Positive-Instance Driven Dynamic Programming for Graph Searching

Max Bannach[1] Sebastian Berndt[2]

[1] Institute for Theoretical Computer Science, Universität zu Lübeck, Lübeck, Germany
bannach@tcs.uni-luebeck.de

[2] Department of Computer Science, Kiel University, Kiel, Germany
seb@informatik.uni-kiel.de

Abstract

Research on the similarity of a graph to being a tree – called the *treewidth* of the graph – has seen an enormous rise within the last decade, but a practically fast algorithm for this task has been discovered only recently by Tamaki (ESA 2017). It is based on dynamic programming and makes use of the fact that the number of positive subinstances is typically substantially smaller than the number of all subinstances. Algorithms producing only such subinstances are called *positive-instance driven* (PID). We give an alternative and intuitive view on this algorithm from the perspective of the corresponding configuration graphs in certain two-player games. This allows us to develop PID-algorithms for a wide range of important graph parameters such as treewidth, pathwidth, and treedepth. We analyse the worst case behaviour of the approach on some well-known graph classes and perform an experimental evaluation on real world and random graphs.

1 Introduction

Treewidth, a concept to measure the similarity of a graph to being a tree, is arguably one of the most used tools in modern combinatorial optimization. It is a cornerstone of parameterized algorithms [14] and its success has led to its integration into many different fields: For instance, treewidth and its close relatives treedepth and pathwidth have been theoretically studied in the context of machine learning [5, 15, 20], model-checking [3, 32], SAT-solving [7, 21, 27], QBF-solving [12, 18], CSP-solving [31, 33], or ILPs [19, 25, 24, 34, 41]. Some of these results (e. g. [3, 7, 12, 21, 27, 31, 32, 33]) show quite promising experimental results giving hope that the theoretical results lead to actual practical improvements.

To utilize the treewidth for this task, we have to be able to compute it quickly. More crucially, most algorithms also need a witness for this fact in form of a tree-decomposition. In theory we have a beautiful algorithm for this task [8], which is unfortunately known to *not* work in practice due to huge constants [39]. We may argue that, instead, a heuristic is sufficient, as the attached solver will work correctly independently of the actual treewidth – and the heuristic may produce a decomposition of "small enough" width. However, even a small error, something as "off by 5," may put the parameter to a computationally intractable range, as the dependency on the treewidth is usually at least exponential. It is therefore a very natural and important task to build practical fast algorithms to determine parameters as the treewidth or treedepth exactly.

To tackle this problem, the fpt-community came up with an implementation challenge: the PACE [16, 17]. Besides many, one very important result of the challenge was a new combinatorial algorithm due to Hisao Tamaki, which computes the treewidth of an input graph exactly and astonishingly fast on a wide range of instances. An implementation of this algorithm by Tamaki himself [42] won the corresponding track in the PACE challenge in 2016 [16] and an alternative implementation due to Larisch and Salfelder [36] won in 2017 [17]. The algorithm is based on a dynamic program by Arnborg et al. [1] for computing tree decompositions. This algorithm has a game theoretic characterisation that we will utilities in order to apply Tamaki's approach to a broader range of problems. It should be noted, however, that Tamaki has improved his algorithm for the second iteration of the PACE by applying his framework to the algorithm by Bouchitté and Todinca [11, 43]. This algorithm has a game theoretic characterisation as well [23], but it is unclear how this algorithm can be generalized to other parameters. Therefore, we focus on Tamaki's first algorithm and analyze it both, from a theoretical and a practical perspective. Furthermore, we will extend the algorithm to further graph parameters, which is surprisingly easy due to the new game-theoretic representation. In detail, our contributions are the following:

Contribution I: A simple description of Tamaki's first algorithm.
We describe Tamaki's algorithm based on a well-known graph searching game for treewidth. This provides a nice link to known theory and allows us to analyze the algorithm in depth.

Contribution II. Extending Tamaki's algorithm to other parameters.
The game theoretic point-of-view allows us to extend the algorithm to various other parameters that can be defined in terms of similar games – including pathwidth, treedepth.

Contribution III: Experimental and theoretical analysis.
We provide, for the first time, theoretical bounds on the runtime of the algorithm on certain graph classes. Furthermore, we count the number of subinstances generated by the algorithm on various random and named graphs.

2 Graph Searching

A *tree decomposition* of a graph $G = (V, E)$ is a tuple (T, ι) consisting of a rooted tree T and a mapping ι from nodes of T to sets of vertices of G (called *bags*) such that (1) for all $v \in V$ the set $\{\, x \mid v \in \iota(x) \,\}$ is nonempty and connected in T, and (2) for every edge $\{v, w\} \in E$ there is a node m in T with $\{v, w\} \subseteq \iota(m)$. The *width* of a tree decomposition is the maximum size of one of its bags minus one, its *depth* is the maximum of the width and the depth of T. The *treewidth* of G, denoted by $\mathrm{tw}(G)$, is the minimum width any tree decomposition of G must have. If T is a path we call (T, ι) a *path decomposition*; if for all nodes x, y of T we have $\iota(x) \subsetneq \iota(y)$ whenever y is a descendent of x we call (T, ι) a *treedepth decomposition*; and if on any path from the root to a leaf there are at most q nodes with more then one children we call (T, ι) a *q-branched tree decomposition*. Analogous to the treewidth, we define the *pathwidth* and *q-branched-treewidth* of G, denoted by $\mathrm{pw}(G)$ and $\mathrm{tw}_q(G)$, respectively. The *treedepth* $\mathrm{td}(G)$ is the minimum depth any treedepth decomposition must have. The various parameters are illustrated in Figure 1.

Another important variant of this parameter is *dependency-treewidth*, which is used primarily in the context of quantified Boolean formulas [18]. For a graph $G = (V, E)$ and a partial order $\lessdot$ of V the dependency-treewidth $\mathrm{dtw}(G)$ is the minimum width any tree-decomposition (T, ι) with the following property must have: Consider the natural partial order $\leq_T$ that T induces on its nodes, where the root is the smallest elements and the leaves form the maximal elements; define for any $v \in V$ the node $F_v(T)$ that is the $\leq_T$-minimal node t with $v \in \iota(t)$ (which is well defined); then define a partial order $<_{\mathcal{T}}$ on V such that $u <_{\mathcal{T}} v \iff F_u(T) \leq_T F_v(T)$; finally for all $u, v \in V$ it must hold that $F_u(T) <_T F_v(T)$ implies that that $u \lessdot v$ does not hold.

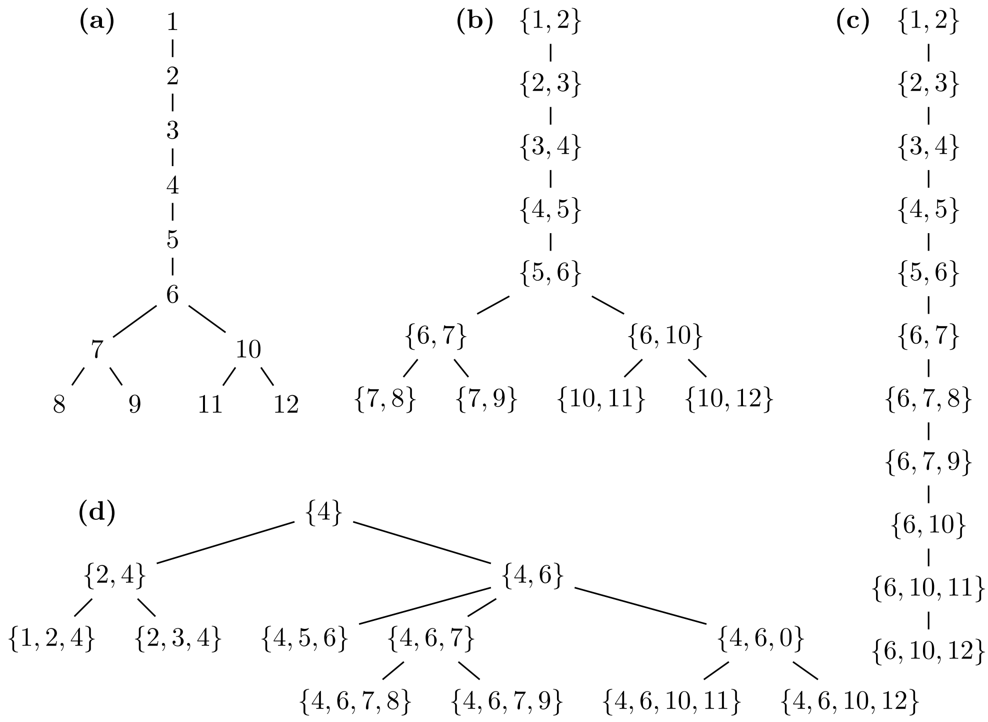


Figure 1: Various tree decompositions of an undirected graph $G = (V, E)$ shown at **(a)**. The decompositions justify **(b)** $\mathrm{tw}(G) \leq 1$, **(c)** $\mathrm{pw}(G) \leq 2$, and **(d)** $\mathrm{td}(G) \leq 3$. With respect to q-branched treewidth the decompositions also justify **(b)** $\mathrm{tw}_2(G) \leq 1$ and **(c)** $\mathrm{tw}_0(G) \leq 2$.

We study classical graph searching in a general setting proposed by Fomin, Fraigniaud, and Nisse [22]. The input is an undirected graph $G = (V, E)$ and a number $k \in \mathbb{N}$, and the question is whether a team of k searchers can catch an *invisible* fugitive on G by the following set of rules: At the beginning, the fugitive is placed at a vertex of her choice and at any time, she knows the position of the searchers. In every turn she may move with *unlimited speed* along edges of the graph, but may never cross a vertex occupied by a searcher. This implies that the fugitive does not occupy a single vertex but rather a subgraph, which is separated from the rest of the graph by the searchers. The vertices of this subgraph are called *contaminated* and at the start of the game all vertices are contaminated. The searchers, trying to catch the fugitive, can perform one of the following operations during their turn:

1. *place* a searcher on a contaminated vertex;
2. *remove* a searcher from a vertex;
3. *reveal* the current position of the fugitive.

When a searcher is placed on a contaminated vertex it becomes *clean*. When a searcher is removed from a vertex v, the vertex may become *recontaminated* if there is a contaminated vertex adjacent to v. The searchers win the game if they manage to clean all vertices, i. e., if they catch the fugitive; the fugitive wins if, at any point, a recontamination occurs, or if she can escape infinitely long. Note that this implies that the searchers have to catch the fugitive in a *monotone* way. A priori one could assume that the later condition gives the fugitive an advantage (recontamination could be necessary for the cleaning strategy), however, a crucial result in graph searching is that "recontamination does not help" in all variants of the game that we consider [6, 26, 35, 40, 37].

2.1 Entering the Arena and the Colosseum

Our primary goal is to determine whether the searchers have a winning strategy. A folklore algorithm for this task is to construct an alternating graph arena$(G,k) = ((V_s \cup V_f), E_{\mathrm{ar}})$ that contains for each position of the searchers ($S \subseteq V$ with $|S| \leq k$) and each position of the fugitive ($f \in V$) two copies of the vertex (S, f), one in V_s and one in V_f (see e. g. Section 7.4 in [14]). Vertices in V_s correspond to a configuration in which the searchers do the next move (they are existential) and vertices in V_f correspond to fugitive moves (they are universal). The edges E_{ar} are constructed according to the possible moves. Clearly, our task is now reduced to the question whether there is an alternating path from a start configuration to some configuration in which the fugitive is caught. Since alternating paths can be computed in linear time (see e. g., Section 3.4 in [28]), we immediately obtain an $O(n^{k+1})$ algorithm.

Modeling a configuration of the game as tuple (S, f) comes, however, with a major drawback: The size of the arena does directly depend on n and k and does *not* depend on some further structure of the input. For instance, the arena of a path of length n and any other graph on n vertices will have the same size for any fixed value k. As the major goal of parameterized complexity is the understanding of structural parameters beyond the input size n, such a fixed-size approach is usually not practically feasible. In contrast, we will define the configuration graph colosseum(G,k), which might be larger then arena(G,k) in general, but is also "prettier" in the sense that it adapts to the input structure of the graph. Moreover, the resulting algorithms are *self-adapting* in the sense that it needs no knowledge about this special structure to make use of it (in constrast to other parameterized algorithms, where the parameter describing this structure needs to be given explicitly).

2.2 Simplifying the Game

Our definition is based upon a similar formulation by Fomin et al. [22], but we simplify the game to make it more accessible to our techniques. First of all, we restrict the fugitive in the following sense. Since she is invisible to the searchers and travels with unlimited speed, there is no need for her to take regular actions. Instead, the only moment when she is actually active is when the searchers perform a reveal. If C is the set of contaminated vertices, consisting of the induced components $C_1, \dots, C_\ell$, a reveal will uncover the component in which the fugitive hides and, as a result, reduce C to C_i for some $1 \leq i \leq \ell$. The only task of the fugitive is, thus, to answer a reveal with such a number i. We call the whole process of the searcher performing a reveal, the fugitive answering it, and finally of reducing C to C_i a *reveal-move.*

We will also restrict the searchers by the concept of *implicit searcher removal.* Let $S \subseteq V(G)$ be the vertices currently occupied by the searchers, and let $C \subseteq V(G)$ be the set of contaminated vertices. We call a vertex $v \in S$ *covered* if every path between v and C contains a vertex $w \in S$ with $w \neq v$.

Lemma 2.1. *A covered searcher can be removed safely.*

Proof. As we have $N(v) \cap C = \emptyset$, the removal of v will not increase the contaminated area. Furthermore, at no later point of the game v can be recontaminated, unless a neighbor of v gets recontaminated as well (in which case the game would already be lost for the searchers). □

Lemma 2.2. *Only covered searchers can be removed safely.*

Proof. Since for any other vertex $w \in S$ we have $N(w) \cap C \neq \emptyset$, the removal of w would recontaminate w and, hence, would result in a defeat of the searchers. □

Both lemmas together imply that the searchers never have to decide to remove a searcher, but rather do it *implicitly*. We thus restrict the possible moves of the searchers to a combined move of placing a searcher and *immediately* removing the searchers from all covered vertices. We call this a *fly-move*. Observe that the sequence of original moves mimicked by a fly-move does not contain a reveal and, thus, may be performed independently of any action of the fugitive.

We are now ready to define the colosseum. We could, as for the arena, define it as an alternating graph. However, as the searcher is the only player that performs actions in our simplified game, we find it more natural to express this game as *edge-alternating graph* – a generalization of alternating graphs. An edge-alternating graph is a triple $H = (V, E, A)$ consisting of a *vertex set* V, an existential edge relation $E \subseteq V \times V$, and an universal edge relation $A \subseteq V \times V$. We define the neighborhood of a vertex v as $N_\exists(v) = \{\, w \mid (v, w) \in E \,\}$, $N_\forall(v) = \{\, w \mid (v, w) \in A \,\}$, and $N_H(v) = N_\exists(v) \cup N_\forall(v)$. An *edge-alternating* s-t-path is a set $P \subseteq V$ such that (1) $s, t \in P$ and (2) for all $v \in P$ with $v \neq t$ we have either $N_\exists(v) \cap P \neq \emptyset$ or $\emptyset \neq N_\forall(v) \subseteq P$ or both. We write $s \prec t$ if such a path exists and define $\mathcal{B}(Q) = \{\, v \mid v \in Q \vee (\exists w \in Q \colon v \prec w) \,\}$ for $Q \subseteq V$ as the set of vertices on edge-alternating paths leading to Q. We say that an edge-alternating s-t-path P is *q-branched*, if (i) H is acyclic and (ii) every (classical) directed path π from s to t in H with $\pi \subseteq P$ uses at most q universal edges.

For an undirected graph $G = (V, E)$ and a number $k \in \mathbb{N}$ we now define the colosseum(G, k) to be the edge-alternating graph H with vertex set $V(H) = \{\, C \mid \emptyset \neq C \subseteq V \text{ and } |N_G(C)| \leq k \,\}$ and the following edge sets: for all pairs $C, C' \in V(H)$ there is an edge $e = (C, C') \in E(H)$ if, and only if, $C \setminus \{v\} = C'$ for some $v \in C$ and $|N_G(C)| < k$; furthermore, for all $C \in V(H)$ with at least two components $C_1, \dots, C_\ell$ we have edges $(C, C_i) \in A(H)$. The *start configuration* of the game is the vertex $C = V$, that is, all vertices are contaminated. We define $Q = \{\, \{v\} \subseteq V \colon |N_G(\{v\})| < k \,\}$ to be the set of *winning configurations*, as at least one searcher is available to catch the fugitive. Therefore, the searchers have a winning strategy if, and only if, $V \in \mathcal{B}(Q)$ and we will therefore refer to $\mathcal{B}(Q)$ as the *winning region*. Observe that the colosseum is acyclic (that is, the digraph $(V, E \cup A)$ is acyclic) as we have for every edge (C, C') that $|C| > |C'|$, and observe further that Q is a subset of the sinks of H. Hence, we can test if $V \in \mathcal{B}(Q)$ in time $O(|\text{colosseum}(G, k))|)$. Finally, note that the size of colosseum(G, k) may be of order 2^n rather than n^{k+1}, giving us a slightly worse overall runtime.

The reader that is familiar with graph searching or with exact algorithms for treewidth will probably notice the similarity of the colosseum and an exact "Robertson–Seymour fashioned" algorithm as sketched in Listing 1.

Listing 1: To get some intuition behind the colosseum, consider the following procedure. It is assumed that an input graph $G = (V, E)$ and a target number $k \in \mathbb{N}$ is globally available in memory. The procedure, when called with parameters $S = \emptyset$ and $C = V$, will determine whether k searcher can catch the fugitive in the search game. Hereby, the set S is always the current position of the searchers and C is the contaminated area. We maintain the invariant $N(C) \subseteq S$, as the searchers would lose otherwise due to recontamination. Observe that from any configuration (S, C) the procedure will, without branching, move to $(N(C), C)$. These are exactly the configurations that are present in the colosseum. In fact, the colosseum is essentially the configuration graph of this procedure if it is used with memoization.

```
procedure generalGraphSearching($S, C$)

  // end of recursion
  if $|S| > k$ then // we need too many searchers
    return false
  end
  if $C = \emptyset$ then // the searchers cleaned the graph
    return true
  end

  // implicit searcher removal
  for $v \in S$ do
    if $N(v) \cap C = \emptyset$ then
      $S \leftarrow S \setminus \{v\}$
      return generalGraphSearching($S, C$)
    end
  end

  // reveal−move
  $C_1, \ldots, C_\ell \leftarrow$ connectedComponents($G[C]$)
  if $\ell > 1$ then
    return $\bigwedge_{i=1}^{\ell}$ generalGraphSearching($S, C_i$)
  end

  // fly−move
  return $\bigvee_{v \in C}$ generalGraphSearching($S \cup \{v\}, C \setminus \{v\}$)
end
```

2.3 Fighting in the Pit

Both algorithms introduced in the previous section run asymptotically in the size of the generated configuration graph $|\text{arena}(G,k)|$ or $|\text{colosseum}(G,k)|$. Both of these graphs might be very large, as the arena has fixed size of order $O(n^{k+1})$, while the colosseum may even have size $O(2^n)$. Additionally, both graphs contain many unnecessary configurations, that is, configurations that are not contained in the winning region of the searchers. In the light of dynamic programming this is the same as listing all possible configurations; and in the light of positive-instance driven dynamic programming we would like to list only the positive instances – which is exactly the winning region in this context.

To realize this idea, we consider the *pit* inside the colosseum, which is the area where only true champions can survive – formally we define $\text{pit}(G,k)$ as the subgraph of $\text{colosseum}(G,k)$ induced by $\mathcal{B}(Q)$, that is, as the induced subgraph on the winning region. The key-insight is that $|\text{pit}(G,k)|$ may be smaller than $|\text{colosseum}(G,k)|$ or even $|\text{arena}(G,k)|$ on various graph classes. Our primary goal for the next section will therefore be the development of an algorithm that computes the pit in time $O(|\text{pit}(G,k)|^2)$.

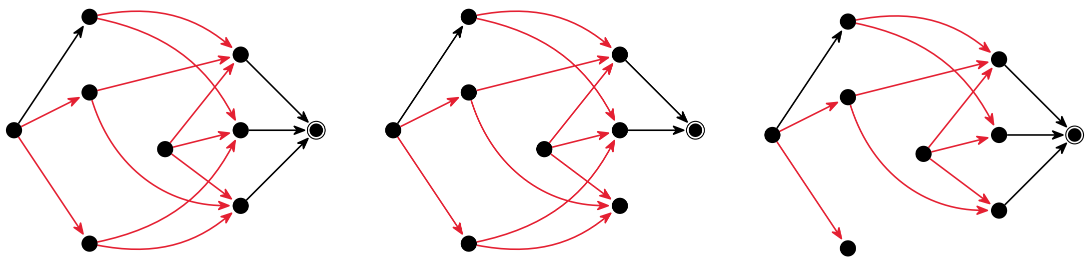
3 Computing the Pit

Our aim for this section is to develop an algorithm that computes $\text{pit}(G,k)$. Of course, a simple way to do this is to compute the whole colosseum and to extract the pit afterwards. However, this will cost time $O(2^n)$ and is surely not what we aim for. Our algorithm traverses the colosseum "backwards" by starting at the set Q of winning configurations and by uncovering $\mathcal{B}(Q)$ layer by layer. In order to achieve this, we need to compute the predecessors of a configuration C. This is easy if C was reached by a fly-move as we can simply enumerate the n possible predecessors. Reversing a reveal-move, that is, finding the universal predecessors, is significantly more involved. A simple approach is to test for every subset of already explored configurations if we can "glue" them together – but this would result in an even worse runtime of $2^{|\text{pit}(G,k)|}$. Fortunately, we can avoid this exponential blow-up as the colosseum has the following useful property:

Definition 3.1 (Universal Consistent)**.** *We say that an edge-alternating graph $H=(V,E,A)$ is* universal consistent *with respect to a set $Q \subseteq V$ if for all $v \in V \setminus Q$ with $v \in \mathcal{B}(Q)$ and $N_\forall(v)=\{w_1,\dots,w_r\}$ we have (1) $N_\forall(v) \subseteq \mathcal{B}(Q)$ and (2) for every $I \subseteq \{w_1,\dots,w_r\}$ with $|I| \geq 2$ there is a vertex $v' \in V$ with $N_\forall(v')=I$ and $v' \in \mathcal{B}(Q)$.*

Intuitively, this definition implies that for every vertex with high universal-degree there is a set of vertices that we can arrange in a tree-like fashion to realize the same adjacency relation. This allows us to glue only two configurations at a time and, thus, removes the exponential dependency. An example of the definition can be found in Example 3.2.

Example 3.2. *Consider the following three edge-alternating graphs, where a black edge is existential and the red edges are universal. The set Q contains a single vertex that is highlighted. From left to right: the first graph is universal consistent; the second and third one are not. The second graph conflicts the condition that $v \in \mathcal{B}(Q)$ implies $N_\forall(v) \subseteq \mathcal{B}(Q)$, as the vertex on the very left is contained in $\mathcal{B}(Q)$ by the top path, while its universal neighbor on the bottom path is not contained in $\mathcal{B}(Q)$. The third graph conflicts the condition that $N_\forall(v)=\{w_1,\dots,w_r\}$ implies that for every $I \subseteq \{w_1,\dots,w_r\}$ with $|I| \geq 2$ there is a vertex $v' \in V$ with $N_\forall(v')=I$ and $v' \in \mathcal{B}(Q)$ as witnessed by the vertex with three outgoing universal edges.*



Lemma 3.3. *For every graph $G = (V, E)$ and number $k \in \mathbb{N}$, the edge-alternating graph* $\mathrm{colosseum}(G, k)$ *is universal consistent.*

Proof. For the first property just observe that "reveals do not harm" in the sense that if the searchers can catch the fugitive without knowing where she hides, they certainly can do if they do know.

For the second property consider any configuration $C \in V(H)$ that has universal edges to $C_1, \dots, C_\ell$. By definition we have $|N(C)| \leq k$ and $N(C_i) \subseteq N(C)$ for all $1 \leq i \leq \ell$. Therefore we have for every $I \subseteq \{1, \dots, \ell\}$ and $C' = \cup_{i \in I} C_i$ that $N(C') \subseteq N(C)$ and $|N(C')| \leq k$ and, thus, $C' \in V(H)$. $\square$

We are now ready to formulate the algorithm for computing the pit shown in Listing 2. In essence, the algorithm runs in three phases: first it computes the set Q of winning configurations; then the winning region $\mathcal{B}(Q)$ (that is, the vertices of $\mathrm{pit}(G, k)$); and finally, it computes the edges of $\mathrm{pit}(G, k)$.

Theorem 3.4. *The algorithm* $\mathsf{Discover}(G, k)$ *finishes in at most $O\big(|\mathcal{B}(Q)|^2 \cdot |V|^2\big)$ steps and correctly outputs* $\mathrm{pit}(G, k)$.

Proof. The algorithm is supposed to compute Q in phase I, $\mathcal{B}(Q)$ in phase II, and the edges of $\mathrm{colosseum}(G, k)[\mathcal{B}(Q)]$ in phase III. First observe that Q is correctly computed in phase I by the definition of Q.

To show the correctness of the second phase we argue that the computed set $V(\mathrm{pit}(G, k))$ equals $\mathcal{B}(Q)$. Let us refer to the set $V(\mathrm{pit}(G, k))$ during the computation as K and observe that this is exactly the set of vertices inserted into the queue. We first show $K \subseteq \mathcal{B}(Q)$ by induction over the ith inserted vertex. The first vertex C_1 is in $\mathcal{B}(Q)$ as $C_1 \in Q$. Now consider C_i. As $C_i \in K$, it was either added in Line 16 or Line 20. In the first case there was a vertex $\tilde{C}_i \in K$ such that $C_i = \tilde{C}_i \cup \{v\}$ for some $v \in N(\tilde{C}_i)$. By the induction hypothesis we have $\tilde{C}_i \in \mathcal{B}(Q)$ and by the definition of the colosseum we have $(C_i, \tilde{C}_i) \in E(H)$ and, thus, $C_i \in \mathcal{B}(Q)$. In the second case there where vertices $\tilde{C}_i$ and $\hat{C}_i$ with $\tilde{C}_i, \hat{C}_i \in K$ and $C_i = \tilde{C}_i \cup \hat{C}_i$. By the induction hypothesis we have again $\tilde{C}_i, \hat{C}_i \in \mathcal{B}(Q)$. Let $t_1, \dots, t_\ell$ be the connected components of $\tilde{C}_i$ and $\hat{C}_i$. Since the colosseum H is universal consistent with respect to Q by Lemma 3.3, we have $t_1, \dots, t_\ell \in \mathcal{B}(Q)$. By the definition of the colosseum we have $N_\forall(C_i) = t_1, \dots, t_\ell$ and, thus, $C_i \in \mathcal{B}(Q)$.

To see $\mathcal{B}(Q) \subseteq K$ consider for a contradiction the vertices of $\mathcal{B}(Q)$ in reversed topological order (recall that H is acyclic) and let C be the first vertex in this order with $C \in \mathcal{B}(Q)$ and $C \notin K$. If $C \in Q$ we have $C \in K$ by phase I and are done, so assume otherwise. Since $C \in \mathcal{B}(Q)$ we have either $N_\exists(C) \cap \mathcal{B}(Q) \neq \emptyset$ or $\emptyset \neq N_\forall(C) \subseteq \mathcal{B}(Q)$. In the first case there is a $\tilde{C} \in \mathcal{B}(Q)$ with $(C, \tilde{C}) \in E(H)$. Therefore, $\tilde{C}$ precedes C in the reversed topological order and, by the choice of C, we have $\tilde{C} \in K$. Therefore, at some point of the algorithm $\tilde{C}$ gets extracted from the queue and, in Line 16, would add C to K, a contradiction.

In the second case there are vertices $t_1, \dots, t_\ell$ with $N_\forall(C) = \{t_1, \dots, t_\ell\}$ and $t_1, \dots, t_\ell \in \mathcal{B}(Q)$. By the choice of C, we have again $t_1, \dots, t_\ell \in K$. Since H is universal consistent with respect to

Q, we have for every $I \subseteq \{1, \ldots, \ell\}$ that $\bigcup_{i \in I} t_i$ is contained in $\mathcal{B}(Q)$. In particular, the vertices $t_1 \cup t_2$, $t_3 \cup t_4$, …, $t_{\ell-1} \cup t_\ell$ are contained in $\mathcal{B}(Q)$, and these elements are added to K whenever the t_i are processed (for simplicity assume here that ℓ is a power of 2). Once these elements are processed, Line 20 will also add their union, that is, vertices of the form $(t_1 \cup t_2) \cup (t_3 \cup t_4)$. In this way, the process will add vertices that correspond to increasing subgraphs of G to K, resulting ultimately in adding $\bigcup_{i=1}^{\ell} t_i = C$ into K, which is the contradiction we have been looking for.

Finally, once the set $\mathcal{B}(Q)$ is known, it is easy to compute the subgraph colosseum$(G, k)[\mathcal{B}(Q)]$, that is, to compute the edges of the subgraph induced by $\mathcal{B}(Q)$. Phase III essentially iterates over all vertices and adds edges according to the definition of the colosseum.

For the runtime, observe that the queue will contain exactly the set $\mathcal{B}(Q)$ and, for every element extracted, we search through the current $K' \subseteq \mathcal{B}(Q)$, which leads to the quadratic timebound of $|\mathcal{B}(Q)|^2$. Furthermore, we have to compute the neighborhood of every extracted element, and we have to test whether two such configurations intersect – both can easily be achieved in time $O(|V|^2)$. Finally, in phase III we have to compute connected components of the elements in $\mathcal{B}(Q)$, but since this is possible in time $O(|V| + |E|)$ per element, it is clearly possible in time $|\mathcal{B}(Q)| \cdot |V|^2$ for the whole graph. □

Listing 2: Discover(G, k)

```
1  V(pit(G,k)) := ∅
2  E(pit(G,k)) := ∅
3  A(pit(G,k)) := ∅
4  initialize empty queue
5
6  // Phase I: compute Q
7  for v ∈ V(G) do
8    insert({v}, k − 1)
9  end
10
11 // Phase II: compute B(Q) = V(pit(G,k))
12 while queue not empty do
13   extract C from queue
14   // reverse fly−moves
15   for v ∈ N(C) do
16     insert(C ∪ {v}, k − 1)
17   end
18   // reverse reveal−moves
19   for C' ∈ V(pit(G,k)) with C ∩ C' = ∅ do
20     insert(C ∪ C', k)
21   end
22 end
23
24 // Phase III: compute E and A
25 discoverEdges()
26
27 return (V(pit(G,k)), E(pit(G,k)), A(pit(G,k)))
```

Listing 3: insert(C, t)

```
if C ∉ V(pit(G,k)) and |N_G(C)| ≤ t then
  add C to V(pit(G,k))
  insert C into queue
end
```

Listing 4: discoverEdges()

```
for C ∈ V(pit(G,k)) do

  // add fly−move edges
  for v ∈ C do
    if C \ {v} ∈ V(pit(G,k)) then
      add (C, C \ {v}) to E(pit(G,k))
    end
  end

  // add reveal−move edges
  let C_1, ..., C_ℓ be
    the connected components of G[C]
  if C_1, ..., C_ℓ ∈ K then
    for i = 1 to ℓ do
      add (C, C_i) to A(pit(G,k))
    end
  end

end
```

4 Distance Queries in Edge-Alternating Graphs

In the previous section we have discussed how to compute the pit for a given graph and a given value k. The computation of treewidth now boils down to a reachability problem within this pit. But, intuitively, the pit should be able to give us much more information. In the present section we formalize this claim: We will show that we can compute shortest edge-alternating paths. To get an intuition of "distance" in edge-alternating graphs think about such a graph as in our game and consider some vertex v. There is always one active player that may decide to take *one* existential edge (a fly-move in our game), or the player may decide to ask the opponent to make a move and, thus, has to handle *all* universal edges (a reveal-move in our game). From the point of view of the active player, the distance is thus the *minimum* over the minimum of the distances of the existential edges and the maximum of the universal edges.

Definition 4.1 (Edge-Alternating Distance)**.** *Let $H = (V, E, A)$ be an edge-alternating graph with $v \in V$ and $Q \subseteq V$, let further $c_0 \in \mathbb{N}$ be a constant and $\omega_E \colon E \to \mathbb{N}$ and $\omega_A \colon A \to \mathbb{N}$ be weight functions. The distance $d(v, Q)$ from v to Q is inductively defined as $d(v, Q) = c_0$ for $v \in Q$ and otherwise:*

$$d(v,Q) = \min\big(\min_{w\in N_\exists(v)}(d(w,Q)+\omega_E(v,w)),\ \max_{w\in N_\forall(v)}(d(w,Q)+\omega_A(v,w))\big).$$

Lemma 4.2. *Given an acyclic edge-alternating graph $H = (V, E, A)$, weight functions $\omega_E \colon E \to \mathbb{N}$ and $\omega_A \colon A \to \mathbb{N}$, a source vertex $s \in V$, a subset of the sinks Q, and a constant $c_0 \in \mathbb{N}$. The value $d(s, Q)$ can be computed in time $O(|V| + |E| + |A|)$ and a corresponding edge-alternating path can be computed in the same time.*

Proof. Since H is acyclic we can compute a topological order of V using the algorithm from [30]. We iterate over the vertices v in reversed order and compute the distance as follows: if v is a sink we either set $d(v, Q) = c_0$ or $d(v, Q) = \infty$, depending on whether we have $v \in Q$. If v is not a sink we have already computed $d(w, Q)$ for all $w \in N(v)$ and, hence, can compute $d(v, Q)$ by the formula of the definition. Since this algorithm has to consider every edge once, the whole algorithm runs in time $O(|V| + |E| + |A|)$. A path from s to Q of length $d(s, Q)$ can be found by backtracking the labels starting at s. □

Theorem 4.3. *Given a graph $G = (V, E)$ and a number $k \in \mathbb{N}$, we can decide in time $O(|\mathrm{pit}(G, k + 1)|^2 \cdot |V|^2)$ whether G has { treewidth, pathwidth, treedepth, q-branched-treewidth, dependency-treewidth } at most k.*

Before we got into the details, let us briefly sketch the general idea of proving the theorem: All five problems have game theoretic characterizations in terms of the same search game with the same configuration set [6, 22, 26]. More precisely, they condense to various distance questions within the colosseum by assigning appropriate weights to the edges.

treewidth: To solve treewidth, it is sufficient to find *any* edge-alternating path from the vertex $C_s = V(G)$ to a vertex in Q. We can find a path by choosing ω_E and ω_A as $(x, y) \mapsto 0$, and by setting $c_0 = 0$.

pathwidth: In the pathwidth game, the searchers are not allowed to perform any reveal [6]. Hence, universal edges cannot be used and we set ω_A to $(x, y) \mapsto \infty$. By setting ω_E to $(x, y) \mapsto 0$ and $c_0 = 0$, we again only need to find some path from $V(G)$ to Q with weight less than ∞.

treedepth: In the game for treedepth, the searchers are not allowed to remove a placed searcher again [26]. Hence, the searchers can only use k existential edges. Choosing ω_E as $(x, y) \mapsto 1$, ω_A as $(x, y) \mapsto 0$, and $c_0 = 1$ is sufficient. We have to search a path of weight at most k.

q-branched-treewidth: For q-branched-treewidth we wish to use at most q reveals [22]. By choosing ω_E as $(x, y) \mapsto 0$, ω_A as $(x, y) \mapsto 1$, and $c_0 = 0$, we have to search for a path of weight at most q.

dependency-treewidth This parameter is in essence defined via graph searching game that is equal to the game we study with some fly- and reveal-moves forbidden. Forbidding a move can be archived by setting the weight of the corresponding edge to ∞ and by searching for an edge-alternating path of weight less then ∞.

Proof. Let us first observe that, by the definition of the colosseum, k searchers in the search game have a winning strategy if, and only if, the start configuration $V(G)$ is contained in $\mathcal{B}(Q)$. With other words, if there is an edge-alternating path from $V(G)$ to some winning configuration in Q. Note that such a path directly corresponds to the strategy by the searchers in the sense that the used edges directly correspond to possible actions of the searchers.

Since for any graph $G = (V, E)$ and any number $k \in \mathbb{N}$ the edge-alternating graph colosseum(G, k) is universal consistent by Lemma 3.3, all vertices of an edge alternating path corresponding to a winning strategy are contained in $\mathcal{B}(Q)$ as well. In fact, *every* edge-alternating path from $V(G)$ to Q (and, thus, *any* winning strategy) is completely contained in $\mathcal{B}(Q)$. Therefore, it will always be sufficient to search such paths within $\mathrm{pit}(G, k)$. By Lemma 4.2 we can find such a path in time $O(|\mathrm{pit}(G, k)|^2)$. In fact, we can even define two weight functions $w_E \colon E \to \mathbb{N}$ and $w_A \colon A \to \mathbb{N}$ and search a *shortest path* from $V(G)$ to Q. To compute the invariants of G as stated in the theorem, we make the following claim:

Claim 4.4. *Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$. Define w_E as $(x, y) \mapsto 0$ and w_A as $(x, y) \mapsto 1$, and set $c_0 = 0$. Then we have $d(V(G), Q) \leq q$ in $\mathrm{pit}(G, k)$ if, and only if, $\mathrm{tw}_q(G) \leq k - 1$.*

Proof. We follow the proof of Theorem 1 in [22] closely. We will use the following well-known fact that easily follows from the observation that in a tree decomposition (T, ι), for each three different nodes $i_1, i_2, i_3 \in T$, we have $\iota(i_1) \cap \iota(i_3) \subseteq \iota(i_2)$ if i_2 is on the unique path from i_1 to i_3 in T.

Fact 4.5. *Let (T, ι) be a tree decomposition of $G = (V, E)$ rooted arbitrarily at some node $r \in T$. Let $i \in T$ be a node and $j \in T$ be a child of i in T. Then, the set $\iota(i) \cap \iota(j)$ is a separator between $C = \big[\bigcup_{d \in \mathrm{Desc}(j)} \iota(d)\big] \setminus \big(\iota(i) \cap \iota(j)\big)$ and $\big(V \setminus C\big) \setminus \big(\iota(i) \cap \iota(j)\big)$, where $\mathrm{Desc}(x)$ denotes the set of descendants of x including x. Hence, every path from some node $u \in C$ to some node $v \in V \setminus C$ contains a vertex of $\iota(i) \cap \iota(j)$.*

From a Tree Decomposition to an Edge-alternating Path: Let (T, ι) be a q-branched tree decomposition of $G = (V, E)$ of width k. Without loss of generality, we can assume that G is connected. We will show how to construct an edge-alternating path from the start configuration V of cost at most q in colosseum$(G, k+1)$. As described above, this is also an edge-alternating path with the same costs in $\mathrm{pit}(G, k+1)$. The first existential edge from V leads to the configuration $V \setminus \iota(r)$, where r is the root of T. Clearly, $N(V \setminus \iota(r)) \subseteq \iota(r)$. Now suppose that we have reached a configuration C with $N(C) \subseteq \iota(i) \in V(\mathrm{colosseum}(G, k+1))$ for some node $i \in T$ and we have

$$C \subseteq \big[\bigcup_{j \in \mathrm{Desc}(i)} \iota(j) \big] \setminus \iota(i)$$

where $\mathrm{Desc}(i)$ are the descendants of i in T. Clearly, for $i = r$, this assumption holds trivially. If i is a leaf in T, there are no more descendants and thus $C = \emptyset$. Hence, have reached a winning configuration in colosseum$(G, k+1)$. Therefore, suppose that i is a non-leaf node. We distinguish two cases:

- If i has exactly one child j, we can find a path P_1 of existential edges leading from C to a configuration C_1 with $N(C_1) \subseteq \iota(i) \cap \iota(j)$. Moreover, we can also find a path P_2 of existential edges from C_1 to a configuration C_2 with $N(C_2) \subseteq \iota(j)$.

  The path P_1 will be constructed by iteratively removing all vertices $v \in C$ with $N(v) \cap [\iota(i) \setminus \iota(j)] \neq \emptyset$. For the remaining vertices C_1, we have $N(C_1) \subseteq \iota(i) \cap \iota(j)$. Clearly, if all configurations that we aim to visit on P_1 exists, the corresponding edges also exists by definition. Hence, assume that we are currently in some configuration C' with $N(C') \cap [\iota(i) \setminus \iota(j)] \neq \emptyset$ and want to remove some vertex $v \in C'$ with $N(v) \cap [\iota(i) \setminus \iota(j)] \neq \emptyset$, but $C' \setminus \{v\} \notin V(\text{colosseum}(G, k+1))$. By definition of colosseum$(G, k+1)$, this means that $|N(C' \setminus \{v\})| \geq k+2$. As we wanted to remove v, we have $N(v) \cap \iota(i) \neq \emptyset$. On the other hand, as $N(C' \setminus \{v\}) \subseteq N(C') \cup \{v\}$ and $|N(C' \setminus \{v\})| \geq k+2$, we know that there is some $u \in C'$ with $v \in N(u)$. Hence, Fact 4.5 implies that $v \in \iota(i) \cap \iota(j)$, a contradiction. Hence, all configurations in P_1 exist.

  Similarly, we construct P_2 by iteratively removing all vertices in $\iota(j)$ from C_1. It is easy to see that the neighborhood of the visited configurations will always be a subset of $\iota(j)$ and hence, all configurations on this path exist.

  We have thus arrived at a configuration C_2 with $N(C_2) \subseteq \iota(j)$ and

  $$C_2 \subseteq [\bigcup_{j' \in \text{Desc}(j)} \iota(j')] \setminus \iota(j)$$

  due to Fact 4.5.

- If node i has a set of children J with $|J| \geq 2$, we will use universal edges. Let $\mathcal{C}$ be the connected components of $G[\bigcup_{j \in \text{Desc}(i)} \iota(j) \setminus \iota(i)]$. We claim, that for each component $\Gamma \in \mathcal{C}$, there is a unique index $j(\Gamma) \in J$ such that $\Gamma \cap \iota(j(\Gamma)) \neq \emptyset$. If no such index exists, we have $\iota(j) = \iota(i)$. We can iteratively remove such bags $\iota(j)$ until this can not happen anymore. If two indices $j_1, j_2 \in J$ exist with $\iota(j_1) \cap \Gamma \neq \emptyset$ and $\iota(j_2) \cap \Gamma \neq \emptyset$, the connectivity property implies that $\iota(i) \cap \Gamma \neq \emptyset$, a contradiction to our assumption. Hence, for each component Γ, we follow the universal edge to Γ and then proceed as above: first, we find a path P_1 of existential edges from Γ to a configuration Γ_1 with $N(\Gamma_1) \subseteq \iota(i) \cap \iota(j(\Gamma))$ and then a path P_2 of existential edges from Γ_1 to a configuration Γ_2 with $N(\Gamma_2) \subseteq \iota(j(\Gamma))$. The same arguments as above imply that all configurations on these paths exist and that we arrive at a configuration Γ_2 with $N(\Gamma_2) \subseteq \iota(j(\Gamma))$ and

  $$\Gamma_2 \subseteq [\bigcup_{j' \in \text{Desc}(j(\Gamma))} \iota(j')] \setminus \iota(j(\Gamma)).$$

This shows that we will eventually reach the leaves of the tree decomposition and thus some wining configuration. Clearly, this is an edge-alternating path in colosseum$(G, k+1)$ and thus in pit$(G, k+1)$. Furthermore, as each path from the root of T to some leaf of T contains at most q nodes with more than one children, this path is q-branched, as we use at most q universal edges from the initial configuration V to any used winning configuration for every induced directed path. Hence, we have found an edge-alternating path in pit$(G, k+1)$ of cost at most q.

From an Edge-alternating Path to a Tree Decompostion: Let $P \subseteq V(\text{pit}(G, k+1))$ be an edge-alternating q-branched path from the initial configuration V to a final configuration $\{v^*\}$ in pit$(G, k+1)$ with $|N(\{v^*\})| \leq k$. We argue inductively on q.

- If $q = 0$, the path P does not use any universal edges. Let $\pi = \pi_1, \ldots, \pi_s$ be any classical directed path from the initial configuration V to some wining configuration $\{v^*\}$

in $\mathrm{pit}(G, k+1)$ that only uses vertices from P. As the initial configuration is $\pi_1 = V$, the winning configuration is $\pi_s = \{v^*\}$, and there are only existential edges (C, C') with $|C'| = |C| - 1$ in $\mathrm{pit}(G, k+1)$, we know that $|\pi_i| = |V| - i + 1$ and thus $s = |V|$. We say that vertex $v \in V$ is *removed at time* i, if $v \in \bigcap_{j=1}^{i} \pi_j$ and $v \notin \bigcup_{j=i+1}^{|V|} \pi_j$. We also say that v^* was removed at time $|V|$. For $i = 1, \ldots, |V|$, let v_i be the vertex removed at time i.

We will now construct a 0-branched tree decomposition (T, ι), i.e. a path decomposition. As T is a path, let $t_1, \ldots, t_{|V|}$ be the vertices on the path in their respective ordering with root t_1. We set $\iota(t_i) = N(\pi_i) \cup \{v_i\}$. For $i = 1, \ldots, |V| - 1$, there is an existential edge leading from π_i to π_{i+1} and thus $|N(\pi_i)| \leq k$. As $\pi_{|V|} = \{v_{|V|}\}$ is a winning configuration, we also have $|N(\pi_{|V|})| \leq k$. Hence, the resulting decomposition T has width at most k. As T is a path, it is also 0-branched.

We now need to verify that (T, ι) is indeed a valid tree decomposition. As every vertex v is removed at some time i, we have $v = v_i$ and thus $v \in \iota(t_i)$. Hence, every vertex is in some bag. Let $\{v_i, v_{i'}\}$ be any edge with $i < i'$. As $v_{i'} \in \pi_{i'}$ and $v_i \notin \pi_{i'}$, we have $v_i \in N(\pi_{i'})$ and thus $\{v_i, v_{i'}\} \subseteq N(\pi_{i'}) \cup \{v_{i'}\} = \iota(t_{i'})$. Hence, every edge is in some bag. Finally, let $v_i \in V$. Clearly, as $v_i \in \pi_1$, $v_i \in \pi_2, \ldots$, $v_i \in \pi_{i-1}$, the first bag where v_i might appear is $\iota(t_i)$. Let $v_{i'} \in N(v_i)$ be the neighbour of v_i that is removed at the latest time. If $i' < i$, we have $N(v_i) \cap \bigcup_{j=i+1}^{|V|} \pi_j = \emptyset$ and v_i thus only appears in $\iota(t_i)$. If $i < i'$, then $v_i \in \bigcap_{j=i+1}^{i'} N(\pi_j)$ and hence $v_i \in \bigcap_{j=i+1}^{i'} \iota(t_j)$.

- Now, assume that $q \geq 1$ and that we can construct for every $q' < q$ a q'-branched tree decomposition of width at most k from any q'-branched edge-alternating path P in $\mathrm{pit}(G, k+1)$. Consider the directed acyclic subgraph H in $\mathrm{pit}(G, k+1)$ induced by P. A configuration $C \in V(H)$ is called a *universal configuration*, if $N_A(C) \subseteq V(H)$ and a *top-level universal configuration* with respect to some directed path π if C is the first universal configuration on π. Note that we can reduce P in such a way that all directed paths π from the initial configuration V to some winning configuration $\{v^*\}$ in H have the same top-level universal configuration, call it C^*. Let $V = \pi_1, \ldots, \pi_i = C^*$ be the shared existential path from V to C^* in H and let $N_A(C^*) = \{C_1, \ldots, C_\ell\}$ be the universal children of C^*. Note that $\{C_1, \ldots, C_\ell\} \subseteq P$ due to the definition of an edge-alternating path. For each child C_j, the edge-alternating path P contains a directed path $\pi^{(j)}$ from C_j to some final configuration in $\mathrm{pit}(G, k+1)$. Furthermore, each $\pi^{(j)}$ contains at most $q' \leq q - 1$ universal edges (otherwise, P would not be q-branched). Hence, by induction hypothesis, we can construct a q'-branched tree decomposition $(T^{(j), \iota^{(j)}})$ for the subgraph induced by the vertices contained in the path $\pi^{(j)}$ with root $r^{(j)}$.

  Now, we use the same construction as above to construct a path $(T' = (t'_1, \ldots, t'_i), \iota')$ from $\pi_1, \ldots, \pi_i$ and for each path $\pi^{(j)}$, we add the root $r^{(j)}$ of the q'-branched tree decomposition $(T^{(j)}, \iota(j))$ as a child of bag t_i to obtain our final tree decomposition (T, ι). As there is a universal edge from C^* to C_j, we know that C_j is a component of C^*. As all $(T^{(j)}, \iota(j))$ are valid $q-1$-branched tree decompositions of width at most k, we can thus conclude that (T, ι) is a valid q-branched tree decomposition of width k. $\square$

Combining the above claim with Theorem 3.4 for computing the pit, we conclude that we can check whether a graph G has q-branched-treewidth k in time $O(|\mathrm{pit}(G, k+1)|^2 \cdot |V|^2)$. We note that the algorithm is fully constructive, as the obtained path (and, hence, the winning strategy of the searchers) directly corresponds to the desired decomposition. Since we have $\mathrm{tw}(G) = \mathrm{tw}_\infty(G)$ and $\mathrm{pw}(G) = \mathrm{tw}_0(G)$, the above results immediately implies the same statement for treewidth and pathwidth by checking $d(V(G), k) < \infty$ or $d(V(G), k) = 0$, respectively.

In order to show the statement for treedepth, we will require another claim for different weight functions. The proof idea is, however, very similar.

Claim 4.6. *Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$. Define w_E as $(x, y) \mapsto 1$ and w_A as $(x, y) \mapsto 0$, and $c_0 = 1$. Then we have $d(V(G), Q) \leq k$ in $\mathrm{pit}(G, k)$ if, and only if, $\mathrm{td}(G) \leq k$.*

Proof. To prove the claim, we use an alternative representation of treedepth [38]. Let $G = (V, E)$ be a graph with connected components $C_1, \dots, C_\ell$, then:

$$\mathrm{td}(G) = \begin{cases} 1 & \text{if } |V| = 1; \\ \max_{i=1}^{\ell} \mathrm{td}(G[C_i]) & \text{if } \ell \geq 2; \\ \min_{v \in V} \mathrm{td}(G[V \setminus \{v\}]) + 1 & \text{otherwise.} \end{cases}$$

Let us reformulate this definition a bit. Let $C \subseteq V$ be a subset of the vertices and let $C_1, \dots, C_\ell$ be the connected components of $G[C]$. Define:

$$\mathrm{td}^*(C) = \begin{cases} 1 & \text{if } |C| = 1; \\ \max_{i=1}^{\ell} \mathrm{td}^*(C_i) & \text{if } \ell \geq 2; \\ \min_{v \in C} \mathrm{td}^*(C \setminus \{v\}) + 1 & \text{otherwise.} \end{cases}$$

Obviously, $\mathrm{td}(G) = \mathrm{td}^*(V)$. We proof that for any $C \subseteq V$ we have $d(C, Q) = \mathrm{td}^*(C)$ in $\mathrm{pit}(G, k)$ for every $k \geq \mathrm{td}(G)$ and $d(C, Q) \geq \mathrm{td}^*(C)$ for all $k < \mathrm{td}(G)$.

For the first part we consider the vertices of $\mathrm{pit}(G, k)$ in inverse topological order and prove the claim by induction. The first vertex C_0 is in Q and thus $d(C_0, Q) = c_0 = 1$. Since the vertices in Q represent sets of cardinality 1, we have $d(C_0, Q) = \mathrm{td}^*(C_0)$. For the inductive step consider C_i and first assume it is not connected in G. Then

$$\begin{aligned} d(C_i, Q) &= \max_{C_j \in N_\forall(C_i)} \big(d(C_j, Q) + w_A(C_i, C_j)\big) \\ &= \max_{C_j \in N_\forall(C_i)} d(C_j, Q) \\ &= \max_{C_j \text{ is a component in } G[C_i]} \mathrm{td}^*(C_j) \\ &= \mathrm{td}^*(C_i). \end{aligned}$$

Note that there could, of course, also be existential edges leaving C_i. However, since the universal edges are "for free," for every shortest path that uses an existential edge at C_i, there is also one that first uses the universal edges.

For the second case, that is C_i is connected, observe that C_i is not incident to any universal edge. Therefore we obtain:

$$\begin{aligned} d(C_i, Q) &= \min_{v \in C_i} \big(d(C_i \setminus \{v\}, Q) + w_E(C_i, C_i \setminus \{v\})\big) \\ &= \min_{v \in C_i} \big(d(C_i \setminus \{v\}, Q) + 1\big) \\ &= \min_{v \in C_i} \big(\mathrm{td}^*(C_i \setminus \{v\}) + 1\big) \\ &= \mathrm{td}^*(C_i). \end{aligned}$$

This completes the part of the proof that shows $d(C, Q) = \mathrm{td}^*(C)$ for $k \geq \mathrm{td}(G)$. We are left with the task to argue that $d(C, Q) \geq \mathrm{td}^*(C)$ for all $k < \mathrm{td}(G)$. This follows by the fact that for every $k' < k$ we have that $\mathrm{pit}(G, k')$ is an induced subgraph of $\mathrm{pit}(G, k)$. Therefore, the distance can only increase in the pit for a smaller k – in fact, the distance can even become infinity if $k < \mathrm{td}(G)$. ☐

Again, combining the claim with Theorem 3.4 yields the statement of the theorem for treedepth. Finally, we will prove the statement for dependency-treewidth. This parameter can be characterized by a small adaption of the graph searching game [18]: In addition to the graph G and the parameter k, one is also given a partial ordering $\lessdot$ on the vertices of G. For a vertex set V', let $\mu_{\lessdot}(V') = \{\, v \in V' \mid \forall w \in V' \setminus \{v\} : (w, v) \notin \lessdot \,\}$ be the minimal elements of V' with regard to $\lessdot$. If $C \subseteq V(G)$ is the contaminated area, we are only allowed to put a searcher on $\mu_{\lessdot}(C)$, rather than on all of C. The *dependency-treewidth* $\mathrm{dtw}_{\lessdot}(G)$ is the minimal number of searchers required to catch the fugitive in this version of the game. Therefore, we just need a way to permit only existential edges (C, C') with $C \setminus C' \subseteq \mu_{\lessdot}(C)$. We show the following stronger claim:

Claim 4.7. *Consider a variant of the search game in which at some configurations C_i some fly-moves are forbidden, and in which furthermore at some configurations C_j no reveals are allowed. Whether k searcher have a winning strategy in this game can be decided in time $O(|\mathrm{pit}(G, k)|^2 \cdot |V|^2)$.*

Proof. First observe that, if the k searcher have a winning strategy S, this strategy corresponds to a path in $\mathrm{pit}(G, k)$. The reason is that searchers that are allowed to use all fly- and reveal-moves (and for which all winning strategies correspond to paths in $\mathrm{pit}(G, k)$) can, of course, use S as well. We compute the pit with Theorem 3.4.

Now to find the restricted winning strategy we initially set w_E and w_A to $(x, y) \mapsto 0$. Then for any existential edge (C_i, C_j) that we wish to forbid we set $w_E(C_i, C_j) = \infty$. Furthermore, for any node C at witch we would like to forbid universal edges we set $w_E(C_i, C_j) = \infty$ for all $C_j \in N_{\forall}(C_i)$. Finally, we search a path from $V(G)$ to Q of weight less then ∞ using Lemma 4.2. □

This completes the proof of Theorem 4.3. □

5 Theoretical Bounds for Certain Graph Classes

In general, it is hard to compare the size of the arena, the colosseum, and the pit. For instance, already simple graph classes as paths (P_n) and stars (S_n) reveal that the colosseum may be smaller or larger than the arena (the arena has size $O(n^3)$ on both, but the colosseum has size $O(n)$ on P_n and $O(2^n)$ on S_n, both with regard to their optimal treewidth 1). However, experimental data of the PACE challenge [16, 17] shows that the pit is very small in practice. In the following, we are thus interested in graph classes where we can give theoretical guarantees on the size of the pit. We will first show that the colosseum is indeed often smaller than the arena (Lemma 5.1) and furthermore, that the pit might be much smaller than the colosseum (Lemma 5.3).

Lemma 5.1. *For every connected claw-free graph $G = (V, E)$ and integer $k \in \mathbb{N}$, it holds that* $|\text{colosseum}(G, k)| \leq \sum_{i=1}^{k} \binom{n}{i} \cdot 2^{2i} \in O(\binom{n}{k} \cdot 4^k)$.

Proof. Observe that in a claw-free graph every $X \subseteq V$ separates G in at most $2 \cdot |X|$ components, as every component is connected to a vertex in X (since G is connected), but every vertex in X may be connected to at most two components (otherwise it forms a claw). In the colosseum, every configuration C corresponds to a separator $N(C)$ of size at most k, and there are at most $\sum_{i=1}^{k} \binom{n}{i}$ such separators. For each separator we may combine its associated components in an arbitrary fashion to build configurations of the colosseum, but since there are at most $2 \cdot i$ components, we can build at most $2^{2 \cdot i}$ configurations. $\square$

We remark that the result of Lemma 5.1 can easily be extended to $K_{1,t}$-free graphs for every fixed t, and that this result is rather tight:

Lemma 5.2. *Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$. It holds that* $|\text{colosseum}(G, k)| \geq \sum_{i=1}^{k} \binom{|V_i|}{i}$, *where $V_i = \{v \in V : |N(v)| \geq i\}$.*

Proof. Let X be any subset of at most i vertices from V_i with $i \leq k$. As $|X| \leq i$, every vertex in X has a neighbour in $V \setminus X$. Hence, $N(V \setminus X) = X$ and thus $|N(V \setminus X)| \leq k$ and $V \setminus X \in V(\text{colosseum}(G, k))$. $\square$

We now show that the pit, on the other hand, can be substantially smaller than the colosseum even for graphs with many high-degree vertices. For $n, k \in \mathbb{N}$ with $n \geq 2k$, we define the graph $P_{n,k}$ on vertices $V(P_{n,k}) = \{v_0, v_1, \dots, v_{n \cdot k}, v_{n \cdot k+1}\}$. For $i = 1, \dots, n$, let $X_i = \{v_{(i-1) \cdot k+1}, v_{(i-1) \cdot k+2}, \dots, v_{(i-1) \cdot k+k}\}$, $X_0 = \{v_0\}$, and $X_{n+1} = \{v_{n \cdot k+1}\}$. The edges $E(P_{n,k})$ are defined as

$$E(P_{n,k}) = \bigcup_{i=1}^{n} \{\{u, v\} \mid u, v \in X_i\} \cup \bigcup_{i=0}^{n} \{\{u, v\} \mid u \in X_i, v \in X_{i+1}\}.$$

Informally, $P_{n,k}$ is constructed by taking a path of length $n + 2$ and replacing the inner vertices by cliques of size k that are completely connected to each other.

Lemma 5.3. *It holds:*

(i) $\text{tw}(P_{n,k}) = \text{pw}(P_{n,k}) = 2k - 1$;

(ii) $|\text{arena}(P_{n,k}, 2k)| = 2 \cdot \binom{n \cdot k+2}{2k+1}$;

(iii) $|\text{colosseum}(P_{n,k}, 2k)| \geq \sum_{i=1}^{2k} \binom{n \cdot k}{i}$;

(iv) $|\text{pit}(P_{n,k}, 2k)| \in O(n^2 + n \cdot 2^{6k})$.

Proof. Property *(i)* holds as $P_{n,k}$ contains a clique of size $2k$ and the path decomposition $[X_0 \cup X_1, X_1 \cup X_2, \ldots, X_n \cup X_{n+1}]$ is valid. Property *(ii)* holds by definition of $\mathrm{arena}(P_{n,k}, 2k)$. To see *(iii)*, observe that for $v \in \bigcup_{i=1}^{n} X_i$, we have $|N(v)| \geq 2k$. Lemma 5.2 thus implies $|\mathrm{colosseum}(P_{n,k}, 2k)| \geq \sum_{i=1}^{2k} \binom{n \cdot k}{i}$.

In order to prove Property *(iv)* we count the number of configurations inserted into the queue by algorithm Discover. Theorem 3.4 shows that this number equals the size of the pit. All configurations either include v_0 (a *left* configuration), $v_{k \cdot n+1}$ (a *right* configuration) or both (a *mixed* configuration), as $\{v_0\}, \{v_{k \cdot n+1}\}$ are the only winning configurations. Left and right configurations can be extended by reverse fly-moves (Line 16) as follows: Starting from $C = X_0$, we can add a vertex $u \in X_1$ to C to generate C'. The neighborhood of C' will be $\big(X_1 \setminus \{u\}\big) \cup X_2$. Adding a vertex w of X_2 to C' would result in a configuration with neighborhood $\big(X_1 \setminus \{u\}\big) \cup \big(X_2 \setminus \{w\}\big) \cup X_3$ greater than $2k$. Hence, further reverse fly-moves have to add X_1 completely to C' before elements of X_2 can be added. An inductive arguments yields that configurations constructed in this way have the form $X_0 \cup X_1 \cup \cdots \cup X_{i-1} \cup \tilde{X}_i$ with $\tilde{X}_i \subseteq X_i$. As the same is true when starting with X_{n+1}, we can generate $2 \cdot n \cdot 2^k$ such configurations.

Now consider reverse reveal-moves (Line 20). We can only unite a left configuration $C_1 = X_0 \cup X_1 \cup \cdots \cup X_{i-1} \cup \tilde{X}_i$ with a right one $C_2 = X_{n+1} \cup X_n \cup \cdots \cup X_{j+1} \cup \tilde{X}_j$. If $\tilde{X}_i = \tilde{X}_j = \emptyset$, $C_1 \cup C_2$ is a legal configuration and there are $\binom{n}{2}$ such configurations. If $|\tilde{X}_i \cup \tilde{X}_j| > 0$, we have $|N(C_1)| + |N(C_2)| > 2k$, and can not add such configurations. For the combinations with $i+1 < j-2$, we have $C_{j-2} \cap (N(C_1) \cup N(C_2)) = \emptyset$ and thus $N(C_1) \cap N(C_2) = \emptyset$, which implies that $C_1 \cup C_2$ is not legal due to $|N(C_1) \cup N(C_2)| = |N(C_1)| + |N(C_2)| > 2k$. The remaining combinations have $i+1 \geq j-2$. For each fixed i, there are at most four such values of j ($j \in \{i, i+1, i+2, i+3\}$). Both $\tilde{X}_i$ and $\tilde{X}_j$ might be arbitrary and we can thus create at most $4 \cdot n \cdot 2^k \cdot 2^k = 4 \cdot n \cdot 2^{2k}$ configurations.

Finally, we can perform reverse fly-moves for mixed configurations of the form $C = X_0 \cup X_1 \cup \cdots \cup X_{i-1} \cup \tilde{X}_i \cup \tilde{X}_j \cup X_{j+1} \cup \cdots \cup X_n \cup X_{n+1}$ with $i+1 \geq j-2$ (otherwise, the neighborhood is too large). Let $U = V \setminus C$ be the uncontaminated vertices. As $U \subseteq X_i \setminus \tilde{X}_i \cup X_{i+1} \cup X_{i+2} \cup X_j \setminus \tilde{X}_j$, we have $|U| \leq 4k$. Hence, for each of the $4 \cdot n \cdot 2^{2k}$ configurations with $i+1 \geq j-2$ there are at most 2^{4k} configurations reachable by reverse fly-moves – yielding at most $4 \cdot n \cdot 2^{6k}$ configurations. Overall, we thus have $n \cdot 2^k + \binom{n}{2} + 4 \cdot n \cdot 2^{6k}$ configurations. □

6 Experimental Estimation of the Pit Size

A heavily optimized version of the treewidth algorithm described above has been implemented in the Java library Jdrasil [4, 2]. To show the usefulness of our general approach, we experimentally compared the size of the pit, the arena, and the colosseum for various named graphs known from the DIMACS Coloring Challenge [29] or the PACE [16, 17]. For each graph the values are taken for the minimal k such that k searchers can win. Note that $|\mathrm{arena}(G,k)| \leq |\mathrm{pit}(G,k)|$ holds only in 6 of 24 cases, emphasized by underlining.

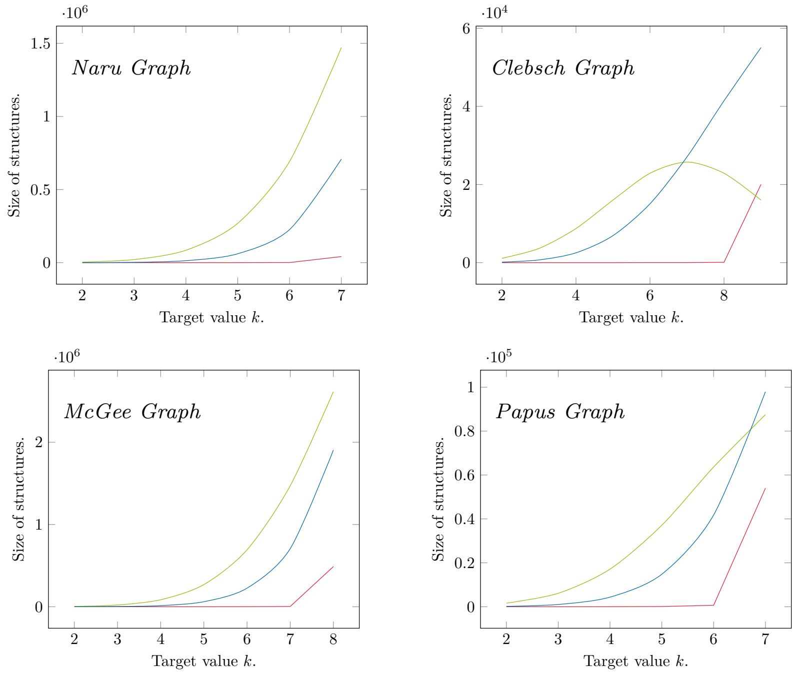
Graph	$\lvert V\rvert$	$\lvert E\rvert$	k	Pit	Arena	Col.
Grotzsch	11	20	6	1,235	660	1,853
Heawood	14	21	6	5,601	6,864	9,984
Chvatal	12	24	7	3,170	990	3,895
Goldner Harary	11	27	4	103	924	639
Sierpinski Gasket	15	27	4	488	6,006	2,494
Blanusa 2. Snark	18	27	5	861	37,128	15,413
Icosahedral	12	30	7	2,380	990	3,575
Pappus	18	27	7	54,004	87,516	97,970
Desargues	20	30	7	85,146	251,940	202,661
Dodecahedral	20	30	7	112,924	251,940	207,165
Flower Snark	20	30	7	79,842	251,940	203,473
Gen. Petersen	20	30	7	78,384	251,940	202,685

Graph	$\lvert V\rvert$	$\lvert E\rvert$	k	Pit	Arena	Col.
Hoffman	16	32	7	5,851	25,740	30,270
Friendship 10	21	30	3	57,554	11,970	58,695
Poussin	15	39	7	3,745	12,870	17,358
Markstroem	24	36	5	13,846	269,192	71,604
McGee	24	36	8	487,883	2,615,008	1,905,241
Naru	24	36	7	41,623	1,470,942	708,044
Clebsch	16	40	9	20,035	16,016	55,040
Folkman	20	40	7	21,661	251,940	151,791
Errera	17	45	7	3,527	48,620	42,418
Shrikhande	16	48	10	50,627	8,736	61,456
Paley	17	68	12	114,479	4,760	129,474
Goethals Seidel	16	72	12	54,833	1,120	65296

We have performed the same experiment on various random graph models. For each model we picked 25 graphs at random and build the mean over all instances, where each instance contributed values for its minimal k. We used all 3 models with $N = 25$ and, for the first two with $p = 0.33$; and for the later two with $K = 5$. For a detailed description of the models see for instance [10].

Model	$\lvert\mathrm{pit}(G, \mathrm{OPT})\rvert$	$\lvert\mathrm{arena}(G, \mathrm{OPT})\rvert$	$\lvert\mathrm{colosseum}(G, \mathrm{OPT})\rvert$
Erdős–Rényi	66,320	342,918	503,767
Watts Strogats	15,323	192,185	108,074
Barabási Albert	61,147	352,716	551,661

Finally, we observe the growth of the pit, the arena, and the colosseum for a fixed graph if we raise k from 2 to the optimal value. While the arena shows its binomial behavior, the colosseum is in many early stages actually smaller then the arena. This effect is even more extreme for the pit, which is *very* small for k that are smaller then the optimum. This makes the technique especially well suited to establish lower bounds, an observation also made by Tamaki [43].



7 Conclusion and Outlook

Treewidth is one of the most useful graph parameters that is successfully used in many different areas. The Positive-Instance Driven algorithm of Tamaki has led to the first practically relevant algorithm for this parameter. We have formalized Tamaki's algorithm in the more general setting of graph searching, which has allowed us to (i) provide a clean and simple formulation; and (ii) extend the algorithm to many natural graph parameters. With a few further modification of the colosseum, our approach can also be used for the notion of *special-treewidth* [13]. We assume that a similar modification may also be possible for other parameters such as *spaghetti-treewidth* [9].

Acknowledgements: The authors would like to thank Jan Arne Telle and Fedor Fomin for helpful discussions about the topic and its presentation.

References

[1] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[2] M. Bannach, S. Berndt, and T. Ehlers . Jdrasil. `https://github.com/maxbannach/Jdrasil`, 2017. [Online; accessed 09-02-2019].

[3] Max Bannach and Sebastian Berndt. Practical access to dynamic programming on tree decompositions. In *ESA*, pages 6:1–6:13, 2018.

[4] Max Bannach, Sebastian Berndt, and Thorsten Ehlers. Jdrasil: A modular library for computing tree decompositions. In *SEA*, pages 28:1–28:21, 2017.

[5] Jeremias Berg, Matti Järvisalo, and Brandon Malone. Learning optimal bounded treewidth bayesian networks via maximum satisfiability. In *Artificial Intelligence and Statistics*, pages 86–95, 2014.

[6] Daniel Bienstock and Paul D. Seymour. Monotonicity in graph searching. *Journal of Algorithms*, 12(2):239–245, 1991.

[7] Per Bjesse, James H. Kukula, Robert F. Damiano, Ted Stanion, and Yunshan Zhu. Guiding SAT diagnosis with tree decompositions. In *SAT*, pages 315–329, 2003.

[8] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[9] Hans L. Bodlaender, Stefan Kratsch, Vincent J. C. Kreuzen, O-joung Kwon, and Seongmin Ok. Characterizing width two for variants of treewidth. *Discrete Applied Mathematics*, 216:29–46, 2017.

[10] Béla Bollobás. Random graphs. In *Modern graph theory*, pages 215–252. Springer, 1998.

[11] Vincent Bouchitté and Ioan Todinca. Listing all potential maximal cliques of a graph. *Theor. Comput. Sci.*, 276(1-2):17–32, 2002.

[12] Günther Charwat and Stefan Woltran. Dynamic programming-based QBF solving. In *QBF*, pages 27–40, 2016.

[13] Bruno Courcelle. On the model-checking of monadic second-order formulas with edge set quantifications. *Discrete Applied Mathematics*, 160(6):866–887, 2012.

[14] M. Cygan, F. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer Berlin Heidelberg, 2015.

[15] Adnan Darwiche. A differential approach to inference in bayesian networks. *Journal of the ACM (JACM)*, 50(3):280–305, 2003.

[16] H. Dell, T. Husfeldt, B. Jansen, P. Kaski, C. Komusiewicz, and F. Rosamond. The First Parameterized Algorithms and Computational Experiments Challenge. In *IPEC 2016*, pages 30:1–30:9, 2017.

[17] H. Dell, C. Komusiewicz, N. Talmon, and M. Weller. The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The Second Iteration. In *IPEC 2017*, 2018.

[18] Eduard Eiben, Robert Ganian, and Sebastian Ordyniak. Small resolution proofs for QBF using dependency treewidth. In *STACS*, volume 96 of *LIPIcs*, pages 28:1–28:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[19] Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. Faster algorithms for integer programs with block structure. In *ICALP*, volume 107 of *LIPIcs*, pages 49:1–49:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[20] Gal Elidan and Stephen Gould. Learning bounded treewidth bayesian networks. *Journal of Machine Learning Research*, 9(Dec):2699–2731, 2008.

[21] Johannes Klaus Fichte, Markus Hecher, Stefan Woltran, and Markus Zisser. Weighted model counting on the GPU by exploiting small treewidth. In *ESA*, pages 28:1–28:16, 2018.

[22] F. Fomin, P. Fraigniaud, and N. Nisse. Nondeterministic graph searching: From pathwidth to treewidth. *Algorithmica*, 53(3):358–373, 2009.

[23] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010.

[24] Robert Ganian and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP. *Artif. Intell.*, 257:61–71, 2018.

[25] Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Going beyond primal treewidth for (M)ILP. In *AAAI*, pages 815–821. AAAI Press, 2017.

[26] A. Giannopoulou, P. Hunter, and D. Thilikos. Lifo-search: A min-max theorem and a searching game for cycle-rank and tree-depth. *Discrete Applied Mathematics*, 160(15):2089–2097, 2012.

[27] Djamal Habet, Lionel Paris, and Cyril Terrioux. A tree decomposition based approach to solve structured SAT instances. In *ICTAI*, pages 115–122, 2009.

[28] N. Immerman. *Descriptive complexity*. Springer, 1999.

[29] David S Johnson and Michael A Trick. *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*, volume 26. American Mathematical Soc., 1996.

[30] A. B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, 1962.

[31] Shant Karakashian, Robert J. Woodward, and Berthe Y. Choueiry. Improving the performance of consistency algorithms by localizing and bolstering propagation in a tree decomposition. In *AAAI*, 2013.

[32] Joachim Kneis, Alexander Langer, and Peter Rossmanith. Courcelle's theorem - A game-theoretic approach. *Discrete Optimization*, 8(4):568–594, 2011.

[33] Arie M. C. A. Koster, Stan P. M. van Hoesel, and Antoon W. J. Kolen. Solving partial constraint satisfaction problems with tree decomposition. *Networks*, 40(3):170–180, 2002.

[34] Martin Koutecký, Asaf Levin, and Shmuel Onn. A parameterized strongly polynomial algorithm for block structured integer programs. In *ICALP*, volume 107 of *LIPIcs*, pages 85:1–85:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[35] A. LaPaugh. Recontamination does not help to search a graph. *ACM*, 40:224–245, 1993.

[36] L. Larisch and F. Salfelder. p17. `https://github.com/freetdi/p17`, 2017. [Online; accessed 02-08-2017].

[37] F. Mazoit and N. Nisse. Monotonicity of non-deterministic graph searching. *TCS*, 399(3):169–178, 2008.

[38] J. Nesetril and P. Ossona de Mendez. *Sparsity*. Springer, 2012.

[39] Hein Röhrig. Tree decomposition: A feasibility study. Diploma thesis, Max-Planck-Institut für Informatik in Saarbrücken, 1998.

[40] P. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *JCT*, 58:22–33, 1993.

[41] Stefan Szeider. On fixed-parameter tractable parameterizations of SAT. In *SAT*, pages 188–202, 2003.

[42] H. Tamaki. treewidth-exact. `github.com/TCS-Meiji/treewidth-exact`, 2016. [Online; accessed 02-08-2017].

[43] H. Tamaki. Positive-instance driven dynamic programming for treewidth. In *ESA*, pages 68:1–68:13, 2017.