Four shorts stories on surprising algorithmic uses of treewidth

Dániel Marx

Dedicated to Hans L. Bodlaender on the occasion of his 60th birthday.

Abstract

This article briefly describes four algorithmic problems where the notion of treewidth is very useful. Even though the problems themselves have nothing to do with treewidth, it turns out that combining known results on treewidth allows us to easily describe very clean and high-level algorithms.

1 Introduction

While the definition of treewidth may seem very technical at first sight, the naturality of treewidth is witnessed by the fact that it was introduced independently at least three times with equivalent definitions by different authors [7, 50, 69]. One may arrive to the study of treewidth from various directions and justify its importance with different arguments. One can, for example, argue that graphs of low treewidth (or some generalization of it) appear naturally in certain applications [14,38,60,73], hence algorithms for such graphs could be of practical interest. Or one could say that algorithms on bounded-treewidth graphs are based on the fundamental idea of recursively splitting the problem along small separators, and the study of treewidth is a good formalization of the study of this basic principle. But perhaps the nicest and most surprising reason for arriving at this notion is when the original goal has nothing to do with treewidth, but suddenly treewidth appears as the right theoretical tool for handling the problem. This article contains four such "war stories," where the notion of treewidth and algorithms for bounded-treewidth graphs give very elegant solutions, which are sometimes in fact more efficient than those that were obtained earlier by involved and problem-specific techniques.

The four stories below are intentionally kept very brief in order to highlight the conceptual simplicity of the arguments. The aim is to show how certain high-level results can be combined in a clean way to achieve our goals. The detailed discussions or proofs of the results we are building on are beyond the scope of this article. Later in this volume, the article of Marcin Pilipczuk contains more advanced examples of algorithmic use of treewidth bounds [63].

2 Bidimensionality

Restricting an algorithmic problem to a certain family of graphs can make it easier than trying to solve it in general on every possible graph. A large part of the literature on algorithmic graph theory concerns algorithms for restricted classes of graphs that are of practical or theoretical significance. Restriction to planar graphs are studied both because of their interesting mathematical properties and as a starting point for modelling, e.g., road networks or 2D geometric problems.

From the viewpoint of polynomial-time solvability vs. NP-hardness, the restriction to planarity does not seem to make the problem significantly easier. Most of the classic NP-hard problems

(e.g., 3-Coloring, Maximum Indepenent Set, Hamiltonian Cycle, etc.) remain NP-hard on planar graphs. The situation is very different from the viewpoint of parameterized complexity. Many of the basic problems that are W[1]-hard on general graphs turn out to be FPT on planar graphs. In fact, it took some time to arrive to the first relatively simple and natural problems that are W[1]-hard on planar graphs [13, 19].

The restriction to planarity can help even for problems that are already FPT for general graphs. One of the main goals of the area of parameterized algorithms is to design algorithms with running time $f(k)n^{O(1)}$ such that the dependence $f(k)$ on the parameter is a function that grows as slowly as possible. For many of the fundamental problems studied in parameterized algorithms (e.g., Vertex Cover, Feedback Vertex Set, k-Path, Odd Cycle Transversal), algorithms with running time $2^{O(k)}n^{O(1)}$ are known. Furthermore, it is very likely that this form of running time is optimal: it is known that, under the Exponential Time Hypothesis (ETH) [51, 52], no algorithm with running time $2^{o(k)}n^{O(1)}$ exists for these problems. When restricted to planar graphs, significantly better algorithms are known for many of these problems, typically with running times of the form $2^{O(\sqrt{k})}n^{O(1)}$ or $2^{O(\sqrt{k}\log k)}n^{O(1)}$. Below we show how a very clean argument based on treewidth delivers such agorithms for certain basic problems; for others, more involved problem-specific ideas are needed [1, 35, 44, 54, 58, 64, 65]. The main argument we present here was described first by Fomin and Thilikos [46] (for the Dominating Set problem) and was further developed under the name "bidimensionality" (see, e.g., [28–31]).

Let us consider the k-Path problem as our running example: given a (planar) graph G and an integer k, we have to decide if G contains a simple path on k vertices. Let us first note that k-Path is FPT parameterized by the treewidth w of the input graph G. More precisely, standard dynamic programming techniques give $2^{O(w\log w)}n^{O(1)}$ running time, while more sophisticated arguments are needed to obtain $2^{O(w)}n^{O(1)}$ time [11, 25, 33, 34, 36, 37, 45] (note that some of these algorithms are randomized and some of these algorithms work only on planar graphs).

Theorem 2.1. *k-Path can be solved in time $2^{O(w)}n^{O(1)}$ if a tree decomposition of width w is given in the input.*

The second ingredient that we need is the Planar Excluded Grid Theorem [48, 68]. A *minor* of a graph G is a graph H that is obtained by a sequence of vertex deletions, edge deletions, and edge contractions. A $k \times k$ *grid* is a graph with vertex set $[k] \times [k]$, where vertices (x, y) and (x', y') are adjacent if and only if $|x - x'| + |y - y'| = 1$. The following theorem states that, in a very tight sense, the existence of a grid minor is the canonical reason why a planar graph has large treewidth:

Theorem 2.2 (Planar Excluded Grid Theorem)**.** *Every planar graph width treewidth at least $4.5k$ has a $k \times k$ grid minor.*

In particular, Theorem 2.2 implies that an n-vertex planar graph has treewidth $O(\sqrt{n})$: it certainly cannot contain a grid minor larger than $\sqrt{n} \times \sqrt{n}$.

Finally, we have to make two simple observations about the k-Path problem:

(1) The $k \times k$ grid contains a path on k^2 vertices: imagine a "snake" that visits the rows one after the other.

(2) If H is a minor of G, then the length of the longest path in H is not larger than in G. This can be proved by verifying that none of vertex deletion, edge deletion, or edge contraction can increase the length of the longest path.

Now the claimed algorithm can be obtained by putting together these ingredients using a win/win approach. For simplicity, we describe an algorithm for the decision version of the problem where only a YES/NO answer has to be returned.

Theorem 2.3. *k-Path on planar graphs can be solved in time $2^{O(\sqrt{k})}n^{O(1)}$.*

Proof. Let $w := 4.5\lceil\sqrt{k}\rceil$. If G is a graph with treewidth at least w, then Theorem 2.2 implies that G contains a $\lceil\sqrt{k}\rceil \times \lceil\sqrt{k}\rceil$ grid minor H. Then the first observation above shows that H contains a path on k vertices and the second observation shows that G also contains a path on k vertices. Therefore, we can conclude that if the input graph G has treewidth at least w, then it is a YES-instance: it surely contains a path on k vertices.

The algorithm proceeds as follows. First, we compute an (approximate) tree decomposition of G. For this purpose, it is convenient to use the algorithm of Bodlaender et al. [12], which, given an integer w and a graph G, in time $2^{O(w)} \cdot n = 2^{O(\sqrt{k})} \cdot n$ either correctly states that treewidth of G is larger than w, or gives a tree decomposition of width at most $5w + 4$. We can complete the computation in both cases:

- If the algorithm states that G has treewidth larger than w, then, as we have seen above, the answer is YES.
- If the algorithm returns a tree decomposition of width at most $5w + 4 = O(\sqrt{k})$, then we can invoke Theorem 2.1 to decide the existence of a path on k vertices and return YES or NO accordingly. The running time is $2^{O(w)}n^{O(1)} = 2^{O(\sqrt{k})}n^{O(1)}$, as required.

Thus we have an algorithm that returns a correct YES/NO-answer in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$. □

The same argument works for Feedback Vertex Set and Vertex Cover. Only the analogs of the two observations (1) and (2) need to be verified: the optimum value is $\Omega(k^2)$ on the $k \times k$ grid and that the minor operation cannot increase the optimum value. A variant of the argument, based on contractions instead of minors, can give algorithms for Independent Set and Dominating Set. There are also less straighforward uses of Theorem 2.2, where it is invoked not on the input graph itself, but on some auxilliary graph defined in a nonobvious way; see the article of Marcin Pilipczuk later in this volume for some examples [63].

3 Exponential-time algorithms for graphs of maximum degree 3

If the task is to find a subset of vertices satisfying certain properties, then we can typically solve the problem in time $2^n \cdot n^{O(1)}$ on graphs with n vertices by enumerating every subset. For many problems, it is easy to improve on this brute force algorithm. For example, in the case of the Maximum Independent Set problem (for graphs with arbitrarily large degree), there is a simple textbook example of an improved branching algorithm that beats the $2^n \cdot n^{O(1)}$ running time. As long as there is a vertex v of degree at least 3, branch into two directions: either the solution avoids v (in which case we can remove v, decreasing the size of the graph by 1) or it contains v (in which case we can remove v and its neighbors from the problem, decreasing the size of the graph by at least 4 vertices). The problem can be solved in polynomial time if every vertex has degree at most 2. Analyzing the algorithm shows that its running time is $1.3803^n \cdot n^{O(1)}$. Further improvements are possible with more and more involved techniques [18, 41, 53, 55, 70, 72] with the current best algorithm having running time $1.1996^n \cdot n^{O(1)}$ [77]. Similar "races" for the best exponential-time algorithm are known for many other problems [43]. Let us remark that for some problems just beating the trivial $2^n \cdot n^{O(1)}$ running time is already highly nontrivial [10, 26, 66].

For the Maximum Independent Set problem on graphs of maximum degree 3, the current best algorithm has running time $1.0836^n \cdot n^{O(1)}$ [76]. Here we would like to highlight an earlier, less efficient algorithm that can be explained using the notion of treewidth very easily. Fomin and

Høie [42] proved, using an earlier result of Monien and Preis [62], that the pathwidth (and hence the treewidth) of an n-vertex graph with maximum degree 3 is essentially at most $n/6$. More precisely:

Theorem 3.1 (Fomin and Høie [42]). *For any $\epsilon > 0$, there is an integer n_ϵ such that the pathwidth of any graph on $n > n_\epsilon$ vertices and maximum degree at most 3 is at most $(1/6 + \epsilon)n$.*

Together with the fact that a Maximum Independent Set on an n-vertex graph can be solved in time $2^w \cdot n^{O(1)}$ if a tree decomposition of width w is given, it follows that the problem can be solved in time $2^{n/6} \cdot n^{O(1)} = 1.1225^n \cdot n^{O(1)}$. The running time obtained as a simple consequence of this pathwidth bound was better than some earlier work at that time [5, 20], but since then improved algorithms with more complicated and problem-specific arguments were found for this problem [17, 18, 67, 76]. In a similar way, algorithms for Minimum Dominating Set and Max Cut follow immediately from Theorem 3.1, which were better than some of the algorithms found by earlier problem specific techniques [42].

4 Finding and counting permutation patterns

Interesting combinatorial and algorithmic problems can be defined on permutations and on the patterns they contain or avoid. A *permutation* of length n is a bijection $\pi : [n] \to [n]$; typically we describe permutations by the sequence $(\pi(1), \pi(2), \ldots, \pi(n))$. We say that a permutation σ of length n *contains* a permutation π of length k if there is a mapping $f : [k] \to [n]$ such that $f(1) < f(2) < \cdots < f(k)$ and $\pi(i) < \pi(j)$ if and only if $\sigma(f(i)) < \sigma(f(j))$. That is, σ contains π if the sequence $(\pi(1), \ldots, \pi(k))$ can be mapped to a subsequence of $(\sigma(1), \ldots, \sigma(n))$ in a way that preserves the relative order of the values. As an example, the permutation $(3, 4, 5, 2, 1, 7, 8, 6)$ contains the permutation $(2, 1, 3, 4)$ (e.g., by the mapping $(f(1), f(2), f(3), f(4)) = (1, 4, 6, 7)$), but it does not contain the permutation $(4, 3, 2, 1)$. Observe that the permutations *not* containing $(1, 2)$ are exactly the decreasing sequences, while the permutations *not* containing $(2, 1)$ are exactly the increasing sequences. As shown by Knuth [56, § 2.2.1], the permutations avoiding $(2, 3, 1)$ are exactly the permutations sortable by a single stack. From the extremal combinatorics point of view, a very natural question is to bound the number of permutations of length n avoiding a fixed permutation π. Marcus and Tardos [61] proved a long-standing conjecture of Stanley and Wilf[1] by showing that for every fixed permutation π, there is a constant $c(\pi)$ such that the number of permutations of length n avoiding π is at most $2^{c(\pi) \cdot n}$. This has to be contrasted with the fact that the total number of permutations of length n is $n! = 2^{O(n \log n)}$.

From the algorithmic point of view, perhaps the most fundamental question is testing for containment: given a permutation σ of length n and a permutation π of length k, does σ contain π? The problem is often called Permutation Pattern Matching and is known to be NP-hard [16], but of course can be solved in time $O(n^k)$ by brute force. Albert et al. [3] improved this to $O(n^{2/3k+1})$ time, Ahal and Rabinovich [2] further improved it to $n^{0.47k+o(k)}$ time, and Berendsohn et al. [6] gave an $n^{0.25k+o(k)}$ time algorithm. Guillemot and Marx [49] showed that Permutation Pattern Matching can be solved in time $2^{O(k^2 \log k)} \cdot n$, that is, it is fixed-parameter tractable (FPT) parameterized by the length of π.

Even though the problem is FPT, algorithms with running time n^{ck} can be still interesting for two reasons. First, if k is fairly large, say, $\Omega(\log n)$, then $2^{O(k^2 \log k)} \cdot n$ is actually worse than $n^{O(k)}$. Thus unless we have $2^{O(k)} \cdot n^{O(1)}$ FPT algorithms for the problem, we need different type of algorithms to understand the complexity of the problem in the regime where k is large. Second,

[1] Marcus and Tardos [61] mentions that the conjecture was formulated around 1992 (but it is hard to find a citable source) and the PhD thesis of Julian West is an even earlier source [75].

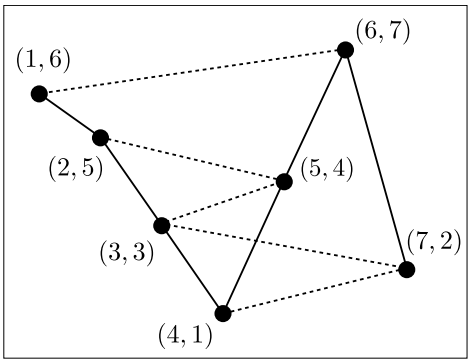

Figure 1: Permutation $\pi = (6, 5, 3, 1, 4, 7, 2)$ and its incidence graph G_π. Solid lines indicate neighbors by index (L-R), dashed lines indicate neighbors by value (U-D). Indices plotted on x-coordinate, values plotted on y-coordinate.

the n^{ck} time algorithms [2, 3, 6] can be easily modified to count the total number of solutions, while the FPT algorithm of Guillemot and Marx [49] returns only a single solution. This is not just a shortcoming of the presentation [49]: the FPT algorithm contains a step where a certain structure is discovered that guarantees that every permutation of length k appears in σ. Then the algorithm stops and does not look for any further occurences of π. Furthermore, it is unlikely that the algorithm can be extended to a counting version: Berendsohn et al. [6] proved that the counting problem is #W[1]-hard.

The n^{ck} algorithms for Permutation Pattern Matching [2, 3, 6] are implicitly or explicitly based on dynamic programming on a certain tree decomposition. Here we follow the presentation of Berendsohn et al. [6], where it is shown how high-level arguments and previous results on treewidth can be combined to obtain an $n^{k/3+o(k)}$ time in a very clean way (a further improvement, based on a technical idea of Cygan et al. [24], reduces the running time to $n^{0.25k+o(k)}$ [6]).

A permutation $\pi : [k] \to [k]$ can be seen as a k-element point set $S_\pi = \{(i, \pi(i)) \mid i \in [k]\}$ (see Figure 1). With this interpretation, σ contains π if S_π can be mapped to a subset of S_σ in a way that the mapping preserves the relative ordering of any two points along both the horizontal axis and the vertical axis. For a point $p \in S_\pi$, we will denote by $p.x$ and $p.y$ the first and second coordinates of p, respectively. For each point $(x, y) \in S_\pi$, we define the four neighbors of (x, y) as follows:

$$\begin{aligned} N^R((x,y)) &= (x+1,\ \pi(x+1)), \\ N^L((x,y)) &= (x-1,\ \pi(x-1)), \\ N^U((x,y)) &= (\pi^{-1}(y+1),\ y+1), \\ N^D((x,y)) &= (\pi^{-1}(y-1),\ y-1). \end{aligned}$$

The superscripts R, L, U, D are meant to evoke the directions *right*, *left*, *up*, *down*, when plotting S_σ in the plane. That is, if we start sweeping the vertical line going through (x, y) to the Right, then $N^R((x, y))$ is the next point that we meet, and similarly with the other directions. Note that some neighbors of a point may coincide.

The *incidence graph* G_π of π is a graph on S_π where each point is connected to its four neighbors (when defined). It is easy to see that G_π is the union of two Hamiltonian paths on the same set S_π of vertices, with one path going in the left-right direction in the plane, while the other path going in the top-bottom direction.

The key lemma that allows a clean abstraction of the problem is the following characterization of solutions.

Lemma 4.1. *Let $\sigma : [n] \to [n]$ and $\pi : [k] \to [k]$ be two permutations. Then σ contains π if and only if there is a function $f : S_\pi \to S_\sigma$ such that for every $p \in S_\pi$*

$$f(N^L(p)).x < f(p).x < f(N^R(p)).x, \quad \text{and} \tag{1}$$
$$f(N^D(p)).y < f(p).y < f(N^U(p)).y, \tag{2}$$

whenever the corresponding neighbor of p is defined.

It is not very difficult to prove Lemma 4.1 using the definitions and we can also see that the functions f satisfying the requirements of Lemma 4.1 are in one to one correspondence with the occurrences of π in σ. The inequalities in the first line ensure that the mapping of points represent the left-to-right ordering, while the inequalities in the second line handle the top-to-bottom ordering. The key observation is that even though we require these inequalities only between neighbors in G_π, it follows as consequence that every pairwise inequality in the definition of containment holds. For example, if $\pi(i) < \pi(j)$, then $(j, \pi(j))$ can be reached from $(i, \pi(i))$ by going through a sequence of U-neighbors, hence a sequence of inequalities ensure that the second coordinate of $f((i, \pi(i))$ is less than the second coordiante of $f((j, \pi(j)))$.

Readers familar with the notion of Constraint Satisfaction Problems (CSPs) may recognize that Lemma 4.1 cleanly transforms the problem into a binary constraint satisfaction problem. A *binary CSP* instance is a triplet (V, D, C), where V is a set of variables, D is a set of admissible values (the *domain*), and C is a set of constraints $C = \{c_1, \ldots, c_m\}$, where each constraint c_i is of the form $((x, y), R)$, where $x, y \in V$, and $R \subseteq D^2$ is a binary relation. A solution of the CSP instance is a function $f : V \to D$ (i.e., an assignment of admissible values to the variables), such that for each constraint $c_i = ((x_i, y_i), R_i)$, the pair of assigned values $(f(x_i), f(y_i))$ is contained in R_i.

The *constraint graph* of the binary CSP instance (also known as *primal graph* or *Gaifman graph*) is a graph whose vertices are the variables V and whose edges connect all pairs of variables that occur together in a constraint. Low treewidth of the constraint graph can be exploited for an efficient solution of the problem:

Theorem 4.2 ([27, 47])**.** *A binary CSP instance (V, D, C) can be solved in time $O(|D|^{t+1})$ where t is the treewidth of the constraint graph.*

To view the Permutation Pattern Matching problem as a binary CSP instance, let $V = S_\pi$ be the set of variables and let $D = S_\sigma$ be the domain. Then we want to find a function f that satisfies the inequalities in Lemma 4.1. Each inequality is a binary constraint between p and $N^\alpha(p)$ for some $\alpha \in \{L, R, D, U\}$, restricting the possible combination of values that $f(p)$ and $f(N^\alpha(p))$ can take. Thus we end up with a CSP instance on k variables, domain size n, and whose constraint graph is exactly G_π.

In order to invoke Theorem 4.2 on this instance, we need to bound the treewidth of G_π. Recall that G_π has k vertices and maximum degree 4. By splitting each degree-4 vertex into two degree-3 vertices connected by an edge, we can create a graph G'_π that has at most $2k$ vertices, maximum degree 3, and G_π is a minor of G'_π. Then Theorem 3.1 shows that G'_π has treewidth $2k/6 + o(k) = k/3 + o(k)$ and G_π being a minor of G'_π shows that the same bound holds for G_π as well. Therefore, we can conclude that Theorem 4.2 solves the instance in time $n^{k/3+o(k)}$. It is not difficult to modify the algorithm to count the number of solutions. Therefore, the combination of an easy observation (Lemma 4.1), a combinatorial treewidth bound (Theorem 3.1), and a known general algorithm (Theorem 4.2) solves the problem in a very clean way.

In Lemma 4.1, the functions f satisfying the requirements are in one to one correspondence with the occurences of π in σ and Theorem 4.2 can be extended to a counting version. Theorem 3.1 is purely combinatorial, thus it is of course irrelevant if we are using it for the decision or the counting problem. Thus the same algorithmic idea goes through.

Theorem 4.3 (Berendsohn et al. [6]). *Given a length-k permutation π and length-n permutation σ, the number of occurrences of π in σ can be counted in time $n^{k/3+o(k)}$.*

5 Counting subgraphs

It is a well-known phenomenon in theoretical computer science that in many cases finding a solution is easier than counting the number of all solutions. For example, it can be checked in polynomial time if a bipartite graph contains a perfect matching, but the seminal result of Valiant shows that counting the number of perfect matchings is #P-hard and hence unlikely to be polynomial-time solvable [74]. By now, many other examples of hard counting problems are known.

Flum and Grohe [39] started the investigation of the complexity of counting in the setting of parameterized complexity. They introduced the notion of #W[1]-hardness to give evidence that certain parameterized counting problems are unlikely to be FPT. As a highly nontrivial example, they considered the k-Path problem: the decision version is known to be FPT by various techniques [4,45], but they showed that the counting version of the problem is #W[1]-hard. In the same paper, they asked as an open question whether the counting version of the polynomial-time solvable k-Matching problem is FPT. This question was resolved in the negative by the #W[1]-hardness proof of Curticapean [21], which used heavy algebraic machinery, and by the later simpler proof given by Curticapean and Marx [23]. More recently, Dell et al. [22] described and exploited a connection beween subgraph counting and homomorphism counting problems. This connection can be useful in two different ways: it gives new subgraph-counting algorithms by reducing it to homomorphism-counting problems, and gives hardness results for subgraph counting (including new and clean #W[1]-hardness proofs of k-Matching and k-Path) based on our understanding of the complexity of counting homomorphisms. Below we give an example of the algorithmic use of this connection.

Given the #W[1]-hardness of k-Path, we cannot hope for an FPT algorithm solving the problem. But it is still an interesting question whether we can improve on the trivial $n^{k+O(1)}$ time brute force algorithm. The "meet in the middle" approach can be used to improve this to $n^{k/2+O(1)}$ time [8,57], which was further improved by Björklund et al. [9] to $n^{0.455k+O(1)}$. Here we describe an algorithm with running time $k^{O(k)} \cdot n^{0.174k+o(k)}$, which has a much smaller exponent for a fixed k and at the same time conceptually much simpler.

Let us first review some basic background on homomorphisms. A *homomorphism* from graph H to graph G is a mapping $f : V(H) \to V(G)$ such that for every edge $uv \in E(H)$, we have $f(u)f(v) \in E(G)$. We will denote by $\#\mathsf{Hom}(H \to G)$ the number of homomorphisms from H to G. Given a tree decomposition of H, standard dynamic programming techniques can be used to compute the number of homomorphisms from H to a given graph G.

Theorem 5.1 (Díaz et al. [32]). *Given graphs H and G, $\#\mathsf{Hom}(H \to G)$ can be computed in time $(|V(H)| + |V(G)|)^{w+O(1)}$, where w is the treewidth of H.*

Note that the algorithm of Theorem 5.1 does not need a decomposition of H, as it can be found in time $|V(H)|^{c+O(1)}$.

A homomorphism $f : V(H) \to V(G)$ is *injective* if $f(u) \neq f(v)$ for any two distinct $u, v \in V(H)$; let $\#\mathsf{Emb}(H \to G)$ denote the number of such homomorphisms. Let us denote by $\#\mathsf{Sub}(H \to G)$

the number of subgraphs of G that are isomorphic to H. It is well known and easy to see that $\#\mathsf{Emb}(H \to G) = \#\mathsf{Sub}(H \to G) \cdot \#\mathsf{Aut}(H)$, where $\#\mathsf{Aut}(H) = \#\mathsf{Emb}(H \to H)$ is the number of automorphisms of the graph H. Therefore, for a fixed H, computing $\#\mathsf{Sub}(H \to G)$ is essentially equivalent to computing $\#\mathsf{Emb}(H \to G)$, the number of injective homomorphisms. In order to explain the connection between counting homomorphisms and subgraphs, it will be more convenient to work with $\#\mathsf{Emb}(H \to G)$ than with $\#\mathsf{Sub}(H \to G)$, as the former is already defined in terms of homomorphisms.

Of course, not every homomorpism from H to G is injective, the images of some vertices may coincide. For example, if H is the 4-cycle on vertices $1, 2, 3, 4$, then a homomorphism from H to a loopless graph G either (1) is injective, (2) identifies 1 with 3, (3) identifies 2 with 4, (4) identifies 1 with 3, and 2 with 4. In case (1), the image of H is a 4-cycle; in cases (2) and (3), the image of H is the path P_3 on three vertices; and in case (4), the image of H is the path P_2 on two vertices. This shows that the following formula holds for the number of homomorphisms:

$$\#\mathsf{Hom}(C_4 \to G) = \#\mathsf{Emb}(C_4 \to G) + 2 \cdot \#\mathsf{Emb}(P_3 \to G) + \#\mathsf{Emb}(P_2 \to G).$$

More generally, we can classify the homomorphisms according to which sets of vertices they identify. To each homomorphism $h : V(G) \to V(H)$, we can associate a partition ρ_h of $V(H)$ with the meaning that, for every $u, v \in V(H)$, we have $h(u) = h(v)$ if and only u and v are in the same block of ρ. For a partition ρ of $V(H)$, let H/ρ be the *quotient graph* obtained by consolidating each block of ρ into a single vertex. The key observation is that the homomorphisms from H to G having type ρ are in one-to-one correspondence with the injective homomorphisms from H/ρ to G. Therefore, we can express the number of homomorphisms from H to G as

$$\#\mathsf{Hom}(H \to G) = \sum_{\rho} \#\mathsf{Emb}(H/\rho \to G), \tag{3}$$

where the sum ranges over every partition ρ of $V(H)$.

Why is this useful for us? Observe that $H = H/\rho$ holds only for the partition ρ_0 where every block has size exactly one and H/ρ has strictly fewer vertices for every other ρ. Therefore, Eq. (3) can be written as

$$\#\mathsf{Hom}(H \to G) = \#\mathsf{Emb}(H \to G) + \sum_{\rho \neq \rho_0} \#\mathsf{Emb}(H/\rho \to G),$$

and hence

$$\#\mathsf{Emb}(H \to G) = \#\mathsf{Hom}(H \to G) - \sum_{\rho \neq \rho_0} \#\mathsf{Emb}(H/\rho \to G). \tag{4}$$

That is, Eq. (4) reduces the problem of computing $\#\mathsf{Emb}(H \to G)$ to the problem of computing $\#\mathsf{Hom}(H \to G)$ and to computing some number of $\#\mathsf{Emb}(H/\rho \to G)$ values, where H/ρ has strictly fewer vertices than $|V(H)|$. Therefore, we can repeat the same argument and recursively replace each term $\#\mathsf{Emb}(H/\rho \to G)$ with a $\#\mathsf{Hom}$ term and some number of $\#\mathsf{Emb}$ terms. As the replacement strictly decreases the number of vertices in the $\#\mathsf{Emb}$ terms, eventually all these terms disappear, and we can express $\#\mathsf{Emb}(H \to G)$ as the linear combination of $\#\mathsf{Hom}(H' \to G)$ values for various graphs H'. This means that we can reduce the problem of computing $\#\mathsf{Emb}(H \to G)$ to computing certain homomorphism values.

Which graphs H' can appear in the $\#\mathsf{Hom}(H' \to G)$ terms when we express $\#\mathsf{Emb}(H \to G)$ this way? It is easy to see that the quotient graph of a quotient of H is also a quotient graph of

H. This means that every graph H' appearing in this linear combination is a quotient graph of H. Thus we can express $\#\mathsf{Emb}(H \to G)$ as

$$\#\mathsf{Emb}(H \to G) = \sum_{\rho} \beta_{\rho,H} \cdot \#\mathsf{Hom}(H/\rho \to G)\,, \tag{5}$$

where $\beta_{\rho,H}$ is a constant depending only on ρ and H. The argument described above gives an algorithm for writing $\#\mathsf{Emb}(H \to G)$ in this form and for computing the constants $\beta_{\rho,H}$ (and the work of Lovász et al. [15,59] gives more explicit formulas for these constants). Given this expression, we can reduce the problem of computing $\#\mathsf{Emb}(H \to G)$ to computing the values $\#\mathsf{Hom}(H/\rho \to G)$. If H has k vertices, then the sum ranges over $k^{O(k)}$ different partitions ρ. Therefore, if every H/ρ has treewidth bounded by c, then invoking Theorem 5.1 for the computation of each $\#\mathsf{Hom}(H/\rho \to G)$ results in an algorithm with running time $k^{O(k)} \cdot n^{c+O(1)}$ for the computation of $\#\mathsf{Emb}(H \to G)$ (and hence of $\#\mathsf{Sub}(H \to G)$).

These considerations show that bounding the running time of our algorithm essentially boils down to a bound on the maximum treewidth of H/ρ. The treewidth of H/ρ can be much larger than the treewidth of H. For example, it is not difficult to see that if H is a matching with k independent edges, then we can obtain any connected graph with k edges as H/ρ for an appropriate partition ρ. However, this operation cannot increase the number of edges: if H has k edges, then H/ρ has at most k edges. We can use the following bound on the treewidth of graphs with at most k edges:

Theorem 5.2 ([40, 71])**.** *Every graph with at most k edges has treewidth $0.174k + o(k)$.*

This immediately gives an upper bound on the running time needed if H has at most k edges.

Theorem 5.3 (Dell et al. [22])**.** *If H has at most k edges, then $\#\mathsf{Emb}(H \to G)$ and $\#\mathsf{Sub}(H \to G)$ can be computed in time $k^{O(k)} \cdot n^{0.174k+o(k)}$.*

In particular, we obtain algorithms with running time $k^{O(k)} \cdot n^{0.174k+o(k)}$ if H is a path with k edges (the k-Path problem) or a matching with k edges (the k-Matching problem). We want to emphasize that for a fixed H, the algorithm is very simple: it consists of invoking Theorem 5.1 for various graphs $H' = H/\rho$ and then taking a linear combination of these values. All the real work is done by the computation of the fixed constants $\beta_{\rho,H}$ and by the algorithm of Theorem 5.1 exploiting low treewidth and tree decompositions.

References

[1] P. Aboulker, N. Brettell, F. Havet, D. Marx, and N. Trotignon. Coloring graphs with constraints on connectivity. *Journal of Graph Theory*, 85(4):814–838, 2017.

[2] S. Ahal and Y. Rabinovich. On complexity of the subpattern problem. *SIAM J. Discrete Math.*, 22(2):629–649, 2008.

[3] M. H. Albert, R. E. L. Aldred, M. D. Atkinson, and D. A. Holton. Algorithms for pattern involvement in permutations. In *Proceedings of the 12th International Symposium on Algorithms and Computation*, ISAAC '01, pages 355–366, London, UK, UK, 2001. Springer-Verlag.

[4] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[5] R. Beigel. Finding maximum independent sets in sparse and general graphs. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA.*, pages 856–857, 1999.

[6] B. A. Berendsohn, L. Kozma, and D. Marx. Finding and counting permutations via CSPs. Accepted to IPEC 2019.

[7] U. Bertelè and F. Brioschi. On non-serial dynamic programming. *J. Comb. Theory, Ser. A*, 14(2):137–148, 1973.

[8] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Counting paths and packings in halves. In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA)*, pages 578–586, 2009.

[9] A. Björklund, P. Kaski, and L. Kowalik. Counting thin subgraphs via packings faster than meet-in-the-middle time. In *Proceedings of the 25th Annual Symposium on Discrete Algorithms (SODA)*, pages 594–603, 2014.

[10] I. Bliznets, F. V. Fomin, M. Pilipczuk, and Y. Villanger. Largest chordal and interval subgraphs faster than 2^n. *Algorithmica*, 76(2):569–594, 2016.

[11] H. L. Bodlaender, M. Cygan, S. Kratsch, and J. Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015.

[12] H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov, and M. Pilipczuk. A $c^k \cdot n$ 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016.

[13] H. L. Bodlaender, D. Lokshtanov, and E. Penninkx. Planar capacitated dominating set is $W[1]$-hard. In *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*, pages 50–60, 2009.

[14] A. Bonifati, W. Martens, and T. Timm. An analytical study of large SPARQL query logs. *PVLDB*, 11(2):149–161, 2017.

[15] C. Borgs, J. Chayes, L. Lovász, V. T. Sós, and K. Vesztergombi. Counting graph homomorphisms. In *Topics in discrete mathematics*, pages 315–371. Springer, 2006.

[16] P. Bose, J. F. Buss, and A. Lubiw. Pattern matching for permutations. *Inf. Process. Lett.*, 65(5):277–283, 1998.

[17] N. Bourgeois, B. Escoffier, and V. T. Paschos. An $O^*(1.0977^n)$ exact algorithm for Max Independent set in sparse graphs. In *Parameterized and Exact Computation, Third International Workshop, IWPEC 2008, Victoria, Canada, May 14-16, 2008. Proceedings*, pages 55–65, 2008.

[18] N. Bourgeois, B. Escoffier, V. T. Paschos, and J. M. M. van Rooij. Fast algorithms for Max Independent set. *Algorithmica*, 62(1-2):382–415, 2012.

[19] L. Cai, M. R. Fellows, D. W. Juedes, and F. A. Rosamond. The complexity of polynomial-time approximation. *Theory Comput. Syst.*, 41(3):459–477, 2007.

[20] J. Chen, I. A. Kanj, and G. Xia. Labeled search trees and amortized analysis: Improved upper bounds for np-hard problems. *Algorithmica*, 43(4):245–273, 2005.

[21] R. Curticapean. Counting matchings of size k is w[1]-hard. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 352–363, 2013.

[22] R. Curticapean, H. Dell, and D. Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 210–223, 2017.

[23] R. Curticapean and D. Marx. Complexity of counting subgraphs: Only the boundedness of the vertex-cover number counts. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 130–139, 2014.

[24] M. Cygan, L. Kowalik, and A. Socala. Improving TSP tours using dynamic programming over tree decompositions. In *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, pages 30:1–30:14, 2017.

[25] M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In R. Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159. IEEE Computer Society, 2011.

[26] M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. O. Wojtaszczyk. Solving the 2-disjoint connected subgraphs problem faster than 2 n. *Algorithmica*, 70(2):195–207, 2014.

[27] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):353 – 366, 1989.

[28] E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Bidimensional parameters and local treewidth. *SIAM J. Discrete Math.*, 18(3):501–511, 2004.

[29] E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Trans. Algorithms*, 1(1):33–47, 2005.

[30] E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

[31] E. D. Demaine, M. T. Hajiaghayi, and D. M. Thilikos. Exponential speedup of fixed-parameter algorithms for classes of graphs excluding single-crossing graphs as minors. *Algorithmica*, 41(4):245–267, 2005.

[32] J. Díaz, M. J. Serna, and D. M. Thilikos. Counting H-colorings of partial k-trees. *Theoretical Computer Science*, 281(1-2):291–309, 2002.

[33] F. Dorn. Dynamic programming and planarity: Improved tree-decomposition based algorithms. *Discrete Applied Mathematics*, 158(7):800–808, 2010.

[34] F. Dorn. Planar subgraph isomorphism revisited. In *27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, March 4-6, 2010, Nancy, France*, pages 263–274, 2010.

[35] F. Dorn, F. V. Fomin, D. Lokshtanov, V. Raman, and S. Saurabh. Beyond bidimensionality: Parameterized subexponential algorithms on directed graphs. *Inf. Comput.*, 233:60–70, 2013.

[36] F. Dorn, F. V. Fomin, and D. M. Thilikos. Catalan structures and dynamic programming in h-minor-free graphs. *J. Comput. Syst. Sci.*, 78(5):1606–1622, 2012.

[37] F. Dorn, E. Penninkx, H. L. Bodlaender, and F. V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. *Algorithmica*, 58(3):790–810, 2010.

[38] W. Fischl, G. Gottlob, D. M. Longo, and R. Pichler. Hyperbench: A benchmark and tool for hypergraphs and empirical findings. In *Proceedings of the 13th Alberto Mendelzon International Workshop on Foundations of Data Management, Asunción, Paraguay, June 3-7, 2019.*, 2019.

[39] J. Flum and M. Grohe. The parameterized complexity of counting problems. *SIAM J. Comput.*, 33(4):892–922, 2004.

[40] F. V. Fomin, S. Gaspers, S. Saurabh, and A. A. Stepanov. On two techniques of combining branching and treewidth. *Algorithmica*, 54(2):181–207, 2009.

[41] F. V. Fomin, F. Grandoni, and D. Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5):25:1–25:32, 2009.

[42] F. V. Fomin and K. Høie. Pathwidth of cubic graphs and exact algorithms. *Inf. Process. Lett.*, 97(5):191–196, 2006.

[43] F. V. Fomin and D. Kratsch. *Exact Exponential Algorithms.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010.

[44] F. V. Fomin, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. Subexponential parameterized algorithms for planar and apex-minor-free graphs via low treewidth pattern covering. In *FOCS 2016*, pages 515–524. IEEE Computer Society, 2016.

[45] F. V. Fomin, D. Lokshtanov, F. Panolan, and S. Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016.

[46] F. V. Fomin and D. M. Thilikos. Dominating sets in planar graphs: Branch-width and exponential speed-up. *SIAM J. Comput.*, 36(2):281–309, 2006.

[47] E. C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proceedings of the Eighth National Conference on Artificial Intelligence - Volume 1*, AAAI'90, pages 4–9. AAAI Press, 1990.

[48] Q. Gu and H. Tamaki. Improved bounds on the planar branchwidth with respect to the largest grid minor size. *Algorithmica*, 64(3):416–453, 2012.

[49] S. Guillemot and D. Marx. Finding small patterns in permutations in linear time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 82–101, 2014.

[50] R. Halin. S-functions for graphs. *Journal of Geometry*, 8(1-2):171–186, 1976.

[51] R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[52] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. System Sci.*, 63(4):512–530, 2001.

[53] T. Jian. $O(2^{0.304n})$ algorithm for solving maximum independent set problem. *IEEE Transactions on Computers*, C-35(9):847–851, 1986. cited By 49.

[54] P. N. Klein and D. Marx. A subexponential parameterized algorithm for Subset TSP on planar graphs. In *SODA 2014*, pages 1812–1830. SIAM, 2014.

[55] J. Kneis, A. Langer, and P. Rossmanith. A fine-grained analysis of a simple independent set algorithm. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, pages 287–298, 2009.

[56] D. E. Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms*. Addison-Wesley, 1968.

[57] I. Koutis and R. Williams. LIMITS and applications of group algebras for parameterized problems. *ACM Transactions on Algorithms*, 12(3):31:1–31:18, 2016.

[58] D. Lokshtanov, S. Saurabh, and M. Wahlström. Subexponential parameterized odd cycle transversal on planar graphs. In *FSTTCS 2012*, volume 18 of *LIPIcs*, pages 424–434. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2012.

[59] L. Lovász. Operations with structures. *Acta Mathematica Hungarica*, 18(3-4):321–328, 1967.

[60] S. Maniu, P. Senellart, and S. Jog. An experimental study of the treewidth of real-world graph data. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*, pages 12:1–12:18, 2019.

[61] A. Marcus and G. Tardos. Excluded permutation matrices and the Stanley-Wilf conjecture. *J. Comb. Theory, Ser. A*, 107(1):153–160, 2004.

[62] B. Monien and R. Preis. Upper bounds on the bisection width of 3- and 4-regular graphs. In *Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS 2001 Marianske Lazne, Czech Republic, August 27-31, 2001, Proceedings*, pages 524–536, 2001.

[63] M. Pilipczuk. Surprising applications of treewidth bounds for planar graphs.

[64] M. Pilipczuk, M. Pilipczuk, P. Sankowski, and E. J. van Leeuwen. Subexponential-time parameterized algorithm for Steiner tree on planar graphs. In *STACS 2013*, volume 20 of *LIPIcs*, pages 353–364. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2013.

[65] M. Pilipczuk, M. Pilipczuk, P. Sankowski, and E. J. van Leeuwen. Network sparsification for Steiner problems on planar and bounded-genus graphs. In *FOCS 2014*, pages 276–285. IEEE Computer Society, 2014.

[66] I. Razgon. Computing minimum directed feedback vertex set in $O(1.9977^n)$. In *Theoretical Computer Science, 10th Italian Conference, ICTCS 2007, Rome, Italy, October 3-5, 2007, Proceedings*, pages 70–81, 2007.

[67] I. Razgon. Faster computation of maximum independent set and parameterized vertex cover for graphs with maximum degree 3. *J. Discrete Algorithms*, 7(2):191–212, 2009.

[68] N. Robertson, P. Seymour, and R. Thomas. Quickly excluding a planar graph. *J. Combin. Theory Ser. B*, 62(2):323–348, 1994.

[69] N. Robertson and P. D. Seymour. Graph minors. III. Planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984.

[70] J. M. Robson. Algorithms for maximum independent sets. *J. Algorithms*, 7(3):425–440, 1986.

[71] A. D. Scott and G. B. Sorkin. Linear-programming design and analysis of fast algorithms for Max 2-CSP. *Discrete Optimization*, 4(3-4):260–287, 2007.

[72] R. E. Tarjan and A. E. Trojanowski. Finding a maximum independent set. *SIAM J. Comput.*, 6(3):537–546, 1977.

[73] M. Thorup. All structured programs have small tree-width and good register allocation. *Inf. Comput.*, 142(2):159–181, 1998.

[74] L. G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[75] J. West. *Permutations with restricted subsequences and stack-sortable permutations.* PhD thesis, MIT, Cambridge, MA, 1990.

[76] M. Xiao and H. Nagamochi. Confining sets and avoiding bottleneck cases: A simple maximum independent set algorithm in degree-3 graphs. *Theor. Comput. Sci.*, 469:92–104, 2013.

[77] M. Xiao and H. Nagamochi. Exact algorithms for maximum independent set. *Inf. Comput.*, 255:126–146, 2017.