Computing Tree Width: From Theory to Practice and Back

Sebastian Berndt

Department of Computer Science, Kiel University
seb@informatik.uni-kiel.de

Abstract. While the theoretical aspects concerning the computation of tree width – one of the most important graph parameters – are well understood, it is not clear how it can be computed practically. As tree width has a wide range of applications, e. g. in bioinformatics or artificial intelligence, this lack of understanding hinders the applicability of many important algorithms in the real world. The Parameterized Algorithms and Computational Experiments (PACE) challenge therefore chose the computation of tree width as one of its challenge problems in 2016 and again in 2017. In 2016, Hisao Tamaki (Meiji University) presented a new algorithm that outperformed the other approaches (including SAT solvers and branch-and-bound) by far. An implementation of Tamaki's algorithm allowed Larisch (King-Abdullah University of Science and Engineering) and Salfelder (University of Leeds) to solve over 50% of the test suite of PACE 2017 (containing graphs with over 3500 nodes) in under six seconds. Before PACE 2016, no algorithm was known to reliably compute tree width on graphs with about 100 nodes. As a wide range of parameterized algorithm require the computation of a tree decomposition as a first step, this breakthrough result allows practical implementations of these algorithms for the first time.
This talks starts with a gentle introduction to tree width and its use in parameterized complexity, followed by an algorithmic approach for the exact computation of the tree width of a graph, based on a variant of the well-studied cops-and-robber game. Finally, we present a streamlined version of Tamaki's algorithm due to Bannach and Berndt based on this game.

Keywords: tree width, algorithms, experimental evaluation, graph searching, parameterized complexity

Talk Summary

Introducing Tree Width

Consider your favorite optimization problem on graphs. With high probability, it is easy, i. e. solvable in polynomial time, if restricted to trees. Let us consider the Maximum Independent Set problem, where we want to find a set of vertices $V' \subseteq V(G)$ of maximum cardinality such that $\{u, v\} \notin E(G)$ for all $u, v \in V'$.

This is one of the classical NP-hard problems, but solving it on trees is fairly simple. Root the tree T at some arbitrary vertex $r \in V(T)$ to obtain the rooted tree T^r. For a vertex $v \in V(T)$, let $\mathrm{ch}(v)$ be the children of v of in T^r and let $T^r[v]$ be the set of descendants of v in T^r. Furthermore, let $I^+[v]$ be the size of a maximal independent set of $T^r[v]$ that contains v and $I^-[v]$ be the size of a maximal independent set of $T^r[v]$ that does not contain v. For all leafs v of T^r, we have $I^+[v] = 1$ and $I^-[v] = 0$. If v is an inner node, we clearly have

$$I^+[v] = 1 + \sum_{w \in \mathrm{ch}(u)} I^-[w],$$
$$I^-[v] = \sum_{w \in \mathrm{ch}(u)} \max\{I^+[w], I^-[w]\}.$$

As $\max\{I^+[r], I^-[r]\}$ is the size of a maximal independent set in $T^r[r] = V(T)$, we can conclude that this problem is indeed solvable in linear time.

Now suppose that we add an edge $\{u, w\}$ to the tree T and obtain a graph G. Intuitively, the problem should not become much harder on G: After all, the graph G is just a tree and a single edge. The simple observation that a maximal independent set can contain u, or w, or none of them allows us to reuse our previous dynamic program with simple adaptions. Generalizing this principle, if G is a graph consisting of a tree and an additional set of edges E', we can compute the size of a maximal independent set in time $\mathcal{O}(2^{|2E'|} \cdot |V(G)|)$ by trying out all $2^{|2E'|}$ independent sets in E'. Note that some of the edges $\{u, v\}, \{u', v'\}$ in E' may be independent of each other in the sense that the inclusion of u into the maximal independent set does not tell us whether u', v' or none of them need to be part of the maximal independent set. See e. g. the edges $\{d, e\}$ and $\{f, h\}$ in Figure 1(a).

The concept of the *tree width* of a graph G captures the idea that a graph is tree-like and also integrates our previous discussion on independent edges. A *tree decomposition* of a graph[1] $G = (V, E)$ is a pair $(\mathcal{T}, \iota)$ such that $\mathcal{T}$ is a tree and ι maps nodes of $\mathcal{T}$ to subsets of V. These subsets are called *bags*. Furthermore, such a tree decomposition must have the following properties: (i) for every vertex $v \in V$, there is a node X in $\mathcal{T}$ such that $v \in \iota(X)$, (ii) for every edge $\{u, v\} \in E$, there is a node X in $\mathcal{T}$ such that $\{u, v\} \subseteq \iota(X)$, and (iii) for every vertex $v \in V$, the set $\{X \mid v \in \iota(X)\}$ is connected in $\mathcal{T}$. The *width* of a tree decomposition $(\mathcal{T}, \iota)$ is the maximum size of one of its bags minus one. As the width corresponds to the complexity of such a decomposition, we want to minimize this quantity. Hence, the *tree width* $\mathrm{tw}(G)$ of a graph G is the minimum width over all tree decompositions of G. See Figure 1 for an example.

The concept of tree decompositions has been used for a wide number of different applications. It is one of the fundamental tools in parameterized complexity (see e. g. [6, 9–11]), but also found applications in bioinformatics (see e. g. [15]) or artificial intelligence (see e. g. [12]).

[1] All graphs in this work are undirected, unless stated otherwise.

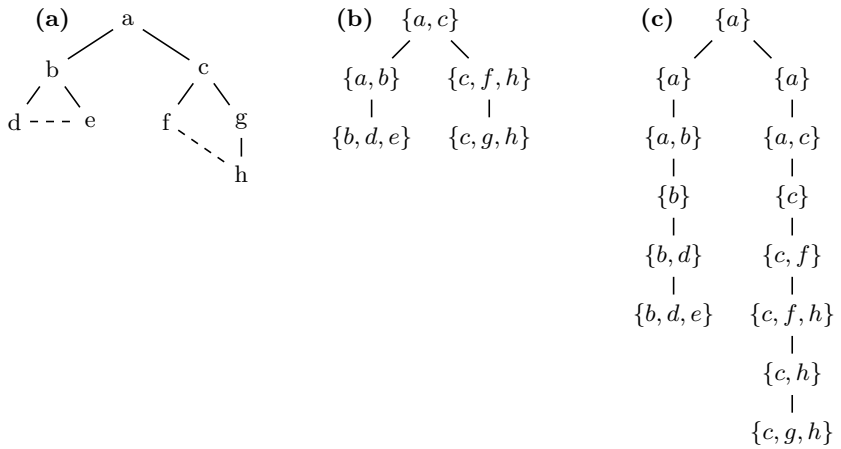

Fig. 1. Two tree decompositions of an undirected graph $G = (V, E)$ shown at **(a)**. The decomposition in **(b)** shows $\mathrm{tw}(G) \leq 2$ and the expanded one in **(c)** gives more intuition on the construction.

Given a graph G on n nodes, one can test in time $f(k) \cdot n$ whether $\mathrm{tw}(G) \leq k$ by the celebrated algorithm of Bodlaender [3]. Hence, this problem is fixed-parameter-tractable (and belongs to the class FPT), but the algorithm is not useful in practice, as the function $f(k)$ is quite large [4]. A wide number of FPT-algorithms compute a tree decomposition as a first step. Hence, any practical implementation of these algorithms relies on a practical algorithm for tree width. See also [2] for a more detailed discussion on this. Finding a practically fast algorithm for tree width – like modern algorithms for the SAT problem – is thus still an open and intriguing problem. Fortunately, due to the Parameterized Algorithms and Computational Experiments (PACE) challenge, huge steps toward such an algorithm were made. The aim of the PACE challenge, initiated in 2016, is the development of practically FPT algorithms [7, 8]. The winning algorithm of PACE 2016, due to Hisao Tamaki, was in fact so successful, that *all* participants in PACE 2017 used a variant of this algorithm. The winning implementation of 2017 by Larisch and Salfelder "is basically Hisao Tamakis implementation" [13]. A variant of this algorithm is described in [17]. In the remainder of this paper, we will give an simplified presentation of the core of Tamaki's algorithm. This presentation is due to Bannach and Berndt [1].

An Algorithmic Approach: Cops-And-Robber

In contrast to other graph parameters, the definition of tree width does not give immediate rise to an exponential-time algorithm: If one tries to find a tree decomposition with width k, the number of possible bags is n^{k+1}, and the number of possible trees on these bags is thus $(n^{k+1})^{n^{k+1}/2}$, giving a super-exponential running time. Nevertheless, there are alternative characterizations of tree width

that allow the design of practical exponential-time algorithms. We will take a closer look at one of them, namely the *cops-and-robber game*.

In the cops-and-robber game (sometimes also called searchers-and-fugitive game or graph-searching game), two players – the *cops* and the *robber* – play a game on the vertices of a graph $G = (V, E)$. The goal of the cops is to catch the robber, while the robber tries to avoid this. In the first turn, the cops player choose k vertices to put the cops on. Then the robber player chooses some vertex to position the robber. In each subsequent turn, the cops player announces that he moves $\ell \leq k$ cops from their current position $v_1, \ldots, v_\ell$ to new vertices $v'_1, \ldots, v'_\ell$. The cops are then removed from $v_1, \ldots, v_\ell$. Before the cops are put on $v'_1, \ldots, v'_\ell$, the robber player may move the robber along the edges of the graph, but is not allowed to visit vertices currently occupied by a cop (hence, he may now visit $v_1, \ldots, v_\ell$, as those are currently not occupied). After the movement of the robber, the cops are put on $v'_1, \ldots, v'_\ell$. The cops player wins, if he places a cop on the vertex occupied by the robber player. The minimum number of cops to win this game on the graph G is denoted as $\mathrm{vs}(G)$. Figure 2 depicts the first turns of such a game.

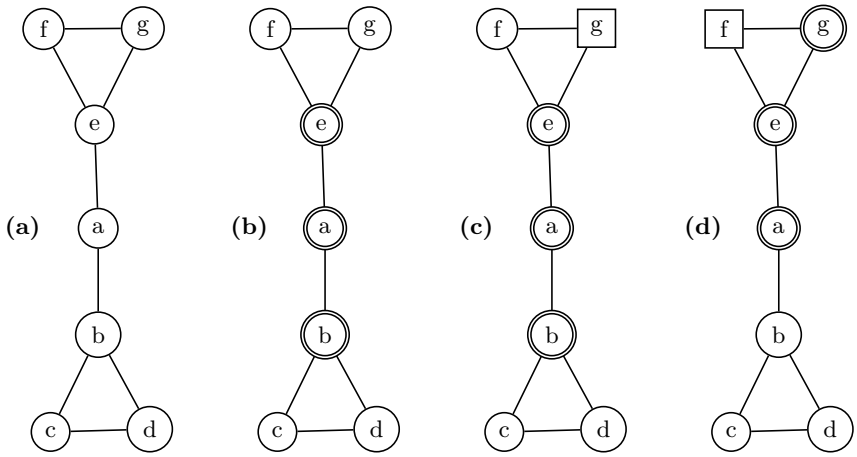

Fig. 2. A possible game on the graph depicted in **(a)**. First, the cops player (depicted by double circles) choose vertices a, b and e in **(b)**. Then, the robber player (depicted as rectangle) chooses vertex g in **(c)**. The cop player announces that he moves a cop from b to g and the robber player moves the robber to f in **(d)**. In order to win, the cops player will now move the cop from a to f.

The close connection between this game and the tree width of G was already observed by Seymour and Thomas who proved that $\mathrm{vs}(G) = \mathrm{tw}(G) + 1$ [14]. We will give some intuition behind this relation. An important aspect of this game is that if $\mathrm{vs}(G) = k$, there is a *monotone* winning strategy for the k cops, i. e. the

area where the robber may move is monotonically decreasing. This non-trivial fact was also shown in [14].

- If we know that $\mathrm{vs}(G) = k + 1$, the cops player can catch the robber with $k + 1$ cops in a monotone way. Intuitively, the positions occupied by the cops correspond to the bags of the tree decomposition. This is a valid tree decomposition of width k, as
  - (i) each vertex must at some point be occupied by some cop (otherwise, the robber would simply stay on this vertex);
  - (ii) for each edge, there must be a situation, where both endpoints of the edge are occupied by cops (otherwise, the robber could alternate between these endpoints);
  - (iii) the connectedness property of the tree decomposition is guaranteed due to the monotonicity of the strategy (otherwise, if the bags containing some vertex v were not connected, the robber may "escape" through v, which would violate the monotonicity).
- On the other hand, if we are given a tree decomposition $(\mathcal{T}, \iota)$ of width k, this corresponds to a valid monotone winning strategy for $k + 1$ cops. Consider some bag $X \in \mathcal{T}$ in the tree decomposition, which we will treat as root. The cops player puts its k cops on $\iota(X)$. The robber player now decides the position v of the robber. The cops player will now determine a child Y of X, such that v occurs in some bag in the tree rooted at Y. Such a child must always exists, as each vertex must occur in some bag. As $v \notin \iota(X)$ (otherwise, the game would be over now), the connectedness property of the tree decomposition guarantees that such a child Y is unique. The cops player then announces that the cops in $\iota(X) \cap \iota(Y)$ stay on their positions and that the cops in $\iota(X) \setminus \iota(Y)$ move to $\iota(Y) \setminus \iota(X)$. The game then goes on inductively until a leaf is reached, where the robber is captured.

For a formal proof of this relation, see e. g. [14].

This connection gives us a simple $\mathcal{O}(n^{k+2})$ algorithm to test whether $\mathrm{tw}(G) \leq k$. Construct the so called *arena graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, which is directed. Each vertex (C, H, r) in $\mathcal{V}$ has three components: (i) a set C of current positions of the cops, (ii) a set H of future positions of the cops (i. e. these positions are not currently occupied by cops, but will be at the end of the turn), and (iii) a vertex $r \in V(G)$ determining the position of the robber or the value $\perp$, if the robber is not yet positioned.

Consider the game in Figure 2. The situation in Figure 2(c) would be described as $(C = \{a, b, e\}, H = \emptyset, r = g)$. As the cops player announces that he moves a cop from b to g, this situation is described as $(C = \{a, e\}, H = \{g\}, r = g)$, as the cop is removed from b. The robber player now moves the robber to f and the resulting situation is $(C = \{a, e\}, H = \{g\}, r = f)$. Finally, the cop is put on g and the resulting situation of Figure 2(d) would be $(C = \{a, e, g\}, H = \emptyset, r = f)$.

As $|C \cup H| \leq k+1$, there are at most n^{k+2} such nodes in $\mathcal{V}$. The *start vertices* are the vertices having the form $(C, \emptyset, \perp)$ for all $C \subseteq V(G)$ with $|C| \leq k + 1$. These vertices correspond to situations, where the cops are positioned in the

first turn, but the robber is not yet placed. The *final vertices* are of the form $(C, \emptyset, r)$ for all $C \subseteq V(G)$ with $|C| \leq k+1$ and $r \in C$. These vertices correspond to situations, where the robber was captured.

For each vertex $(C, \emptyset, r)$, we add a *cops edge* to each vertex (C', H, r), where $H \subseteq V(G)$ with $|H| \leq |C|$ and $C' \subseteq C$ with $|C'| \leq |C| - |H|$. Such an edge corresponds to the announced move of the cops player. It also allows us to forget some cops that will never appear again. This will be useful later on to reduce the final vertices. For each vertex (C, H, r) with $H \neq \emptyset$, we add a *robber edge* to each vertex (C, H, r'), where r' is reachable from r in the graph $G \setminus C$. We also add an robber edge from each start vertex $(C, \emptyset, \bot)$ to each vertex $(C, \emptyset, r)$. Finally, if there is a robber edge between (C, H, r) and (C, H, r'), we place a *placement edge* between (C, H, r') and $(C \cup H, \emptyset, r')$.

We can now determine whether there is a winning strategy by using the following *backward labeling*: First, mark all final vertices. Every vertex (C, H, r) that reaches a marked vertex $(C \cup H, \emptyset, r)$ via a placement edge is also marked. If a vertex $(C, \emptyset, r)$ reaches some marked vertex (C', H, r) via a cops edge, it is also marked. Finally, we mark a vertex (C, H, r) if *all* its children reachable by robber edges are marked. One can easily see that a start vertex $(C, \emptyset, \bot)$ is marked by this process iff there is a winning strategy for $k+1$ cops that start at C. One can thus simply construct the arena graph $\mathcal{G}$ in time $\mathcal{O}(n^{k+2})$ and perform this backward labeling.

Tamaki's Algorithm

The following description of Tamaki's algorithm is due to Bannach and Berndt [1]. A closer look at the algorithm on the arena graph $\mathcal{G}$ in the previous section reveals some places that might be optimized. First of all, there are n^{k+1} start vertices, but most of these start positions will be clearly unsuited for a winning strategy. But listing all of these start vertices takes nearly as much time as the complete algorithm. Furthermore, there might be a lot of *dead ends* in the graph, i. e. non-final vertices that can not lead to final vertices. Finally, we actually do not need every final vertex $(C, \emptyset, r)$, but only those of the form $(N[r], \emptyset, r)$, where $N[r]$ denotes the neighbourhood of r in G (including r itself). Note that we modeled our arena graph in such a way that we might forget some cops along the way in order to make this work. The naive approach described in the previous section lists all of these superfluous vertices.

We still want to perform the backward labeling, but we do not want to build the complete arena graph in order to do so. We will rather build the *active* vertices of the arena graph iteratively by using a queue-like structure. First, we add all n final vertices $(N[r], \emptyset, r)$ to the queue. While this queue is non-empty, we dequeue one vertex (C, H, r) and add all vertices with outgoing edges to (C, H, r) to the queue. While doing this, we also maintain the same backward labeling as before. If we ever mark a start vertex, we know that a winning strategy with $k+1$ cops exists and have thus proved that $\mathrm{tw}(G) \leq k$. Intuitively, this approach allows us to only look at situations that really might occur within in the game and get rid of the superfluous ones. Furthermore, by ordering the vertices within the queue

in such a way that vertices with short distance to the start vertices are preferred, we might also be able to leave non-necessary parts of the arena graph unexplored.

This approach of getting rid of superfluous situations was coined *positive instance driven* (PID) dynamic programming [17]. It was used in the PACE challenge in 2016 by Hisao Tamaki [16] on the normal arena graph. In 2017, Tamaki [18] made use of the fact that one can indeed reduce the vertices in the arena graph to those corresponding to *maximal potential clliques*, as described by Bouchitté and Todinca in [5].

Acknowledgments The author likes to thank Max Bannach for many fruitful discussions around theoretical and practical approaches to tree width.

References

1. Bannach, M., Berndt, S.: Positive-instance driven dynamic programming for graph searching, unpublished
2. Bannach, M., Berndt, S., Ehlers, T.: Jdrasil: A modular library for computing tree decompositions. In: Proc. SEA. LIPIcs, vol. 75, pp. 28:1–28:21. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017)
3. Bodlaender, H.L.: A Linear Time Algorithm for Finding Tree-Decompositions of Small Treewidth. In: Proc. STOC. pp. 226–234. ACM (1993)
4. Bodlaender, H.L., Koster, A.M.C.A.: Treewidth Computations I. Upper bounds. Information and Computation 208(3), 259–275 (2010)
5. Bouchitté, V., Todinca, I.: Listing all potential maximal cliques of a graph. Theor. Comput. Sci. 276(1-2), 17–32 (2002)
6. Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer Berlin Heidelberg (2015)
7. Dell, H., Husfeldt, T., Jansen, B.M.P., Kaski, P., Komusiewicz, C., Rosamond, F.A.: The First Parameterized Algorithms and Computational Experiments Challenge. In: Proc. IPEC. LIPIcs, vol. 63, pp. 30:1–30:9. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016)
8. Dell, H., Komusiewicz, C., Talmon, N., Weller, M.: The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The Second Iteration. In: Proc. IPEC. pp. 30:1–30:12. LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2018)
9. Downey, R.G., Fellows, M.R.: Fundamentals of parameterized complexity. Springer (2016)
10. Downey, R.G., Fellows, M.R.: Parameterized complexity. Springer (2012)
11. Flum, J., Grohe, M.: Parameterized complexity theory. Springer (2006)
12. Karger, D.R., Srebro, N.: Learning Markov Networks: Maximum Bounded Tree-Width Graphs. In: Proc. SODA. pp. 392–401. ACM/SIAM (2001)
13. Larisch, L., Salfelder, F.: p17. `https://github.com/freetdi/p17` (2017), [Online; accessed 02-01-2018]
14. Seymour, P.D., Thomas, R.: Graph searching and a min-max theorem for tree-width. J. Comb. Theory, Ser. B 58, 22–33 (1993)
15. Song, Y., Liu, C., Malmberg, R.L., Pan, F., Cai, L.: Tree Decomposition Based Fast Search of RNA Structures Including Pseudoknots in Genomes. In: Proc. CSB. pp. 223–234. IEEE Computer Society (2005)

16. Tamaki, H.: treewidth-exact. https://github.com/TCS-Meiji/treewidth-exact (2016), [Online; accessed 02-01-2018]
17. Tamaki, H.: Positive-instance driven dynamic programming for treewidth. In: Proc. ESA. LIPIcs, vol. 87, pp. 68:1–68:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017)
18. Tamaki, H., Ohtsuka, H.: tw-exact. https://github.com/TCS-Meiji/PACE2017-TrackA (2017), [Online; accessed 02-01-2018]