log W)\log(n)g(k))$ when $\mathrm{pw}(H) \geq 2$,*
2. *expected time $O((n^2W + nW \log W)g(k))$ and time $O((n^2W + nW \log W)\log(n)g(k))$ when $\mathrm{pw}(H) = 1$*

where $n := |V(G)|, k := |V(H)|$, and W is the maximum absolute weight in the image of w.

We conclude Section 6 by showing that for node-weighted instances of bounded pathwidth or of treewidth 1, the algorithms can be sped up slightly more. In particular, let $\mathrm{MM}(n, n, x)$ be the time in which a $n \times n$ matrix can be multiplied with with a $n \times x$ matrix. We show the following improvement.

▶ **Theorem 5.** *There are algorithms which, given an arbitrary instance $\phi = (H, G, w)$ of the node-weighted* Exact Weight Subgraph Isomorphism *problem where H is a tree, solve ϕ in expected time$O((\mathrm{MM}(n, n, W) + nW \log W)g(k))$ and time $O((\mathrm{MM}(n, n, W) + nW \log W)\log(n)g(k))$.*

▶ **Theorem 6.** *There are algorithms which, given an arbitrary instance $\phi = (H, G, w)$ of the node-weighted* Exact Weight Subgraph Isomorphism *problem, solve ϕ in expected time $O(\mathrm{MM}(n, n, n^{\mathrm{pw}(H)-1}W)g(k))$ and time $O(\mathrm{MM}(n, n, n^{\mathrm{pw}(H)-1}W)\log(n)g(k))$.*

For $W = O(n^{\gamma})$, the running time of Theorem 5 is $O(n^{\omega(\gamma)}\,\mathrm{poly}(k))$. Using results from [54] (see also Section 4.7), this implies several interesting facts. Firstly, for $\gamma < 0.31$, the node-weighted problem on trees can be solved in time $O(n^2\,\mathrm{poly}(k))$, meaning it can be solved in the same running time as the unweighted case. In particular, this applies to $W = \mathrm{polylog}(n)$ or $W = O(\sqrt[4]{n})$.

Secondly, for arbitrary γ we now have a running time of $O(n^{\omega(\gamma)-\gamma}W \operatorname{poly}(k))$. Trivially, for any $\gamma > 0$, $\omega(\gamma) - \gamma < 2$. Indeed, for $\gamma \geq 5$ we have $\omega(\gamma) - \gamma < 1.16$ by [54]. This shows that for node weighted trees, there cannot be a lower bound of $n^{1.16}W$, let alone n^2W. Indeed, it is known that $\lim_{\gamma\to\infty} \omega(\gamma) - \gamma = 1$ [39], which implies that when restricting to instances where $W = \Theta(n^\gamma)$, there cannot be a lower bound of $n^{1+\varepsilon}W$ for any $\varepsilon > 0$ that holds for any constant $\gamma > 0$. Similar results hold for bounded-pathwidth graphs.

In Section 7, we proceed to prove conditional lower bounds for unweighted and weighted Subgraph Isomorphism. We start with the case of bounded treewidth, and show the following obstacles to algorithms beating the running times from Theorems 1 and 2. The k-clique hypothesis states that the k-clique problem cannot be solved in time than $O(n^{k\omega/3-\varepsilon})$ for any $\varepsilon > 0$. For $h \geq 3$, the h-uniform hyperclique hypothesis states that for no $k > h$ can there be an algorithm which correctly determines if a h-uniform graph contains a h-uniform k-hyperclique[3] in time $O(n^{k-\varepsilon})$ for $\varepsilon > 0$.

▶ **Theorem 7.** *For each $t \in \mathbb{N}^*$ there is a connected,bipartite graph $\mathcal{H}_t$ of treewidth t such that there cannot be an algorithm which solves all instances from $\{(\mathcal{H}_t, G, f) \mid G$ is a graph$\}$ of Subgraph Isomorphism in time*

1. *$O(n^{t+1-\varepsilon})$ for $t \geq 3$, unless the h-uniform hyperclique hypothesis fails for all $3 \leq h \leq t$,*
2. *$O(n^{t-\varepsilon})$ for any $t \geq 2$, unless the h-uniform hyperclique hypothesis fails for all $h \geq 3$, or*
3. *$O(n^{(t+1)\omega/3-\varepsilon})$ for $t \geq 2$, unless the $(t+1)$-clique hypothesis fails.*

▶ **Corollary 8.** *For no $t \in \mathbb{N}$ can there be an algorithm solving Exact Weight Subgraph Isomorphism for pattern graphs of treewidth t as in parts 1-3 of Theorem 7.*

▶ **Theorem 9.** *For both the node- and edge weighted variant of the Exact Weight Subgraph Isomorphism problems, for any $t \in \mathbb{N}$ and any $\gamma \in \mathbb{R}^+$, there is a connected, bipartite graph $\mathcal{H}_{t,\gamma}$ of treewidth t such that there cannot be an algorithm which solves all instances from $\{(\mathcal{H}_{t,\gamma}, G, f, w) \mid G$ is a graph and $W := \max_{z\in \mathrm{Im}(w)} |z| = \Theta(n^\gamma)\}$ in time*

1. *$O(n^{t+1-\varepsilon}W)$ or $O(n^{t+1}W^{1-\varepsilon})$ for $t \geq 3$, unless the h-uniform hyperclique hypothesis fails for all $3 \leq h \leq t$,*
2. *$O(n^{t-\varepsilon}W)$ or $O(n^tW^{1-\varepsilon})$ for any t, unless the h-uniform hyperclique hypothesis fails for all $h \geq 3$, or*
3. *$O(n^{(t+1)\omega/3-\varepsilon}W^{\omega/3})$ or $O(n^{(t+1)\omega/3}W^{\omega/3-\varepsilon})$ for any t, unless the clique hypothesis fails.*

Note that item 1 only gives lower bounds for $t \geq 3$, while items 2 and 3 are mostly interesting for the case $t \in \{1, 2\}$.

▶ **Corollary 10.** *For no $t \in \mathbb{N}$ and no $\gamma \in \mathbb{R}^+$ can there be an algorithm solving Exact Weight Subgraph Isomorphism for pattern graphs of treewidth t and maximum weight $W = \Theta(n^\gamma)$ as in parts 1-3 of Theorem 9.*

Comparing these results with Theorems 1 and 2, we have tight lower bounds for all unweighted cases, and an essentially tight lower bound for the weighted case with $\mathrm{tw}(H) \geq 3$. For weighted $\mathrm{tw}(H) = 2$, we have a lower bound which is tight except for the exponent of $\omega/3$ to W – it is unclear whether this can be strengthened. The lower bound for weighted graphs with $\mathrm{tw}(H) = 1$ is not tight at all: We have an upper bound of $O(n^2W + nW \log W)$, but our

[3] i.e. a set of k vertices such that any size h subset of the vertices is connected by a hyperedge

lower bounds only state that it requires time $O(n^{1-o(1)}W^{1-o(1)})$ and $O(n^{2\omega/3-o(1)}W^{\omega/3-o(1)})$. Tighter lower bounds for this case remain an open problem.

With a very similar proof of the preceding theorems, we also get lower bounds for the case of bounded pathwidth for weighted instances. And while we furthermore get lower bounds for unweighted instances when considering Colored Subgraph Isomorphism, our equivalence lemma is not strong enough to hoist these results to the uncolored case. We leave it as an open problem whether this is even possible.

▶ **Theorem 11** (Theorem 9 for pathwidth). *Parts 2 and 3 of Theorem 9 also hold when replacing the treewidth t by the pathwidth p. Part 1 does not hold.*

▶ **Corollary 12.** *For no $p \in \mathbb{N}$ and no $\gamma \in \mathbb{R}^+$ can there be an algorithm solving Exact Weight Subgraph Isomorphism for pattern graphs of pathwidth p and maximum weight $W = \Theta(n^\gamma)$ as in parts 2 and 3 of Theorem 9 (when replacing the pathwidth p by the treewidth t).*

For $\text{pw}(H) \geq 3$, these lower bounds are obviously not tight, unless significant advances in matrix multiplication techniques are made. For $\text{pw}(H) = 1, 2$, the situation is the same as with treewidth.

It is natural to think that the exponents $\frac{\omega}{3}$ to W in the lower bounds of Theorems 9 and 11 are only artefacts of the reduction, and that with more advanced methods, this exponent can be improved to 1. However, Theorems 5 and 6 disprove this notion for $\text{tw}(H) = 1$ and $\text{pw}(H) = 1, 2$. Indeed, for $tw(H) = 1$ (or $pw(H) = 1$) and $W = n$, these bounds are tight, so further general improvements on the exponent are impossible.

As a related result, we also prove the following theorem about the Subset Sum problem, where one is given a set A of n numbers and a target T, and has to check whether some subset of A sums to T.

▶ **Theorem 13.** *For no $\varepsilon > 0$ can there be an algorithm which solves Subset Sum in time $O(T^{1-\varepsilon}\,\text{poly}(n))$ unless the h-uniform Hyperclique hypothesis fails for all $h \geq 3$.*

In [6], a slightly better lower bound for Subset Sum is proven under a stronger hypothesis: They prove that unless SETH fails, Subset Sum cannot have an algorithm running in time $O(T^{1-\varepsilon}2^{o(n)})$.

Let us quickly argue why SETH is actually a (much) stronger hypothesis – indeed, why the h-uniform Hyperclique hypothesis failing for all $h \geq 3$ has many important implications. There is a known reduction that shows that if the h-uniform Hyperclique hypothesis fails for some h, then Max-h-Sat[4] can be solved in time $O(2^{(1-\delta)n})$ for some $\delta > 0$ [83, 68]. Hence if it fails for all $h \geq 3$, Max-Sat can be solved in time $O(2^{(1-\delta)n})$ for any constant clause width. In particular, it also breaks the popular Max-3-Sat hypothesis, which states that no such algorithm exists for clause width 3. This would be a very strong result. Even if it were only true for sparse formulas, it would already imply via the Sparsification Lemma [58] that SETH is false.[5]

Note that the hypotheses we use are very much believable when looking at the current state of algorithms for these problems. For hyperclique, we are very far from any algorithm

[4] Given a CNF formula with clause width h and a number k, check if at least k clauses of the formula can be satisfied.

[5] Note that for sparse, constant clause width Max-Sat, the other direction is also true: If SETH is false, then Max-Sat with constant clause width can be solved faster-than-naively for sparse formulas. This is proven in [5] for the more general problem of deciding the satisfiability of linear-size threshold circuits of logarithmic depth with only AND, OR and NOT gates.

beating the hypothesized lower bound; while we do have an algorithm running in time $O(m^{k-1}/2^{\Theta(\sqrt{(\log n)})})$ where m is the number of clauses [68, 55], this only beats the naive algorithm for sparse (or nearly sparse) instances. Nothing significant is known for dense instances. And for Max-Sat: Again for sparse formulas, fast algorithms for Max-q-Sat with specific q have been proven [46, 37], but there are – to the best of our knowledge – no such results for non-sparse Max-q-Sat instances for any $q \geq 3$.

4 Preliminaries

4.1 General Notation and Nomenclature

We denote by $\mathbb{N}$ the set of positive integers. For $p \in \mathbb{N}$, we use $[p]$ to denote the set $\{1, \ldots, p\}$. For a statement or predicate P, we define the Iverson bracket

$$[P] := \begin{cases} 1 & \text{if P is true} \\ 0 & \text{otherwise} \end{cases}$$

For a fuction $f : \mathcal{A} \to \mathcal{B}$ and a set $S \subseteq \mathcal{A}$, we denote with $f|_S : S \to \mathcal{B}$ the function function f restricted to S. That is, $\forall s \in S : f|_S(s) = f(s)$. Furthermore, for $u \notin \mathcal{A}$ and $v \notin \mathcal{B}$ we define the function extension $(f \cup \{u \mapsto v\}) : \mathcal{A} \cup \{u\} \to \mathcal{B} \cup \{v\}$ as

$$(f \cup \{u \mapsto v\})(c) := \begin{cases} v & \text{if } c = u \\ f(c) & \text{otherwise} \end{cases}$$

The set of functions from X to Y is called $X \to Y$.

We use standard notation for graphs. In particular, for a graph G, we let $V(G)$ be its set of vertices and $E(G)$ its set of edges. For a set $X \subseteq V(G)$, we denote the induced subgraph by $G[X]$. For a vertex $v \in V(G)$, we denote its neighbourhood as $N(v)$. We denote the treewidth and pathwidth (see Section 4.5) of G as $tw(G)$ and $pw(G)$, respectively. All graphs are, unless otherwise stated, simple, undirected and without self-loops.

We use $\text{poly}(n)$ to denote functions which are upper-bounded by a polynomial n^c, and $\text{polylog}(n)$ to denote functions upper-bounded by some $\log^c(n)$, $c \in \mathbb{N}$. We use the O^*-notation to suppress polynomial factors, that is $O^*(f(n)) = O(f(n)\,\text{poly}(n))$.

4.2 Notation and Nomenclature for Colored Subgraph Isomorphism

We now define some nomenclature for the (Exact Weight) Colored Subgraph Isomorphism problem. Fix an instance (G, H, f) or (G, H, f, w). H is the pattern graph which is to be found in the large graph G when given homomorphism $f : V(G) \to V(H)$ and weight function w. For a subset $I \subseteq V(H)$, we call a function $R : I \to f^{-1}(I)$ a **configuration of** I if $\forall v \in I : R(v) \in f^{-1}(v)$. We define $\mathcal{Conf}(I) \subseteq (I \to f^{-1}(I))$ to be the set of configurations of I. A configuration of I is called **valid configuration** if $\forall uv \in E(H[I]) : R(u)R(v) \in E(G)$. Finally, we call a configuration R of I a **partial solution of** I **in** J, for some $I \subseteq J \subseteq V(H)$, if there is a valid configuration S of J such that $S|_I = R$. We may shorten this to R being a **partial solution for** J if I is clear from context.

For a valid configuration R of $I \subseteq V(H)$, we call $w(R)$ its **weight**. The exact definition of the weight $w(R)$ depends on whether G is node-weighted or edge-weighted. If G is node-weighted with weight function $w : V(G) \to \mathbb{Z}$, we define $w(R) := \sum_{u \in I} w(f^{-1}(u))$. If it is edge-weighted with weight function $w : E(G) \to \mathbb{Z}$, we define $w(R) := \sum_{uv \in E(H[I])} w(R(u)R(v))$, where it is guaranteed that $R(u)R(v) \in E(G)$ because R is a valid configuration. Furthermore, we say that a partial solution R of I in J **has an extension of weight** W' if there is a valid configuration S of J such that $S|_I = R$ and the nodes (respectively: the edges) of S which are not in I have combined weight W', i.e. $w_{ext}(S, R) := w(S) - w(R) = W'$.

For brevity, we define the following indicator functions (using the Iverson bracket):

- $\mathrm{ParSol}(R; I; J) = [R \text{ is a partial solution of } I \text{ in } J]$
- $\mathrm{ParSolE}(R; I; J; W) = [R \text{ is a partial solution of } I \text{ in } J \text{ with an extension of weight } W]$
- $\mathrm{ValConf}(R; I) = R \text{ is a valid configuration of } I$

Note that the first two take values from $\{0, 1\}$, while the last is either true or false.

4.3 Problem Definitions

In the following, we define all problems considered in this paper. Note that for the colored variants, we do not actually talk about colors; instead, to simplify our discussions, we talk about a graph homomorphism which basically simulates the colors.

Subgraph Isomorphism Given a graph H and a graph G, does G have a (not necessarily induced) subgraph isomorphic to H? Instances are tuples of the form (H, G).

Colored Subgraph Isomorphism Given a graph H and a graph G along with a graph homomorphism $f : V(G) \to V(H)$ where all preimages are of equal size, is it possible to pick a set S with exactly one vertex from the preimage of each $v \in V(H)$ such that the subgraph induced by S is isomorphic to H? Instances are tuples of the form (H, G, f).

Exact Weight Subgraph Isomorphism May be either node- or edge-weighted. The same as Subgraph Isomorphism, except that the edges (respectively: vertices) each have an integer weight assigned to them via a weight function w. A solution to a problem instance is a solution to the Subgraph Isomorphism problem, except that the edge weights (respectively: node weights) of the subgraph induced by S must sum to zero. Instances are of the form (H, G, w).

Exact Weight Colored Subgraph Isomorphism May be either node- or edge-weighted. The same as Colored Subgraph Isomorphism, except that it is weighted just as the Exact Weight Subgraph Isomorphism. Instances are of the form (H, G, f, w).

Max-Sat Given a Sat instance ϕ as CNF and a number k, determine whether there is an assignment to the variables such that at least k clauses of ϕ are satisfied. Instances are of the form (ϕ, k).

Max-q-Sat Like Max-Sat, but all clauses have exactly q literals.

k-Clique Given a graph G and a number k, determine whether there is a subset $C \subseteq V(G)$ with $|C| = k$ such that $G[C]$ is the complete graph on k vertices.

h-Uniform k-Hyperclique Given an h-uniform hypergraph G and a number k, determine whether there exists a subset $C \subseteq V(G)$ with $|C| = k$ such that $G[C]$ is a complete graph. That is, any size h subset of C is connected by a hyperedge.

Subset Sum Given a set $A \subseteq \mathbb{N}$ of n numbers and a target $T \in \mathbb{N}$, determine whether $\exists B \subseteq A : \sum_{b \in B} b = T$.

k-Sum In the k-sum problem, we are given k sets $A_1, \dots A_k \subseteq \mathbb{N}$ of n numbers and a target $T \in \mathbb{N}$, and we wish to determine whether $\exists a_1 \in A_1, \dots a_k \in A_k : \sum_{i=1}^{k} a_i = T$.

To avoid special cases with the running time, we assume without further mention that in all weighted problems, the target weight or maximum absolute weight is at least 1.

For a discussion on current algorithms and hardness results on the Subgraph Isomorphism problems, see the introduction and related work. For Max-q-Sat, Hyperclique and k-Clique, see the next subsection.

We briefly discuss Subset Sum and k-Sum here. The Subset Sum problem has a well-known $O(Tn)$ time algorithm using dynamic programming [77]. Very recently, suprising new algorithms with running time $\widetilde{O}(\sqrt{n}T)$ [62, 63] and $\widetilde{O}(T+n)$ [28] have been shown, the latter matching several conditional lower bounds from SETH [6], Set Cover [44] and k-clique (observed in [28] via techniques from [44, 9]). The algorithm in [28] is slightly simplified in [61], with improvements in the log factors of the running time. Moving on, the k-Sum problem is basically a modified version of Subset Sum. It has a folklore $O(n^{\lceil k/2\rceil})$ time algorithm, and it is conjectured that no $O(n^{\lceil k/2\rceil-\varepsilon})$ time algorithm exists. An unconditional lower bound of $\Omega(n^{\lceil k/2\rceil})$ is known for a restricted type of decision trees [51].

We also remark that the restriction of the target weight to zero in the Exact Weight Subgraph Isomorphism problems is simply for ease of discussion. We use this version to both simplify our algorithms and circumvent any problems that might arise from having T be part of the input. Trivially, all of our results also hold for the problem where a target $T = O(W)$ is given.

4.4 Conditional Lower Bounds and Relevant Hypotheses

When searching for faster algorithms for computational problems, it is important to have a clear picture of how hard the problem is, and what barriers are in place that might hinder that search. Ideally, one would like to have an upper bound via a fast algorithm, and a matching unconditional lower bound showing that no improvements whatsoever are possible. However, such unconditional lower bounds are very hard to come by, even in the setting of very restricted algorithms. Instead, great progress has been made by relating the hardness of one problem to that of another via means of a reduction: If problem A has a faster algorithm, then so does problem B. Ideally, problem B has seen decades of intense study, with no indications of considerable progress towards such a faster algorithm existing; hence one may conclude that problem A is seemingly also very hard.

These ideas have recently been ported to show conditional lower bounds for problems solvable in polynomial time (so called Fine-Grained Complexity Theory), with great success. Our lower bound results for the unweighted case – and perhaps arguably also the weighted case – are part of this.

Throughout the paper, we refer to the following hypotheses, each of which has been standing for several decades.

MaxSat Hypothesis There is no $\varepsilon > 0$ such that MaxSat with clauses of constant width can be solved in time $O^*(2^{(1-\varepsilon)n})$.

Max3Sat Hypothesis There is no $\varepsilon > 0$ such that Max3Sat can be solved in time $O^*(2^{(1-\varepsilon)n})$.

k-Clique Hypothesis For no $k \geq 3$ is there an $\varepsilon > 0$ such that k-Clique can be solved in time $O(n^{k\omega/3-\varepsilon})$.

h-uniform Hyperclique Hypothesis For no $k \geq h+1$ is there an $\varepsilon > 0$ such that h-uniform k-Hyperclique can be solved in time $O(n^{k-\varepsilon})$. This hypothesis is only defined for $h \geq 3$.

SETH Originally from [57]. For any $\varepsilon > 0$ there exists $k \geq 3$ such that k-Sat on n variables cannot be solved in time $O^*(2^{(1-\varepsilon)n})$.

Of these four hypotheses, SETH is beyond doubt the most widely used for conditional lower bounds for problems in P, see e.g. [5, 11, 10, 6, 27, 31, 78]. The k-clique hypothesis has found many application in string problems (see e.g. [2, 19, 30, 32]), but also problems on graphs [68]. The Max-Sat hypothesis seems to be somewhat more rarely used, implying both lower bounds on MAX-FLOW [12] and, crucially and by a well-known reduction, that the q-UNIFORM k-HYPERCLIQUE problem has no $O(n^{k-\varepsilon})$ algorithm for $q \geq 3$. Finally, the MAX3SAT-hypothesis is widely used, seeing use in problems on graphs and hypergraphs [13, 29, 67, 68, 84] as well as CSPs [59].

The MAXSAT hypothesis is, of course, weaker than both SETH and the MAX3SAT hypothesis. Indeed, we are very far from finding faster-than-trivial algorithms for MAXSAT: Not even a $2^n/poly(n)$ algorithm seems to be known [12].

The MAX3SAT hypothesis can also be formulated for MAX-q-SAT for an arbitrary $q \geq 3$ and reasonably be conjectured to be true. The restriction $q \geq 3$ is necessary: For MAX-2-SAT, faster algorithms of running time $O^*(2^{\omega n/3})$ are known [84]. Also note that, as discussed in the introduction, faster algorithms for MAX-q-SAT are known when the instances are guaranteed to be sparse [46, 37]. However, for non-sparse instances, research seems to indicate that it is a very reasonable assumption that the trivial $O(2^n)$ algorithm is optimal up to minor improvements.

For k-CLIQUE, the algorithm with running time $O(n^{k\omega/3})$ for k divisible by three can be found in [74]; it is well-known. The problem has resisted significant improvements in running time for a long time, and not for lack of trying. Under the Exponential Time Hypothesis (ETH), k-clique does not have algorithms with running time $n^{o(k)}$ [35], and for tensor networks, an unconditional lower bound of $\Omega(n^{\lceil k \cdot 2/3 \rceil})$ is known [18].

The h-UNIFORM HYPERCLIQUE hypothesis for $h \geq 3$ is, as was said, weaker than the MAX-SAT hypothesis. For a discussion of its believability, as well as more context, we refer to [68, Section 7].

4.5 Treewidth and Pathwidth

We give a very short introduction to treewidth and pathwidth, and state some auxiliary definitions and notation used throughout the paper.

▶ **Definition 14** (Tree Decomposition). *Let H be a graph. A **tree decomposition** of H is a pair $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ consisting of a tree T and along with a set of "bags" $X_t \subseteq V(H)$, one for each vertex of T. It must satisfy the following properties:*
(T1) $\bigcup_{t \in V(T)} X_t = V(H)$
(T2) $\forall uv \in E(H) : \exists t \in V(T) : \{u, v\} \subseteq X_t$
(T3) $\forall u \in V(H)$: *The subgraph induced by $T_u := \{t \in V(T) | u \in X_t\}$ is a connected subtree*

▶ **Definition 15** (Treewidth). *Let $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ be a tree decomposition of H. We define its **width** to be $max_{t \in V(T)} |X_t| - 1$. We define the **treewidth** of H to be the minimum width of all tree decompositions of H and denote it as* $\mathrm{tw}(H)$.

▶ **Definition 16** (Path Decomposition). *Let H be a graph. A **path decomposition** of H is a tree decomposition where T is a path.*

▶ **Definition 17** (Pathwidth). *The **width** of path decompositions is defined as for tree decompositions. The **pathwidth** of H is defined to be the minimum width of all path decompositions of H, and is denoted as* $\mathrm{pw}(H)$.

A classic algorithm by Bodlaender [23] computes an optimal tree decomposition or path decomposition for an input graph H in time $O(f(|tw(H)|)|V(H)|)$ for a computable function f. For our purposes, this is almost excessive: For our results, we only need an algorithm which computes an optimal tree decomposition in time $g(|V(H)|)$, for some computable function g.

Clearly, the treewidth of a graph is always smaller than or equal to its pathwidth. It should also be noted that while graphs of treewidth one are exactly the class of tree graphs, graphs of pathwidth one encompass more than just paths. Rather, they are the class of graphs where each connected component is a caterpillar graph, i.e. consists of a single path with arbitrarily many degree-one nodes attached at any node of the path [76]. The latter is vital for our conditional lower bounds for pathwidth one.

When working with a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t\in T})$, we root it at an arbitrary vertex r. We often use path decompositions as if they were tree decompositions with the path T rooted at one of its endpoints to unify notation. For a tree or path decomposition with underlying tree T and for $u \in V(T)$, we define T_u to be the subtree rooted at u. The **cone** V_u is then defined to be $V_u := \bigcup_{v\in V(T_u)} X_v$.

For an excellent and detailed introduction to treewidth, pathwidth and their many applications, we refer the reader to [45, Chapter 7].

4.6 k-Wise Matrix Product

In our algorithms, we use the following generalization of matrix multiplication to tensors, as defined in its general form in [56] and explored further algorithmically in [68]:

Given k tensors $A^1, \ldots, A^k$ of order k with dimensions $\overbrace{n \times \ldots \times n}^{\text{k times}}$, we define the k-wise matrix product $MP_k(A^1, \ldots, A^k)$ to be the tensor given by

$$MP_k(A^1, \ldots, A^k)[i_1, \ldots, i_k] := \sum_{\ell\in[n]} A^1[\ell, i_2, \ldots, i_k] \cdot A^2[i_1, \ell, i_3, \ldots, i_k] \cdots A^k[i_1, \ldots, i_{k-1}, \ell]$$

Clearly, for $k = 2$ this product is exactly matrix multiplication. Furthermore, the k-wise matrix product can trivially be computed in time $O(n^{k+1})$ for all k. Unfortunately, as Lincoln, Williams and Williams observe in [68], it is impossible that faster Strassen-like algorithms with running time $O(n^{k+1-\varepsilon})$ with $\varepsilon > 0$ exist for $k \geq 3$: Just as the operation of $n \times n$ by $n \times n$ matrix multiplication has a corresponding order-6 tensor of dimensions $n \times \ldots \times n$, there is an order-$k(k+1)$ tensor of dimension $n \times \ldots \times n$ corresponding to the k-wise matrix product. For $k \geq 3$, this tensor has border rank $k+1$, i.e. it cannot be expressed as the limit of a sequence of tensors of rank smaller than $k+1$.

It turns out that calculation of this product is the bottleneck for two of the algorithms for bounded-treewidth graphs we develop. With a few exceptions, all tensor calculations are done using operations from the ring $\mathbb{Z}$. In particular, though most of our tensors have entries from $\{0, 1\}$, the k-wise matrix product of such tensors might have larger entries.

4.7 Rectangular Matrix Multiplication

Multiplication of $n \times n$ by $n \times n$ matrices is perhaps the most ubiquitous open problem in computer science, with the central question being whether this can be done in $O(n^2)$ time. The matrix multiplication exponent ω has been slowly inching toward, but not quite reaching, a value of 2 over the last few decades. In our algorithms, however, we also multiply rectangular matrices. It turns out that the techniques used to show fast algorithms for square matrix multiplication can also be generalized to the rectangular case. In the following, let $\mathrm{MM}(s, r, t)$ denote the time needed to multiply a matrix of size $r \times s$ with a matrix of size $s \times t$. We are mostly interested in the case that $s = n, r = n$ and $t = n^k$ for some $k \in \mathbb{R}^+$.

A simple, well-known result is that MM is both convex and symmetrical in its arguments (see e.g. [69, 81]). In particular, we have $\forall x \in \mathbb{R}^+ : \mathrm{MM}(n, n, n^{k+x}) \leq n^x \cdot MM(n, n, n^k)$

and $\mathrm{MM}(n, n, n^k) = \mathrm{MM}(n, n^k, n)$. Letting $\omega(k) := \log_n MM(n, n, n^k)$, we immediately get $\forall k \geq 1 : \omega(k) < k + 1.373$ via the current bounds of $\omega(1) = \omega < 2.373$ [65].

We can, however, do better: Le Gall [54] has shown that there are faster algorithms based on the Coppersmith-Winograd method [40, 65] used for square matrix multiplication. Among other values, he shows $\omega(0.31) = 2$, $\omega(2) < 3.26$, $\omega(3) < 4.2$, $\omega(4) < 5.18$ and $\omega(5) < 6.16$ (see [54] for an extensive table of such values).

5 Equivalence of Subgraph Isomorphism and Colored Subgraph Isomorphism

We begin our discussion of (Exact Weight) Subgraph Isomorphism and (Exact Weight) Colored Subgraph Isomorphism by showing the equivalence of these problems with respect to running times. The reductions from (Exact Weight) Subgraph Isomorphism to (Exact Weight) Colored Subgraph Isomorphism and the reduction from Exact Weight Colored Subgraph Isomorphism to Exact Weight Subgraph Isomorphism leaves H unmodified. The reduction from Colored Subgraph Isomorphism to Subgraph Isomorphism, however, modifies H in such a way that preserves treewidth, but may modify pathwidth.

We say that Subgraph Isomorphism or Colored Subgraph Isomorphism have a $T(n, k, \rho(H))$ algorithm (for some graph parameter ρ) if there is an algorithm $\mathcal{A}$ which decides a given instance $\phi = (H, G, f)$ of either problem in time $T(n, k, \rho(H))$. Analogously, we define the phrase that Exact Weight Subgraph Isomorphism or Exact Weight Colored Subgraph Isomorphism has a $T(n, k, \rho(H), W)$ algorithm, the only difference being that $\phi = (H, G, f, w)$. W denotes the maximum absolute value of the weight function w.

Parts 1 and 2 of the lemma follows directly from the Color Coding technique [15].

▶ **Lemma 18.** *Let ρ be any graph parameter.*

1. *If there is a $T(n, k, \rho(H))$ time deterministic algorithm for Colored Subgraph Isomorphism, then there is a $O(T(kn, k, \rho(H))g(k))$ expected time algorithm and furthermore a $O(T(kn, k, \rho(H)) \log(n) g(k))$ time deterministic algorithm for Subgraph Isomorphism, for some computable function g.*
2. *If there is a $T(n, k, \rho(H), W)$ time deterministic algorithm for Exact Weight Colored Subgraph Isomorphism, then there is a $O(T(kn, k, \rho(H), W)g(k))$ expected time algorithm and furthermore a $O(T(kn, k, \rho(H), W) \log(n) g(k))$ time deterministic algorithm for Exact Weight Subgraph Isomorphism, for some computable function g.*
3. *Let $\mathrm{tw}(H) \geq 2$. If there is a $T(n, k, \mathrm{tw}(H))$ time algorithm for Subgraph Isomorphism, then there is a $O(T(\mathrm{poly}(k)n, \mathrm{poly}(k), \mathrm{tw}(H)) + \mathrm{poly}(k)n^2)$ time algorithm for Colored Subgraph Isomorphism.*
4. *If there is a $T(n, k, \rho(H), W)$ time algorithm for Exact Weight Subgraph Isomorphism, then there is a $O(T(2n, 2k, \rho(H), 2^k W) + \mathrm{poly}(k)n^2)$ time algorithm for Exact Weight Colored Subgraph Isomorphism.*

Regarding treewidth, the only case the above lemma does not cover is how to transform an algorithm for unweighted Subgraph Isomorphism to an algorithm for Colored Subgraph Isomorphism for $\mathrm{tw}(H) = 1$. Thus, we also cannot transfer lower bounds for the latter to lower bounds for the former when H is a tree. For our purposes, this is not a problem, since Subgraph Isomorphism for trees already has a trivial unconditional lower bound of $\Omega(n^2)$, which is tight. Note that for this unconditional lower bound, we must assume that the graph is dense, i.e. has $\Theta(n^2)$ edges.

The lemma also cannot transform algorithms for unweighted Subgraph Isomorphism to an algorithm for Colored Subgraph Isomorphism for bounded pathwidth. This is a shortcoming that we could not fix, and we leave its resolution as an open problem.

Proof (of Lemma 18). We start by showing **part 1** and **part 2**. As mentioned, this follows directly from a standard application of the Color Coding technique [15]. Briefly speaking, they use random colorings of the vertices of G to make the potential solution subgraph multicolored with some probability depending only on k. In our case, we may then try all $k!$ mappings from colors to vertices of H to obtain the randomized algorithm; we delete any monochromatic edges to make sure that f is a homomorphism. The authors of [15] also explain how to derandomize the algorithm using k-perfect hash functions, which results in the deterministic algorithm with an additional factor of $2^{O(k)}\log(n)$.

The factor of k in front of n in $T(kn, k, \mathrm{tw}(H))$ comes from the fact that in the Colored Subgraph Isomorphism problem, we consider n to be the size of the preimages of f, while in the Subgraph Isomorphism problem, we consider it to be the size of $V(G)$.

Now we show **part 3**. Given an instance ϕ of Colored Subgraph Isomorphism, where H has k vertices, with preimages in G of n vertices each, we construct an equivalent Subgraph Isomorphism instance ϕ'. This is done in two steps. First, we modify ϕ into an equivalent, but more structured Colored Subgraph Isomorphism instance $\widetilde{\phi}$, which we then reduce to ϕ'. See Figure 1 for an example of this reduction.

Subdividing all edges: In the first step, we construct $\widetilde{H}$, which consists of a subdivided copy of H, where each vertex has a unique "signature" structure attached to it. These signatures have a triangle as a key component. Abusing notation, we write $V(H) = \{1, \dots, k\}$ and use the vertices as numbers. First, for each $i \in V(H)$, we add a vertex $\widetilde{i}$ to $\widetilde{H}$. Then, for each edge $ij \in E(H)$ with $i < j$, we add a vertex $\widetilde{x}_{ij}$ and create two edges $\widetilde{i}\widetilde{x_{ij}}$ and $\widetilde{x}_{ij}\widetilde{j}$, hence subdividing the edge ij. This ensures that for now, the new graph has no triangles. In $\widetilde{G}$, we populate the preimages as follows: For each $i \in V(H)$, let $f^{-1}(i) = \{a_i^1, \dots, a_i^n\}$ and add n vertices $\{\widetilde{a}_i^1, \dots, \widetilde{a}_i^n\}$ to $f^{-1}(\widetilde{i})$. For each ℓ, the vertex $\widetilde{a}_i^\ell$ corresponds to a_i^ℓ. The preimages of $\widetilde{x}_{ij}$ are populated with n vertices $\{b_{ij}^1, \dots, b_{ij}^n\}$ via $\widetilde{f}$. For each edge $ij \in E(H)$ with $i < j$, we add an edge $a_i^\ell b_{ij}^\ell$ for every $\ell \in [n]$. We also go through each edge $a_i^\ell a_j^m \in E(G)$ and add a corresponding edge $b_{ij}^\ell \widetilde{a}_j^m$ to $E(\widetilde{G})$.

Signatures: We now add the signatures. For each $i \in [k]$, we add a new vertex $\widetilde{t}_i$ and a new triangle $\widetilde{u}_i\widetilde{v}_i\widetilde{w}_i$ to $\widetilde{H}$, and connect $\widetilde{t}_i$ to both $\widetilde{u}_i$ and $\widetilde{i}$ from $V(\widetilde{H})$. Furthermore, we connect $\widetilde{v}_i$ to $i+1$ other newly created vertices $\widetilde{y}_i^1, \dots \widetilde{y}_i^{i+1}$. Let the set of all newly created vertices $\widetilde{t}_i, \widetilde{u}_i, \widetilde{v}_i, \widetilde{w}_i, \widetilde{y}_i^\ell$ $(i \in [n], \ell \in [i+1])$ be named X. In $\widetilde{G}$, we populate the preimages of these new vertices by adding n vertices $\{z_1, \dots, z_n\}$ to $V(\widetilde{G})$ for each vertex $v \in X$, with $\forall i \in [n] : \widetilde{f}(z_i) = v$. Now, for each $u \in X$, we pick an arbitrary node from $f^{-1}(u)$ and call it active. Furthermore, for all $\widetilde{i} \in V(\widetilde{H})$, we call all vertices of $f^{-1}(i)$ active. Now for each $v \in X$, we connect its active vertex in $f^{-1}(v)$ to all active vertices from the neighbourhood $f^{-1}(N(v))$. Note that of the vertices in $f^{-1}(X)$, only the active ones have edges at all. Indeed, for each $i \in [k]$, $\widetilde{G}$ contains exactly one triangle such that one of its vertices has degree $i+3$.

We set $\widetilde{\phi}$ to be the instance $(\widetilde{H}, \widetilde{G}, \widetilde{f})$. This concludes the construction of $\widetilde{\phi}$.

Clearly, the new instance $\widetilde{\phi}$ has a solution iff ϕ has one, and the size of the new instance is only a $\mathrm{poly}(k)$ factor larger than the size of ϕ. We must also show that the reduction preserves treewidth. Note that $\widetilde{H}$ is obtained from H via two operations: Subdividing edges

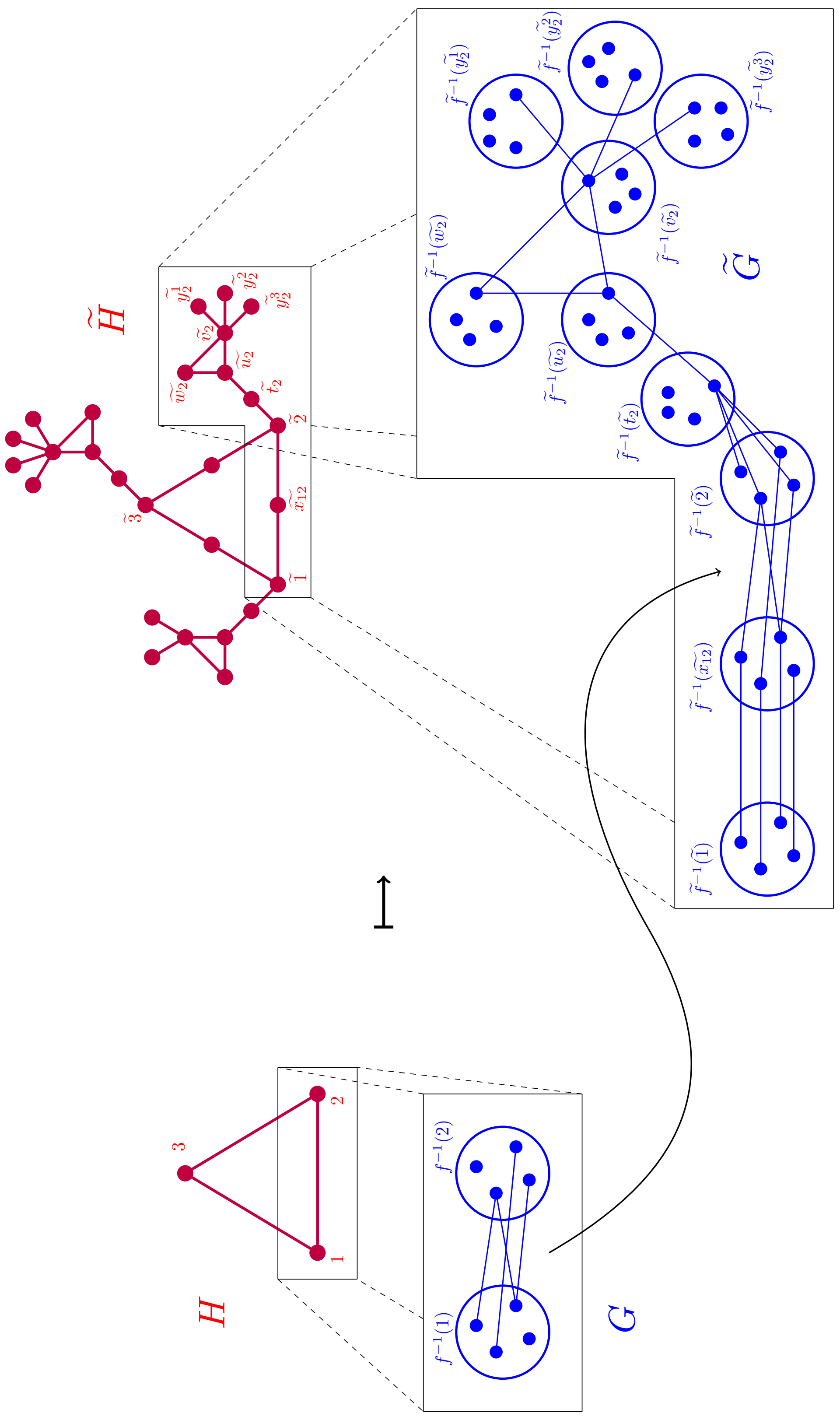
H
3
2
1
$f^{-1}(1)$
$f^{-1}(2)$
G
$\widetilde{H}$
$\widetilde{1}$
$\widetilde{2}$
$\widetilde{3}$
$\widetilde{x_{12}}$
$\widetilde{t_2}$
$\widetilde{u_2}$
$\widetilde{v_2}$
$\widetilde{w_2}$
$\widetilde{y_2^1}$
$\widetilde{y_2^2}$
$\widetilde{y_2^3}$
$\widetilde{G}$
$\widetilde{f}^{-1}(\widetilde{1})$
$\widetilde{f}^{-1}(\widetilde{x_{12}})$
$\widetilde{f}^{-1}(\widetilde{2})$
$\widetilde{f}^{-1}(\widetilde{t_2})$
$\widetilde{f}^{-1}(\widetilde{u_2})$
$\widetilde{f}^{-1}(\widetilde{w_2})$
$\widetilde{f}^{-1}(\widetilde{v_2})$
$\widetilde{f}^{-1}(\widetilde{y_2^1})$
$\widetilde{f}^{-1}(\widetilde{y_2^2})$
$\widetilde{f}^{-1}(\widetilde{y_2^3})$

Figure 1 An example of the construction of the instance $\widetilde{\phi}$ from $\phi = (H, G, f)$ where H is a triangle

and connecting a graph of smaller or equal treewidth via a single edge. It is easy to see that both operations do not change the treewidth.

Now, to construct ϕ', we simply get rid of the mapping $\widetilde{f}$. In other words, $\phi' = (\widetilde{H}, \widetilde{G})$. Obviously, if $\widetilde{\phi}$ has a solution, then ϕ' has one. For the other direction, suppose ϕ' has a solution, i.e. a subgraph S' of $\widetilde{G}$ along with an isomorphism $h : V(\widetilde{H}) \to V(G[S'])$ of $\widetilde{G}$. Since both $\widetilde{H}$ and $\widetilde{G}$ have, for each i, exactly one triangle with a vertex of degree i, h must map these triangles to their respective counterparts in $\widetilde{G}$. In particular, each of the vertices in X is mapped to the active vertex in its preimage. Since the active node in $f^{-1}(\widetilde{t}_i)$ is connected only to nodes of $f^{-1}(\widetilde{i})$ (apart from the active node of $f^{-1}(\widetilde{u}_i)$, which is already in the image of $h(\widetilde{u}_i)$), we know that $h(\widetilde{i}) \in f^{-1}(\widetilde{i})$. Analogously, $h(\widetilde{x}_{ij}) \in f^{-1}(x_{ij})$. Hence, S' takes exactly one vertex from each preimage of vertices from $\widetilde{H}$. Thus, $\widetilde{\phi}$ also has a solution.

We thus obtain a way to reduce ϕ to ϕ' with a size factor of only poly(k). The reduction obviously runs in $O(\text{poly}(k)n^2)$ time. This shows part 3.

Finally, we show **part 4**. We begin with the node-weighted version. Given an instance ϕ of Exact Weight Colored Subgraph Isomorphism where the pattern graph H has k vertices, we create an instance ϕ' of Exact Weight Subgraph Isomorphism by simply dropping f and modifying the weights. We have to ensure that a solution of ϕ' takes exactly one node from each preimage of H. To do this, we encode a checklist in the weights of the nodes. Again, let $V(H) = \{1, \dots, k\}$. Let $u \in f^{-1}(i)$ for $i \in V(H)$, and consider its weight $w(u)$. We modify it by multiplying it with 2^k and adding 2^i. We call the added weight its "signature". Now, since any solution must pick exactly k vertices, the only way that the signatures of the solution vertices sum up to $2^k - 1$ is to pick vertices which have a sum of weight 0 according to the original weight function and furthermore have exactly one vertex with added weight 2^i for each $i = 1, \dots, k$. To complete our reduction, we pick an arbitrary vertex $v \in V(G)$ and subtract $2^k - 1$ from all vertices in $f^{-1}(v)$, making the new target zero. This reduction does not alter G or H, and instead only modifies the weight function, resulting in the stated time bounds.

For the edge-weighted version, we can use essentially the same construction as for the node-weighted version. Again, all weights are multiplied by 2^k, and each vertex of H has a unique "signature". However, this time, we have to add the signatures to the edge weights. Consequently, for each $i \in V(H) = \{1, \dots, k\}$, we pick an arbitrary incident edge $e \in E(H)$. Let $e = \{i, j\}$. We add the signature 2^i to every edge in the preimage of e. That is, to every edge $\{e' \in E(G) \mid e' = \{u, v\}$ and $f(u) = i$ and $f(v) = j\}$. This way, we must still pick a node from each preimage to ensure that the signatures sum to $T := 2^k - 1$. Again, we pick an arbitrary edge $e \in E(H)$ and subtract T from all edges in its preimage. This almost completes the proof. However, we still have to handle nodes of degree 0 in H, since we cannot pick an incident edge for them. However, an isolated vertex i in H may be mapped to any vertex in G. Hence, we may simply skip the signature of i. We also have to modify the target T to be $T - 2^i$. ◀

6 Algorithmic Results

6.1 Colored Subgraph Isomorphism for Bounded Treewidth

We begin by looking at the unweighted version of Colored Subgraph Isomorphism. As was previously mentioned for Theorem 1, these results essentially follow from [15] and [43], but are now unified via a single technique.

▶ **Theorem 19.** *There is an algorithm which, given an arbitrary instance $\phi = (H, G, f)$ of* Colored Subgraph Isomorphism, *solves ϕ in time*

1. $O(n^{\mathrm{tw}(H)+1} \operatorname{poly}(k) + g(k))$ *when* $\mathrm{tw}(H) \geq 3$,
2. $O(n^{\omega} \operatorname{poly}(k) + g(k))$ *when* $\mathrm{tw}(H) = 2$, *where ω is the exponent of matrix multiplication, and*
3. $O(n^2 \operatorname{poly}(k) + g(k))$ *when* $\mathrm{tw}(H) = 1$.

where $k := |V(H)|$, n is the size of the preimages of f, and g is a computable function.

Obviously, a proof of this theorem suffices to prove Theorem 1, since we can simply plug the algorithm into Lemma 18.

Proof (of part 1 of Theorem 19). We describe an algorithm which, given G, H and $f : V(G) \to V(H)$, first calculates an optimal tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ for H in time $g(k)$, then finds a solution via dynamic programming over the tree decomposition. By assumption, $\mathrm{tw}(H) \geq 3$. We use a slightly more complicated framework than necessary, because it generalizes nicely to a proof of part 2 and to a proof of parts 1 and 2 of Theorem 20.

To make our algorithm as easy as possible, we begin by modifying the tree decomposition. This modification is analogous to that which [43] describes for treewidth 2 graphs. In particular, we will obtain the following properties:

1. The bags of the root node r as well as of all leaves have size $\mathrm{tw}(H)$, and every other bag has size exactly $\mathrm{tw}(H) + 1$
2. The root node r has a unique child r'
3. For every $t \in V(T)$ and every child t' of t, we have $|X_t \cap X_{t'}| = \mathrm{tw}(H)$
4. For every inner node $t \in V(T)$ with set of children C_t, we have $\forall u \in X_t : \exists t' \in C_t : X_t \cap X_{t'} = X_t \setminus \{u\}$

To obtain these properties, we follow the outline of [43]. First, we merge adjacent nodes $t_1, t_2 \in V(T)$ with $X_{t_1} = X_{t_2}$. If there exists a node t with $|X_t| < \mathrm{tw}(H) + 1$ and a neighbor t' such that $X_{t'} \not\subseteq X_t$, then we simply add an element of $X_{t'} \setminus X_t$ to X_t. Now all bags, including root and leaves, have size exactly $\mathrm{tw}(H) + 1$. To achieve property 3, we take any adjacent pair of nodes (t, t') with $|X_t \cap X_{t'}| < \mathrm{tw}(H)$ and, letting $u \in X_t \setminus X_{t'}$ and $v \in X_{t'} \setminus X_t$, insert a bag with vertices $(X_t \cup \{v\}) \setminus \{u\}$ between them. Applying this rule exhaustively satisfies property 3. Now we satisfy property 1 and 2 by taking the root and all leaves and, for each one, copying any $\mathrm{tw}(H)$ of its vertices into a new bag which we attach only to the respective root or leaf. Finally, to satisfy property 4, we take any inner node which violates this property for some $u \in X_t$ and attach a leaf with vertices $X_t \setminus \{u\}$ to it.

We now do dynamic programming over the modified tree decomposition. Using notation and nomenclature from Section 4.2, we only store values for each configuration of the separators $X_t \cap X_{t'}$. In particular, let t' be a non-root node with parent t. For each such t', we store the following function of finite domain from configurations of $X_t \cap X_{t'}$ to truth values. Remember that $\mathrm{ParSol}(S; I; J)$ is 1 iff S is a partial solution of I in J, for $I \subseteq J \subseteq V(H)$.

$$\begin{aligned} d_{t'} &: \mathcal{Conf}(X_t \cap X_{t'}) \to \{0, 1\} \\ d_{t'}(R) &:= \mathrm{ParSol}(R; X_t \cap X_{t'}; V_{t'}) \end{aligned} \tag{1}$$

I.e. we store for each configuration whether it is a partial solution of $X_t \cap X_{t'}$ in the cone $V_{t'}$.

Since there are n^{tw} many configurations for $X_t \cap X_{t'}$, the function $d_{t'}$ can be specified using n^{tw} many bits.

We calculate all functions $d_{t'}$ for $t' \in T$ in a bottom-up manner. We begin with the case that t' is a leaf with parent t. Since $X_t \cap X_{t'} = X_{t'}$, any configuration of $X_t \cap X_{t'}$ is a configuration of $X_{t'}$, and $X_{t'} = V_{t'}$. Thus, to calculate $d_{t'}(R)$ as in equation 1, we simply have to check whether R is a valid configuration. The latter can be done in time $\mathrm{poly}(k)$, which leads to a total time of $O(n^{\mathrm{tw}(H)}\,\mathrm{poly}(k))$ per leaf.

Now let t' with parent t be an inner node of T, let $C_{t'}$ be its set of child vertices in T. We may partition the set $C_{t'}$ into $\mathrm{tw}(H)+1$ subsets according to which vertices the bag of the child shares with $X_{t'}$. Specifically, for $u \in X_{t'}$, let $C_{t'}^{(u)}$ denote the set of children $t'' \in C_{t'}$ such that $X_{t'} \cap X_{t''} = X_{t'} \setminus \{u\}$. We define the "partial cone" $V_{t'}^{(u)} := \bigcup_{t'' \in C_{t'}^{(u)}} V_{t''}$.

Now, let R be a configuration of $X_t \cap X_{t'}$. Letting $X_{t'} \setminus X_t = \{v\}$, we obtain the following alternate characterization of $d_{t'}(R)$:

$$d_{t'}(R) = \Big[\exists v' \in f^{-1}(v) : \forall u \in X_{t'} : \mathrm{ParSol}((R \cup \{v \mapsto v'\})|_{X_{t'} \setminus \{u\}}; X_{t'} \setminus \{u\}; V_{t'}^{(u)}) = 1\Big]$$

We now show how to calculate the values of $d_{t'}$ via a $\mathrm{tw}(H)$-wise matrix product, with 0-1-tensors of dimensions $n \times \ldots \times n$. For convenience, all tensors from this point onward are indexed via configurations, each dimension indexed by a single vertex. Formally, we call a tensor A **indexed by configurations of** $X \subseteq V(H)$ when it is an order $|X|$ tensor of dimensions $n \times \ldots \times n$. Abusing notation, we rename the vertices of $f^{-1}(u)$ for each $u \in X$ as $\{1, \ldots, n\}$ and use them as if they were numbers. We fix some ordering $(v_1, \ldots, v_{|X|})$ of X, and for a configuration R of X define $A[R] := A[R(v_1), \ldots, R(v_{|X|})]$.

For our $\mathrm{tw}(H)$-wise matrix product, each input tensor is indexed by configurations of one of the subsets of size $\mathrm{tw}(H)$ of $X_{t'}$, excluding $X_{t'} \setminus \{v\}$. Note that in particular, all input tensors have one dimension indexed by the configuration of v.

The input tensor $p_{t'}^{(u)}$, to be defined in a moment, is indexed by configurations of $X_{t'} \setminus \{u\}$ and is responsible for all children $t'' \in C_{t'}^{(u)}$. By definition, for all such children t'', we have $X_{t''} \cap X_{t'} = X_{t'} \setminus \{u\}$. We define

$$p_{t'}^{(u)}[R'] := \mathrm{ParSol}(R'; X_{t'} \setminus \{u\}; V_{t'}^{(u)})$$

Letting $X_{t'} = \{v, u_1, \ldots, u_{\mathrm{tw}(H)}\}$, and permuting the indices of the tensors $p_{t'}^{(u_i)}$ such that the i-th index always corresponds to v, we now calculate the $\mathrm{tw}(H)$-wise matrix product $D := MP_{\mathrm{tw}(H)}(p_{t'}^{(u_1)}, \ldots, p_{t'}^{(u_{\mathrm{tw}(H)})})$. Via the formula for the $\mathrm{tw}(H)$-wise matrix product, it can be seen that for any configuration R of $X_t \cap X_{t'}$, we have $D[R] \geq 1$ iff $\exists v' \in f^{-1}(v) : \forall u \in X_{t'} \setminus \{v\} : p_{t'}^{(u)}[(R \cup \{v \mapsto v'\})|_{X_{t'} \setminus \{u\}}] = 1$. We arrive at the following formula for $d_{t'}(R)$:

$$\begin{aligned}
d_{t'}(R) &= \Big[\exists v' \in f^{-1}(v) : \forall u \in X_{t'} : \mathrm{ParSol}((R \cup \{v \mapsto v'\})|_{X_{t'} \setminus \{u\}}; X_{t'} \setminus \{u\}; V_{t'}^{(u)}) = 1\Big] \\
&= \Big[\mathrm{ParSol}(R; X_{t'} \setminus \{v\}; V_{t'}^{(v)}) = 1 \wedge \\
&\qquad \exists v' \in f^{-1}(v) : \forall u \in X_{t'} \setminus \{v\} : \mathrm{ParSol}((R \cup \{v \mapsto v'\})|_{X_{t'} \setminus \{u\}}; X_{t'} \setminus \{u\}; V_{t'}^{(u)}) = 1\Big] \\
&= \Big[p_{t'}^{(v)}[R] = 1 \wedge \exists v' \in f^{-1}(v) : \forall u \in X_{t'} \setminus \{v\} : p_{t'}^{(u)}[(R \cup \{v \mapsto v'\})|_{X_{t'} \setminus \{u\}}] = 1\Big] \\
&= \Big[p_{t'}^{(v)}[R] = 1 \wedge D[R] \geq 1\Big]
\end{aligned}$$

This gives an easy algorithm for calculating $d_{t'}$ from the tensors $p_{t'}^{(u)}$. The $\mathrm{tw}(H)$-wise matrix product to calculate D can be done in $O(n^{\mathrm{tw}(H)+1})$ given the input tensors. From that, we can calculate $d_{t'}$ in time $O(n^{\mathrm{tw}(H)}\,\mathrm{poly}(k))$ using the above formula. To calculate the input tensors, note that

$$\begin{aligned} p_{t'}^{(u)}[R'] &= \left[\forall t'' \in C_{t'}^{(u)} : \mathrm{ParSol}(R'|_{X_{t'}\setminus\{u\}}; X_{t'} \setminus \{u\}; V_{t''}) = 1\right] \\ &= \left[\forall t'' \in C_{t'}^{(u)} : d_{t''}(R'|_{X_{t'}\setminus\{u\}}) = 1\right] \end{aligned}$$

Obviously, calculation of a single entry $p_{t'}^{(u)}[R']$ takes time $O(|C_{t'}^{(u)}|) = \mathrm{poly}(k)$(since the tree decomposition only has $\mathrm{poly}(k)$ nodes). There are $\mathrm{tw}(H) = \mathrm{poly}(k)$ input tensors, one for each $u \in X_{t'}$, and each one has $n^{\mathrm{tw}(H)}$ entries. Hence, the entries of all of the input tensors $p_{t'}^{(u)}$ can be calculated in total time $O(n^{\mathrm{tw}(H)}\,\mathrm{poly}(k))$. Thus we obtain a total running time of $O(n^{\mathrm{tw}(H)+1}\,\mathrm{poly}(k))$ for each inner node.

Now to output the answer, remember that r' is the unique child of the root node r of T. We know by definition that $d_{r'}(R)$ is 1 iff R is a partial solution for X_r in $V(H)$. Thus, the input is a YES-instance for Colored Subgraph Isomorphism iff there is an R such that $d_{r'}(R)$ is 1.

It remains to analyze the running time. As was discussed, we need time $O(n^{\mathrm{tw}(H)}\,\mathrm{poly}(k))$ for each leaf, and time $O(n^{\mathrm{tw}(H)+1}\,\mathrm{poly}(k))$ for each inner node. The final check of $d_{r'}(R)$ can be done in $O(n^{\mathrm{tw}(H)})$. Since the number of nodes of T is polynomial in k, we have an overall running time of $O(n^{\mathrm{tw}(H)+1}\,\mathrm{poly}(k))$. ◀

So far, we have only looked at the case that $\mathrm{tw}(H) \geq 3$. However, this exact algorithm also achieves the second result.

Proof (of part 2 of Theorem 19). Note that in the algorithm for part 1, all steps run in time $O(n^{\mathrm{tw}(H)}\,\mathrm{poly}(k))$, except for the $\mathrm{tw}(H)$-wise matrix product, which can be done in time $O(n^{\mathrm{tw}(H)+1})$. However, for $\mathrm{tw}(H) = 2$, $\mathrm{tw}(H)$-wise matrix product is exactly matrix multiplication, which runs in time $O(n^{\omega})$. Thus we obtain our second result. ◀

The third result with $\mathrm{tw}(H) = 1$ cannot be achieved by this algorithm directly. We shortly outline why. Rooting G in some arbitrary node, consider the tree decomposition that takes exactly the edges of G as bags, and constructs T such that two nodes are connected by an edge iff their bags have a non-empty intersection. Now consider a graph G which contains nodes u, v, w such that u is the parent of v and v is the parent of w. Let $t, t' \in T$ be such that $X_t = \{u, v\}$, $X_{t'} = \{v, w\}$. But now the edge vw is not covered by any of the subsets $X_{t'} \setminus \{v\}$ or $X_{t'} \setminus \{w\}$ and is thus not considered in the algorithm at all. This leads to the algorithm failing. For $\mathrm{tw}(H) \geq 2$, this does not happen, since any pair of nodes is contained in a bag X_t of size ≥ 3, hence there is some u such that the edge is covered by the subset $X_t \setminus \{u\}$.

Thus, to obtain our third result, we must employ a different technique. However, this part of the theorem turns out to be easy.

Proof (of part 3 of Theorem 19). We only sketch the result, since it is easy to see. It suffices to employ the trivial dynamic programming solution on the tree G, which already has a running time of $O(n^2\,\mathrm{poly}(k))$. ◀

6.2 Exact Weight Colored Subgraph Isomorphism for Bounded Treewidth

We now move on to the weighted version of Colored Subgraph Isomorphism. Specifically, we show how to solve the Exact Weight Colored Subgraph Isomorphism problem for bounded-treewidth pattern graphs using dynamic programming. Remember that the instances we consider here may be either node- or edge-weighted.

▶ **Theorem 20.** *There is an algorithm which, given an arbitrary instance $\phi = (H, G, f, w)$ of the* Exact Weight Colored Subgraph Isomorphism *problem, solves ϕ in time*

1. $O((n^{\mathrm{tw}(H)+1}W + n^{\mathrm{tw}(H)}W \log W)\,\mathrm{poly}(k) + g(k))$ *when* $\mathrm{tw}(H) \geq 3$,
2. $O((n^{\omega}W + n^2 W \log W)\,\mathrm{poly}(k) + g(k))$ *when* $\mathrm{tw}(H) = 2$, *and*
3. $O((n^2 W + nW \log W)\,\mathrm{poly}(k) + g(k))$ *when* $\mathrm{tw}(H) = 1$.

where $k := |H|$, n is the size of the preimages of f, and W is the maximum absolute weight in the image of w.

Again, note that this implies Theorem 2 via a simple application of Lemma 18 to the resulting algorithm.

Before we prove this, we establish two important lemmata. First, we prove that in the algorithms, we can always assume that the instances are edge-weighted. For the lower bounds, it also shows that we must only prove bounds for node-weighted instances to immediately also get bounds for edge-weighted instances.

▶ **Proposition 21.** *Given an instance of* Exact Node Weight Colored Subgraph Isomorphism *where each vertex $v \in V(G)$ with $w(v) \neq 0$ has degree at least one, we can transform it into an equivalent instance of* Exact Edge Weight Colored Subgraph Isomorphism *in such a way that only the weight function changes. Furthermore, this reduction runs in time linear in the input size.*

Proof. We construct a new weight function $w' : E(G) \to \mathbb{Z}$. Initially, $w'(e) = 0$ for all $e \in E(G)$. The idea is to push the weight of each vertex of non-zero weight in G onto one of its edges. Hence let $v \in V(G)$ with $w(v) \neq 0$. Then there must be $u \in V(G)$ with $uv \in E(G)$. We add $w(v)$ to $w'(uv)$. This completes the reduction.

It can easily be seen that if there is a solution in the node-weighted instance with w, then that same solution must work with w' and vice versa. ◀

As our second lemma, we show that you can basically do the k-wise matrix product of tensors of polynomials faster than naively by utilizing the Fast Fourier Transform. Recall that a Laurent polynomial $p \in \mathbb{C}[X, X^{-1}]$ is simply a polynomial which may have negative powers of the X.

▶ **Lemma 22.** *Let $m \in \mathbb{N}$ tensors $A^1, \ldots, A^m$ of order m be given, each of dimensions $n \times \ldots \times n$ and such that each of their entries $A^i_{j_1,\ldots,j_m} \in \mathbb{C}[Z, Z^{-1}]$ ($i \in [m]$ and $\forall \ell \in [m] : j_\ell \in [n]$) is a Laurent polynomial of degree bounded by W in both the positive and negative direction. Then their m-wise matrix product can be computed in time*

1. $O(n^{m+1}Z + mn^m W \log W)$ *for* $m \geq 3$ *and*
2. $O(n^{\omega}Z + n^2 W \log W)$ *for* $m = 2$.

Proof. It suffices to prove the result for standard polynomials of degree bounded by $2W$, since we can shift the exponents of the polynomials such that they only have positive exponents, do the m-wise matrix product, then shift back.

We assume for now that $m \geq 3$. Our algorithm is a generalization of the Fast Fourier Transform algorithm for standard polynomials (e.g. [38]). Specifically, we evaluate each A^i at the set S of the $2W$-th roots of unity, obtaining $m \cdot |S|$ tensors with complex entries. Since each entry of A^i is a polynomial, this can be done separately for each entry. Then, for each $s \in S$, we compute $MP_m(A^1(s), \ldots, A^m(s))$, obtaining $|S|$ tensors with complex entries.

Obviously, these are exactly the evaluations of $MP_m(A^1, \dots, A^m)$ at S. At this point we can use interpolation via the inverse Fast Fourier Transform separately for each entry to recover the result.

Evaluating all entries of the tensors A^i at the roots of unity can be done using the classic Fast Fourier Transform algorithm. Since there are $m \cdot n^m$ such entries, each with polynomials of degree bounded by W, this runs in time $O(mn^m W \log W)$. Similarly, the interpolation of the result can be done by the inverse Fast Fourier Transform algorithm. Since there are n^m entries to interpolate, each with a degree bound of $O(W)$, this runs in time $O(mn^m W \log W)$. Finally, computing the m-wise matrix products for each root of unity can be done in $|S| \cdot n^{m+1} = O(n^{m+1}W)$. Thus, our total running time is $O(n^{m+1}W + mn^m W \log W)$.

For the case that $m = 2$, note that in the above algorithm, all steps except for the m-wise matrix product run in time $O(n^m W \log W)$. For $m = 2$, the m-wise matrix product is exactly matrix multiplication, which can be done in $O(n^\omega)$. Thus the $O(W)$ matrix multiplications can be done in $O(n^\omega W)$. ◀

We use this lemma as an important tool in our proof of Theorem 20. The framework of the proof is somewhat analogous to that of Theorem 19, but instead of storing a single tensor for each node of T in the dynamic programming table, they have several such tensors, one for each achievable weight. The computation of the dynamic programming table entries is slightly more complex, with some shifting of the entries being required.

Proof (of part 1 of Theorem 20). Due to Proposition 21, we only need to consider the case of edge weights.

We describe an algorithm with inputs G, H, $f : V(G) \to V(H)$ and a weight function w describing edge weights. It calculates an optimal tree decomposition $\mathcal{T} := (T, \{X_t\}_{t \in V(T)})$ for H in time $g(k)$ and then computes a solution via dynamic programming over the tree decomposition. By assumption, $\operatorname{tw}(H) \geq 3$.

We use the same modified version of the tree decomposition as described in the proof of part 1 of Theorem 19. We continue using the terms and notation as described in Section 4.2. We also continue using the the notation and nomenclature for indexing tensors via configurations, as described in the proof of part 1 of Theorem 19.

In deviation from the proof of Theorem 19, our dynamic programming table is now structured differently. Instead of having only a single function of finite domain for each t', we have one function for each t' and achievable weight. Since the instance is node-weighted, the achievable weights must all lie in $\mathcal{W} := \{-kW, \dots, kW\}$.

Specifically, for each non-root node $t' \in T$ with parent t, for each weight $W' \in \mathcal{W}$ and for each configuration R of $X_t \cap X_{t'}$, we store

$$d_{t',W'} : \mathcal{Conf}(X_t \cap X_{t'}) \to \{0, 1\}$$
$$d_{t',W'}(R) := \operatorname{ParSolE}(R; X_t \cap X_{t'}; V_{t'}; W')$$

Again, we calculate these values in a bottom-up manner. Specifically, for a node $t \in T$ with children C_t, we calculate $d_{t',W}$ for all $t' \in C_t, W' \in \mathcal{W}$ before calculating $d_{t',W}$ for all $W \in \mathcal{W}$. In the case that t' is a leaf with parent t, we have that $d_{t',W'}(R)$ is 1 iff R is a valid configuration and $W' = 0$. Even the simplest implementation of this takes time at most $O(n^{\operatorname{tw}(H)} W \operatorname{poly}(k))$.

Now consider the case where $t' \in T$ is an inner node. For $u \in X_{t'}$ we again denote by $C_{t'}^{(u)}$ the set of children t'' with $X_{t''} \cap X_{t'} = X_{t'} \setminus \{u\}$ and define the partial cone $V_{t'}^{(u)} := \bigcup_{t'' \in C_{t'}^{(u)}} V_{t''}$. First, we deal with calculating the weight of an extension. Let R' be a partial solution of $X_{t'} \cap X_t$ in $V_{t'}$, let S be an extension of R' and let $X_{t'} \setminus X_t = \{v\}$. Notice that we get the following formula for the weight of the extension S:

$$w_{ext}(S, R') = w(S) - w(S|_{X_{t'} \setminus \{v\}}) = \underbrace{\left(\sum_{u \in X_{t'}} w_{ext}(S|_{V_{t'}^{(u)}}, S|_{X_{t'} \setminus \{u\}}) \right)}_{\text{Part A}} + \underbrace{w_{ext}(S|_{X_{t'}}, S|_{X_{t'} \setminus \{v\}})}_{\text{Part B}}$$

Part A is the combined weight of the edges which are not in $X_{t'}$. Compared to the right hand side, it is missing the weight of all edges going from v to $X_{t'} \setminus \{v\}$, which is exactly what is then added in Part B. Recall that$w_{ext}(S|_{X_{t'}}, S|_{X_{t'} \setminus \{v\}}) = \sum_{u \in X_{t'} \setminus \{v\}} w(S(v)S(u))$.

Again, we name the vertices of $X_{t'}$ as $X_{t'} =: \{v, u_1, \ldots, u_{\text{tw}(H)}\}$.

We call a tensor indexed by configurations of a subset as defined in the proof of part 1 of Theorem 19. In keeping with that, we now define for each $u \in X_{t'}$ and each $\widetilde{W} \in \mathcal{W}$ a 0-1-tensor $p^{(u)}_{t',\widetilde{W}}$ of dimensions $n \times \ldots \times n$, which is indexed by configurations of the subset $X_{t'} \setminus \{u\}$ and thus is responsible for the set of children $C^{(u)}_{t'}$. However, to be able to compute the full weight of a configuration from these tensors, we want the weight fo part B from above to be encoded in these tensors. Hence, the entry of $p^{(u)}_{t',\widetilde{W}}$ corresponding to a configuration R' of $X_{t'} \cap X_t$ is defined as

$$p^{(u)}_{t',W'}[R'] := \begin{cases} \mathcal{P}^{(u)}_{t',\widetilde{W} - \sum_{i=2}^{\text{tw}(H)} w(R'|_{\{v,u_i\}})}[R'] & \text{if } u = u_1 \\ \mathcal{P}^{(u)}_{t',\widetilde{W} - w(R'|_{\{v,u_1\}})}[R'] & \text{if } u = u_2 \\ \mathcal{P}^{(u)}_{t',\widetilde{W}}[R'] & \text{otherwise} \end{cases}$$

where $\mathcal{P}^{(u)}_{t',\widetilde{W}}$ is another 0-1-tensor indexed by the same set of configurations, which we define via

$$\mathcal{P}^{(u)}_{t',\widetilde{W}}[R'] := \text{ParSolE}(R'; X_{t'} \setminus \{u\}; V^{(u)}_{t'}; \widetilde{W})$$

Now, the weight of v is encoded in the tensors $p^{(u_1)}_{t',*}$ and $p^{(u_2)}_{t',*}$ (for $* \in \mathcal{W}$). We will specify how to compute all of these tensors later. For now, consider how we can calculate $d_{t',W'}$ from them. We have the following formula for $d_{t',W'}(R)$:

$$\begin{aligned} d_{t',W'}(R) = \Big[&\exists W_v, W_{u_1}, \ldots, W_{u_{\text{tw}(H)}} : \exists v' \in f^{-1}(v) : \\ &W_v + \sum_{i=1}^{\text{tw}(H)} W_{u_i} + w_{ext}((R \cup \{v \mapsto v'\}), R) = W' \wedge \\ &\forall u \in X_{t'} : \text{ParSolE}((R \cup \{v \mapsto v'\})|_{X_{t'} \setminus \{u\}}; X_{t'} \setminus \{u\}; V^{(u)}_{t'}; W_u) = 1 \Big] \end{aligned}$$

We now substitute.

$$W_u := \begin{cases} \widetilde{W}_{u_1} - \sum_{i=2}^{\text{tw}(H)} w(v' R(u_i)) & \text{if } u = u_1 \\ \widetilde{W}_{u_2} - w(v', R(u_1)) & \text{if } u = u_2 \\ \widetilde{W}_u & \text{otherwise} \end{cases}$$

This cancels out with the weight of v being added. Furthermore, we again reorder the tensors $p^{(u_i)}_{t',*}$ ($* \in \mathcal{W}, i \in [\text{tw}(H)]$) such that the i-th index corresponds to v. As a result, their

$\mathrm{tw}(H)$-wise matrix product iterates over v. Defining the tensor $D_{\widetilde{W}}$ as

$$D_{\widetilde{W}}[R] := \left[\exists W_{u_1}, \ldots, W_{u_{\mathrm{tw}(H)}} : \sum_{i=1}^{\mathrm{tw}(H)} W_{u_i} = \widetilde{W} \wedge MP_{\mathrm{tw}(H)}(p^{(u_1)}_{t',W_{u_1}}, \ldots, p^{(u_{\mathrm{tw}(H)})}_{t',W_{u_{\mathrm{tw}(H)}}})[R] \geq 1\right]$$

one can now show equivalence of $d_{t',W'}$ to the following:

$$\begin{aligned}
d_{t',W'}(R) &= \Bigg[\exists \widetilde{W_v}, \widetilde{W_{u_1}}, \ldots, \widetilde{W_{u_{\mathrm{tw}(H)}}} : \exists v' \in f^{-1}(v) : \widetilde{W_v} + \sum_{i=1}^{\mathrm{tw}(H)} \widetilde{W_{u_i}} = W' \wedge \\
&\qquad \forall u \in X_{t'} : p^{(u)}_{t',\widetilde{W_u}}[(R \cup \{v \mapsto v'\})|_{X_{t'} \setminus \{u\}}] = 1\Bigg] \\
&= \Bigg[\exists \widetilde{W_v}, \widetilde{W_{u_1}}, \ldots, \widetilde{W_{u_{\mathrm{tw}(H)}}} : \widetilde{W_v} + \sum_{i=1}^{\mathrm{tw}(H)} \widetilde{W_{u_i}} = W' \wedge p^{(v)}_{t',\widetilde{W_v}}[R] = 1 \wedge \\
&\qquad \exists v' \in f^{-1}(v) : \forall u \in X_{t'} \setminus \{v\} : p^{(u)}_{t',\widetilde{W_u}}[(R \cup \{v \mapsto v'\})|_{X_{t'} \setminus \{u\}}] = 1\Bigg] \\
&= \left[\exists \widetilde{W_v}, \widetilde{W} : \widetilde{W_v} + \widetilde{W} = W' \wedge p^{(v)}_{t',\widetilde{W_v}}[R] = 1 \wedge D_{\widetilde{W}}[R] = 1\right] \qquad (1)
\end{aligned}$$

We are now ready to describe the algorithm for computing $d_{t',W'}$ for each $W' \in \mathcal{W}$ in detail. It consists of the following three steps:

1. Compute $p^{(u)}_{t',W'}$ each $W' \in \mathcal{W}$ and $u \in X_{t'}$
2. Compute $D_{\widetilde{W}}$ for each $\widetilde{W} \in \mathcal{W}$ using the results of step 1
3. Compute $d_{t',W'}$ for each and $W' \in \mathcal{W}$ using (1) and the results of step 2

We have to show that each of these steps can be done in time $O(n^{\mathrm{tw}(H)+1} W \,\mathrm{poly}(k) + n^{\mathrm{tw}(H)} W \log W \,\mathrm{poly}(k))$. Starting with **step 1**, we wish to compute $p^{(u)}_{t',W'}$ for arbitrary $W' \in \mathcal{W}$ and $u \in X_{t'}$. It suffices to specify how to compute $\mathcal{P}^{(u)}_{t',W'}$, since from that point onward calculating $p^{(u)}_{t',W'}$ boils down to shifting entries among the tensors according to the weight difference.

Let R' be a configuration of $X_{t'} \setminus \{u\}$. We set $c = |C^{(u)}_{t'}|$ and number the child nodes $C^{(u)}_{t'}$ as $C^{(u)}_{t'} = \{t''_1, \ldots, t''_c\}$. To calculate the entry $\mathcal{P}_{t',W'}[R']$, note that R' is a partial solution for $V^{(u)}_{t'}$ with an extension of weight W' iff there are weights $W_1, \ldots, W_c$ with sum W' such that $\forall i : R'|_{X_{t'} \setminus \{u\}}$ is a partial solution for the cone $V_{t''_i}$ with an extension of weight W_i. Hence,

$$\mathcal{P}_{t',W'}[R'] = \left[\exists W_1, \ldots W_c : \sum_{i=1}^{c} W_i = W \wedge \forall i : d_{t''_i,W_i}(R') = 1\right]$$

This may be calculated via a discrete convolution. Specifically, for the configuration R' we define the finitely supported functions $f^1_{t',R'}, \ldots, f^c_{t',R'} : \mathbb{Z} \to \{0,1\}$ as $f^i_{t',R'}(x) := d_{t''_i,x}(R')$ if $x \in \mathcal{W}$, and 0 otherwise. We now use the discrete convolution of these functions, defined as $(f^i_{t',R'} * f^j_{t',R'})(x) := \sum_{m=-\infty}^{\infty} f^i_{t',R'}(m) f^j_{t',R'}(x-m)$. Note how $(f^i_{t',R'} * f^j_{t',R'}) \geq 0$ iff $\exists m : f^i_{t',R'}(m) f^j_{t',R'}(x-m) = 1$. We get that $\mathcal{P}^{(u)}_{t',W'}[R] = \left[(f^1_{t',R'} * \ldots * f^c_{t',R'})(W') \geq 0\right]$. This is a convolution of $\mathrm{poly}(k)$ finitely supported functions on $\mathbb{Z}$, with supported values ranging over an interval of size $O(W)$. This can be computed via a Fast Fourier Transform (e.g. the classical algorithm in [38]) in time $O(W \log W \operatorname{poly}(k))$. Since we this for each entry, $\mathcal{P}^{(u)}_{t',W'}$ can be computed in total time $O(n^{\operatorname{tw}(H)} W \log W \operatorname{poly}(k))$.

For **step 2**, we want to compute $D_{\widetilde{W}}$ for each $\widetilde{W} \in \mathcal{W}$. Note how, with the usual definition of tensor addition, we have

$$D_{\widetilde{W}}[R] = \left[\left(\sum_{\substack{W_1,\dots,W_{\operatorname{tw}(H)} \in \mathcal{W} \\ W_1+\dots+W_{\operatorname{tw}(H)} = \widetilde{W}}} MP_{\operatorname{tw}(H)}(p^{(u_1)}_{t',W_1}, \dots, p^{(u_{\operatorname{tw}(H)})}_{t',W_{\operatorname{tw}(H)}}) \right) [R] \geq 1 \right]$$

We now show how to calculate this using a single tw(H)-wise matrix product of tensors with polynomials as entries. Let $T_{\operatorname{tw}(H)}$ be the additive group of order-tw(H) tensors of dimensions $n \times \ldots \times n$ with entries from $\mathbb{Z}$. Define $T_{\operatorname{tw}(H)}[X, X^{-1}]$ as the group of Laurent polynomials with elements of $T_{\operatorname{tw}(H)}$ as coefficients. Note how elements of $T_{\operatorname{tw}(H)}[X, X^{-1}]$ may also be viewed as tensors with Laurent polynomials as entries, or as functions $f : \mathbb{Z} \to T_{\operatorname{tw}(H)}$ when using the usual definition of scalar-tensor multiplication.

We define $p^{(u_i)}_{t'}(X) := \sum_{j \in \mathcal{W}} p^{(u_i)}_{t',j} \cdot X^j \in T_{\operatorname{tw}(H)}[X, X^{-1}]$ for all i. Viewing them as tensors of polynomials, we may calculate their tw(H)-wise matrix product. Viewing $MP_{\operatorname{tw}(H)}(p^{(u_1)}_{t'}, \dots p^{(u_{\operatorname{tw}(H)})}_{t'})$ as polynomial again, it can be easily seen that the coefficient tensor for $X^{\widetilde{W}}$ is exactly

$$\sum_{\substack{W_1,\dots,W_{\operatorname{tw}(H)} \in \mathcal{W} \\ W_1+\dots+W_{\operatorname{tw}(H)} = \widetilde{W}}} MP_{\operatorname{tw}(H)}(p^{(u_1)}_{t',W_{u_1}}, \dots, p^{(u_{\operatorname{tw}(H)})}_{t',W_{u_{\operatorname{tw}(H)}}})$$

Thus, to compute $D_{\widetilde{W}}[R]$, it suffices to compute $MP_{\operatorname{tw}(H)}(p^{(u_1)}_{t'}, \dots, p^{(u_{\operatorname{tw}(H)})}_{t'})$.

Hence, we have reduced the problem of calculating $D_{\widetilde{W}}$ to computing the tw(H)-wise matrix product of tensors whose entries are Laurent polynomials of degree bounded (in both directions) by $O(W)$. By Lemma 22, this can be done in time $O(n^{\operatorname{tw}(H)+1} W + n^{\operatorname{tw}(H)} W \log W poly(k))$.

Finally, **step 3** is just another discrete convolution. Let R be a configuration of $X_{t'}$. Defining $f_v, f_D : \mathbb{Z} \to \{0,1\}$ as $f_v(x) := p^{(v)}_{t',x}[R]$ and $f_D(x) := D_x[R]$ if $x \in \mathcal{W}$ and 0 everywhere else, equation (1) impllies $d_{t',W'}(R) = [(f_v * f_D)(W') \geq 1]$. Again, the discrete convolution may be implemented via a Fast Fourier Transform, leading to a running time of $O(n^{\operatorname{tw}(H)} W \log W)$.

Thus the case where $t' \in T$ is an inner node can be handled in time $O(n^{\operatorname{tw}(H)+1} W + n^{\operatorname{tw}(H)} W \log W \operatorname{poly}(k))$.

After calculating $d_{t',W'}$ for each $t' \in T, W' \in \mathcal{W}$, outputting the result is simple. Letting r' be the unique child of the root r of T, we output YES iff $\exists R : d_{r',-w(R)}(R) = 1$. By the definition of $d_{t',-w(R)}$, this is the case iff R is a partial solution of $X_{r'} \cap X_r = X_r$ in the cone $V_{r'} = V(H)$ with an extension of weight $-w(R)$. This is the case iff R can be expanded to a configuration for all of $V(H)$ such that its total weight is $-w(R) + w(R) = 0$. Hence, $\exists R : d_{t',-w(R)}(R) = 1$ iff the instance has a solution. Checking whether such an R exists can obviously be done in time $O(n^{\operatorname{tw}(H)} \operatorname{poly}(k))$.

We have shown that for each node of the tree decomposition, we need time at most $O(n^{\operatorname{tw}(H)+1} W + n^{\operatorname{tw}(H)} W \log W \operatorname{poly}(k))$. The tree decomposition has poly(k) nodes, so the total running time of the dynamic programming algorithm is $O((n^{\operatorname{tw}(H)+1} W + n^{\operatorname{tw}(H)} W \log W) \operatorname{poly}(k))$. ◀

We now consider the second part of the theorem. Similarly to the weighted case, it suffices to employ the algorithm from part 1.

Proof (of part 2 of Theorem 20). In the algorithm for part 1, all running times except for the $\mathrm{tw}(H)$-wise matrix product are bounded by $O(n^{\mathrm{tw}(H)} W \log W \operatorname{poly}(k))$. For $\mathrm{tw}(H) = 2$, the $\mathrm{tw}(H)$-wise matrix product is simply matrix multiplication, which can be done in $O(n^{\omega})$. Thus, for $\mathrm{tw}(H) = 2$ the running time is $O(n^{\omega} W \operatorname{poly}(k) + n^2 W \log W \operatorname{poly}(k))$. ◀

Finally, we come to the third part of the theorem. For reasons outlined in the proof of the previous theorem, the algorithm from part 1 does not work for $\mathrm{tw}(H) = 1$. Again, however, the result turns out to be quite simple for this case.

Proof (of part 3 of Theorem 20). Again, we only sketch the result, since it is easy to see. A simple dynamic programming algorithm on trees can be applied, storing for each node which configurations together with which weights can be achieved.

◀

6.3 Colored Subgraph Isomorphism for Bounded Pathwidth

Surprisingly, Colored Subgraph Isomorphism can be solved slightly faster on graphs of bounded pathwidth. This algorithm leverages rectangular matrix multiplication. In the following, for $z \in \mathbb{R}^+$, let $\omega(z)$ be the smallest real number such that multiplying a $n \times n$ matrix with a $n \times n^z$ matrix can be done in time $O(n^{\omega(z)})$. For a discussion of current best running times, see Section 4.7.

▶ **Theorem 23.** *There is an algorithm which, given an arbitrary instance $\phi = (H, G, f)$ of Colored Subgraph Isomorphism, solves ϕ in time*
1. $O(n^{\omega(\mathrm{pw}(H)-1)} \operatorname{poly}(k) + g(k))$ *when* $\mathrm{pw}(H) \geq 2$*, and*
2. $O(n^2 \operatorname{poly}(k)) + g(k)$ *when* $\mathrm{pw}(H) = 1$

where $k := |H|$, n is the size of the preimages of f, and g is a computable function.

This theorem implies Theorem 3 from the results section.

Proof. **Part 2** of the theorem is trivial, since for any graph, its treewidth is smaller than its pathwidth. Hence, by application of Theorem 19, we achieve the desired running time.

For **part 1**, let G, H and $f : V(G) \to V(H)$ be given. The algorithm first computes an optimal path decomposition $\mathcal{P} = (P, \{X_t\}_{t \in V(T)})$ of H in time $g(k)$ (see preliminaries), then does dynamic programming over $\mathcal{P}$.

To unify nomenclature and notation with the case of treewidth, we talk about a path as a tree rooted at one of its endpoints. As in the proof of Theorem 19, we modify the path decomposition $\mathcal{P}$ to satisfy certain properties. Specifically, we wish to obtain the following properties:
1. The bags of the root and leaf of the path have size $\mathrm{pw}(H)$, and every other bag has size $\mathrm{pw}(H) + 1$
2. For every $t \in V(P)$ with child t', we have $|X_t \cap X_{t'}| = \mathrm{pw}(H)$

These properties can be obtained via the techniques described in the proof of part 1 of Theorem 19.

We now do dynamic programming on this modified path decomposition. We only store values for each configuration of the separators $X_t \cap X_{t'}$. In particular, let t' be a node other than the root, and let t be its parent. As in Theorem 19, we only store

$$\begin{aligned} &d_{t'} : \mathcal{Conf}(X_t \cap X_{t'}) \to \{0,1\} \\ &d_{t'}(R) := \mathrm{ParSol}(R; X_t \cap X_{t'}; V_{t'}) \end{aligned} \tag{1}$$

We calculate these functions bottom-up. The case that t' is the leaf is analogous to the corresponding case in Theorem 19, taking time $O(n^{\mathrm{pw}(H)} \operatorname{poly}(k))$.

Now let t' with parent t and child t'' be an inner node of P. We define $\hat{v}$ to be the unique element with $\hat{v} \in X_{t'} \setminus X_t$ and similarly, $\hat{w} \in X_{t'} \setminus X_{t''}$ (or, if we would have $\hat{v} = \hat{w}$, we take $\hat{w}$ to be some vertex from $X_{t'} \setminus \{\hat{v}\}$ instead), and finally $E := X_{t'} \setminus \{\hat{v}, \hat{w}\}$. Now if $\hat{v}\hat{w} \notin E(H)$, the calculation is easy. Hence assume $\hat{v}\hat{w} \in E(H)$. For a configuration R of $X_t \cap X_{t'}$, we get the following alternate characterization of $d_{t'}(R)$:

$$\begin{aligned} d_{t'}(R) = [&\mathrm{ValConf}(R; \{\hat{w}\} \cup E) \wedge \\ &\exists v' \in f^{-1}(\hat{v}) : \mathrm{ParSol}((R \cup \{\hat{v} \mapsto v'\})|_{X_{t'} \setminus \{\hat{w}\}}; \{\hat{v}\} \cup E; V_{t''}) = 1 \wedge v'R(w) \in E(G)] \end{aligned}$$

We describe how to calculate $d_{t'}$ via rectangular matrix multiplication. Much like in Theorem 19, the matrices are indexed by configurations. In contrast to the former, however, one of the two dimensions of the matrix might correspond to the configuration of multiple vertices. Formally, for a vertex subset $Y \subseteq V(H)$ and a vertex $x \in V(H), x \notin Y$, we call a matrix A **indexed by configurations of** (x, Y) when it is of dimensions $n \times n^{|Y|}$. We use two arbitrary bijections $g_{\{x\}} : \mathcal{Conf}(\{x\}) \to [n]$ and $g_Y : \mathcal{Conf}(Y) \to [n^{|Y|}]$ which will help us map configurations of x and Y to indices of the matrix. Hence, for a configuration R of $\{x\} \cup Y$, we define $A[R] := A[g_x(R|_{\{x\}}), g_Y(R|_Y)]$.

For our rectangular matrix product, we define a $n \times n^{\mathrm{pw}(H)-1}$ matrix $B_{t'}$ indexed by configurations of $(\hat{v}, E)$. For a configuration R' of $\{\hat{v}\} \cup E$, we define

$$B_{t'}[R'] := d_{t''}(R') = \mathrm{ParSol}(R'; \{\hat{v}\} \cup E; V_{t''})$$

Now consider the adjacency matrix $Adj_{\hat{v},\hat{w}}$ of $f^{-1}(\hat{v})$ and $f^{-1}(\hat{w})$, indexed by configurations of $(\hat{w}, \{\hat{v}\})$. We make sure that the indexing bijection $g_{\{\hat{v}\}}$ as defined above is the same for both $Adj_{\hat{v},\hat{w}}$ and $B_{t'}$ and then calculate the matrix product $Adj_{\hat{v},\hat{w}} \cdot B_{t'}$. Naturally, the product is indexed by configurations of $(\hat{w}, E)$ and can be expressed as

$$\begin{aligned} (Adj_{\hat{v},\hat{w}} \cdot B_{t'})[R'] = \big[&\exists v' \in f^{-1}(\hat{v}) : \mathrm{ParSol}((R' \cup \{\hat{v} \mapsto v'\})|_{X_{t'} \setminus \{\hat{w}\}}; \{\hat{v}\} \cup E; V_{t''}) = 1 \\ &\wedge v'R(w) \in E(G)] \end{aligned}$$

Hence, we may write $d_{t'}(R)$ as

$$d_{t'}(R) = [\mathrm{ValConf}(R; \{\hat{w}\} \cup E) \wedge (Adj_{\hat{v},\hat{w}} \cdot B_{t'})[R] = 1] \tag{2}$$

The algorithm to calculate $d_{t'}$ is immediate. First, we calculate the rectangular matrix product of $Adj_{\hat{v},\hat{w}}$ and $B_{t'}$ in time $O(n^{\omega(\mathrm{pw}(H)-1)})$, then calculate $d_{t'}$ via formula 2. Checking whether R is a valid configuration of $\{\hat{w}\} \cup E$ can be done in time $\operatorname{poly}(\mathrm{pw}(H)) \leq \operatorname{poly}(k)$, giving us a total time of at most $O(n^{\omega(\mathrm{pw}(H)-1)} \operatorname{poly}(k))$ per inner node.

To output the answer, consider the child r' of the root r. We have by definition that $d_{r'}(R)$ is 1 iff R is a partial solution for X_r in $V(H)$. Thus, the input is a YES-instance for COLORED SUBGRAPH ISOMORPHISM iff there is an R such that $d_{r'}(R)$ is 1.

Since there is only a single leaf and $\operatorname{poly}(k)$ inner nodes, total running time of the dynamic programming algorithm is $O(n^{\omega(\mathrm{pw}(H)-1)} \operatorname{poly}(k))$. ◀

6.4 Exact Weight Colored Subgraph Isomorphism for Bounded Pathwidth

The techniques of using rectangular matrix multiplication for the case of pathwidth can also be applied to the weighted case, and they lead to improvements in the expected way. Again, the instances may be either node- or edge-weighted.

▶ **Theorem 24.** *There is an algorithm which, given an arbitrary instance $\phi = (H, G, f, w)$ of the* Exact Weight Colored Subgraph Isomorphism *problem, solves ϕ in time*
1. $O((n^{\omega(\mathrm{pw}(H)-1)}W + n^{\mathrm{pw}(H)}W \log W)\,\mathrm{poly}(k) + g(k))$ *when* $\mathrm{pw}(H) \geq 2$*, and*
2. $O((n^2W + nW \log W)\,\mathrm{poly}(k) + g(k))$ *when* $\mathrm{pw}(H) = 1$

where $k := |V(H)|$, n is the size of the preimages of f, W is the maximum absolute weight in the image of w, and g is a computable function.

From this, Theorem 4 from the results section follows directly via Lemma 18.

▶ **Lemma 25.** *Let two matrices A, B of dimensions $r \times s$ and $s \times t$ be given, such that each of their entries $A_{i,j}, B_{j,k} \in \mathbb{C}[X, X^{-1}]$ (for all $i \in [r], j \in [s], k \in [t]$) is a Laurent polynomial of degree bounded by Z in both the positive and the negative direction. Then their product can be computed in time $O(MM(r, s, t)Z + (rs + st)Z \log Z)$*

Proof. Analogous to Lemma 22. ◀

Proof (of theorem 24). Again, we only need to prove an algorithm for the edge-weighted case due to Proposition 21.

Part 2 is a corollary of Theorem 20, since for any graph, its treewidth is smaller than its pathwidth.

For **part 1**, we only sketch the proof, since it is a straightforward combination of the techniques used in the proofs of Theorems 20 and 23. We use notation and phrasing from both of those proofs without further mention.

Given G, H and f, we compute an optimal path decomposition $(P, \{X_t\}_{t \in V(P)})$ for H and do dynamic programming on the modified path decomposition as described in the proof of Theorem 23. For each non-root node $t' \in P$ with parent t, for each weight $W' \in \mathcal{W}$ and for each configuration R of $X_t \cap X_{t'}$, we store

$$d_{t',W'} := \mathrm{ParSolE}(R; X_t \cap X_{t'}; V_{t'}; W')$$

The case that t' is a leaf is clear. For the case that t' is an inner node with parent t and child t'', let $\hat{v}, \hat{w}$ be the unique elements with $\hat{v} \in X_{t'} \setminus X_t, \hat{w} \in X_{t'} \setminus X_{t''}$ (or, if we would have $\hat{v} = \hat{w}$, we take $\hat{w}$ to be some vertex from $X_{t'} \setminus \{\hat{v}\}$ instead) and $E = X_{t'} \setminus \{\hat{v}, \hat{w}\}$. If $\hat{v}\hat{w} \notin E(H)$, the calculation is easy, hence assume $\hat{v}\hat{w} \in E(H)$. For each weight W', we build a rectangular matrix $A_{t'}^{W'}$ indexed by $(\hat{v}, E)$. The entry corresponding to a configuration R of $\{\hat{v}\} \cup E$ tells us whether R has an extension of weight $W' + x$, where x is the weight contributed by $\hat{v}$ in $\{\hat{v}\} \cup E$. In particular $x := \sum_{\hat{u} \in E} w(R|_{\{\hat{u},\hat{v}\}})$.

We also use, for each weight W', an adjacency matrix $Adj_{\hat{v},\hat{w}}^{W'}$ defined for node-weighted instances as

$$Adj_{\hat{v},\hat{w}}^{W'}[v', w'] = [v'w' \in E(G) \wedge w(v'w') = W']$$

We then create two Laurent polynomials of degree $|\mathcal{W}|$, one with the matrices $Adj_{\hat{v},\hat{w}}^{W'}$ as coefficients, one with the matrices $A_{t'}^{W'}$. These can also be seen as matrices with Laurent polynomials as entries. Using Lemma 25, we then calculate their matrix product, which tells us for each weight W' and each configuration R of $\{\hat{w}\} \cup E$ if R is a potential solution of $\{\hat{w}\} \cup E$ in $V_{t'}$ with an extension of weight W', potentially missing edges between the preimage of w and the preimages of E. The latter can be checked for each entry. This leads to the desired running time.

◀

6.5 Improvements for the Node-Weighted Case

In the case of node weights instead of edge weights, some of the algorithms can be slightly improved using rectangular matrix multiplication. However, these improvements only work for the case that that the treewidth of H is 1 and for the case of bounded pathwidth.

Specifically, we show the following two results, which imply Theorems 5 and 6 from the results section.

▶ **Theorem 26.** *There is an algorithm which, given an arbitrary instance $\phi = (H, G, f, w)$ of the node-weighted* Exact Weight Colored Subgraph Isomorphism *problem where H is a tree, solves ϕ in time $O(\mathrm{MM}(n, n, W)\,\mathrm{poly}(k) + nW \log W\,\mathrm{poly}(k))$.*

▶ **Theorem 27.** *There is an algorithm which, given an arbitrary instance $\phi = (H, G, f, w)$ of the node-weighted* Exact Weight Colored Subgraph Isomorphism *problem, solves ϕ in time $O(MM(n, n, n^{\mathrm{pw}(H)-1}W)\,\mathrm{poly}(k) + g(k))$.*

Proof (of Theorem 26). Let $G, f : V(G) \to V(H)$ and $w : V(G) \to \mathbb{Z}$ be given. All achievable total weights must lie in $\mathcal{W} := \{-kW, \ldots, kW\}$.

We do dynamic programming on the tree H. Rooting H in an arbitrary vertex, we define T_u to be the subtree rooted at a node $u \in H$. For each weight $W \in \mathcal{W}$, each node $v \in H$, we store the following function of finite domain:

$$\begin{aligned} &d_{v,W} : \mathcal{Conf}(\{v\}) \to \{0, 1\} \\ &d_{v,W}(R) := \mathrm{ParSolE}(R; \{v\}; T_v; W) \end{aligned}$$

We calculate the entries of these functions bottom-up, starting at the leaves of H. If v is a leaf, $d_{v,W}(R)$ is 1 iff $W = 0$. Now suppose v is a non-leaf node of H with set of children C_v. For each child $u \in C_v$, we construct a rectangular matrix p_v^u. This matrix is indexed by w and $\mathcal{W}$. Formally, we call a matrix A **indexed by w and $\mathcal{W}$** if it has dimensions $n \times |\mathcal{W}|$. We use two arbitrary bijections $g_{\{v\}} : \mathcal{Conf}(\{v\}) \to [n]$ and $g_{\mathcal{W}} : \mathcal{W} \to [|\mathcal{W}|]$ to help us map weights from $\mathcal{W}$ and configurations of v to indices of A. Correspondingly, we define $A[R, W] := A[g_v(R), g_W(W)]$.

We define p_v^u as

$$p_v^u[R, W] := \mathrm{ParSolE}(R; \{v\}; \{v\} \cup T_u; W)$$

We may calculate this as follows. Let $Adj_{\{v,u\}}$ be the adjacency matrix of $f^{-1}(v)$ and $f^{-1}(u)$ indexed by $(v, \{u\})$ (as defined in the proof of Theorem 23), and let $\widetilde{d}_u$ be the rectangular matrix indexed by u and $\mathcal{W}$ and defined as $\widetilde{d}_u[R, W] := d_{u,W-w(R)}(R)$. We make sure that the indexing bijection $g_{\{u\}}$ as defined above is the same for both $Adj_{v,u}$ and $\widetilde{d}_u$ and then calculate the matrix product $Adj_{v,u} \cdot \widetilde{d}_u$. The product is indexed by configurations of v and $\mathcal{W}$ and can be expressed as

$$\begin{aligned} (Adj_{v,u} \cdot \widetilde{d}_u)[R, W] &= [\exists u' \in f^{-1}(u) : R(v)u' \in E(G) \wedge \mathrm{ParSolE}(R; \{u\}; T_u; W - w(u')) = 1] \\ &= \mathrm{ParSolE}(R; \{v\}; \{v\} \cup T_u; W) \\ &= p_v^u[R, W] \end{aligned}$$

Writing $C_v = \{u_1, \dots, u_c\}$ with $c = |C_v|$, the function $d_{v,W}$ may then be expressed as

$$d_{v,W}(R) = [\exists W_1, \dots, W_c : \sum_{i=1}^{c} W_i = W \wedge \forall i : p_v^{u_i}[R, W_i] = 1]$$

Just as in the proof of Theorem 20, this may be calculated using a discrete convolution. Accordingly, we define for each configuration R of v the finitely supported functions $f^1_{v,R}, \dots, f^c_{v,R} : \mathbb{Z} \to \{0, 1\}$ as $f^i_{v,R}(x) := p_v^{u_i}(R)$ if $x \in \mathscr{W}$, and 0 otherwise. By a simple calculation, we get $d_{v,W}(R) = [(f^1_{v,R} * \dots * f^c_{v,R})(W)]$.

Finally, after having calculated all values of $d_{t',W}(R)$ for all t', W and R, we wish to output the result. Let r be the root of H. By definition of $d_{r,W}$, there is some configuration R of r such that $d_{r,-w(R)}(R) = 1$ iff the instance has a solution.

It remains to analyze the running time. For the leaves of H, the calculation takes time $O(nW)$. For inner nodes, the calculation of the matrix product $\mathit{Adj}_{\{v,u\}} \cdot \widetilde{d_u}$ takes time $\mathrm{MM}(n, n, W)$. Finally, calculating the discrete convolutions takes time $O(nW \log W)$, since any vertex of v is involved as a child in at most one discrete convolution. Hence, we arrive at the running time from the theorem. ◀

Proof (of Theorem 27). The proof uses a combination of the techniques from the proofs of Theorem 26 and Theorem 23.

Let $G, H, f : V(G) \to V(H)$ and $w : V(G) \to \mathbb{Z}$ be given. We compute an optimal path decomposition in time $g(k)$, modify it as described in Theorem 23, obtaining a modified path decomposition $\mathscr{P} = (P, \{X_t\}_{t \in P})$, and then do dynamic programming on $\mathscr{P}$. Note that all achievable weights must lie in $\mathscr{W} := \{-kW, \dots, kW\}$.

We store, for each non-root node t' with parent t of P, the following function of finite domain:

$$d_{t',W}(R) := \mathrm{ParSolE}(R; X_t \cap X_{t'}; V_{t'}; W)$$

For t' a leaf, the calculations is clear. Let t' be an inner node with parent t and child t'' and define $\hat{v}$ to be the unique element with $\hat{v} \in X_{t'} \setminus X_t$ and similarly $\hat{w} \in X_{t'} \setminus X_{t''}$ (or, if we would have $\hat{v} = \hat{w}$, we take $\hat{w}$ to be some arbitrary vertex from $X_{t'} \setminus \{v\}$ instead) and finally $E := X_{t'} \setminus \{\hat{v}, \hat{w}\}$. If $\hat{v}\hat{w} \notin E(H)$, the calculation is easy, hence assume $\hat{v}\hat{w} \in E(H)$. For a configuration R of $X_t \cap X_{t'}$, we get the following alternate characterization of $d_{t',W}(R)$:

$$\begin{aligned} d_{t',W}(R) = \big[&\mathrm{ValConf}(R; \{\hat{w}\} \cup E) \wedge \exists v' \in f^{-1}(\hat{v}) : v'R(\hat{w}) \in E(G) \wedge \\ &\mathrm{ParSolE}((R \cup \{\hat{v} \mapsto v'\})|_{X_{t'} \setminus \{\hat{w}\}}; \{\hat{v}\} \cup E; V_{t''}; W - w(v')) = 1\big] \end{aligned}$$

We now set up our rectangular matrix product. For a vertex $x \in V(H), x \notin E$, we call a matrix A indexed by x, E and $\mathscr{W}$ if it has dimensions $n \times n^{|E|}|\mathscr{W}|$. We use two arbitrary bijections $g_{\{x\}} : \mathscr{Conf}(\{x\}) \to [n]$ and $g_{E,\mathscr{W}} : \mathscr{Conf}(E) \times \mathscr{W}) \to [n^{|E|}|\mathscr{W}|]$ to help index the matrix and define, for a configuration R of $\{x\} \cup E$, $A[R, W] := A[g_x(R|_x), g_{E,\mathscr{W}}(R|_E, W)]$.

We define the $n \times n^{\mathrm{pw}(H)-1}|\mathscr{W}|$ matrix $B_{t'}$, which is to be indexed by $\hat{v}, E$ and $\mathscr{W}$, as follows:

$$B_{t'}[R', W'] := d_{t',W'-w(R'|_{\hat{v}})}(R')$$

As in the proof of Theorem 23, we also use the adjacency matrix $\mathit{Adj}_{\hat{v},\hat{w}}$ indexed by configurations of $(\hat{w}, \{\hat{v}\})$. Ensuring that the indexing bijections $g_{\{\hat{v}\}}$ are the same for both $B_{t'}$ and

$Adj_{\hat{v},\hat{w}}$, we calculate $Adj_{\hat{v},\hat{w}} \cdot B_{t'}$ and obtain a matrix indexed by $\hat{w}, E$ and $\mathcal{W}$. Its entries can be expressed as

$$(Adj_{\hat{v},\hat{w}} \cdot B_{t'})[R', W'] = \big[\exists v' \in f^{-1}(\hat{v}) : v'R'(w) \in E(G) \ \wedge \\ \text{ParSolE}((R' \cup \{\hat{v} \mapsto v'\})|_{X_{t'} \setminus \{\hat{w}\}}; \{\hat{v}\} \cup E; V_{t''}; W' - w(v')) = 1\big]$$

Thus, $d_{t'}(R)$ can be expressed as

$$d_{t',W}(R) = [\text{ValConf}(R; \{\hat{w}\} \cup E) \wedge (Adj_{\hat{v},\hat{w}} \cdot B_{t'})[R, W] = 1]$$

This concludes the description of the computation of $d_{t',W}$. For the computation of the answer, consider the child r' of the root r. By definition of $d_{r',W}(R)$, we have that there exists a configuration R such that $d_{t',-w(R|_{X_r}}(R) = 1$ iff the instance has a solution.

By a simple argument, this dynamic programming algorithm has a running time of $O(MM(n, n, n^{\text{pw}(H)-1}W)\,\text{poly}(k))$.

◀

7 Hardness Results

7.1 The Main Lemma

We use a central Main Lemma to obtain all our lower bounds. This main lemma reduces Hyperclique instances to Exact Weight Colored Subgraph Isomorphism instances, where the pattern graph H is of a special form defined below.

▶ **Definition 28** (Twin Water Lily). *We call a graph H a **h-uniform Twin Water Lily of order** (s_1, s_2) the following holds: $V(H)$ consists of two independent sets S_1 and S_2 with r_1 and r_2 vertices, respectively. Additionally, for every size h subset $\{(v_1, s_1), \ldots, (v_h, s_h)) \in \binom{(S_1 \cup S_2) \times [h]}{h}$, it has a vertex v which is connected to all vertices in $\{v_1, \ldots, v_h\} \cap S_2$. We define P to be the set of all such v created in this way.*

Note that S_1 consists only of isolated vertices, and that H is bipartite.

▶ **Proposition 29.** *If a graph H is a h-uniform Twin Water Lily of order (s_1, s_2), then its treewidth is bounded by*

$$tw(H) \leq \begin{cases} s_2 - 1 & \text{if } s_2 > h \\ s_2 & \text{otherwise} \end{cases}$$

and its pathwidth is bounded by $pw(H) \leq s_2$.

Proof. There is a very useful characterization of treewidth using a graph-theoretic game: A graph G has treewidth $\leq k$ if and only if $k + 1$ cops can catch a visible robber on G [80]. The game works as follows: The $k + 1$ cops are placed on vertices of the graph. Then the robber may choose his or her starting vertex. The cops can always see the robber and adapt their strategy accordingly. Similarly, the robber can see the cops. The game now proceeds in steps, where in each step, one of the cops chooses an arbitrary destination vertex and takes off via helicopter in the direction of that vertex. While the cop is travelling, the robber sees where they will land and may now move arbitrarily along edges of the graph, as long as he does not pass through stationary cops. When the robber has finished moving, the cop lands. The cops win if and only if they are guaranteed to catch the cop after a finite number of moves, and lose otherwise.

To show the bound on the treewidth of H, simply place the cops on all vertices of S_2. No matter where the the robber starts, it is surrounded by cops or is on an isolated vertex. If $s_2 > h$, then there must exist some cop which is not adjacent to the robber, whom we can use to catch him in a single step. If $s_2 \leq h$, it is not guaranteed that there is a non-adjacent cop. However, we have an additional cop which can start at any vertex. As soon as the robber is positioned, we use the additional cop to capture them.

A similar characterization exists for pathwidth: A graph G has pathwidth $\leq k$ if and only if $k+1$ cops can catch an invisible robber on G [49]. The game can be formulated such that it is the same as the one for treewidth, but the cops simply cannot see the robber and must therefore have a universal strategy for catching him on G. This is also often referred to as the "infection cleansing" game.

For the pathwidth of H, we again place all cops on the vertices of S_2 and have one left over. We use this cop to go through all vertices not in S_2, one in each step. The robber is always surrounded by the cops in S_2 an hence cannot move, so after going through all vertices with the additional cop, we must have caught him. ◀

First, we state the Main Lemma. This result enables us to prove lower bounds for not only Exact Weight Colored Subgraph Isomorphism, but also for Colored Subgraph Isomorphism and Subset Sum.

▶ **Lemma 30** (The Main Lemma). *For any $\varepsilon \in (0,1)$ and any constant parameters $h \in \mathbb{N} \setminus \{1\}, r_1 \in \mathbb{N}, r_2 \in \mathbb{N}, \beta \in (0,1) \cap \mathbb{Q}$, there exists a $k \in \mathbb{N}$ and an algorithm $\mathcal{A}$ which*

(a) *accepts as input an instance $\mathcal{I} = G$ of h-uniform k-Hyperclique.*

(b) *produces an equivalent instance $\mathcal{I}' = (H', G', f', w')$ of Exact Weight Colored Subgraph Isomorphism, where H' is a h-uniform Twin Water Lily of order (r_1, r_2). The preimages of $\mathcal{I}'$ have size at most $\max\{n^{\beta k/r_1}, n^{(1-\beta)k/r_2}\}$, and the maximum weight is $W = \Theta(n^{(1+\varepsilon)\beta k})$.*

(c) *runs in time $O(poly(n) + (n^{\beta k/(hr_1)})^m)$ for some universal constant $m \in \mathbb{N}$.*

We also allow two further special cases of parameters.

1. *All parameters are as before, but $\beta = 0$ and $r_1 = 0$. In this case, the preimages in $\mathcal{I}'$ instead have size n^{k/r_2}, and all weights are zero. Furthermore, the running time of the reduction is only $poly(n)$.*
2. *All parameters are as before, but $\beta = 1$ and $r_2 = 0$. In this case, the preimages in $\mathcal{I}'$ instead have size n^{k/r_1}, and the maximum weight is $W = \Theta(n^{(1+\varepsilon)k})$.*

Intuitively, the parameter β indicates what percentage of the instance $\mathcal{I}$ should be encoded in which part of the Twin Water Lily. A percentage of β is encoded in the weights, while the remaining percentage of $(1-\beta)$ is encoded in the edges.

We now prove the Main Lemma.

Proof. Let us first focus on the case that $\beta \in (0,1)$, postponing the special cases to the end of the proof. For some k chosen later, let an instance $\mathcal{I}_0 = G_0$ of h-uniform k-Hyperclique be given, where k is chosen later. We convert this to an Exact Weight Colored Subgraph Isomorphism instance with a Twin Water Lily as pattern graph in five steps, each of which we explain in detail below: First, we convert it to a Colored k-Hyperclique instance in a standard way. Second, we split the instance into the part that we want to encode in the weights and the part that we want to encode in the edges. In both of these parts, we merge large groups of preimages such that we are left with only hr_1 in the weight part, and hr_2 in the edges part. Third, we go from hypercliques to Colored Subgraph Isomorphism in a standard way while preserving the preimages. Fourth, we convert the weight part of the instance into actually using weights by replacing edge constraints by weight constraints, using a construction known as k-average free sets. Finally, we merge preimages in both parts again to obtain the final Twin Water Lily instance.

1. **Converting to Colored Hyperclique:** We convert $\mathcal{I}_0$ to a new h-uniform Colored Hyperclique instance with k colors. The converted instance should have a hyperclique where all vertices has different colors if and only if the old instance has a hyperclique. The new instance have the form $\mathcal{I}_1 = (G_1, f_1)$, where the color homomorphism $f_1 : V(G_1) \to V(C_k)$ assigns each vertex of G_1 a vertex in the k-hyperclique C_k.
   Let $V(C_k) = \{1, \dots, k\}$. For each i, the preimage $f_1^{-1}(i)$ is a copy of $V(G_0)$. Let $g : V(G_1) \to V(G_0)$ be a function between sets that indicates which vertex in G_0 the vertex in G_1 is a copy of. Now for each set $\{v_1, \dots, v_h\} \in \binom{V(C_k)}{h}$, we go through all tuples $(w_1, \dots, w_h) \in f_1^{-1}(v_1) \times \dots \times f_1^{-1}(v_h)$ and create the edge $\{w_1, \dots, w_h\} \in E(G_1)$ if and only if $\{g(w_1), \dots, g(w_h)\} \in E(G_0)$.
   The correctness of this construction is easy to see.
2. **Merging Preimages:** We now convert $\mathcal{I}_1$ to a new h-uniform Colored $(hr_1 + hr_2)$-Hyperclique instance, by condensing groups of (small) preimages into single (large) preimages. In the converted instance $\mathcal{I}_2 = (G_2, f_2)$ with color homomorphism $f_2 : V(G_2) \to V(C_{r_1+r_2})$, we ensure that $V(C_{hr_1+hr_2})$ can be divided into two sets H_1, H_2 such that $|H_1| = hr_1, |H_2| = hr_2$ and $\forall v \in H_1 : |f_2^{-1}(v)| = n^{\beta k/(hr_1)}, \forall v \in H_2 : |f_2^{-1}(v)| = n^{(1-\beta)k/(hr_2)}$. These two sets correspond to the two water lilies constructed in later steps.
   At this point, we must make our choice of k. We will need that $k_1 := \beta k$ is an integer and divisible by hr_1. Furthermore, $k_2 := (1-\beta)k$ must also be an integer and divisible by hr_2. Letting $\beta = \frac{p}{q} \in (0,1) \cap \mathbb{Q}$ where $p, q \in \mathbb{N}$, it hence suffices to choose $k = hr_1r_2q$. Now, we split $V(C_k) = \{1, \dots, k\}$ into two groups $V_1 = \{1, \dots, k_1\}$ and $V_2 = \{k_1 + 1, \dots, k_2\}$. We split each of these sets further: V_1 is split into hr_1 disjoint groups $V_1, \dots, V_{hr_1}$ of size $\frac{k_1}{hr_1}$ each. Analogously, we split V_2 into hr_2 many disjoint groups $V_{hr_1+1}, \dots, V_{hr_1+hr_2}$ of size $\frac{k_2}{hr_2}$ each.
   Define $V(C_{hr_1+hr_2}) = \{v_1, \dots, v_{hr_1+hr_2}\}$ and furthermore let $S_1' = \{v_1, \dots, v_{hr_1}\}$, $S_2' = \{v_{hr_1+1}, \dots, v_{hr_1+hr_2}\}$. The vertex v_i corresponds to V_i, for every i. The vertices in their preimages represent all the configurations of the sets. In particular, we let every $v_i' \in f_2^{-1}(v_i)$ represent a configuration $\mathrm{conf}(v_i') \in \mathcal{C}onf(V_i)$. Hence, for $v_i \in S_1'$ the preimage $f_2^{-1}(v_i)$ has size $n^{k_1/(hr_1)}$, and for $v_i \in S_2'$ the preimage $f_2^{-1}(v_i)$ has size $n^{k_2/(hr_2)}$.
   Now for the edges. For every subset $\{v_{i_1}, \dots, v_{i_h}\} \in \binom{S_1' \cup S_2'}{h}$, we iterate over all $(v_{i_1}', \dots, v_{i_h}') \in f_2^{-1}(v_{i_1}) \times \dots, \times f_2^{-1}(v_{i_h})$. We combine the configurations that they represent by defining $R \in \mathcal{C}onf(V_{i_1} \cup \dots V_{i_h})$ as $\forall \ell \in [h] : \forall v_{i_\ell}' \in f_2^{-1}(v_{i_\ell}) : R(v_{i_\ell}') := \mathrm{conf}(v_{i_\ell}')$. Now we add $\{w_1, \dots, w_h\}$ as a hyperedge to $E(G_2)$ iff R is a valid configuration. That is, if the image of R induces a hyperclique in G_1.
   Again, correctness is easy to see.
3. **Representing Hyperedges by Intermediate Vertices:** We now go from the Colored Hyperclique instance $\mathcal{I}_2 = (G_2, f_2)$ to a (structured) Colored Subgraph Isomorphism instance $\mathcal{I}_3 = (H_3, G_3, f_3)$. The reduction is done in a standard way: We replace each hyperedge in G_2 with a vertex connected to all its endpoints.
   Formally, we need to construct H_3 and G_3. H_3 has three sets of vertices S_1', S_2' and P. S_1' and S_2' copy S_1' and S_2' from the last step, including their preimages. Accordingly, we write $S_1' = \{v_1, \dots, v_{hr_1}\}$ and $S_2' = \{v_{hr_1+1}, \dots, v_{hr_1+hr_2}\}$. In P, we have one vertex u for every subset $\{w_1, \dots, w_h\} \in \binom{C_{hr_1+hr_2}}{h}$, and we have $\forall \ell \in [h] : uw_\ell \in E(H_3)$. Now for

every hyperedge $\{w'_1, \ldots, w'_h\} \in E(G_2)$ with $\forall \ell \in [h] : w'_\ell \in f_2^{-1}(G_2)$, we add a vertex $u' \in f_3^{-1}(u)$ which is connected to all vertices $w'_1, \ldots, w'_h$. Note here that u' is only connected to one vertex from each preimage, which is a property that will be needed in the fourth step. This concludes the construction of $\mathcal{I}_3$.

Correctness of this construction is again easy to see. As for the size, note that preimages of vertices in S'_1 and S'_2 still have size $n^{k_1/(hr_1)}$ and $n^{k_2/(hr_2)}$, respectively. The preimages of vertices in P, however, have at most $(\max\{n^{k_1/(hr_1)}, n^{k_2/(hr_2)}\})^h = \max\{n^{k_1/r_1}, n^{k_2/r_2}\}$ vertices.

4. Replacing Some of the Edges with Weights We now come to the crucial step of converting some of the edge constraints to weight constraints. We convert the Colored Subgraph Isomorphism instance $\mathcal{I}_3 = (H_3, G_3, f_3)$ of the preceding step into an Exact Weight Colored Subgraph Isomorphism instance $\mathcal{I}_4 = (H_4, G_4, f_4, w_4)$. To do this, we will need so-called k-average free sets.

▶ **Definition 31** (k-average free sets)**.** *A set $S \subseteq \mathbb{Z}$ is called k-average-free if, for any $s_1, \ldots, s_{k'+1} \in S$ with $k' \leq k$, we have $s_1 + \ldots + s_{k'} = k' \cdot s_{k'+1}$ iff $s_1 = \ldots = s_{k'+1}$. In other words, the average of $s_1, \ldots, s_{k'} \in S$ is in S iff all s_i are equal.*[6]

We use the following construction for k-average free sets, originally proven in [21], modified into a more useful version in [9] and formulated in this form in [6].

▶ **Lemma 32.** *There exists a universal constant $c > 0$ such that, for all constants $\varepsilon \in (0, 1)$ and $k \geq 2$, a k-average-free set S of size n with $S \subseteq [0, k^{c/\varepsilon} n^{1+\varepsilon}]$ can be constructed in time* $\text{poly}(n)$.

Specifically, we use Lemma 32 with $\varepsilon' = \varepsilon, k' = \lambda := |P|$ and $n' = n^{k_1/(hr_1)}$. Letting $B := \lambda^{c/\varepsilon} n^{(1+\varepsilon)k_1/(hr_1)}$, this yields a λ-average free set $S \subseteq [0, B]$ of size $n^{k_1/(hr_1)}$. From this, we can construct an arbitrary bijection $\varrho_S : [n^{k_1/(hr_1)}] \to S$.

Now, to construct $\mathcal{I}_4$, we first copy $\mathcal{I}_3$, giving each node a default weight of "infinity" (i.e. something otherwise unobtainable, e.g. $k^2W + 1$). Now we delete all edges in H_3 which are incident to a vertex in S'_1, along with the corresponding edges in G_3. These are the edges that we replace by weight constraints.

Hence we now describe the weights. To make our construction easier, we specify a target value T (instead of the default target zero). We can easily get rid of this again by picking some vertex $\hat{w} \in V(H_3)$ and subtracting T from the weights of all of its preimages. The binary representation of T consists of $h \cdot a$ blocks of $\lceil \log(2\lambda B) \rceil$ bits, each containing the binary representation of λB. The i-th block represents the vertex $v_i \in S'_1$.

We move to the weights of the vertices, starting with vertices in the preimages of S'_1. Somewhat abusing notation, we define $\forall i \in [hr_1] : f_4^{-1}(v_i) = \{1, \ldots, n^{k_1/(hr_1)}\}$. The weight of vertex $v'_i \in f_4^{-1}(v_i)$ has a value of $\lambda B - |N(v_i)| \cdot \varrho_S(v'_i)$ in the i-th block, and a value of zero in all other blocks. Now for vertices in the preimages of$P$. Let $u \in P$ correspond to the set $\{w_1, \ldots, w_h\} \in \binom{C_{hr_1+hr_2}}{h}$. As observed in the preceding step, each $u' \in f_3^{-1}(u)$ is connected to exactly one vertex w'_i from each preimage $f_3^{-1}(w_i)$. In the current step, for each $i \in [h]$ with $w_i \in S'_1$, we have deleted the edges $u'w'_i$. To replace them, for each such i, we give u' a value of $\varrho_S(w'_i)$ in the i-th block. We have just changed the weight of u' at $|N(u) \cap S'_1|$ blocks of its binary representation. All other blocks have a value of zero.

[6] These k-average-free sets are a tool which are very useful for weighted problems, especially when they have additive elements. Such problems include k-sum, Subset Sum, Bin Packing, various scheduling problems, Tree Partitioning, Max-Cut, Maximum/Minimum Bisection, a Dominating Set variant with capacities, and similar [6, 7, 9, 17, 52, 60, 47]. Other uses of k-average-free sets in computer science include constructions in extremal graph theory, see e.g. [3, 4, 14].

All vertices with so far unspecified weight have weight zero. This concludes the construction of $\mathcal{I}_4$.

We show correctness of this construction. It suffices to show that any configuration R that is a solution for $\mathcal{I}_3$ is also a solution for $\mathcal{I}_4$ and vice versa. Hence, suppose R is a solution for $\mathcal{I}_3$. Then all edge constraints of H_4 are trivially fulfilled and we need only show that the total weight is T. For ease of discussion, we denote by $\alpha[i]$ the value of the i-th block of a weight α. Consider the blocks of the binary representation of the sum of weights $w(R) = \sum_{v \in \mathrm{Im}(R)} w_4(v)$, and let i be fixed. The large block size prevents overflow, so $w(R)[i] = \sum_{v \in \mathrm{Im}(R)} w_4(v)[i]$. By construction, we have $w(R)[i] = w(R(v_i))[i] + \sum_{u \in N(v_i)} w(R(u))[i]$. However since R is a valid configuration in $\mathcal{I}_3$, we have that for each u, $\forall v_j \in N(u) : R(u)R(v_j) \in E(G_3)$. In particular, $R(u)R(v_i) \in E(G_3)$ and hence by construction $w(R(u))[i] = \varrho_S(v_i)$. We conclude $w(R)[i] = \lambda B - |N(v_i)| \cdot \varrho_S(R(v_i)) + \sum_{i \in N(v_i)} \varrho_S(R(v_i)) = \lambda B = T[i]$. Hence $w(R)$ is equal to T in each of its blocks, which was to be proven.

Conversely, suppose R is a solution for $\mathcal{I}_4$. Then all edge constraints in $G_3[V(G_3) \setminus S_1']$ are trivially satisfied and we need only show that $\forall v_i \in S_1' : \forall u \in N(v_i) : R(v_i)R(u) \in E(G_3)$. Fix $v_i \in S_1'$. We have that $\lambda B = T[i] = w(R)[i] = w(R(v_i))[i] + \sum_{u \in N(v_i)} w(R(u))[i]$. Let $N(v_i) = \{u_1, \dots, u_{|N(v_i)|}\}$. For each $\ell \in |N(v_i)|$, we have that $R(u_\ell)$ is connected to some vertex $v_i^{(\ell)} \in f_3^{-1}(v_i)$, and hence that $w(R(u_\ell))[i] = \varrho_S(v_i^{(\ell)})$. Hence we have that $\lambda B = \lambda B - |N(v_i)| \cdot \varrho_S(R(v_i)) + \sum_{\ell \in |N(v_i)|} \varrho_S(v_i^{(\ell)})$. Hence $|N(v_i)| \cdot \varrho_S(R(v_i)) = \sum_{\ell \in |N(v_i)|} \varrho_S(v_i^{(\ell)})$. But because the values in the image of the bijection ϱ_S are a λ-average free set and $N(v_i) \subseteq P$ certainly has size less than $|P| = \lambda$, we have that $\forall \ell : v_i^{(\ell)} = R(v_i)$. Thus for all ℓ, $R(u_\ell)$ is connected to $R(v_i)$, which was to be proven.

5. Merging Preimages in S_1' and S_2': Lastly, we go from $\mathcal{I}_4 = (H_4, G_4, f_4, w_4)$ to the final instance $\mathcal{I}_5 = (H_5, G_5, f_5, w_5)$ where H_5 is a Twin Water Lily. Note that $\mathcal{I}_4$ is "almost" a the instance we want, save for the fact that the preimages P are much larger (size up to $\max\{n^{k_1/r_1}, n^{k_2/r_2}\}$) than the preimages of S_1' and S_2' (size $n^{k_1/(hr_1)}$ and $n^{k_2/(hr_2)}$, respectively). We rectify this by merging groups of vertices within S_1' and S_2'. In both sets, these groups have size h.

We split S_1' into r_1 groups $X_1, \dots, X_{r_1}$ of size h, and we split S_2' into r_2 groups $X_{r_1+1}, \dots, X_{r_1+r_2}$ of size h. In H_5, we have for every $i \in [r_1 + r_2]$ a vertex x_i representing X_i. Each vertex $x_i' \in f_5^{-1}(x_i)$ corresponds to a configuration $\mathrm{conf}(x_i') \in \mathcal{Conf}(X_i)$. The weight of x_i' is $w(\mathrm{conf}(x_i'))$, i.e. the sum of the weights of the vertices in the image of the configuration. In accordance with the definition of a Twin Water Lily, we define $S_1 = \{x_1, \dots, x_{r_1}\}$ and $S_2 = \{x_{r_1+1}, \dots, x_{r_1+r_2}\}$.

The set $P \subseteq V(H_5)$ remains the same as in the preceding step, including its preimages and the weights of the vertices in the preimages. For each $u \in P$, we go through the vertices v in the neighbourhood of u in H_4, and connect u to x_i such that $v \in X_i$. Note that each $u \in P$ is still connected to at most h other vertices (note, however, that it can be less if multiple vertices of its neighborhood came from the same group). Finally, we connect a vertex $u' \in f^{-1}(u)$ to a vertex $x_i' \in f_5^{-1}(x_i)$ if and only if $\forall z \in \mathrm{Im}(\mathit{conf}(x_i')) : u'z \in E(G_4)$.

Correctness is easy to see. Note that H_5 is a Twin Water Lily now. The set S_1 has size r_1 and S_2 has size r_2, with respective preimages of size n^{k_1/r_1} and n^{k_2/r_2}. The preimages of P still have size $\max\{n^{k1/r_1}, n^{k_2/r_2}\}$. Furthermore, the weights constructed in step 4 have hr_1 blocks of $\lceil \log(2\lambda B) \rceil$ bits, hence the maximum weight is $\Theta(2^\zeta)$ where $\zeta = hr_1 \cdot (\log(\lambda B) + O(1)) = hr_1 \cdot \log(|P|^{1+c/\varepsilon} n^{(1+\varepsilon)k_1/(hr_1)}) + O(1) = hr_1 \cdot \log(n^{(1+\varepsilon)k_1/(hr_1)}) + O(1) = (1+\varepsilon)k_1 \log(n) + O(1)$, hence as promised in the statement of the lemma the maximum weight is $\Theta(n^{(1+\varepsilon)k_1})$.

We have shown a reduction that has properties (b) and (c) from the lemma. We still have to analyze the running time. It is easy to see that steps 1, 2 and 3 run in time $poly(n)$, where the polynomial degree depends only on r_1, r_2 and h. In step 4, we need to construct the k-average free set S, which is done in time $poly(n^{k_1/(hr_1)}) = O((n^{k_1/(hr_1)})^m)$ for some universal constant $m \in \mathbb{Z}$. The rest of step 4 as well as step 5 can again be done in time $poly(n)$.

This proves the main case of the lemma. We still must prove the two special cases. In special case 1, where $\beta = 0$ and $a = 0$, we wish to encode everything in the weights. The above reduction still works. We must simply purge all parts of the construction that relate to the weights part. That is, no construction of V_1 in step 2, no construction of S_1' in steps 3 and 5 and no construction of S_2 in step 5. We also do not need to construct the λ-average free set in step 4, which means that we do not need the additional running time of $O((n^{(1+\varepsilon)k_1/(hr_1)})^m)$.

The second special case is analogous, except that we purge all parts of the construction that relate to the edge part. Of course, we still need the λ-average free set, so the running time does not change.

◀

7.2 Lower Bound Results

We now use the Main Lemma to prove our lower bound results. The main case of the lemma is used to prove the lower bounds for Exact Weight Colored Subgraph Isomorphism of bounded treewidth, while the first and second special case give lower bounds for Colored Subgraph Isomorphism and Subset Sum, respectively.

7.2.1 Exact Weight Colored Subgraph Isomorphism

The following theorem implies Theorem 9 via Lemma 18.

▶ **Theorem 33.** *For both the node- and edge weighted variant of the Exact Weight Colored Subgraph Isomorphism problems, for any $t \in \mathbb{N}$ and any $\gamma \in \mathbb{R}^+$, there is a connected, bipartite graph $\mathcal{H}_{t,\gamma}$ of treewidth t such that there cannot be an algorithm which solves all instances from $\{(\mathcal{H}_{t,\gamma}, G, f, w) \mid G$ is a graph and $W := \max_{z\in \mathrm{Im}(w)} |z| = \Theta(n^\gamma)\}$ in time*

1. *$O(n^{t+1-\varepsilon}W)$ or $O(n^{t+1}W^{1-\varepsilon})$ for $t \geq 3$, unless the h-uniform HYPERCLIQUE hypothesis fails for all $3 \leq h \leq t$,*
2. *$O(n^{t-\varepsilon}W)$ or $O(n^{t}W^{1-\varepsilon})$ for any t, unless the h-uniform HYPERCLIQUE hypothesis fails for all $h \geq 3$, or*
3. *$O(n^{(t+1)\omega/3-\varepsilon}W^{\omega/3})$ or $O(n^{(t+1)\omega/3}W^{\omega/3-\varepsilon})$ for any t, unless the CLIQUE hypothesis fails.*

Note that item 1 only gives lower bounds for $t \geq 3$, while items 2 and 3 are mostly interesting for the case $t \in \{1, 2\}$.

The following implies Theorem 11 from the results section.

▶ **Theorem 34** (Theorem 33 for pathwidth)**.** *Parts 2 and 3 of Theorem 33 also hold when replacing the treewidth t by the pathwidth p. Part 1 does not hold.*

▶ Remark 35. By the algorithm presented in Theorem 23, we cannot hope to obtain a lower bound as in part 1 of Theorem 33 for the case of pathwidth.

Proof (of theorem 33). Note that by Proposition 21, it suffices to prove lower bounds for the node-weighted case.

We begin with **part 1** of the theorem. Let $t \in \mathbb{N}$ and $\gamma \in \mathbb{R}^+$, as well as $3 \leq h \leq t$ be given. We apply the Main Lemma (Lemma 30) with

- some $\varepsilon' > 0$ chosen later,
- some $\beta' \in (0,1) \cap \mathbb{Q}$ chosen later,
- $h' := h$,
- $r_2' := t+1$ and
- some arbitrary $r_1' \in \mathbb{N}$ with
  - $r_1' > \frac{m\beta'}{h}$ (this ensures that the running time $O(poly(n) + n^{m\beta' k/(hr_1')})$ of the reduction is equal to $O(poly(n) + n^{k-\varepsilon})$ for some $\varepsilon > 0$, and can hence be ignored in the analysis) and
  - $r_1' > \frac{\beta'(t+1)}{1-\beta'}$ (this ensures that $\max\{n^{\beta' k/r_1'}, n^{(1-\beta')k/(t+1)}\} = n^{(1-\beta')k/(t+1)}$).

This produces a $k \in \mathbb{N}$ and a reduction algorithm $\mathcal{A}$ with the properties from the lemma. In particular, the reduction algorithm produces instances where the pattern graph H is a Twin Water Lily of order (r_1, r_2), which we define to be our graph $\mathcal{H}_{t,\gamma}$.

Now suppose there is an algorithm for the Exact Weights Colored Subgraph Isomorphism problem on pattern graph $\mathcal{H}_{t,\gamma}$ running in time $O(N^{t+1-\varepsilon}W)$ (the case $O(N^{t+1}W^{1-\varepsilon})$ is analogous). We show that the h-uniform Hyperclique hypothesis fails by showing that there is an algorithm for h-uniform k-Hyperclique running in time $O(n^{k-\varepsilon})$ for some $\varepsilon > 0$.

Given an h-uniform k-Hyperclique instance, we use algorithm $\mathcal{A}$ to obtain an equivalent instance of Exact Weight Colored Subgraph Isomorphism where the pattern graph is a Twin Water Lily of order $(r_1', t{+}1)$, the preimages have size $N = \max\{n^{\beta' k/r_1'}, n^{(1-\beta')k/(t+1)}\} = n^{(1-\beta')k/(t+1)}$, and the maximum weight is $W = \Theta(n^{(1+\varepsilon')\beta' k})$.

First, we make sure that $W = \Theta(N^\gamma)$ by choosing β' and ε' accordingly. Substituting, we get $n^{(1+\varepsilon')\beta' k} = \Theta(n^{\gamma(1-\beta')k/(t+1)})$, which is true if and only if

$$(1+\varepsilon')\beta' k = \frac{\gamma(1-\beta')k}{t+1} \iff \frac{\beta'}{1-\beta'} = \frac{\gamma}{(t+1)(1+\varepsilon')} \iff \varepsilon' = \frac{\gamma(1-\beta')}{(t+1)\beta'} - 1$$

Hence we choose ε' as such. However, to apply the Main Lemma, we must have $\varepsilon' \in (0,1)$. Hence we get the following two constraints for β':

$$\frac{\gamma(1-\beta')}{(t+1)\beta'} - 1 > 0 \iff \beta' < \frac{\gamma}{(t+1)+\gamma}$$
$$\frac{\gamma(1-\beta')}{(t+1)\beta'} - 1 < 1 \iff \beta' > \frac{\gamma}{2(t+1)+\gamma}$$

We incorporate these constraints later.

Now we need to ensure that the new running time we get is also small. We solve the Exact Weight Colored Subgraph Isomorphism instance in time $O(N^{t+1-\varepsilon}W) = O(N^{t+1+\gamma-\varepsilon}) = O(n^{(t+1+\gamma-\varepsilon)(1-\beta')k/(t+1)})$. Hence we get the following additional constraints on β':

$$\frac{(t+1+\gamma-\varepsilon)(1-\beta')}{t+1} < 1 \iff 1-\beta' < \frac{t+1}{t+1+\gamma-\varepsilon} \iff \beta' > \frac{\gamma-\varepsilon}{(t+1)+\gamma-\varepsilon}$$

Combining these three constraints on β', we get

$$\max\left\{\frac{\gamma-\varepsilon}{(t+1)+\gamma-\varepsilon}, \frac{\gamma}{2(t+1)+\gamma}\right\} < \beta' < \frac{\gamma}{(t+1)+\gamma}$$

It is always possible to choose a $\beta' \in (0,1) \cap \mathbb{Q}$ such that this is true.

Part 2 of the theorem is very much analogous. Note that the loss of the 1 in the exponent is due to the weaker bound in Proposition 29.

Part 3 is completely analogous for $t \geq 2$; we simply always choose $h = 2$. The $\omega/3$ in the bound comes from the Clique hypothesis.

However, a small trick has to be used for $t = 1$, since a 2-uniform Twin Water Lily of order $(r_1, 2)$ has treewidth 2, not 1. To get the better lower bound, we have to slightly modify the proof of the Main Lemma for $h = 2$ in step 3. Instead of replacing the edges between vertices of S_2 by intermediate vertices, we simply leave them as-is. Now the resulting graph is not a Twin Water Lily anymore, but does always have treewidth r_2. We omit the details. ◀

Finally, we prove the same theorem for pathwidth.

Proof (of Theorem 34). Proving part 2 is exactly analogous to part 2 of the theorem for treewidth.

Now remember that we needed a slight modification of the proof of the Main Lemma for part 3 of the theorem for treewidth for $t = 1$. For part 3 of the theorem for pathwidth, we actually need that modification for all t. Using this, the rest of the proof is clear. Again, we omit the details. ◀

7.2.2 Unweighted Colored Subgraph Isomorphism

▶ **Theorem 36.** *For each $t \in \mathbb{N}$ there is a connected,bipartite graph $\mathcal{H}_t$ of treewidth t such that there cannot be an algorithm which solves all instances from $\{(\mathcal{H}_t, G, f) \mid G \text{ is a graph}\}$ of Colored Subgraph Isomorphism in time*

1. *$O(n^{t+1-\varepsilon})$ for $t \geq 3$, unless the h-uniform hyperclique hypothesis fails for all $3 \leq h \leq t$,*
2. *$O(n^{t-\varepsilon})$ for any $t \geq 2$, unless the h-uniform hyperclique hypothesis fails for all $h \geq 3$, or*
3. *$O(n^{(t+1)\omega/3-\varepsilon})$ for $t \geq 2$, unless the $(t+1)$-clique hypothesis fails.*

Trivially, we get the following corollary.

▶ **Corollary 37.** *For no $t \in \mathbb{N}$ can there be an algorithm solving Exact Weight Colored Subgraph Isomorphism for pattern graphs of treewidth t as in parts 1-3 of Theorem 36.*

▶ **Theorem 38** (Theorem 36 for pathwidth). *Part 2 of Theorem 36 also holds when replacing the treewidth t by the pathwidth p. Part 3 only holds when replacing $t + 1$ by p. Part 1 does not hold.*

▶ **Corollary 39.** *For no $p \in \mathbb{N}$ can there be an algorithm solving Exact Weight Colored Subgraph Isomorphism for pattern graphs of pathwidth p as in parts 1-3 of Theorem 38.*

Unfortunately, Lemma 18 cannot be applied to Theorem 38, and hence we have no lower bounds for the uncolored, unweighted case of bounded pathwidth. We believe it is unlikely that the techniques used to prove 18 for the uncolored, unweighted case of bounded treewidth generalize to pathwidth. Note, however, that Theorem 36 implies Theorem 7 by Lemma 18.

We now prove the theorems above.

Proof (of Theorem 36). For the proof of this theorem, we use the first special case of the Main Lemma (Lemma 30).

We begin with **part 1**. Let $t \in \mathbb{N}, \gamma \in \mathbb{R}^+$ and $3 \leq h \leq t$ be given. We apply the first special case of the Main Lemma with

- some $\varepsilon' > 0$ chosen later
- $h' := h$
- $r_2' = t + 1$

This produces a $k \in \mathbb{N}$ and a reduction algorithm $\mathcal{A}$ with the properties from the lemma. In particular, the reduction algorithm produces instances where the patern graph H is a Twin Water Lily of order $(0, r_2)$, which we define to be our graph $\mathcal{H}_{t,\gamma}$.

Now suppose there is an algorithm for the Exact Weights Colored Subgraph Isomorphism problem on pattern graph $\mathcal{H}_{t,\gamma}$ running in time $O(N^{t+1-\varepsilon}W)$ (the case $O(N^{t+1}W^{1-\varepsilon})$ is analogous). We show that the h-uniform Hyperclique hypothesis fails by showing that there is an algorithm for h-uniform k-Hyperclique running in time $O(n^{k-\varepsilon})$ for some $\varepsilon > 0$.

Given an h-uniform k-Hyperclique instance, we use algorithm $\mathcal{A}$ to obtain an equivalent instance of Exact Weight Colored Subgraph Isomorphism where the pattern graph is a Twin Water Lily of order $(0, t+1)$, the preimages have size $N = n^{k/(t+1)}$, and all weights are zero.

By simply removing the weights, this is a Colored Subgraph Isomorphism instance with the same properties. Solving this with the fast algorithm from above, we get a running time of $O(N^{t+1-\varepsilon}) = O(n^{(t+1-\varepsilon)k/(t+1)} = O(n^{k-\varepsilon/t+1})$. Hence we are done.

Part 2 is analogous. Again, as in Theorem 33 the loss of 1 in the exponent is due to the weaker bound in Proposition 29.

Part 3 is obvious: We simply choose $\mathcal{H}_t$ to be the $(t+1)$-clique and use the clique hypothesis. The Main Lemma is not needed.

◀

Proof (of Theorem 38). Completely analogous. ◀

7.2.3 Subset Sum

We now prove the conditional lower for Subset Sum, restated below for convenience.

▶ **Theorem 13.** *For no $\varepsilon > 0$ can there be an algorithm which solves Subset Sum in time $O(T^{1-\varepsilon}\,\mathrm{poly}(n))$ unless the h-uniform Hyperclique hypothesis fails for all $h \geq 3$.*

Before proving this result, we need a lemma that shows that we can reduce k-sum to Subset Sum with minimal overhead. This theorem is already known, but we could not find a formal proof of it in the literature. Therefore we provide one here.

▶ **Lemma 40** (Reducing k-Sum to Subset Sum). *There is an algorithm $\mathcal{B}$ which, given as input a k-sum instance with N values from $[0, D]$ per set, as well as a target T, constructs an equivalent Subset Sum instance with $k \cdot N$ values in $[0, Dg(k)]$ and a target T' bounded by $Tg(k)$ for a computable function g. Furthermore, for constant k, $\mathcal{B}$ runs in time linear in the input size.*

Proof. The values of the Subset Sum instance are the union of the k sets. However, we modify the weights and target as follows. At the front of the binary representation of the weights and the target, we add a buffer of $\lceil \log(k) \rceil$ zero bits to avoid overflow, then another k bits constituting a “checklist”, then in front of that another buffer of $\lceil \log(k) \rceil$ zero bits and finally another $\lceil \log(k) \rceil$ bits which contains a counter for the number of nodes of the solution.

The target T has the binary representation of k in the counter bits and only ones in the checklist bits. Each weight has the binary representation of 1 in the counter bits, ensuring that we take exactly k weights. Furthermore, if the weight comes from the i-th set of the k-SUM instances, its checklist bits is zero except for the i-th position.

Now if one picks more than $2^{\lceil\log(k)\rceil}$ values, the counter at the front overflows the length of the target, so that selection cannot be a solution. Since $2^{\lceil\log(k)\rceil} < 2k$ and since there is a buffer of $\lceil\log(k)\rceil$ bits, the checklist cannot overflow into the counter. Hence any solution must pick exactly k weights. Hence, the only way to achieve all ones in the checklist bits of a sum of k weights is to pick exactly one weight from each of the k sets. This completes the reduction.

Since we add $3\lceil\log(k)\rceil + k$ bits to the weights and the target, their value is multiplied by at most $2^k \cdot 2^3 \cdot k^3$. Choosing $g(k) = 2^{k+3}k^3$, we obtain the bounds from the lemma. ◀

We now prove the theorem about SUBSET SUM. Essentially, the instances we get from the second special case of the Main Lemma are k-SUM instances that we can then reduce to SUBSET SUM via the lemma above.

Proof (of Theorem 13). Suppose there is an algorithm solving SUBSET SUM in time $O(T^{1-\varepsilon}N^z)$ for some $z \in \mathbb{N}$. We use the second special case of the Main Lemma with

- some ε' chosen later,
- $h' := h$, and
- some arbitrary $r_1' \in \mathbb{N}$ such that
  - $r_1' > \frac{m\beta'}{h}$ (this ensures, again, that the running time $O(poly(n) + n^{m\beta' k/(hr_1')})$ of the reduction is equal to $O(poly(n) + n^{k-\varepsilon})$ for some $\varepsilon > 0$, and can hence be ignored in the analysis), and
  - $r_1' > z\varepsilon'$ (this ensures that $n^{zk/r_1'} < n^{\varepsilon' k}$).

This yields a $k \in \mathbb{N}$ and a reduction algorithm $\mathcal{A}$ with the properties from the lemma. In particular, the reduction algorithm produces an EXACT WEIGHT COLORED SUBGRAPH ISOMORPHISM where H consists only of isolated vertices with preimages of size $O(n^{k/r_1})$ and maximum absolute weight $W = \Theta(n^{(1+\varepsilon)k})$. We make all weights positive by adding a large number to each, such that the target is $T = \Theta(n^{(1+\varepsilon)k})$. Note that this instance is also a $(r_1 + \binom{hr_1}{h})$-SUM instance, with each set of numbers being the set of weights in a preimage.

We now use the algorithm $\mathcal{B}$ from Lemma 40 to convert this to a Subset Sum instance with $N = O(n^{k/r_1})$ values in $[0, \Theta(n^{(1+\varepsilon')k})]$ and target $T = \Theta(n^{(1+\varepsilon')k})$.

We can now solve the instance in time $O(n^{(1-\varepsilon)(1+\varepsilon')k} n^{zk/r_1}) = O(n^{((1-\varepsilon)(1+\varepsilon')+\varepsilon')k})$. Hence it suffices to choose ε' such that

$$(1-\varepsilon)(1+\varepsilon') + \varepsilon' < 1 \iff \varepsilon' < \frac{\varepsilon}{2-\varepsilon}$$

Since $\varepsilon < 1$ and hence $\frac{\varepsilon}{2-\varepsilon} > 0$, this is always possible.

◀

8 Open Problems

We conclude with some open problems.

1. Can the algorithms for weighted trees be improved? We have shown that some small improvements can be made for node-weighted trees (see Theorem 5), but are these optimal? What about edge-weighted trees?
2. Are there faster algorithms for unweighted SUBGRAPH ISOMORPHISM on graphs of bounded pathwidth, independent of running time improvements for rectangular matrix multiplication? If so, can they be adjusted for the weighted case? See Theorems 3 and 4.

3. Relatedly, are there good lower bounds for unweighted Subgraph Isomorphism on graphs of bounded pathwidth? These could be attained via a modification of the proof of part 2 of Lemma 18 for pathwidth (though we saw no way to do this), or with completely new techniques.
4. Do our algorithms and lower bounds also work for other types of subgraph homomorphisms, and for counting the number of solutions? It seems like techniques from [43] should work.

References

1 Amir Abboud, Arturs Backurs, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Or Zamir. Subtree isomorphism revisited. *ACM Transactions on Algorithms (TALG)*, 14(3):1–23, 2018.

2 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is Valiant's parser. *SIAM Journal on Computing*, 47(6):2527–2555, 2018.

3 Amir Abboud and Greg Bodwin. The 4/3 additive spanner exponent is tight. *Journal of the ACM (JACM)*, 64(4):1–20, 2017.

4 Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. *SIAM Journal on Computing*, 47(6):2203–2236, 2018.

5 Amir Abboud, Karl Bringmann, Holger Dell, and Jesper Nederlof. More consequences of falsifying SETH and the Orthogonal Vectors conjecture. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 253–266, 2018.

6 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for Subset Sum and Bicriteria Path. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 41–57. SIAM, 2019.

7 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. Scheduling lower bounds via AND Subset Sum. *arXiv preprint arXiv:2003.07113*, 2020.

8 Amir Abboud and Kevin Lewi. Exact weight subgraphs and the k-SUM conjecture. In *International Colloquium on Automata, Languages, and Programming*, pages 1–12. Springer, 2013.

9 Amir Abboud, Kevin Lewi, and Ryan Williams. Losing weight by gaining edges. In *European Symposium on Algorithms*, pages 1–12. Springer, 2014.

10 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 434–443. IEEE, 2014.

11 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *International Colloquium on Automata, Languages, and Programming*, pages 39–51. Springer, 2014.

12 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM Journal on Computing*, 47(3):1098–1122, 2018.

13 Josh Alman and Virginia Vassilevska Williams. OV graphs are (probably) hard instances. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

14 Noga Alon. Testing subgraphs in large graphs. *Random Structures & Algorithms*, 21(3-4):359–370, 2002.

15 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.

16 Omid Amini, Fedor V Fomin, and Saket Saurabh. Counting subgraphs via homomorphisms. In *International Colloquium on Automata, Languages, and Programming*, pages 71–82. Springer, 2009.

17 Zhao An, Qilong Feng, Iyad Kanj, and Ge Xia. The complexity of tree partitioning. In *Workshop on Algorithms and Data Structures*, pages 37–48. Springer, 2017.

18 Per Austrin, P Kaski, and K Kubjas. Tensor network complexity of multilinear maps. In *10th Innovations in Theoretical Computer Science, ITCS 2019, 10-12 January 2019, San Diego, United States*. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2019.

19 Arturs Backurs and Christos Tzamos. Improving Viterbi is hard: Better runtimes imply faster clique algorithms. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 311–321. JMLR. org, 2017.

20 MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Saeed Seddighin, and Cliff Stein. Fast algorithms for knapsack via convolution and prediction. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1269–1282, 2018.

21 Felix A Behrend. On sets of integers which contain no three terms in arithmetical progression. *Proceedings of the National Academy of Sciences of the United States of America*, 32(12):331, 1946.

22 Hans L Bodlaender. A tourist guide through treewidth. *Acta cybernetica*, 11(1-2):1, 1994.

23 Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.

24 Hans L Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical computer science*, 209(1-2):1–45, 1998.

25 Hans L Bodlaender. Discovering treewidth. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 1–16. Springer, 2005.

26 David Bremner, Timothy M Chan, Erik D Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, and Perouz Taslakian. Necklaces, convolutions, and x+ y. In *European Symposium on Algorithms*, pages 160–171. Springer, 2006.

27 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 661–670. IEEE, 2014.

28 Karl Bringmann. A near-linear pseudopolynomial time algorithm for Subset Sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1073–1084. SIAM, 2017.

29 Karl Bringmann, Nick Fischer, and Marvin Künnemann. A fine-grained analogue of Schaefer's theorem in P: Dichotomy of exists^k-forall-quantified first-order graph properties. In *34th Computational Complexity Conference (CCC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

30 Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 307–318. IEEE, 2017.

31 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 79–97. IEEE, 2015.

32 Karl Bringmann and Philip Wellnitz. Clique-Based Lower Bounds for Parsing Tree-Adjoining Grammars. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching (CPM 2017)*, volume 78 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: `http://drops.dagstuhl.de/opus/volltexte/2017/7332`, `doi:10.4230/LIPIcs.CPM.2017.12`.

33 Danilo Bruschi, Lorenzo Martignoni, and Mattia Monga. Detecting self-mutating malware using control-flow graph matching. In *International conference on detection of intrusions and malware, and vulnerability assessment*, pages 129–143. Springer, 2006.

34 Timothy M Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 31–40, 2015.

35 Jianer Chen, Xiuzhen Huang, Iyad A Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346–1367, 2006.

36 Lijie Chen, Shafi Goldwasser, Kaifeng Lyu, Guy N Rothblum, and Aviad Rubinstein. Fine-grained complexity meets IP = PSPACE. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–20. SIAM, 2019.

37 Ruiwen Chen and Rahul Santhanam. Improved algorithms for sparse MAX-SAT and MAX-k-CSP. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 33–45. Springer, 2015.

38 James W Cooley and John W Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

39 Don Coppersmith. Rapid multiplication of rectangular matrices. *SIAM Journal on Computing*, 11(3):467–471, 1982.

40 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 1–6, 1987.

41 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. 2009.

42 Bruno Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.

43 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 210–223, 2017.

44 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Transactions on Algorithms (TALG)*, 12(3):1–24, 2016.

45 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 4. Springer, 2015.

46 Evgeny Dantsin and Alexander Wolpert. MAX-SAT for formulas with constant clause density can be solved faster than in $\mathcal{O}(2^n)$ time. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 266–276. Springer, 2006.

47 Bartłomiej Dudek, Paweł Gawrychowski, and Tatiana Starikovskaya. All non-trivial variants of 3-LDT are equivalent. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 974–981, 2020.

48 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1-3):57–67, 2004.

49 John A Ellis, Ivan Hal Sudborough, and Jonathan S Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1):50–79, 1994.

50 David Eppstein. Subgraph Isomorphism in planar graphs and related problems. In *Graph Algorithms and Applications I*, pages 283–309. World Scientific, 2002.

51 Jeff Erickson et al. Lower bounds for linear satisfiability problems. In *SODA*, pages 388–395, 1995.

52 Fedor V Fomin, Petr A Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM Journal on Computing*, 43(5):1541–1563, 2014.

53 Fedor V Fomin, Daniel Lokshtanov, Venkatesh Raman, Saket Saurabh, and BV Raghavendra Rao. Faster algorithms for finding and counting subgraphs. *Journal of Computer and System Sciences*, 78(3):698–706, 2012.

54 François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1029–1046. SIAM, 2018.

55 Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and Ryan Williams. Completeness for first-order properties on sparse structures with algorithmic applications. *ACM Transactions on Algorithms (TALG)*, 15(2):1–35, 2018.

56 Edinah K Gnang, Ahmed Elgammal, and Vladimir Retakh. A spectral theory for tensors. In *Annales de la Faculté des sciences de Toulouse: Mathématiques*, volume 20, pages 801–841, 2011.

57 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

58 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*, pages 653–662. IEEE, 1998.

59 Bart MP Jansen and Michał Włodarczyk. Optimal polynomial-time compression for Boolean Max CSP. *arXiv preprint arXiv:2002.03443*, 2020.

60 Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79(1):39–49, 2013.

61 Ce Jin and Hongxun Wu. A simple near-linear pseudopolynomial time randomized algorithm for subset sum. In *2nd Symposium on Simplicity in Algorithms (SOSA 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

62 Konstantinos Koiliaris and Chao Xu. A faster pseudopolynomial time algorithm for Subset Sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1062–1072. SIAM, 2017.

63 Konstantinos Koiliaris and Chao Xu. Subset Sum made simple. *arXiv preprint arXiv:1807.08248*, 2018.

64 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

65 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*, pages 296–303, 2014.

66 Yuan Li, Alexander Razborov, and Benjamin Rossman. On the AC^0 complexity of Subgraph Isomorphism. *SIAM Journal on Computing*, 46(3):936–971, 2017.

67 Andrea Lincoln and Nikhil Vyas. Algorithms and lower bounds for cycles and walks: Small space and sparse graphs. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

68 Andrea Lincoln, Virginia Vassilevska Williams, and Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1236–1252. SIAM, 2018.

69 Grazia Lotti and Francesco Romani. On the asymptotic complexity of rectangular matrix multiplication. *Theoretical Computer Science*, 23(2):171–185, 1983.

70 Josef Malík, Ondřej Suchý, and Tomáš Valla. Efficient implementation of Color Coding algorithm for Subgraph Isomorphism problem. In *International Symposium on Experimental Algorithms*, pages 283–299. Springer, 2019.

71 Dániel Marx. Can you beat treewidth? In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 169–179. IEEE, 2007.

72 Dániel Marx and Michal Pilipczuk. Everything you always wanted to know about the parameterized complexity of Subgraph Isomorphism (but were afraid to ask). In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, volume 25 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 542–553, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer

Informatik. URL: `http://drops.dagstuhl.de/opus/volltexte/2014/4486`, `doi:10.4230/LIPIcs.STACS.2014.542`.

73 Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

74 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

75 Kevin Pratt. Waring rank, parameterized and exact algorithms. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 806–823. IEEE, 2019.

76 Andrzej Proskurowski and Jan Arne Telle. Classes of graphs with restricted interval models. *Discrete Mathematics and Theoretical Computer Science*, 3(4), 1999.

77 Bellman Richard. Dynamic programming. *Princeton University Press*, 89:92, 1957.

78 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 515–524, 2013.

79 Gregory Rosenthal. Beating treewidth for average-case subgraph isomorphism. In *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

80 Paul D Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–33, 1993.

81 Andrew James Stothers. On the complexity of matrix multiplication. 2010.

82 Edward H Sussenguth. A graph-theoretic algorithm for matching chemical structures. *Journal of Chemical Documentation*, 5(1):36–43, 1965.

83 R Ryan Williams. *Algorithms and Resource Requirements for Fundamental Problems*. PhD thesis, Carnegie Mellon University Pittsburgh, PA, 2007.

84 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005.