State Canonization and Early Pruning in Width-Based Automated Theorem Proving

Mateus de Oliveira Oliveira
Department of Computer and Systems Sciences, Stockholm University, Sweden
Department of Informatics, University of Bergen, Norway

Farhad Vadiee
Department of Informatics, University of Bergen, Norway

Abstract

Width-based automated theorem proving is a framework where counter-examples for graph theoretic conjectures are searched width-wise relative to some graph width measure, such as treewidth or pathwidth. In a recent work it has been shown that dynamic programming algorithms operating on tree decompositions can be combined together with the purpose of width-based theorem proving. This approach can be used to show that several long-standing conjectures in graph theory can be tested in time $2^{2^{k^{O(1)}}}$ on the class of graphs of treewidth at most k. In this work, we give the first steps towards evaluating the viability of this framework from a practical standpoint. At the same time, we advance the framework in two directions. First, we introduce a state-canonization technique that significantly reduces the number of states evaluated during the search for a counter-example of the conjecture. Second, we introduce an early-pruning technique that can be applied in the study of conjectures of the form $\mathbb{P}_1 \rightarrow \mathbb{P}_2$, for graph properties $\mathbb{P}_1$ and $\mathbb{P}_2$, where $\mathbb{P}_1$ is a property closed under subgraphs.

As a concrete application, we use our framework in the study of graph theoretic conjectures related to coloring triangle free graphs. In particular, our algorithm is able to show that Reed's conjecture for triangle free graphs is valid on the class of graphs of pathwidth at most 5, and on graphs of treewidth at most 3. Perhaps more interestingly, our algorithm is able to construct in a completely automated way counter-examples for non-valid strengthenings of Reed's conjecture. These are the first results showing that width-based automated theorem proving is a promising avenue in the study of graph-theoretic conjectures.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Width-Based Automated Theorem Proving, Dynamic Programming, Parameterized Complexity

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.33

Funding Research Council of Norway, Project Number 288761

1 Introduction

Width-based automated theorem proving is a framework where parameterized algorithms are employed to search for counterexamples to graph-theoretic conjectures. Within this framework, the search for counterexamples is conducted width-wise, relative to some specific width measure for graphs, such as treewidth or pathwidth. More specifically, given a conjecture C and a positive integer k, the objective is to determine whether C holds on the class of graphs of width at most k. If C does not hold on this class of graphs, a counterexample of width at most k that invalidates the conjecture should be produced.

This approach is relevant for two main reasons. First, many interesting classes of graphs have small width with respect to some graph width measure. For example, trees and forests have treewidth at most 1, series-parallel graphs have treewidth at most 2 and outerplanar graphs have treewidth at most 2 [16, 2, 5]. It has also been shown that k-outerplanar graphs have treewidth $O(k)$ [1, 15], and that k-caterpilars have pathwidth at most k [20]. Second, many important conjectures in graph theory are not known to hold on classes of graphs of small treewidth or pathwidth, and it is therefore natural to try to determine whether such conjectures hold when restricted to such classes of graphs. In many cases, structural properties of graphs of small treewidth or pathwidth have been used to produce analytic proofs of special cases of several well studied conjectures [11, 6, 12, 13, 19, 22, 13, 17]. The framework of width-based automated theorem proving is an avenue for automatizing this approach for certain classes of conjectures.

In a recent work [9], a new approach for width-based automated theorem proving was introduced. Within this approach, instead of specifying graph properties using logical formulas, such properties are specified using dynamic programming algorithms operating on graph decompositions. More specifically, it was shown in [9] that given dynamic programming cores $\mathsf{D}_1, \mathsf{D}_2, \ldots, \mathsf{D}_r$ specifying graph properties $\mathbb{P}_1, \mathbb{P}_2, \ldots, \mathbb{P}_r$, then for any Boolean combination $\mathbb{P}$ of these properties, there is an algorithm $\mathfrak{A}$ that takes a number k as input and decides whether all graphs of treewidth at most k belong to $\mathbb{P}$. Furthermore, if not all graphs of treewidth k belong to $\mathbb{P}$, then the algorithm outputs a certificate that can be used to extract a counter-example. This approach takes double-exponential time with respect to the number of bits needed to represent local witnesses used by the dynamic programming algorithms. Given that many interesting graph properties have DP-algorithms that use local witnesses of size $k^{O(1)}$ when processing tree decompositions of width k, the dynamic programming approach developed in [9] implies that many interesting conjectures can be tested in time double-exponential in $k^{O(1)}$ on the class of graphs of treewidth at most k. This includes long-standing conjectures such as Hadwiger's conjecture (for a fixed number of colors) [4], Tutte's flow conjectures [14, 18], Reed triangle free conjecture etc.

1.1 Our Results

In this work we give the first steps towards evaluating the viability of the width-based automated theorem proving framework introduced in [9] from a practical perspective. At the same time, we advance this framework by introducing a new width based deduction algorithm that produces significantly less states when compared with the algorithm originally introduced in [9]. Our new algorithm leverages on the introduction of two techniques. First, we introduce a suitable notion of state canonization. The second is an early pruning technique that can be used in the study of conjectures of the form $\mathbb{P}_1 \rightarrow \mathbb{P}_2$, where $\mathbb{P}_1$ is a graph property closed under subgraphs. As a concrete case study, we use our implementation to study graph-theoretic statements related to colorings of graphs. Our algorithm was able to produce non-trivial counter-examples for false statements, and also to confirm a well known conjecture due to Reed [21] on the class of graphs of pathwidth at most 5, and on the class of graphs of treewidth at most 3. Together, our results provide the first evidence that the width-based ATP framework introduced in [9] may be a promising avenue in the study of graph-theoretic conjectures.

1.2 Related Work

The approach for width-based automated theorem proving introduced in [9], and further developed in the present work, is heavily based on dynamic programming algorithms deciding graph theoretic properties. Another suitable approach for the study of width-based automated theorem proving is based on logic [7, 8]. In this context, instead of using dynamic-programming algorithms to define graph properties, such graph properties are defined using logical formulas. For example, it can be shown that for each formula φ in the monadic second order logic of graphs, there is an algorithm that takes a positive integer k as input and determines whether every graph of treewidth at most k satisfies φ [7]. The drawback with this approach is that the running time of the deduction algorithm is governed by a function that grows as a tower of exponentials on the quantifier depth of the formula φ.

It is worth noting that several interesting graph theoretic properties have dynamic programming algorithms that produce local witnesses of size $k^{O(1)}$ when processing tree decompositions of width at most k, while at the same time, require large quantifier depth to be expressed in MSO logic [9]. For such properties, while the time complexity implied by the logic approach is upper bounded by a tower of exponentials on k, the time complexity implied by the dynamic-programming approach is upper bounded double exponential in $k^{O(1)}$.

2 Basic Definitions

Basic Notation. We let $\mathbb{N} \doteq \{0, 1, \dots\}$ denote the set of natural numbers and $\mathbb{N}_+ = \mathbb{N}\backslash\{0\}$ denote the set of positive natural numbers. Given a number $n \in \mathbb{N}$, we let $[n] \doteq \{1, \dots, n\}$. In particular, $[0] = \emptyset$. Given a set S, the set of finite subsets of S is denoted by $\mathcal{P}_{\text{fin}}(S)$, the set of all subsets of S is denoted by $\mathcal{P}(S)$.

Graphs. In this work, a *graph* is a triple $G = (V, E, \rho)$ where $V \subseteq \mathbb{N}$ is a *finite* set of *vertices*, $E \subseteq \mathbb{N}$ is a finite set *edges*, and $\rho \subseteq E \times V$ is an *incidence relation* with the property that each edge is incident to exactly two vertices. For each edge $e \in E$, we let $\text{endpts}(e) = \{v \in V \ : \ (e, v) \in \rho\}$ be the *endpoints* of e. In what follows, we may write $V(G)$, $E(G)$ and ρ_G to denote the sets V, E and ρ respectively. We let $|G|= |V(G)|+|E(G)|$ be the *size* of G. We let $\textsc{Graphs}$ denote the set of all graphs. For us, the *emtpy graph* is the graph $(\emptyset, \emptyset, \emptyset)$ with no vertices and no edges.

Graph Isomorphisms. An *isomorphism* from a graph G to a graph H is a pair $\varphi = (\phi, \nu)$ where $\phi : V(G) \to V(H)$ is a bijection from the vertices of G to the vertices of H and $\nu : E(G) \to E(H)$ is a bijection from the edges of G to the edges of H with the property that for each vertex $v \in V(G)$ and each edge $e \in E(G)$, $(v, e) \in \rho_G$ if and only if $(\phi(v), \nu(e)) \in \rho_H$. If such a bijection exists, we say that G and H are *isomorphic*, and denote this fact by $G \sim H$.

Graph Properties. A *graph property* is any subset $\mathbb{P} \subseteq \textsc{Graphs}$ closed under isomorphisms. That is to say, for each two isomorphic graphs G and H in $\textsc{Graphs}$, $G \in \mathbb{P}$ if and only if $H \in \mathbb{P}$. Note that the sets $\emptyset$ and $\textsc{Graphs}$ are graph properties. Given a set S of graphs, the *isomorphism closure* of S is defined as the set $\mathsf{ISO}(S) = \{G \in \textsc{Graphs} \ : \ \exists H \in S, G \sim H\}$.

Ranked Alphabet. A *ranked alphabet* is a finite set Σ together with function $\mathfrak{a} : \Sigma \to \mathbb{N}$, which intuitively specifies the arity of each symbol in Σ. A term over Σ is a pair $\tau = (T, \lambda)$ where T is a rooted tree and $\lambda : \mathsf{Nodes}(T) \to \Sigma$ is a function that labels each node p in $\mathsf{Nodes}(T)$ with a symbol from Σ in such a way that if $\lambda(p)$ is a symbol of arity r, then p has r children $p_1, \dots, p_r$. In particular, leaf nodes are labeled with symbols of arity 0. We assume that the children of p are ordered from left to right, so it makes sense to speak about th i-th child of a node. We may write $\mathsf{Nodes}(\tau)$ to refer to $\mathsf{Nodes}(T)$. We write $|\tau|$ to denote $|\mathsf{Nodes}(T)|$. The height of τ is defined as the height of T. We denote by $\text{Terms}(\Sigma)$ the set of all terms over Σ.

Terms. If $\tau_1 = (T_1, \lambda_1), ..., \tau_r = (T_r, \lambda_r)$ are terms in Terms(Σ), and $a \in \Sigma$ is a symbol of arity r, then we let $a(\tau_1, ..., \tau_r)$ denote the term $\tau = (T, \lambda)$ where $\mathsf{Nodes}(T) = \{u\} \cup \mathsf{Nodes}(T_1) \cup \ldots \cup \mathsf{Nodes}(T_r)$ for some fresh node u, $\mathsf{root}(T) = u$, $\lambda(u) = a$, and $\lambda|_{\mathsf{Nodes}(T_j)} = \lambda_j$ for each $j \in [r]$.

3 Instructive Dynamic Programming Cores

In this work, we represent graphs of treewidth at most k using k-instructive tree decompositions (k-ITDs) [9]. For each $k \in \mathbb{N}$, consider the following alphabet

$$\Sigma_k = \Big\{\texttt{Leaf},\ \texttt{IntroVertex}\{u\}, \texttt{IntroEdge}\{u, v\}, \\ \texttt{ForgetVertex}\{u\},\ \texttt{Join}\ :\ u, v \in [k+1], u \neq v\Big\}, \tag{1}$$

where $\texttt{Leaf}$ is a symbol of arity 0, $\texttt{IntroVertex}\{u\}$, $\texttt{ForgetVertex}\{u\}$ and $\texttt{IntroEdge}\{u, v\}$ are symbols of arity 1, and $\texttt{Join}$ is a symbol of arity 2. We call Σ_k the *k-instructive alphabet.* Intuitively, elements of Σ_k represent instructions that can be used for the construction of a graph of treewidth at most k, together with a set $\mathfrak{b} \subseteq [k+1]$ of *active labels*, where each active label labels exacly one vertex of the graph.

1. In the base case, the instruction $\texttt{Leaf}$ creates an empty graph with an empty set of active labels.
2. Now, let G be a graph with set of active labels $\mathfrak{b}$.
   a. For each $u \in [k+1] \backslash \mathfrak{b}$, the instruction $\texttt{IntroVertex}\{u\}$ adds a new vertex to G, labels this vertex with u, and adds u to $\mathfrak{b}$.
   b. For each $u \in \mathfrak{b}$, the instruction $\texttt{ForgetVertex}\{u\}$ erases the label from the current vertex labeled with u, and removes u from $\mathfrak{b}$. The intuition is that the label u is now free and may be used later in the creation of another vertex.
   c. For each $u, v \in \mathfrak{b}$, the instruction $\texttt{IntroEdge}\{u, v\}$ introduces a new edge between the current vertex labeled with u and the current vertex labeled with v. We note that multiedges are allowed in our graphs.
3. Finally, if G and G' are two graphs, each having $\mathfrak{b}$ as the set of active labels, then the instruction $\texttt{Join}$ creates a new graph by identifying, for each $u \in \mathfrak{b}$, the vertex of G labeled with u with the vertex of G' labeled with u.

Such a construction process can be formalized using a term τ over Σ_k, which specifies an inductive construction from the leaves towards the root. More specifically, leaves are labeled with the $\texttt{Leaf}$ instruction, nodes with a single child are labeled with an instruction of type $\texttt{IntroVertex}\{u\}$, $\texttt{ForgetVertex}\{u\}$, or $\texttt{IntroEdge}\{u, v\}$, and nodes with two children are labeled with the $\texttt{Join}$ instruction. We let $\mathcal{G}(\tau)$ be the graph associated with the root of τ. We let $B(\tau)$ be the set of active labels after processing all operations in τ, and let $\theta[\tau] : B(\tau) \to V(\mathcal{G}(\tau))$ be the map that sends each label in $B(\tau)$ to its corresponding vertex in $\mathcal{G}(\tau)$.

We note that not all terms over Σ_k give rise to legal graphs. For instance, if the set $\mathfrak{b}$ does not contain a label u then the instruction $\texttt{ForgetVertex}\{u\}$ is not well defined. Similarly, if u is already in $\mathfrak{b}$, then the instruction $\texttt{IntroVertex}\{u\}$ is not well defined. In order to specify the set of all terms over Σ_k that do correspond to graphs, we may use a tree automaton A_k. More specifically, we let $A_k = (\Sigma_k, Q_k, F_k, \Delta_k)$ be the tree automaton where $Q_k = F_k = \mathcal{P}([k+1])$, and

$$\begin{aligned}\Delta_k &= \{\texttt{Leaf} \to \emptyset\}\\ &\cup \{\texttt{IntroVertex}\{u\}(\mathfrak{b}) \to \mathfrak{b} \cup \{u\} \mid u \notin [k+1]\backslash\mathfrak{b}\}\\ &\cup \{\texttt{ForgetVertex}\{u\}(\mathfrak{b}) \to \mathfrak{b}\backslash\{u\} \mid u \in \mathfrak{b}\}\\ &\cup \{\texttt{IntroEdge}\{u,v\}(\mathfrak{b}) \to \mathfrak{b} \mid u,v \in \mathfrak{b}, u \neq v\}\\ &\cup \{\texttt{Join}(\mathfrak{b},\mathfrak{b}') \to \mathfrak{b} \mid \mathfrak{b} = \mathfrak{b}'\}.\end{aligned}$$

We let $\mathsf{ITD}_k = \mathcal{L}(A_k)$ where $\mathcal{L}(A_k)$ is the set of terms accepted by A_k. The terms in ITD_k are called *k-instructive decompositions.* It turns out that graphs that can be represented by k-instructive tree decompositions are precisely the graphs of treewidth at most k.

▶ **Lemma 1** ([9]). *Let $G \in$ Graphs and $k \in \mathbb{N}$. Then G has treewidth at most k if and only if there exists a k-instructive tree decomposition τ such that $\mathcal{G}(\tau) \simeq G$.*

Dynamic programming algorithms operating on tree decompositions can be formalized using the notion of an *instructive dynamic programming core* (instructive DP-core), as defined below.

▶ **Definition 2** (Instructive DP-Cores). *An* instructive dynamic programming core *is a sequence of 6-tuples $\mathsf{D} = \{(\Sigma_k, \mathcal{W}_k, \texttt{Final}_k, \Delta_k, \texttt{Clean}_k, \texttt{Inv}_k)\}_{k\in\mathbb{N}}$ where for each $k \in \mathbb{N}$,*

1. *Σ_k is the k-instructive alphabet;*
2. *$\mathcal{W}_k \subseteq \{0,1\}^*$ is a decidable subset of $\{0,1\}^*$;*
3. *$\texttt{Final}_k : \mathcal{W}_k \to \{0,1\}$ is a function;*
4. *Δ_k is a set containing*
   - *A* finite *subset $\texttt{Leaf} \subseteq \mathcal{W}$.*
   - *A function $\texttt{IntroVertex}\{u\} : \mathcal{W} \to \mathcal{P}_{\text{fin}}(\mathcal{W})$ for each $u \in [k+1]$.*
   - *A function $\texttt{ForgetVertex}\{u\} : \mathcal{W} \to \mathcal{P}_{\text{fin}}(\mathcal{W})$ for each $u \in [k+1]$.*
   - *A function $\texttt{IntroEdge}\{u,v\} : \mathcal{W} \to \mathcal{P}_{\text{fin}}(\mathcal{W})$ for each $\{u,v\} \in \mathcal{P}([k+1],2)$.*
   - *A function $\texttt{Join} : \mathcal{W} \times \mathcal{W} \to \mathcal{P}_{\text{fin}}(\mathcal{W})$.*
5. *$\texttt{Clean}_k : \mathcal{P}_{\text{fin}}(\mathcal{W}) \to \mathcal{P}_{\text{fin}}(\mathcal{W})$ is a function;*
6. *$\texttt{Inv}_k : \mathcal{P}_{\text{fin}}(\mathcal{W}) \to \{0,1\}^*$ is a function.*

For each $k \in \mathbb{N}$, we let $\mathsf{D}[k] = (\Sigma_k, \mathcal{W}_k, \texttt{Final}_k, \Delta_k, \texttt{Clean}_k, \texttt{Inv}_k)$ denote the k-th tuple of D. We may write $\mathsf{D}[k].\Sigma$ to denote the set Σ_k, $\mathsf{D}[k].\mathcal{W}$ to denote the set $\mathcal{W}_k$, and so on. Intuitively, for each k, $\mathsf{D}[k]$ is a description of a dynamic programming algorithm that operates on k-instructive tree decompositions. Such an algorithm processes a k-instructive tree decomposition τ from the leaves towards the root, and assigns a set of local witnesses to each node of τ, depending on which instruction labels the node and on the sets assigned to the children of the node. Some dynamic programming algorithms use a function that removes redundant local witnesses from the set of local witnesses constructed at each node. In our framework, this is formalized by the function $\texttt{Clean}_k$. In this work, we assume that $\texttt{Clean}_k$ is the identity function. Finally, the function $\texttt{Inv}_k$ is used whenever we want to use dynamic programming algorithms to compute graph invariants. In this work, we will not be concerned with the computation of invariants, and therefore, we assume that $\texttt{Inv}_k$ is the Boolean function that assigns 1 to a set of local witnesses if and only if it contains some final witness. This process is formalized by the notion of *dynamization*, which we define below.

▶ **Definition 3** (Dynamization). *Let $k \in \mathbb{N}$ and D be an instructive DP-core. The k-dynamization of D is the function $\Gamma[\mathsf{D},k] : \mathsf{ITD}_k \to \mathcal{P}_{\text{fin}}(\mathsf{D}[k].\mathcal{W})$ inductively defined as follows for each $\tau \in \mathsf{ITD}_k$.*

1. *If $\tau = \texttt{Leaf}$, then $\Gamma[\mathsf{D},k](\tau) = \mathsf{D}[k].\texttt{Leaf}$.*

2. *If* $\tau = \texttt{IntroVertex}\{u\}(\sigma)$, *then* $\Gamma[\mathsf{D}, k](\tau) = \mathsf{D}[k].\texttt{Clean}(\texttt{IntroVertex}\{u\}(\Gamma[\mathsf{D}, k](\sigma)))$.
3. *If* $\tau = \texttt{ForgetVertex}\{u\}(\sigma)$, *then* $\Gamma[\mathsf{D}, k](\tau) = \mathsf{D}[k].\texttt{Clean}(\texttt{ForgetVertex}\{u\}(\Gamma[\mathsf{D}, k](\sigma)))$.
4. *If* $\tau = \texttt{IntroEdge}\{u, v\}(\sigma)$, *then* $\Gamma[\mathsf{D}, k](\tau) = \mathsf{D}[k].\texttt{Clean}(\texttt{IntroEdge}\{u, v\}(\Gamma[\mathsf{D}, k](\sigma)))$.
5. *If* $\tau = \texttt{Join}(\sigma_1, \sigma_2)$, *then* $\Gamma[\mathsf{D}, k](\tau) = \mathsf{D}[k].\texttt{Clean}(\texttt{Join}(\Gamma[\mathsf{D}, k](\sigma_1), \Gamma[\mathsf{D}, k](\sigma_2)))$.

We say that $\mathsf{D}[k]$ accepts τ if $\Gamma[\mathsf{D}, k](\tau)$ has a final local witness, i.e. a local witness w with the property that $\mathsf{D}[k].\texttt{Final}(\mathsf{w}) = 1$. We let $\mathbb{G}(\mathsf{D}[k]) = \mathsf{ISO}(\{\mathcal{G}(\tau) : \tau \text{ is accepted by } \mathsf{D}[k]\})$ be the isomorphism closure of the set of graphs associated with terms accepted by $\mathsf{D}[k]$. We note that $\mathbb{G}(\mathsf{D}[k])$ is a graph property, and that all graphs in $\mathbb{G}(\mathsf{D}[k])$ have treewidth at most k. We let $\mathbb{G}(\mathsf{D}) = \bigcup_{k \in \mathbb{N}} \mathbb{G}(\mathsf{D}[k])$ be the graph property defined by D.

▶ **Definition 4** (Coherency). *Let* D *be an instructive DP-core. We say that* D *is coherent if for each* $k, k' \in \mathbb{N}$, *each* $\tau \in \mathsf{ITD}_k$, *and each* $\tau' \in \mathsf{ITD}_{k'}$ *if* $\mathcal{G}(\tau) \simeq \mathcal{G}(\tau')$ *then* $\mathsf{D}[k]$ *accepts* τ *if and only if* $\mathsf{D}[k']$ *accepts* τ'.

Let D be a coherent instructive DP-core, and $k \in \mathbb{N}$. A (k, D)-state is a pair of the form $(\mathfrak{b}, S)$ where $\mathfrak{b} \subseteq [k+1]$ and $S \subseteq \mathsf{D}.\mathcal{W}$. Such a state is said to be (k, D)-*inconsistent*, if S has no final local witness. The initial (k, D)-state is the pair $(\emptyset, \mathsf{D}.\texttt{Leaf})$.

▶ **Definition 5** ((k, D)-Refutation). *Let* D *be an instructive DP-core. A* (k, D)-*refutation is a sequence of pairs* $R \equiv (\mathfrak{b}_0, S_0)(\mathfrak{b}_1, S_1) \dots (\mathfrak{b}_m, S_m)$ *satisfying the following conditions.*

1. $(\mathfrak{b}_0, S_0) = (\emptyset, \mathsf{D}[k].\texttt{Leaf})$.
2. $(\mathfrak{b}_m, S_m)$ *is* (k, D)-*inconsistent.*
3. *For each* $i \in [m]$, *there is some* $j \in [i]$, *such that* $(\mathfrak{b}_i, S_i)$ *is equal to one of the following pairs.*
   a. $(\mathfrak{b}_j \cup \{u\}, \texttt{IntroVertex}\{u\}(S_j))$ *with* $u \notin \mathfrak{b}_j$.
   b. $(\mathfrak{b}_j \setminus \{u\}, \texttt{ForgetVertex}\{u\}(S_j))$ *with* $u \in \mathfrak{b}_j$.
   c. $(\mathfrak{b}_j, \texttt{IntroEdge}\{u, v\}(S_j))$ *with* $u, v \in \mathfrak{b}_j$.
   d. $(\mathfrak{b}_j, \texttt{Join}(S_j, S_l))$ *with* $l \in [i]$.

Intuitively, a (k, D)-refutation is a certificate that some inconsistent (k, D)-state is reachable from the initial (k, D)-state. It turns out that if D is a coherent instructive DP-core, then constructing a (k, D)-refutation is equivalent to showing that $\mathbb{G}(\mathsf{D})$ does not contain all graphs of treewidth at most k.

▶ **Theorem 6** ([9]). *Let* D *be a coherent instructive DP-core. Then there is a* (k, D)-*refutation if and only if some graph of treewidth at most* k *does not belong to the graph property* $\mathbb{G}(\mathsf{D})$.

4 Example: An Instructive DP-Core for Chromatic Number at Most r

Let S be a finite set, and $r \in \mathbb{N}$. An r-partition of S is a partition of S with at most r cells. Let G be a graph. We say that G is r-colorable if there is an r-partition of $V(G)$ such that for each edge $e \in E(G)$, the endpoints of e belong to distinct cells. Let $\textsc{Colorable}_r$ be the graph property consisting of all graphs that are r-colorable. In this section, we specify a DP-core C-$\textsc{Colorable}_r$ for the property $\textsc{Colorable}_r$. We start by defining the notion of a C-$\textsc{Colorable}_r[k]$ local witness, for each $k \in \mathbb{N}$.

▶ **Definition 7.** *Let* $k \in \mathbb{N}$. *A* C-$\textsc{Colorable}_r[k]$ *local witness is any* r-*partition of a subset of* $[k+1]$.

▶ **Definition 8.** *Let $r \in \mathbb{N}$. We let C-$\textsc{Colorable}_r$ be the instructive DP-core D specified below. For each $k \in \mathbb{N}$, we define C-$\textsc{Colorable}_r[k] = \mathsf{D}[k]$. We let $u, v \in [k+1]$, w and w' be C-$\textsc{Colorable}_r[k]$ local witnesses, and S be a set of such local witnesses.*

1. $\mathsf{D}[k].\mathcal{W} = \{\mathsf{w} : \mathsf{w}$ *is a* C-$\textsc{Colorable}_r[k]$ *local witness* $\}$.
2. $\mathsf{D}[k].\mathtt{Leaf} = \{\emptyset\}$.
3. $\mathsf{D}[k].\mathtt{IntroVertex}\{u\}(\mathsf{w}) = \begin{cases} \{(\mathsf{w} \setminus \{p\}) \cup \{p \cup \{u\}\} : p \in \mathsf{w}\} & \text{if } |\mathsf{w}| = r, \\ \{\mathsf{w} \cup \{\{u\}\}\} \cup \{(\mathsf{w} \setminus \{p\}) \cup \{p \cup \{u\}\} : p \in \mathsf{w}\} & \text{if } |\mathsf{w}| < r. \end{cases}$
4. $\mathsf{D}[k].\mathtt{ForgetVertex}\{u\}(\mathsf{w}) = \{p \setminus \{u\} \ : \ p \in \mathsf{w}\} \setminus \{\emptyset\}$ [1].
5. $\mathsf{D}[k].\mathtt{IntroEdge}\{u, v\}(\mathsf{w}) = \begin{cases} \{\mathsf{w}\} & \text{if } u \text{ and } v \text{ are not in a same cell,} \\ \emptyset & \text{Otherwise.} \end{cases}$
6. $\mathsf{D}[k].\mathtt{Join}(\mathsf{w}, \mathsf{w}') = \begin{cases} \{\mathsf{w}\} & \text{if } \mathsf{w} = \mathsf{w}', \\ \emptyset & \text{Otherwise.} \end{cases}$
7. $\mathsf{D}[k].\mathtt{Final}(\mathsf{w}) = 1$ *for every* $\mathsf{w} \in \mathsf{D}.\mathcal{W}$.
8. $\mathsf{D}[k].\mathtt{Clean}(S) = S$ *for every* $S \subseteq \mathsf{D}.\mathcal{W}$.
9. $\mathsf{D}[k].\mathtt{Inv}(S) = \begin{cases} 1 & \text{if } S \text{ has a final witness,} \\ 0 & \text{Otherwise.} \end{cases}$

Next, we define a predicate relating k-instructive tree decompositions with local witnesses.

▶ **Definition 9.** *We let P-$\textsc{Colorable}_r[k]$ be the predicate that is true on a pair $(\tau, \mathsf{w}) \in \mathsf{ITD}_k \times$ C-$\textsc{Colorable}_r[k].\mathcal{W}$ if and only if the following conditions are satisfied.*

1. $\bigcup_{c \in \mathsf{w}} c = dom(\theta[\tau])$.
2. *There is an r-partition α of $V(\mathcal{G}(\tau))$ such that for every $u, v \in B(\tau)$, $\theta[\tau](u)$ and $\theta[\tau](v)$ belong to the same cell in α if and only if u and v belong to the same cell in w.*

▶ **Proposition 10.** *For each $\tau \in \mathsf{ITD}_k$, a local witness w belongs to $\Gamma[$C-$\textsc{Colorable}_r, k](\tau)$ if and only if P-$\textsc{Colorable}_r[k](\tau, \mathsf{w}) = 1$.*

The next corollary states that the predicate P-$\textsc{Colorable}[k, r]$ characterizes those pairs (τ, w) for which w is a witness in $\Gamma[$C-$\textsc{Colorable}_r[k], \tau]$. Below, $\mathsf{Accepted}($C-$\textsc{Colorable}_r[k])$ is the set of k-instructive tree decompositions accepted by C-$\textsc{Colorable}_r[k]$.

▶ **Corollary 11.** *Let τ be a k-instructive tree decomposition. Then $\mathcal{G}(\tau)$ is r-colorable if and only if $\tau \in \mathsf{Accepted}($C-$\textsc{Colorable}_r[k])$.*

Since a C-$\textsc{Colorable}_r[k]$ local witness is an r-partition of a subset of $[k+1]$, we can represent such a partition using $O(k \cdot \log r)$ bits.

▶ **Observation 12.** *C-$\textsc{Colorable}_r[k]$ has bit-length $O(k \cdot \log r)$.*

[1] There is at most one cell containing u. If this cell is a singleton, the whole cell is deleted from w.

5 Width Based ATP with Symmetry Breaking

In this section, we introduce our main technical result. More specifically, we introduce a width-based automated deduction algorithm endowed with a symmetry breaking procedure that allows us to remove redundant states during the search for counter-example for a given conjecture. At the core of our technique, lies the notion of a *witness action*. Intuitively, functions that satisfy the axioms of a witness action can be used to define permuted versions of local witnesses generated by a DP-core. This allows us to define the notion of canonical form of a state generated during the search process. Instead of keeping track of all inferred states, we only keep their canonical forms. This leads to a significant reduction of the search space because states with the same canonical form are identified. Our main theorem (Theorem 19) states that this process preserves provability.

Let $\mathcal{F}_k = \{f : \mathfrak{b} \to [k+1] \mid \mathfrak{b} \subseteq [k+1]\}$ be the set of all injective functions $f : \mathfrak{b} \to [k+1]$ from some subset $\mathfrak{b} \subseteq [k+1]$ to $[k+1]$. We call the elements of $\mathcal{F}$ *relabeling functions*. Given such a function $f \in \mathcal{F}_k$, and a subset $\mathfrak{b} \subseteq [k+1]$, we let $f(\mathfrak{b}) = \{f(u) \ : \ u \in \mathfrak{b}\}$ be the image of $\mathfrak{b}$ under f. Next, we introduce the notion of a *witness action* for a DP-core (Definition 13). Actions will be used later to define the notion of a canonical form for a $\mathsf{D}[k]$-state.

▶ **Definition 13** (Witness Action). *Let* D *be a DP-core and* $k \in \mathbb{N}$*. We say that a function* $\rho_{\mathsf{D}}^k : \mathcal{F}_k \times \mathsf{D}[k].\mathcal{W} \to \mathsf{D}[k].\mathcal{W}$ *is an action for* $\mathsf{D}[k]$ *if the following conditions are satisfied for each* $f \in \mathcal{F}_k$*, and each* $\mathsf{w} \in \mathsf{D}[k].\mathcal{W}$.

1. ρ_{D}^k *preserves acceptance:* $\mathsf{w} \in \mathsf{Accepted}(\mathsf{D}[k])$ *if and only if* $\rho_{\mathsf{D}}^k(f, \mathsf{w}) \in \mathsf{Accepted}(\mathsf{D}[k])$.
2. $\rho_{\mathsf{D}}^k(f^{-1}, \rho_{\mathsf{D}}^k(f, \mathsf{w})) = \mathsf{w}$.
3. $\rho_{\mathsf{D}}^k(f \circ f', \mathsf{w}) = \rho_{\mathsf{D}}^k(f, \rho_{\mathsf{D}}^k(f', \mathsf{w}))$.
4. $\rho_{\mathsf{D}}^k(f, \mathtt{IntroVertex}\{u\}(\mathsf{w})) = \mathtt{IntroVertex}\{f(u)\}(\rho_{\mathsf{D}}^k(f, \mathsf{w}))$.
5. $\rho_{\mathsf{D}}^k(f, \mathtt{ForgetVertex}\{u\}(\mathsf{w})) = \mathtt{ForgetVertex}\{f(u)\}(\rho_{\mathsf{D}}^k(f, \mathsf{w}))$.
6. $\rho_{\mathsf{D}}^k(f, \mathtt{IntroEdge}\{u, v\}(\mathsf{w})) = \mathtt{IntroEdge}\{f(u), f(v)\}(\rho_{\mathsf{D}}^k(f, \mathsf{w}))$.
7. $\rho_{\mathsf{D}}^k(f, \mathtt{Join}(\mathsf{w}_1, \mathsf{w}_2)) = \mathtt{Join}(\rho_{\mathsf{D}}^k(f, \mathsf{w}_1), \rho_{\mathsf{D}}^k(f, \mathsf{w}_2))$.

We extend Definition 13 to subsets of witnesses by setting

$$\rho_{\mathsf{D}}^k(f, S) = \{\rho_{\mathsf{D}}^k(f, \mathsf{w}) \ : \ \mathsf{w} \in S\}$$

for each $S \subseteq \mathsf{D}[k].\mathcal{W}$. Next, given a DP-core D, we will define a notion of canonization for a pair $(\mathfrak{b}, S)$ where $\mathfrak{b} \subseteq [k+1]$ and $S \subseteq \mathcal{P}_{\text{fin}}(\mathsf{D}.\mathcal{W})$.

Let U be an ordered set of elements, and $X \subseteq U$. We let $\text{vec}(X)$ be the vector obtained by ordering the elements of X from the smallest value to the largest value. For instance, if $X = \{2, 3, 5\}$, then $\text{vec}(X) = (2, 3, 5)$. Given two such subsets $X, X' \subseteq U$, we say that $X < X'$ if $\text{vec}(X)$ is lexicographically smaller than $\text{vec}(X)$.

Let D be a DP-core and fix an arbitrary order for the set $\mathsf{D}[k].\mathcal{W}$. For instance, this order can be simply the lexicographic order on strings. We say that a pair $(\mathfrak{b}, S)$ is smaller than $(\mathfrak{b}', S')$ if the pair $(\text{vec}(\mathfrak{b}), \text{vec}(S))$ is lexicographically smaller than the pair $(\text{vec}(\mathfrak{b}'), \text{vec}(S'))$.

▶ **Definition 14** (Canonical Pair). *Let* $\mathfrak{b} \subseteq [k+1]$*,* $S \subseteq \mathsf{D}[k].\mathcal{W}$*, and*

$$\text{CAN}_{\mathsf{D}}^k(\mathfrak{b}, S) = \min\{(f(\mathfrak{b}), \rho_{\mathsf{D}}^k(f, S)) \mid f \in \mathcal{F}_k, dom(f) = \mathfrak{b}\}.$$

We call the function $f : \mathfrak{b} \to [k+1]$ *where the minimum in the above equation is achieved the* canonical relabeling of $(\mathfrak{b}, S)$.

Intuitively, given a fixed action ρ^k_{D} the canonical form $\mathrm{CAN}^k_{\mathsf{D}}(\mathfrak{b}, S)$ of a pair $(\mathfrak{b}, S)$ is the lexicographically smallest pair obtained by relabeling the elements of $\mathfrak{b}$ and each string $\mathsf{w} \in S$ according to some relabeling function $f : \mathfrak{b} \to [k+1]$. The canonical relabeling of $(\mathfrak{b}, q)$ is the unique function $f : \mathfrak{b} \to [k+1]$ with the property that $(f(\mathfrak{b}), \rho^k_{\mathsf{D}}(f, S)) = \mathrm{CAN}^k_{\mathsf{D}}(\mathfrak{b}, S)$.

Our next step is to define the notion of a relabeled refutation. Intuitively, relabelings will be used to produce refutations where all states are in canonical form.

▶ **Definition 15** (F-Relabeled Refutation). *Let $F = (f_1, \dots, f_m)$ be a sequence of relabeling functions in $\mathcal{F}_k$ and D be a DP-core. An F-relabeled (k, D)-refutation is a sequence of pairs*

$$R \equiv (\mathfrak{b}_0, S_0)(\mathfrak{b}_1, S_1) \dots (\mathfrak{b}_m, S_m)$$

satisfying the following conditions:

1. $(\mathfrak{b}_0, S_0) = (\emptyset, \mathsf{D}[k].\mathtt{Leaf})$.
2. $(\mathfrak{b}_m, S_m)$ *is (k, D)-inconsistent, i.e., S_m has no final local witness.*
3. *For each $i \in [m]$, there is some $j \in [i]$, such that $(\mathfrak{b}_i, S_i)$ is equal to one of the following pairs.*
   a. $(f_i(\mathfrak{b}_j \cup \{u\}), \rho^k_{\mathsf{D}}(f_i, \mathtt{IntroVertex}\{u\}(S_j)))$ *for some $u \notin \mathfrak{b}_j$.*
   b. $(f_i(\mathfrak{b}_j \setminus \{u\}), \rho^k_{\mathsf{D}}(f_i, \mathtt{ForgetVertex}\{u\}(S_j)))$ *for some $u \in \mathfrak{b}_j$.*
   c. $(f_i(\mathfrak{b}_j), \rho^k_{\mathsf{D}}(f_i, \mathtt{IntroEdge}\{u, v\}(S_j)))$ *for some $u, v \in \mathfrak{b}_j$.*
   d. $(f_i(\mathfrak{b}_j), \rho^k_{\mathsf{D}}(f_i, \mathtt{Join}(S_j, \rho^k_{\mathsf{D}}(f, S_l))))$ *for some $l \in [i]$ and some $f : \mathfrak{b}_l \to [k+1]$ with $f(\mathfrak{b}_l) = f_i(\mathfrak{b}_j)$.*

If the sequence R in Definition 15 satisfies Conditions 1 and 3, but not Condition 2, then we say that R is a semi F-relabeled (k, D)-refutation.

Below, given an instructive tree decomposition τ, we let $\mathrm{Sub}(\tau)$ denote the set of all subterms of τ.

▶ **Lemma 16.** *Let D be a DP-core, $F = (f_1, \dots, f_m)$ be a sequence of k-relabeling functions. Let $R = (\mathfrak{b}_0, S_0)(\mathfrak{b}_1, S_1) \dots (\mathfrak{b}_m, S_m)$ be a semi F-relabeled (k, D)-refutation, and $g : \mathfrak{b}_m \to [k+1]$ be a k-relabeling. Then, there is a k-instructive tree decomposition $\tau_R \in \mathsf{ITD}_k$ and a function $T : \mathrm{Sub}(\tau_R) \to \mathcal{P}_{\mathrm{fin}}(\mathsf{D}.\mathcal{W})$ such that the following conditions are satisfied for each subterm τ of τ_R.*

1. *If $\tau = \tau_R$, then $T(\tau) = \rho^k_{\mathsf{D}}(g, S_m)$.*
2. *if $\tau = \mathtt{Leaf}$, then $T(\tau) = \mathsf{D}[k].\mathtt{Leaf}$.*
3. *if $\tau = \mathtt{IntroVertex}\{u\}(\tau_1)$ for some subterm τ_1, then $T(\tau) = \mathtt{IntroVertex}\{u\}(T(\tau_1))$.*
4. *if $\tau = \mathtt{ForgetVertex}\{u\}(\tau_1)$ for some subterm τ_1, then $T(\tau) = \mathtt{ForgetVertex}\{u\}(T(\tau_1))$*
5. *if $\tau = \mathtt{IntroEdge}\{u, v\}(\tau_1)$ for some subterm τ_1, then $T(\tau) = \mathtt{IntroEdge}\{u, v\}(T(\tau_1))$.*
6. *if $\tau = \mathtt{Join}(\tau_1, \tau_2)$ for some subterms τ_1 and τ_2, then $T(\tau) = \mathtt{Join}(T(\tau_1), T(\tau_2))$.*

Proof. The proof of Lemma 16 follows by induction on m. In the base case, we have that $R = (\mathfrak{b}_0, S_0)$. In this case, by setting $\tau_R = \mathtt{Leaf}$ and $T(\tau_R) = \mathsf{D}[k].\mathtt{Leaf}$, the conditions 1-6 of Lemma 16 are satisfied.

Assume that the lemma holds for all $n < m$. We show that the lemma holds for $n = m$. Below, for each $i \in [n]$, we let $R_i = (\mathfrak{b}_0, S_0)(\mathfrak{b}_1, S_1) \dots (\mathfrak{b}_i, S_i)$. By the induction hypothesis, for each such i, and each $h : \mathfrak{b}_i \to [k+1]$, there is a k-instructional tree decomposition τ^h_i and a function $T^h_i : \mathrm{Sub}(\tau') \to \mathcal{P}_{\mathrm{fin}}(\mathsf{D}.\mathcal{W})$ such that the conditions 1-6 of Lemma 16 are satisfied. There are four cases to be analyzed.

1. In the first case $(\mathfrak{b}_m, S_m) = (f_m(\mathfrak{b}_i \cup \{u\}), \rho^k_{\mathsf{D}}(f_m, \texttt{IntroVertex}\{u\}(S_i)))$ for some $i < m$. Let $\tau = \texttt{IntroVertex}\{g \circ f_m(u)\}(\tau_i^{g\circ f_m})$, and $T : \mathrm{Sub}(\tau) \to \mathcal{P}_{\mathrm{fin}}(\mathsf{D}[k].\mathcal{W})$ be such that $T(\tau) = \texttt{IntroVertex}\{g \circ f_m(u)\}(\rho^k_{\mathsf{D}}(g \circ f_m, S_i))$ and $T|_{\mathrm{Sub}(\tau_i^{g\circ f_m})} = T_i^{g\circ f_m}$. It should be clear that conditions 2-6 are satisfied. Finally, to show that Condition 1 is satisfied, we note that

$$\begin{aligned}&\texttt{IntroVertex}\{g \circ f_m(u)\}(\rho^k_{\mathsf{D}}(g \circ f_m, S_i))\\ &\ = \rho^k_{\mathsf{D}}(g \circ f_m, \texttt{IntroVertex}\{u\}(S_i))\\ &\ = \rho^k_{\mathsf{D}}(g, \rho^k_{\mathsf{D}}(f_m, \texttt{IntroVertex}\{u\}(S_i)))\\ &\ = \rho^k_{\mathsf{D}}(g, S_m).\end{aligned}$$

where the first equality follows from Definition 13.4, the second equality follows from Definition 13.3 and the third equality follows from the fact that $S_m = \rho^k_{\mathsf{D}}(f_m, \texttt{IntroVertex}\{u\}(S_i))$.

2. In the second case $(\mathfrak{b}_m, S_m) = (f_m(\mathfrak{b}_i \setminus \{u\}), \rho^k_{\mathsf{D}}(f_m, \texttt{ForgetVertex}\{u\}(S_i)))$ for some $i < m$. Let $\tau = \texttt{ForgetVertex}\{g \circ f_m(u)\}(\tau_i^{g\circ f_m})$, and $T : \mathrm{Sub}(\tau) \to \mathcal{P}_{\mathrm{fin}}(\mathsf{D}[k].\mathcal{W})$ be such that $T(\tau) = \texttt{ForgetVertex}\{g \circ f_m(u)\}(\rho^k_{\mathsf{D}}(g \circ f_m, S_i))$ and $T|_{\mathrm{Sub}(\tau_i^{g\circ f_m})} = T_i^{g\circ f_m}$. It should be clear that conditions 2-6 are satisfied. Finally, to show that Condition 1 is satisfied, we note that

$$\begin{aligned}&\texttt{ForgetVertex}\{g \circ f_m(u)\}(\rho^k_{\mathsf{D}}(g \circ f_m, S_i))\\ &\ = \rho^k_{\mathsf{D}}(g \circ f_m, \texttt{ForgetVertex}\{u\}(S_i))\\ &\ = \rho^k_{\mathsf{D}}(g, \rho^k_{\mathsf{D}}(f_m, \texttt{ForgetVertex}\{u\}(S_i)))\\ &\ = \rho^k_{\mathsf{D}}(g, S_m).\end{aligned}$$

where the first equality follows from Definition 13.5, the second equality follows from Definition 13.3 and the third equality follows from the fact that $S_m = \rho^k_{\mathsf{D}}(f_m, \texttt{ForgetVertex}\{u\}(S_i))$.

3. In the third case $(\mathfrak{b}_m, S_m) = (f_m(\mathfrak{b}_i), \rho^k_{\mathsf{D}}(f_m, \texttt{IntroEdge}\{u, v\}(S_i)))$ for some $i < m$. Let $\tau = \texttt{IntroEdge}\{g \circ f_m(u), g \circ f_m(v)\}(\tau_i^{g\circ f_m})$, and $T : \mathrm{Sub}(\tau) \to \mathcal{P}_{\mathrm{fin}}(\mathsf{D}[k].\mathcal{W})$ be such that $T(\tau) = \texttt{IntroEdge}\{g \circ f_m(u), g \circ f_m(v)\}(\rho^k_{\mathsf{D}}(g \circ f_m, S_i))$ and $T|_{\mathrm{Sub}(\tau_i^{g\circ f_m})} = T_i^{g\circ f_m}$. It should be clear that conditions 2-6 are satisfied. Finally, to show that Condition 1 is satisfied, we note that

$$\begin{aligned}&\texttt{IntroEdge}\{g \circ f_m(u), g \circ f_m(v)\}(\rho^k_{\mathsf{D}}(g \circ f_m, S_i))\\ &\ = \rho^k_{\mathsf{D}}(g \circ f_m, \texttt{IntroEdge}\{u, v\}(S_i))\\ &\ = \rho^k_{\mathsf{D}}(g, \rho^k_{\mathsf{D}}(f_m, \texttt{IntroEdge}\{u, v\}(S_i)))\\ &\ = \rho^k_{\mathsf{D}}(g, S_m).\end{aligned}$$

where the first equality follows from Definition 13.6, the second equality follows from Definition 13.3 and the third equality follows from the fact that $S_m = \rho^k_{\mathsf{D}}(f_m, \texttt{IntroEdge}\{u, v\}(S_i))$.

4. In the fourth case $(\mathfrak{b}_m, S_m) = (f_m(\mathfrak{b}_i), \rho^k_{\mathsf{D}}(f_m, \texttt{Join}(S_i, S_j)))$ for some $j, i < m$. Let $\tau = \texttt{Join}(\tau_i^{g\circ f_m}, \tau_j^{g\circ f_m})$, and $T : \mathrm{Sub}(\tau) \to \mathcal{P}_{\mathrm{fin}}(\mathsf{D}[k].\mathcal{W})$ be such that $T(\tau) = \texttt{Join}(\rho^k_{\mathsf{D}}(g \circ f_m, S_i), \rho^k_{\mathsf{D}}(g \circ f_m, S_j)$ and $T|_{\mathrm{Sub}(\tau_i^{g\circ f_m})} = T_i^{g\circ f_m}$ and $T|_{\mathrm{Sub}(\tau_j^{g\circ f_m})} = T_j^{g\circ f_m}$. It should be clear that conditions 2-6 are satisfied. Finally, to show that Condition 1 is satisfied, we note that

$$\begin{aligned}&\texttt{Join}(\rho^k_{\mathsf{D}}(g \circ f_m, S_i), \rho^k_{\mathsf{D}}(g \circ f_m, S_j))\\ &\ = \rho^k_{\mathsf{D}}(g \circ f_m, \texttt{Join}(S_i, S_j))\\ &\ = \rho^k_{\mathsf{D}}(g, \rho^k_{\mathsf{D}}(f_m, \texttt{Join}(S_i, S_j)))\\ &\ = \rho^k_{\mathsf{D}}(g, S_m).\end{aligned}$$

where the first equality follows from Definition 13.7, the second equality follows from Definition 13.3 and the third equality follows from the fact that $S_m = \rho^k_{\mathsf{D}}(f_m, \texttt{Join}(S_i, S_j))$.

◀

The next theorem states that if D is a coherent DP-core, then from each relabeled (k,D)-refutation, one can extract a k-instructive tree decomposition whose graph does not belong to $\mathbb{G}(\mathsf{D})$.

▶ **Theorem 17.** *Let $F = (f_1,\dots,f_m)$ be a sequence of relabeling functions and D be a coherent DP-core. If there is an F-relabeled (k,D)-refutation, then there exists a k-instructive tree decomposition $\tau \in \mathsf{ITD}_k$ where $\mathcal{G}(\tau) \notin \mathbb{G}(\mathsf{D})$.*

Proof. Let $R = (\mathfrak{b}_0, S_0)(\mathfrak{b}_1, S_1)\dots(\mathfrak{b}_m, S_m)$ be a F-relabeled (k,D)-refutation. By Lemma 16, there is a k-instructive tree decomposition τ and a map $T : \mathrm{Sub}(\tau) \to \mathcal{P}_{\mathrm{fin}}(\mathsf{D}.\mathcal{W})$ such that conditions 1-6 of Lemma 16 are satisfied. Conditions 2-6 imply that the value of T on τ is equal to the value of the dynamization of D on τ. In particular we have that $S_m = \Gamma[\mathsf{D}, k](\tau)$. Since R is a (k,D)-refutation, we have that S_m has no final local witness. Therefore, the definition of acceptance for a DP-core, τ is not accepted by D. Since D is coherent, then $\mathcal{G}(\tau)$ does not belong to $\mathbb{G}(\mathsf{D})$. ◀

Next, we define the notion of a canonized refutation. Intuitively, such a refutation is obtained by canonizing the (k,D)-pairs occurring in a (k,D)-refutation.

▶ **Definition 18** (Canonized Refutation)**.** *Let D be a coherent DP-core, and $k \in \mathbb{N}$, $F = (f_1,\dots,f_m)$ be a sequence of relabelings, and $R \equiv (\mathfrak{b}_0, S_0)(\mathfrak{b}_1, S_1)\dots(\mathfrak{b}_m, S_m)$ be an F-relabeled (k,D)-refutation. We say that R is canonized if for each $i \in \mathbb{N}$, the pair $(\mathfrak{b}_i, S_i)$ is canonical.*

Our main theorem (Theorem 19) states that if D is a coherent DP-core, then the existence of a canonized (k,D)-refutation is equivalent to the existence of a graph of treewidth at most k that does not belong to the property $\mathbb{G}(\mathsf{D})$ defined by D.

▶ **Theorem 19.** *Let D be a coherent DP-core and $k \in \mathbb{N}$. Then there is a canonized (k,D)-refutation if and only if some graph of treewidth at most k does not belong to $\mathbb{G}(\mathsf{D})$.*

Proof. Let G be a graph of treewidth at most k that does not belong to $\mathbb{G}(\mathsf{D})$. Then, by Theorem 6, there is a (k,D)-refutation $R = (\mathfrak{b}_0, S_0)(\mathfrak{b}_1, S_1)\dots(\mathfrak{b}_m, S_m)$. Let $R' = (\mathfrak{b}'_0, S'_0)(\mathfrak{b}'_1, S'_1)\dots(\mathfrak{b}'_m, S'_m)$ be the sequence of pairs where for each $i \in \{0, 1, ..., m\}$, $(\mathfrak{b}'_i, S'_i)$ is the canonical form of $(\mathfrak{b}_i, S_i)$. Then R' is a canonized DP-refutation.

For the converse, suppose there is a canonized (k,D)-refutation R. By Definition 18, R is a F-relabeled refutation for some relabeling sequence F. By Theorem 17, there is a k-instructive tree decomposition τ where $\mathcal{G}(\tau) \notin \mathbb{G}(\mathsf{D})$. ◀

6 Early Pruning

Our second main contribution is an early-pruning procedure that can be applied in the study of conjectures of the form $\mathbb{P}_1 \to \mathbb{P}_2$ where $\mathbb{P}_1$ is closed under subgraphs. This reduces the search space even more because it allows us to avoid the computation of states that do not contribute in the search for a counter-example.

A graph property $\mathbb{P}$ is said to be closed under sub-graphs if whenever a graph G belongs to $\mathbb{P}$, we have that every sub-graph of G also belongs to $\mathbb{P}$. If the graph property $\mathbb{G}(\mathsf{D})$ specified by a given coherent DP-core D is of the form $\mathbb{G}(\mathsf{D}_1) \to \mathbb{G}(\mathsf{D}_2)$ for coherent DP-cores D_1 and D_2, such that $\mathbb{G}(\mathsf{D}_1)$ is closed under subgraphs, then when running our inference algorithm to determine whether some graph of treewidth at most k is not contained in $\mathbb{G}(\mathsf{D})$ we may prune the search earlier. The following simple, but crucial observation is the basis of our specialized search.

▶ **Observation 20.** *Let $\mathbb{P}$ be a graph property closed under subgraphs. Let τ and τ' be k-instructive tree decompositions such that τ is a subterm of τ'. Then, if $\mathcal{G}(\tau) \notin \mathbb{P}$, then $\mathcal{G}(\tau') \notin \mathbb{P}$.*

Proof. Assume that $\mathcal{G}(\tau) \notin \mathbb{P}$. Since τ is a subterm of τ', we have that $\mathcal{G}(\tau)$ is isomorphic to a subgraph of $\mathcal{G}(\tau')$. Suppose $\mathcal{G}(\tau') \in \mathbb{P}$, then since $\mathbb{P}$ is closed under subgraphs, we have that $\mathcal{G}(\tau) \in \mathbb{P}$. This contradicts the assumption that $\mathcal{G}(\tau) \notin \mathbb{P}$. ◀

Observation 20 implies that in order to determine whether there is a graph of treewidth at most k that does not belong to $\mathbb{G}(\mathsf{D})$, instead of searching for inconsistent (k, D)-pairs, we may instead search for inconsistent $(k, \mathsf{D}_1, \mathsf{D}_2)$-*triples.* Such a triple, is a triple of the form $(\mathfrak{b}, S_1, S_1)$ satisfying the following properties:

1. $(\mathfrak{b}, S_1)$ is a (k, D_1)-pair,
2. $(\mathfrak{b}, S_2)$ is a (k, D_2)-pair,
3. S_1 has a final local witness for D_1 but S_2 does not have a final local witness for D_2.

This allows a more efficient search because, since $\mathbb{G}(\mathsf{D}_1)$ is assumed to be closed under subgraphs, as soon as we have reached a $(k, \mathsf{D}_1, \mathsf{D}_2)$-triple $(\mathfrak{b}, S_1, S_2)$ where $(\mathfrak{b}, S_1)$ is an inconsistent (k, D_1)-pair, we know that no triple $(\mathfrak{b}, S_1', S_2')$ derived from $(\mathfrak{b}, S_1, S_2)$ will be inconsistent (because S_1' does not contain a final witness for D_1). Therefore, we do not need to consider $(k, \mathsf{D}_1, \mathsf{D}_2)$-triples derived from $(\mathfrak{b}, S_1, S_2)$. This construction is carried out in details in the full version of this work.

7 Reed's Conjecture Parameterized by Treewidth

In this section, we provide a concrete example of how dynamic programming algorithms can be used to provide asymptotic upper bounds on the time-complexity of verifying whether a given graph theoretic conjecture is valid on the class of graphs of width at most k. More specifically, we analyze the following well known conjecture due to Reed [21], which establishes an upper bound on the chromatic number $\chi(G)$ of a triangle-free graph G in terms of the maximum degree $\Delta(G)$ of G.

▶ **Conjecture 21.** *For any simple, triangle-free, undirected graph G, $\chi(G) \leq \lceil \frac{\Delta(G)+3}{2} \rceil$.*

It is worth noting that graphs of treewidth at most k are $(k+1)$-colorable [3]. The following theorem due to Dvorák and Kawarabayashi establishes a better upper bound for the chromatic number of triangle-free graphs in terms of treewidth.

▶ **Theorem 22** ([10]). *For any triangle-free graph G of treewidth $\leq k$, $\chi(G) \leq \lceil \frac{k+3}{2} \rceil$.*

Therefore, in order to prove that every graph of treewidth at most k satisfies Conjecture 21, it is enough to show that for each $s \in \{0, \ldots, k-1\}$, every graph of treewidth at most k and maximum degree at most s has chromatic number at most $\lceil \frac{s+3}{2} \rceil$, since for larger values of s, Theorem 22 implies that the conjecture is true. Now, let $\textsc{Colorable}_r$ denote the graph property consisting of all graphs that are r-colorable, $\textsc{MaxDeg}_d$ denote the graph property consisting of all graphs that have maximum degree at least d, $\textsc{Clique}_\omega$ be the property consisting of all graphs that have clique number at least ω, and $\textsc{MultiEdge}$ be the property consisting of all graphs that have some multiple edges. Let $\textsc{Reed}(s)$ be the graph property

$$\textsc{Reed}(s) \equiv \neg\textsc{MultiEdge} \wedge \neg\textsc{Clique}_3 \wedge \neg\textsc{MaxDeg}_{s+1} \rightarrow \textsc{Colorable}_{\lceil (s+3)/2 \rceil}. \qquad (2)$$

Then determining whether all graphs of treewidth at most k satisfy Reed conjecture is equivalent to determining whether for each $s \in \{0, 1, \dots, k-1\}$, the set of all graphs of treewidth at most k is contained in property $\textsc{Reed}(s)$. Now, $\textsc{Reed}(s)$ is a Boolean combination of four properties, each of which can be decided coherent DP-cores whose bitlength is polynomial in k. The *bitlength of a DP-core* D is a function $\beta(k)$ that measures the maximum number of bits of a local witness produced during the processing of a k-instructive tree decomposition. The specification of each of the four DP-cores can be found in the appendix. Such implementations are of independent interest, because each such core may be viewed as an algorithm that takes a k-instructive tree decomposition τ as input and decides whether the graph $\mathcal{G}(\tau)$ associated with τ satisfies the corresponding property represented by the core.

▶ **Theorem 23** (DP-Cores for Reed's Conjecture). *There exist coherent, instructive DP-cores $C\text{-}\textsc{MaxDeg}$, $C\text{-}\textsc{Colorable}$, $C\text{-}\textsc{Clique}_\omega$, $C\text{-}\textsc{MultiEdge}$ satisfying the following properties.*

1. *$C\text{-}\textsc{MaxDeg}_d$ has bit-length $O(k \cdot \log d)$ and $\mathbb{G}(C\text{-}\textsc{MaxDeg}_d) = \textsc{MaxDeg}_d$.*
2. *$C\text{-}\textsc{Colorable}_r$ has bit-length $O(k \cdot \log r)$ and $\mathbb{G}(C\text{-}\textsc{Colorable}_r) = \textsc{Colorable}_r$.*
3. *$C\text{-}\textsc{Clique}_\omega$ has bit-length $O(k \cdot \log \omega)$ and $\mathbb{G}(C\text{-}\textsc{Clique}_\omega) = \textsc{Clique}_\omega$.*
4. *$C\text{-}\textsc{MultiEdge}$ has bit-length $O(k^2)$ and $\mathbb{G}(C\text{-}\textsc{MultiEdge}) = \textsc{MultiEdge}$*

Therefore, as a consequence of Theorem 19, Theorem 23, and Equation 2 we have the following corollary.

▶ **Corollary 24.** *For each $k \in \mathbb{N}$, Reed's conjecture for triangle-free graphs can be tested in time double-exponential in $O(k^2)$.*

Actually, the bound in Corollary 24 can be improved to double-exponential in $O(k \cdot \log k)$ by noting that the DP-core C-$\textsc{MultiEdge}$ is deterministic.

8 Experimental Evaluation

We have implemented our width-based automated theorem proving framework on a software called *TreeWidzard*. Our software provides an interface that facilitates the implementation of dynamic programming algorithms parameterized by treewidth and pathwidth, and the integration of such algorithms with the purpose of width based automated theorem proving. In this section we evaluate our implementation on the task of producing counter-examples for wrong graph-theoretic statements, and also to provide a verification of Reed's conjectures on graphs of small treewidth and pathwidth.

8.1 Constructing Counterexamples for Wrong Mathematical Statements

One of the most remarkable applications of the framework of width-based automated theorem proving lies in its ability to search for counter-examples width-wise. The advantage of this approach is that the width of a graph may be significantly smaller than its number of vertices.

As stated in Theorem 22, triangle-free graphs of treewidth at most k have chromatic number at most $\lceil \frac{k+3}{2} \rceil$. It can be shown by an analytic proof that for each $k \geq 1$, this bound is tight [10], in the sense that there are graphs of treewidth at most k that cannot be properly colored with $\lceil \frac{k+3}{2} \rceil - 1$ colors. In Figure 1:Left we depict a triangle-free graph with 14 vertices, 27 edges and chromatic number 4. Apart from the drawing, for which we took some artistic liberty, this graph was obtained as a counter-example for the following statement: *triangle-free graphs have chromatic number at most* 3. In particular, the graph was found when restricting our search to graphs of pathwidth at most 4. Note that it follows from [10] that this statement is true for graphs of pathwidth at most 3, and our search terminates without counter-examples in this case.

In a similar vein, the graph depicted in Figure 1:Right is a triangle-free graph of maximum degree 4 and chromatic number 4. This graph was obtained as a counter example for the following statement which is a (false) strengthening of Conjecture 21: *triangle-free graphs of maximum degree at most* 4 *have chromatic number at most* 3. The graph was found when restricting our search to graphs of pathwidth at most 4. It is worth noting that it also follows from [10] that this statement is true for graphs of pathwidth at most 3. In this case, our algorithm concluded the search without a counter-example.

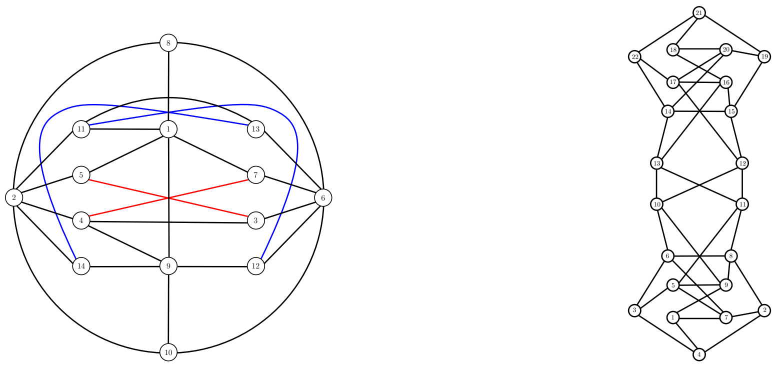

Figure 1 Left: G is a triangle-free graph of pathwidth 4 and chromatic number 4. This is a counter-example for a strengthening of a theorem of Reed and Dvorak. Right: G is a triangle-free graph of pathwidth 4, chromatic number 4, and maximum degree 4. This is a counter-example for a strengthening of Reed's conjecture for triangle free graphs.

8.2 Validating Reed's Triangle Free Conjecture on Graphs of Small Pathwidth and Treewidth

When testing whether a given graph-theoretic conjecture is valid on the class of graphs of pathwidth/treewidth at most k, our deduction algorithm may terminate without producing a counter-example. This means that the conjecture is valid on the class of graphs of pathwidth/treewidth at most k.

When combining both the symmetry-breaking technique introduced in Section 5 with the early-pruning technique introduced in Section 6, our software was able to confirm Conjecture 21 in the class of graphs of pathwidth at most 5. To the best of our knowledge, an analytic proof of Conjecture 21 on this class of graphs is lacking in the literature. It is worth noting that to confirm this case, it is enough to consider graphs of maximum degree at most 3, since for graphs of larger degree Reed's conjecture follow from Theorem 22.

In the verification of Conjecture 21 in the case of graphs of pathwidth at most 5, our search, using both techniques introduced in this work, produced 746187 states, took about 21 hours, and consumed 35 GB of memory on a cluster with processors of type Intel Xeon Gold 6130, with 2.1 Ghz. The search in this particular case was executed with 64 cores and 128 threads. It is worth noting that this case becomes prohibitively large if we deactivate either the canonization procedure or the subgraph-closed premise search. Therefore, the techniques introduced in this work were crucial in this regard.

For graphs of treewidth at most 3, and graphs of pathwidth at most 4, we performed a comparison between the original deduction algorithm introduced in [9], and the algorithm augmented only with the symmetry breaking procedure, only with the early pruning procedure, and with both (See Table 1). Both improvements when applied isolated decrease significantly the number of states considered during the search process. Nevertheless, when combining both approaches, the reduction of the search space was very expressive. For example, in the case of pathwidth 3 and maximum degree 2, the method with no improvement produced more than 20 million states. The method with only the symmetry breaking improvement produced about 1 million states, the method with only the early pruning improvement produced about 1916 states, and the method with both improvements produced only 141 states.

Table 1 Number of states generated by our program when testing Reed's (ω, Δ, χ)-conjecture for $\omega = 2$ (triangle-free case) on graphs of constant pathwidth and treewidth. The entries in blue correspond to experiments that did not terminate, and the entries with a star correspond to experiments that stopped due to a memory limit.

	BFS			BFS-premise		
Δ / pw	2	3	4	2	3	4
pw = 2	2503	4814	9877	149	341	617
pw = 3	> 20738085*	> 21164080*	5193467	1916	9850	27720
pw = 4	> 9463445*	> 6019042*	> 8333258*	29184	1156954	2438694
	ISO-BFS			ISO-BFS-premise		
pw = 2	520	945	1843	38	76	128
pw = 3	1031545	1050960	223920	141	579	1451
pw = 4	> 9449990*	> 9903864*	> 5029614*	486	12375	24494
	ISO-BFS			ISO-BFS-premise		
tw = 2	>25351	>25960	>27632	57	153	330
tw = 3	>69621	>71183	>72891	194	1268	4080
tw = 4	>71858	>73532	>75426	616	>4942	>13438

9 Conclusion

In this work, we have given the first steps towards evaluating the width-based automated theorem proving approach introduced in [9] from a practical perspective. At the same time, we have introduced two techniques that have together drastically reduced the space of states to be explored during the search for a counter-example. While the first technique is quite general and can be applied in the study of any conjecture involving DP-cores for which a witness action can be defined, the second improvement can be applied in the case of conjectures of the form $\mathbb{P}_1 \to \mathbb{P}_2$, where $\mathbb{P}_1$ is a property closed under subgraphs. To illustrate the applicability of our methods, we have used our implementation to produce counter-examples for false graph-theoretic statements, and also to confirm Reed's triangle-free conjecture on the class of graphs of pathwidth at most 5 and on the class of graphs of treewidth at most 3.

It is worth highlighting the modularity of our approach. While the implementation of instructive dynamic programming cores requires specialized knowledge from the part of the programmer, the use of such cores once they have been implemented is straightforward. For instance, in our framework, the critical case for testing Reed's conjecture on graphs of pathwidth at most 5 is the case where the degree is at most 3. More precisely, a conjecture stating that all simple, triangle-free graphs of maximum degree at most 3 have chromatic number at most 3. This conjecture is stated in our framework using the following intuitive lines of code, where the first four lines correspond each to a graph property, and the last line corresponds to the conjecture being tested.

```
x := MaxDegree_AtLeast(4)
y := CliqueNumber_AtLeast(3)
z := HasMultipleEdges
w := Colorable(3)
Formula
NOT x AND NOT y AND NOT z IMPLIES w
```

The idea is that dynamic programming cores deciding graph properties are implemented as plugins that need to be implemented only once by a specialist and then used without difficulty by graph theorists. We believe that our approach has the potential to create a nice interchange of knowledge between the community of researchers working on parameterized complexity theory, and researchers working in automated theorem proving. In essence, our framework shows that the 3 decades of accumulated knowledge obtained in the development of faster width-based parameterized algorithms for model checking may now be put into use in the context of automated theorem proving.

References

1 Therese C. Biedl. On triangulating k-outerplanar graphs. *Discret. Appl. Math.*, 181(1):275–279, 2015. doi:10.1016/j.dam.2014.10.017.

2 Hans L. Bodlaender. *Classes of graphs with bounded tree-width*, volume 86. Unknown Publisher, 1986.

3 Hans L Bodlaender and Fedor V Fomin. Equitable colorings of bounded treewidth graphs. *Theoretical Computer Science*, 349(1):22–30, 2005.

4 Béla Bollobás, Paul A Catlin, and Paul Erdös. Hadwiger's conjecture is true for almost every graph. *Eur. J. Comb.*, 1(3):195–199, 1980.

5 Andreas Brandstädt, Van Bang Le, and Jeremy P Spinrad. *Graph classes: a survey*. SIAM, 1999.

6 CN Campos and Yoshiko Wakabayashi. On dominating sets of maximal outerplanar graphs. *Discrete Applied Mathematics*, 161(3):330–335, 2013.

7 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

8 Bruno Courcelle and Irène Durand. Automata for the verification of monadic second-order graph properties. *Journal of applied logic*, 10(4):368–409, 2012.

9 Mateus de Oliveira Oliveira and Farhad Vadiee. From width-based model checking to width-based automated theorem proving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37(5), pages 6297–6304, 2023.

10 Zdeněk Dvořák and Ken-ichi Kawarabayashi. Triangle-free graphs of tree-width t are $\lceil (t+3)/2 \rceil$-colorable. *European Journal of Combinatorics*, 66:95–100, 2017.

11 DJ Guan and Xuding Zhu. Game chromatic number of outerplanar graphs. *Journal of Graph Theory*, 30(1):67–70, 1999.

12 Daniel Heldt, Kolja Knauer, and Torsten Ueckerdt. On the bend-number of planar and outerplanar graphs. *Discrete Applied Mathematics*, 179:109–119, 2014.

13 Pavol Hell and Xuding Zhu. The circular chromatic number of series-parallel graphs. *Journal of Graph Theory*, 33(1):14–24, 2000.

14 Jesper Lykke Jacobsen and Jesús Salas. Is the five-flow conjecture almost false? *Journal of Combinatorial Theory, Series B*, 103(4):532–565, 2013.

15 Frank Kammer. Determining the smallest k such that g is k-outerplanar. In *European Symposium on Algorithms*, pages 359–370. Springer, 2007.

16 Ton Kloks. *Treewidth: computations and approximations*. Springer, 1994.

17 Zhishi Pan and Xuding Zhu. Tight relation between the circular chromatic number and the girth of series-parallel graphs. *Discrete mathematics*, 254(1-3):393–404, 2002.

18 Léo Vieira Peres and Ricardo Dahab. Tutte's 3-flow conjecture for almost even graphs. *Procedia Computer Science*, 195:280–288, 2021.

19 Alexandre Pinlou and Éric Sopena. Oriented vertex and arc colorings of outerplanar graphs. *Information Processing Letters*, 100(3):97–104, 2006.

20 Andrzej Proskurowski and Jan Arne Telle. From bandwidth to pathwidth k. *_THEORETICAL-E*, page 90, 1996.

21 Bruce Reed. ω, δ, and χ. *Journal of Graph Theory*, 27(4):177–212, 1998.

22 Bo Zhou. Upper bounds for the zagreb indices and the spectral radius of series-parallel graphs. *International Journal of Quantum Chemistry*, 107(4):875–878, 2007.