Guarantees and Limits of Preprocessing in Constraint Satisfaction and Reasoning

Serge Gaspers† Stefan Szeider‡

Abstract

We present a first theoretical analysis of the power of polynomial-time preprocessing for important combinatorial problems from various areas in AI. We consider problems from Constraint Satisfaction, Global Constraints, Satisfiability, Nonmonotonic and Bayesian Reasoning under structural restrictions. All these problems involve two tasks: (i) identifying the structure in the input as required by the restriction, and (ii) using the identified structure to solve the reasoning task efficiently. We show that for most of the considered problems, task (i) admits a polynomial-time preprocessing to a problem kernel whose size is polynomial in a structural problem parameter of the input, in contrast to task (ii) which does not admit such a reduction to a problem kernel of polynomial size, subject to a complexity theoretic assumption. As a notable exception we show that the consistency problem for the AtMost-NValue constraint admits a polynomial kernel consisting of a quadratic number of variables and domain values. Our results provide a firm worst-case guarantees and theoretical boundaries for the performance of polynomial-time preprocessing algorithms for the considered problems.

Keywords: Fixed-Parameter Tractability; Kernelization; Constraint Satisfaction; Reasoning; Computational Complexity

1 Introduction

Many important computational problems that arise in various areas of AI are intractable. Nevertheless, AI research has been very successful in developing and implementing heuristic solvers that work well on real-world instances. An important component of virtually every solver is a powerful polynomial-time preprocessing procedure that reduces the problem input. For instance, preprocessing techniques for the propositional satisfiability problem are based on Boolean Constraint Propagation (see, e.g., [27]), CSP solvers make use of various local consistency algorithms that filter the domains of variables (see, e.g., [4]); similar preprocessing methods are used by solvers for Nonmonotonic and Bayesian reasoning problems (see, e.g., [38], [13], respectively). The history of preprocessing, like applying reduction rules to simplify truth functions, can be traced back to the 1950's [55]. A natural question in this regard is how to measure the quality of preprocessing rules proposed for a specific problem.

Until recently, no provable performance guarantees for polynomial-time preprocessing methods have been obtained, and so preprocessing was only subject of empirical studies. A possible reason for the lack of theoretical results is a certain inadequacy of the P vs NP framework for such an analysis: if we could reduce in polynomial time an instance of an NP-hard problem just by one bit, then we can solve the entire problem in polynomial time by repeating the reduction step a polynomial number of times, and P = NP follows.

With the advent of parameterized complexity [25], a new theoretical framework became available that provides suitable tools to analyze the power of preprocessing. Parameterized complexity considers a problem in a two-dimensional setting, where in addition to the input size $n$, a problem parameter $k$ is taken into consideration. This parameter can encode a structural aspect of the problem instance. A problem is called fixed-parameter tractable (FPT) if it can be solved in time $f(k)p(n)$ where $f$ is a function of the parameter $k$ and $p$ is a polynomial of the input size $n$. Thus, for FPT problems, the combinatorial explosion can be confined to the parameter and is independent of the input size. It is known that a

---

*Preliminary and shorter versions of this paper appeared in the proceedings of IJCAI 2011 [50] and AAAI 2011 [62].
†UNSW Australia and NICTA, Sydney, Australia
‡Institute of Information Systems, Vienna University of Technology, Austria
problem is fixed-parameter tractable if and only if every problem input can be reduced by polynomial-time preprocessing to an equivalent input whose size is bounded by a function of the parameter [3]. The reduced instance is called the problem kernel, the preprocessing is called kernelization. The power of polynomial-time preprocessing can now be benchmarked in terms of the size of the kernel. Once a small kernel is obtained, we can apply any method of choice to solve the kernel: brute-force search, heuristics, approximation, etc. [12]. Because of this flexibility a small kernel is generally preferable to a less flexible branching-based fixed-parameter algorithm. Thus, small kernels provide an additional value that goes beyond bare fixed-parameter tractability.

Kernelization is an important algorithmic technique that has become the subject of a very active field in state-of-the-art combinatorial optimization (see, e.g., the references in [28, 32, 15, 57]). Kernelization can be seen as a preprocessing with performance guarantee that reduces a problem instance in polynomial time to an equivalent instance, the kernel, whose size is a function of the parameter [28, 33, 12, 15].

Once a kernel is obtained, the time required to solve the instance is a function of the parameter only and therefore independent of the input size. While, in general, the time needed to solve an instance does not necessarily depend on the size of the instance alone, the kernelization view is that it preprocesses the easy parts of an instance, leaving a core instance encoding the hard parts of the problem instance. Naturally one aims at kernels that are as small as possible, in order to guarantee good worst-case running times as a function of the parameter, and the kernel size provides a performance guarantee for the preprocessing. Some NP-hard combinatorial problems such as \textit{k-Vertex Cover} admit polynomially sized kernels, for others such as \textit{k-Path} an exponential kernel is the best one can hope for [11].

As an example of a polynomial kernel, consider the \textit{k-Vertex Cover} problem, which, for a graph \( G = (V,E) \) and an integer parameter \( k \), is to decide whether there is a set \( S \) of at most \( k \) vertices such that each edge from \( E \) has at least one endpoint in \( S \). Buss’ kernelization algorithm for \textit{k-Vertex Cover} (see [14]) computes the set \( U \) of vertices with degree at least \( k+1 \) in \( G \). If \( |U| > k \), then reject the instance, i.e., output a trivial No-instance (e.g., the graph \( K_2 \) consisting of one edge and the parameter 0), since every vertex cover of size at most \( k \) contains each vertex from \( U \). Otherwise, if \( G \setminus U \) has more than \( k(k−|U|) \) edges, then reject the instance, since each vertex from \( G \setminus U \) covers at most \( k \) edges. Otherwise, output the instance \((G \setminus (U \cup L), k − |U|)\), where \( L \) is the set of degree-0 vertices in \( G \setminus U \). This instance has \( O(k^2) \) vertices and edges. Thus, Buss’ kernelization algorithm gives a quadratic kernel for \textit{k-Vertex Cover}.

In previous research several NP-hard AI problems have been shown to be fixed-parameter tractable. We list some important examples from various areas:

1. Constraint satisfaction problems (CSP) over a fixed universe of values, parameterized by the induced width [11].
2. Consistency and generalized arc consistency for intractable global constraints, parameterized by the cardinalities of certain sets of values [5].
3. Propositional satisfiability (SAT), parameterized by the size of backdoors [50].
4. Positive inference in Bayesian networks with variables of bounded domain size, parameterized by size of loop cutsets [32, 9].
5. Nonmonotonic reasoning with normal logic programs, parameterized by feedback width [41].

All these problems involve the following two tasks.

(i) \textit{Structure Recognition Task}: identify the structure in the input as required by the considered parameter.

(ii) \textit{Reasoning Task}: use the identified structure to solve a reasoning task efficiently.

For most of the considered problems we observe the following pattern: the Structure Recognition Task admits a polynomial kernel, in contrast to the Reasoning Task, which does not admit a polynomial kernel, unless the Polynomial Hierarchy collapses to its third level.

A negative exception to this pattern is the recognition problem for CSPs of small induced width, which most likely does not admit a polynomial kernel.

A positive exception to this pattern is the AtMost-NValue global constraint, for which we obtain a polynomial kernel. As in [5], the parameter is the number of holes in the domains of the variables, measuring how close the domains are to being intervals. More specifically, we present a linear time
preprocessing algorithm that reduces an AtMost-NValue constraint $C$ with $k$ holes to a consistency-equivalent AtMost-NValue constraint $C'$ of size polynomial in $k$. In fact, $C'$ has at most $O(k^2)$ variables and $O(k^2)$ domain values. We also give an improved branching algorithm checking the consistency of $C'$ in time $O(1.6181^k + n)$. The combination of kernelization and branching yields efficient algorithms for the consistency and propagation of (AtMost-)NValue constraints.

**Outline**

This article is organized as follows. Parameterized complexity and kernelization are formally introduced in Section 2. Section 3 describes the tools we use to show that certain parameterized problems do not have polynomial kernels. Sections 4–8 prove kernel lower bounds for parameterized problems in the areas of constraint networks, satisfiability, global constraints, Bayesian reasoning, and nonmonotonic reasoning. Each of these sections also gives all necessary definitions, relevant background, and related work for the considered problems. In addition, Section 6 describes a polynomial kernel for the consistency problem for the AtMost-NValue parameterized by the number of holes in the variable domains, and an FPT algorithm that uses this kernel as a first step. The correctness and performance guarantees of the kernelization algorithm are only outlined in Section 6 and proved in detail in Appendix A. The conclusion, Section 9, broadly recapitulates the results and suggests the study of Turing kernels to overcome the shortcomings of (standard) kernels for many fundamental AI and Reasoning problems.

**2 Formal Background**

A parameterized problem $P$ is a subset of $\Sigma^* \times \mathbb{N}$ for some finite alphabet $\Sigma$. For a problem instance $(x, k) \in \Sigma^* \times \mathbb{N}$ we call $x$ the main part and $k$ the parameter. We assume the parameter is represented in unary. For the parameterized problems considered in this paper, the parameter is a function of the main part, i.e., $k = \pi(x)$ for a function $\pi$. We then denote the problem as $P(\pi)$, e.g., $U$-CSP(width) denotes the problem $U$-CSP parameterized by the width of the given tree decomposition.

A parameterized problem $P$ is fixed-parameter tractable if there exists an algorithm that solves any input $(x, k) \in \Sigma^* \times \mathbb{N}$ in time $O(f(k) \cdot p(|x|))$ where $f$ is an arbitrary computable function of $k$ and $p$ is a polynomial in $|x|$.

A kernelization for a parameterized problem $P \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, outputs in time polynomial in $|x| + k$ a pair $(x', k') \in \Sigma^* \times \mathbb{N}$ such that

1. $(x, k) \in P$ if and only if $(x', k') \in P,$ and
2. $|x'| + k' \leq g(k)$, where $g$ is an arbitrary computable function, called the size of the kernel.

In particular, for constant $k$ the kernel has constant size $g(k)$. If $g$ is a polynomial then we say that $P$ admits a polynomial kernel.

Every fixed-parameter tractable problem admits a kernel. This can be seen by the following argument due to Downey et al. [24]. Assume we can decide instances $(x, k)$ of problem $P$ in time $f(k)|x|^{O(1)}$. We kernelize an instance $(x, k)$ as follows. If $|x| \leq f(k)$ then we already have a kernel of size $f(k)$. Otherwise, if $|x| > f(k)$, then $f(k)|x|^{O(1)} = |x|^{O(1)}$ is a polynomial; hence we can decide the instance in polynomial time and replace it with a small decision-equivalent instance $(x', k')$. Thus we always have a kernel of size at most $f(k)$. However, $f(k)$ is super-polynomial for NP-hard problems (unless $P = \text{NP}$), hence this generic construction does not provide polynomial kernels.

We understand preprocessing for an NP-hard problem as a polynomial-time procedure that transforms an instance of the problem to a (possible smaller) solution-equivalent instance of the same problem. Kernelization is such a preprocessing with a performance guarantee, i.e., we are guaranteed that the preprocessing yields a kernel whose size is bounded in terms of the parameter of the given problem instance. In the literature also different forms of preprocessing have been considered. An important one is knowledge compilation, a two-phases approach to reasoning problems where in a first phase a given knowledge base is (possibly in exponential time) preprocessed (“compiled”), such that in a second phase various queries can be answered in polynomial time [15].

**3 Tools for Kernel Lower Bounds**

In the sequel we will use recently developed tools to obtain kernel lower bounds. Our kernel lower bounds are subject to the widely believed complexity theoretic assumption NP $\not\subseteq$ coNP/poly. In other words,
the tools allow us to show that a parameterized problem does not admit a polynomial kernel unless \( \text{NP} \subseteq \text{coNP/poly} \). In particular, \( \text{NP} \subseteq \text{coNP/poly} \) would imply the collapse of the Polynomial Hierarchy to the third level: \( \text{PH} = \Sigma^p_3 \).

A composition algorithm for a parameterized problem \( P \subseteq \Sigma^* \times \mathbb{N} \) is an algorithm that receives as input a sequence \((x_1, k), \ldots, (x_t, k) \in \Sigma^* \times \mathbb{N}\), uses time polynomial in \( \sum_{i=1}^{t} |x_i| + k \), and outputs \((y, k') \in \Sigma^* \times \mathbb{N}\) with (i) \((y, k') \in P\) if and only if \((x_i, k) \in P\) for some \(1 \leq i \leq t\), and (ii) \(k'\) is polynomial in \(k\). A parameterized problem is compositional if it has a composition algorithm. With each parameterized problem \( P \subseteq \Sigma^* \times \mathbb{N}\) we associate a classical problem

\[
\text{UP}[P] = \{ x\#1^k : (x, k) \in P \}
\]

where \(1\) denotes an arbitrary symbol from \(\Sigma\) and \(\#\) is a new symbol not in \(\Sigma\). We call \(\text{UP}[P]\) the unparameterized version of \(P\).

The following result is the basis for our kernel lower bounds.

**Theorem 1** (Erik Jonsson). Let \(P\) be a parameterized problem whose unparameterized version is \(\text{NP}\)-complete. If \(P\) is compositional, then it does not admit a polynomial kernel unless \(\text{NP} \subseteq \text{coNP/poly}\).

Let \(P, Q \subseteq \Sigma^* \times \mathbb{N}\) be parameterized problems. We say that \(P\) is polynomial parameter reducible to \(Q\) if there exists a polynomial time computable function \(K : \Sigma^* \times \mathbb{N} \rightarrow \Sigma^* \times \mathbb{N}\) and a polynomial \(p\), such that for all \((x, k) \in \Sigma^* \times \mathbb{N}\) we have (i) \((x, k) \in P\) if and only if \(K(x, k) = (x', k') \in Q\), and (ii) \(k' \leq p(k)\). The function \(K\) is called a polynomial parameter transformation.

The following theorem allows us to transform kernel lower bounds from one problem to another.

**Theorem 2** (N. Robertson). Let \(P\) and \(Q\) be parameterized problems such that \(\text{UP}[P]\) is \(\text{NP}\)-complete, \(\text{UP}[Q]\) is in \(\text{NP}\), and there is a polynomial parameter transformation from \(P\) to \(Q\). If \(Q\) has a polynomial kernel, then \(P\) has a polynomial kernel.

## 4 Constraint Networks

Constraint networks have proven successful in modeling everyday cognitive tasks such as vision, language comprehension, default reasoning, and abduction, as well as in applications such as scheduling, design, diagnosis, and temporal and spatial reasoning [21]. A constraint network is a triple \(I = (V, U, \mathcal{C})\) where \(V\) is a finite set of variables, \(U\) is a finite universe of values, and \(\mathcal{C} = \{C_1, \ldots, C_m\}\) is a set of constraints. Each constraint \(C_i\) is a pair \((S_i, R_i)\) where \(S_i\) is a list of variables of length \(r_i\) called the constraint scope, and \(R_i\) is an \(r_i\)-ary relation over \(U\), called the constraint relation. The tuples of \(R_i\) indicate the allowed combinations of simultaneous values for the variables \(S_i\). A solution is a mapping \(\tau : V \rightarrow U\) such that for each \(1 \leq i \leq m\) and \(S_i = \{x_1, \ldots, x_{r_i}\}\), we have \((\tau(x_1), \ldots, \tau(x_{r_i})) \in R_i\). A constraint network is satisfiable if it has a solution.

With a constraint network \(I = (V, U, \mathcal{C})\) we associate its constraint graph \(G = (V, E)\) where \(E\) contains an edge between two variables if and only if they occur together in the scope of a constraint. A width \(w\) tree decomposition of a graph \(G\) is a pair \((T, \lambda)\) where \(T\) is a tree and \(\lambda\) is a labeling of the nodes of \(T\) with sets of vertices of \(G\) such that the following properties are satisfied: (i) every vertex of \(G\) belongs to \(\lambda(p)\) for some node \(p\) of \(T\); (ii) every edge of \(G\) is contained in \(\lambda(p)\) for some node \(p\) of \(T\); (iii) for each vertex \(v\) of \(G\) the set of all tree nodes \(p\) with \(v \in \lambda(p)\) induces a connected subtree of \(T\); (iv) \(|\lambda(p)| - 1 \leq w\) holds for all tree nodes \(p\). The treewidth of \(G\) is the smallest \(w\) such that \(G\) has a width \(w\) tree decomposition. The induced width of a constraint network is the treewidth of its constraint graph [22].

Kernelization fits perfectly into the context of Constraint Processing where preprocessing and data reduction (e.g., in terms of local consistency algorithms, propagation, and domain filtering) are key methods [3] [63].

Let \(U\) be a fixed universe containing at least two elements. We consider the following parameterized version of the constraint satisfaction problem (CSP).

\[\text{U-CSP}(\text{width})\]

**Instance:** A constraint network \(I = (V, U, \mathcal{C})\) and a width \(w\) tree decomposition of the constraint graph of \(I\).

**Parameter:** The integer \(w\).

**Question:** Is \(I\) satisfiable?
Associated with this problem is also the task of recognizing instances of small treewidth. We state this problem in form of the following decision problem.

**Rec-U-CSP**(width)

**Instance:** A constraint network \( I = (V, U, \mathcal{C}) \) and an integer \( w \geq 0 \).

**Parameter:** The integer \( w \).

**Question:** Does \( I \) admit a tree decomposition of width \( \leq w \)?

It is well known that \( U\text{-CSP}(\text{width}) \) is fixed-parameter tractable over any fixed universe \( U \) (for generalizations see [61]). We contrast this classical result and show that it is unlikely that \( U\text{-CSP}(\text{width}) \) admits a polynomial kernel, even in the simplest case where \( U = \{0, 1\} \).

**Theorem 3.** \( \{0, 1\}\text{-CSP}(\text{width}) \) does not admit a polynomial kernel unless \( \text{NP} \subseteq \text{coNP}/\text{poly} \).

**Proof.** We show that \( \{0, 1\}\text{-CSP}(\text{width}) \) is compositional. Let \( (I_i, T_i), 1 \leq i \leq t \), be a given sequence of instances of \( \{0, 1\}\text{-CSP}(\text{width}) \) where \( I_i = (V_i, \{0, 1\}, \mathcal{C}_i) \) is a constraint network and \( T_i \) is a width \( w \) tree decomposition of the constraint graph of \( I_i \). We may assume, w.l.o.g., that \( V_i \cap V_j = \emptyset \) for \( 1 \leq i < j \leq t \) (otherwise we can simply change the names of variables). We form a new constraint network \( I = (V, \{0, 1\}, \mathcal{C}) \) as follows. We put \( V = \bigcup_{i=1}^t V_i \cup \{a_1, \ldots, a_t, b_0, \ldots, b_t\} \) where \( a_i, b_i \) are new variables. We define the set \( \mathcal{C} \) of constraints in three groups.

1. For each \( 1 \leq i \leq t \) and each constraint \( C = (x_1, \ldots, x_r, R) \in \mathcal{C}_i \), we add to \( \mathcal{C} \) a new constraint \( C' = ((x_1, \ldots, x_r, a_i), R') \) where \( R' = \{ (u_1, \ldots, u_r, 0) : (u_1, \ldots, u_r) \in R \} \cup \{(1, \ldots, 1)\} \).
2. We add \( t \) ternary constraints \( C_1^*, \ldots, C_t^* \) where \( C_i^* = ((b_{i-1}, b_i, a_i), R^*) \) and \( R^* = \{(0, 0, 1), (0, 1, 0), (1, 1, 1)\} \).
3. Finally, we add two unary constraints \( C^0 = ((b_0), (0)) \) and \( C^1 = ((b_t), (1)) \) which force the values of \( b_0 \) and \( b_t \) to 0 and 1, respectively.

Let \( G, G_i \) be the constraint graphs of \( I \) and \( I_i \), respectively. Fig. 1 shows an illustration of \( G \) for \( t = 4 \). We observe that \( a_1, \ldots, a_t \) are cut vertices of \( G \). Removing these vertices separates \( G \) into independent parts \( P, P^i_1, \ldots, P^i_t \), \( P \) is the path \( b_0, b_1, \ldots, b_t \), and \( G^i_i \) is isomorphic to \( G_i \). By standard techniques (see, e.g., [22]), we can put the given width \( w \) tree decompositions \( T_1, \ldots, T_t \) of \( G^1, \ldots, G^t \) and the trivial width \( 1 \) tree decomposition of \( P \) together to a width \( w + 1 \) tree decomposition \( T \) of \( G \). Clearly \( (I, T) \) can be obtained from \( (I_i, T_i), 1 \leq i \leq t \), in polynomial time.

We claim that \( I \) is satisfiable if and only if at least one of the \( I_i \) is satisfiable. This claim can be verified by means of the following observations: The constraints in groups (2) and (3) provide that for any satisfying assignment there will be some \( 0 \leq i \leq t - 1 \) such that \( b_0, \ldots, b_i \) are all set to 0 and \( b_{i+1}, \ldots, b_t \) are all set to 1; consequently \( a_i \) is set to 0 and all \( a_j \) for \( j \neq i \) are set to 1. The constraints in group (1) provide that if we set \( a_t \) to 0, then we obtain from \( C' \) the original constraint \( C \); if we set \( a_t \) to 1 then we obtain a constraint that can be satisfied by setting all remaining variables to 1. We conclude that \( \{0, 1\}\text{-CSP}(\text{width}) \) is compositional.

In order to apply Theorem 1 we need to establish that the unparameterized version of \( \{0, 1\}\text{-CSP}(\text{width}) \) is NP-complete. Deciding whether a constraint network \( I \) over the universe \( \{0, 1\} \) is satisfiable is well-known to be NP-complete (say by reducing 3-SAT). To a constraint network \( I \) on \( n \) variables we can always add a trivial width \( w = n - 1 \) tree decomposition of its constraint graph (taking a single tree node \( t \) where \( \lambda(t) \) contains all variables of \( I \)). Hence UP\([\{0, 1\}\text{-CSP}(\text{width})]\) is NP-complete. \( \square \)
Let us turn now to the recognition problem \textbf{Rec-U-CSP}(width). By Bodlaender’s Theorem \cite{10}, the problem is fixed-parameter tractable. However, the problem is unlikely to admit a polynomial kernel. In fact, Bodlaender et al. \cite{11} showed that the related problem of testing whether a graph has treewidth at most \( w \) does not have a polynomial kernel (taking \( w \) as the parameter), unless a certain "AND-conjecture" fails. In turn, Drucker \cite{26} showed that a failure of the AND-conjecture implies \( \text{NP} \subseteq \text{coNP}/\text{poly} \). The combination of these two results relates directly to \textbf{Rec-U-CSP}(width).

**Proposition 1.** \textbf{Rec-\{0,1\}-CSP}(width) does not admit a polynomial kernel unless \( \text{NP} \subseteq \text{coNP}/\text{poly} \).

## 5 Satisfiability

The \textit{propositional satisfiability problem} (SAT) was the first problem shown to be NP-hard \cite{18}. Despite its hardness, SAT solvers are increasingly leaving their mark as a general-purpose tool in areas as diverse as software and hardware verification, automatic test pattern generation, planning, scheduling, and even challenging problems from algebra \cite{40}. SAT solvers are capable of exploiting the hidden structure present in real-world problem instances. The concept of \textit{backdoors}, introduced by Williams et al. \cite{65}, provides a means for making the vague notion of a hidden structure explicit. Backdoors are defined with respect to a "sub-solver" which is a polynomial-time algorithm that correctly decides the satisfiability for \( \text{A} \)-backdoor of size \( k \), we can decide the satisfiability of \( F \) by running \( A \) on \( 2^k \) instances \( F[\tau] \), yielding a time bound of \( O(2^k n^c) \). Hence SAT decision is fixed-parameter tractable in the backtrack size \( k \) for any sub-solver \( A \). Hence the following problem is clearly fixed-parameter tractable for any sub-solver \( A \).

**SA T**(A-backdoor)

\begin{itemize}
  \item \textbf{Instance:} A CNF formula \( F \), and an A-backdoor \( B \) of \( F \) of size \( k \).
  \item \textbf{Parameter:} The integer \( k \).
  \item \textbf{Question:} Is \( F \) satisfiable?
\end{itemize}

We also consider for every subsolver \( A \) the associated recognition problem.

**Rec-SA T**(A-backdoor)

\begin{itemize}
  \item \textbf{Instance:} A CNF formula \( F \), and an integer \( k \geq 0 \).
  \item \textbf{Parameter:} The integer \( k \).
  \item \textbf{Question:} Does \( F \) have an A-backdoor of size at most \( k \)?
\end{itemize}

With the problem \textbf{SA T}(A-backdoor) we are concerned with the question of whether instead of trying all \( 2^k \) possible partial assignments we can reduce the instance to a polynomial kernel. We will establish a very general result that applies to all possible sub-solvers.

**Theorem 4.** \textbf{SA T}(A-backdoor) does not admit a polynomial kernel for any sub-solver \( A \) unless \( \text{NP} \subseteq \text{coNP}/\text{poly} \).
**Theorem 5.** Neither 3SAT(HORN-backdoor) nor 3SAT(2CNF-backdoor) admit a polynomial kernel unless NP \( \subseteq \text{coNP/poly}. \)

*Proof.* Let \( \ell \in \{\text{HORN, 2CNF}\}. \) We show that 3SAT(\( \ell \)-backdoor) is compositional. Let \( (F_i, B_i) \), \( 1 \leq i \leq t \), be a given sequence of instances of 3SAT(\( \ell \)-backdoor) where \( F_i \) is a 3CNF formula and \( B_i \) is a \( \ell \)-backdoor set of \( F_i \), of size \( k \). We distinguish two cases.

**Case 1:** \( t > 2^k \). Let \( \|F_i\| := \sum_{C \in F_i} |C| \) and \( n := \max_{i=1}^t \|F_i\| \). Whether \( F_i \) is satisfiable or not can be decided in time \( O(2^k n) \) since the satisfiability of a Horn or 2CNF formula can be decided in linear time. We can check whether at least one of the formulas \( F_1, \ldots, F_t \) is satisfiable in time \( O(2^k n) = O(t^2 n) \) which is polynomial in \( t + n \). If some \( F_i \) is satisfiable, we output \( (F_i, B_i) \); otherwise we output \( (F_1, B_1) \) (\( F_1 \) is unsatisfiable). Hence we have a composition algorithm.

**Case 2:** \( t \leq 2^k \). This case is more involved. We construct a new instance \((F, B)\) of 3SAT(\( \ell \)-backdoor) as follows.

Let \( s = \lceil \log_2 t \rceil \). Since \( t \leq 2^k \), \( s \leq k \) follows.

Let \( V_i \) denote the set of variables of \( F_i \). We may assume, w.l.o.g., that \( B_1 = \cdots = B_t \) and that \( V_i \cap V_j = B_1 \) for all \( 1 \leq i < j \leq t \) since otherwise we can change names of variable accordingly. In a first step we obtain from each \( F_i \) a CNF formula \( F'_i \) as follows. For each variable \( x \in V_i \setminus B_1 \) we take \( s + 1 \) new variables \( x_0, \ldots, x_s \). We replace each positive occurrence of a variable \( x \in V_i \setminus B_1 \) in \( F_i \) with the literal \( x_0 \) and each negative occurrence of \( x \) with the literal \( \neg x_s \).

We add all clauses of the form \((\neg x_{j-1} \lor x_j)\) for \( 1 \leq j \leq s \); we call these clauses “connection clauses.” Let \( F''_i \) be the formula obtained from \( F_i \) in this way. We observe that \( F''_i \) and \( F_i \) are SAT-equivalent, since the connection clauses form an implication chain. Since the connection clauses are both Horn and 2CNF, \( B_1 \) is also a \( \ell \)-backdoor of \( F''_i \).

For an illustration of this construction see Example 1 below. We take a set \( Y = \{y_1, \ldots, y_s\} \) of new variables. Let \( C_1, \ldots, C_{2^s} \) be the sequence of all \( 2^s \) possible clauses (modulo permutation of literals within a clause) containing each variable from \( Y \) either positively or negatively. Consequently we can write \( C_i \) as \((\ell_1 \lor \cdots \lor \ell_t)\) where \( \ell_j \in \{y_j, \neg y_j\} \).

For \( 1 \leq i \leq t \) we add to each connection clause \((\neg x_{j-1} \lor x_j)\) of \( F''_i \) the literal \( \ell_j \in C_i \). Let \( F''_i \) denote the 3CNF formula obtained from \( F''_i \) this way.

For \( t < i \leq 2^s \) we define 3CNF formulas \( F''_{i'} \) as follows. If \( s \leq 3 \) then \( F''_{i'} \) consists just of the clause \( C_i \). If \( s > 3 \) then we take new variables \( z_1, \ldots, z_{s-2} \) and let \( F''_{i'} \) consist of the clauses \((\ell_1 \lor \ell_2 \lor \neg z_1), (\ell_3 \lor z_2 \lor \neg z_1), \ldots, (\ell_{s-2} \lor z_{s-3} \lor \neg z_{s-2}), (\ell_{s-1} \lor \ell_s \lor z_{s-2}) \). Finally, we let \( F \) be the 3CNF formula containing all the clauses from \( F''_1, \ldots, F''_{2^s} \). Any assignment \( \tau \) to \( Y \cup B_1 \) that satisfies \( C_i \) be
extended to an assignment that satisfies $F''_i$ since such assignment satisfies at least one connection clause $(x_j \lor x_j \lor f_j)$ and so the chain of implications from from $x_j$ to $x_i$ is broken.

It is not difficult to verify the two claims. (i) $F$ is satisfiable if and only if at least one of the formulas $F_i$ is satisfiable. (ii) $B = Y \cup B_1$ is a $\mathcal{C}$-backdoor of $F$. Hence we have also a composition algorithm in Case 2, and thus $3\text{SAT}(\mathcal{C}\text{-backdoor})$ is compositional. Clearly $\text{UP}[3\text{SAT}(\mathcal{C}\text{-backdoor})]$ is NP-complete, hence the result follows from Theorem\[1\].

\begin{example}
We illustrate the constructions of this proof with a running example, where we let $s = 2$, $t = 4$, $i = 2$, and $B_1 = \{b\}$.
Assume that we have

$$F_i = (x \lor \neg u \lor v) \land \neg (x \lor u \lor v) \land \neg (x \lor \neg u).$$

From this we obtain the following formula, containing four connection clauses

$$F_i' = (x_0 \lor \neg u_2 \lor v) \land (\neg x_2 \lor u_0 \lor v) \land (\neg x_2 \lor \neg u_2) \land$$
$$\quad (\neg x_0 \lor x_1) \land (\neg x_1 \lor x_2) \land (\neg u_0 \lor u_1) \land (\neg u_1 \lor u_2).$$

Now assume $C_i = (y_1 \lor \neg y_2)$. We add to the connection clauses literals from $C_i$ and obtain

$$F_i'' = (x_0 \lor \neg u_2 \lor v) \land (\neg x_2 \lor u_0 \lor v) \land (\neg x_2 \lor \neg u_2) \land$$
$$\quad (\neg x_0 \lor x_1 \lor y_1) \land (\neg x_1 \lor x_2 \lor \neg y_2) \land (\neg u_0 \lor u_1 \lor y_1) \land (\neg u_1 \lor u_2 \lor \neg y_2).$$

Assigning $y_1$ to false and $y_2$ to true reduces $F''_i$ to $F_i'$. The other three possibilities of assigning truth values to $y_1, y_2$ break the connection clauses and make the formula trivially satisfiable. $\blacksquare$

We now turn to the recognition problem $\text{Rec-SAT}(A\text{-backdoor})$, in particular for $A \in \{\text{HORN, 2CNF}\}$ for which, as mentioned above, the problem is known to be fixed-parameter tractable\[50\]. Here we are able to obtain positive results.

\begin{proposition}
Both $\text{Rec-SAT}(\text{HORN-backdoor})$ and $\text{Rec-SAT}(2\text{CNF-backdoor})$ admit polynomial kernels, with a linear and quadratic number of variables, respectively.
\end{proposition}

\begin{proof}
Let $(F, k)$ be the instance of $\text{Rec-SAT}(\text{HORN-backdoor})$. We construct a graph $G(F)$ whose vertices are the variables of $F$ and which contains an edge between two variables $u, v$ if and only if both variables appear as positive literals together in a clause. It is well-known and easy to see that the vertex covers of $G(F)$ are exactly the HORN-backdoor sets of $F$\[60\]. Recall that a vertex cover of a graph is a set of vertices that contains at least one end of each edge of the graph. Now, we apply the known kernelization algorithm for vertex covers\[17\] to $(G(F), k)$ and obtain in polynomial time an equivalent instance $(G', k')$ where $G'$ has at most $2k$ vertices. Now it only remains to consider $G'$ as a CNF formula $F'$ where each edge gives rise to a binary clause on two positive literals. Since evidently $G(F') = G'$, we conclude that $(F', k')$ constitutes a polynomial kernel for $\text{Rec-SAT}(\text{HORN-backdoor})$.

For $\text{Rec-SAT}(2\text{CNF-backdoor})$ we proceed similarly. Let $(F, k)$ be an instance of this problem. We construct a 3-uniform hypergraph $H(F)$ whose vertices are the variables of $F$ and which contains a hyperedge on any three variables that appear (positively or negatively) together in a clause of $F$. Again, it is well-known and easy to see that the hitting sets of $H(F)$ are exactly the 2CNF-backdoor sets of $F$\[60\]. Recall that a hitting set of a hypergraph is a set of vertices that contains at least one vertex from each hyperedge. Now we apply a known kernelization algorithm for the hitting set problem on 3-uniform hypergraphs (3HS)\[1\] to $(H(F), k)$ and obtain in polynomial time an equivalent instance $(H', k')$ where $H'$ has at most $O(k^2)$ vertices. It remains to consider $H'$ as a CNF formula $F'$ where each hyperedge gives rise to a ternary clause on three positive literals. Since evidently $H(F') = H'$, we conclude that $(F', k')$ constitutes a polynomial kernel for $\text{Rec-SAT}(2\text{CNF-backdoor})$. $\blacksquare$
\end{proof}

\section{Global Constraints}

Constraint programming (CP) offers a powerful framework for efficient modeling and solving of a wide range of hard problems\[55\]. At the heart of efficient CP solvers are so-called \textit{global constraints} that specify patterns that frequently occur in real-world problems. Efficient propagation algorithms for global constraints help speed up the solver significantly\[63\]. For instance, a frequently occurring pattern is that we require that certain variables must all take different values (e.g., activities requiring the same resource must all be assigned different times). Therefore most constraint solvers provide a global \texttt{ALLDIFFERENT}
constraint and algorithms for its propagation. Unfortunately, for several important global constraints a complete propagation is NP-hard, and one switches therefore to incomplete propagation such as bound consistency [5].

In their AAAI’08 paper, Bessière et al. [5] showed that a complete propagation of several intractable constraints can efficiently be done as long as certain natural problem parameters are small, i.e., the propagation is fixed-parameter tractable [25]. Among others, they showed fixed-parameter tractability of the AtLeast-NValue and Extended Global Cardinality (EGC) constraints parameterized by the number of “holes” in the domains of the variables. If there are no holes, then all domains are intervals and complete propagation is polynomial by classical results; thus the number of holes provides a way of scaling up the nice properties of constraints with interval domains.

In the sequel we bring this approach a significant step forward, picking up a long-term research objective suggested by Bessière et al. [5] in their concluding remarks: whether intractable global constraints admit a reduction to a problem kernel or kernelization.

More formally, a global constraint is defined for a set $S$ of variables, each variable $x \in S$ ranges over a finite domain $\text{dom}(x)$ of values. For a set $X$ of variables we write $\text{dom}(X) = \bigcup_{x \in X} \text{dom}(x)$. An instantiation is an assignment $\alpha : S \to \text{dom}(S)$ such that $\alpha(x) \in \text{dom}(x)$ for each $x \in S$. A global constraint defines which instantiations are legal and which are not. This definition is usually implicit, as opposed to classical constraints, which list all legal tuples. Examples of global constraints include:

1. The global constraint NValue is defined over a set $S$ of variables and a variable $N$ and requires from a legal instantiation $\alpha$ that $|\{ \alpha(x) : x \in X \}| = \alpha(N)$;
2. The global constraint AtMost-NValue is defined for fixed values of $N$ over a set $X$ of variables and requires from a legal instantiation $\alpha$ that $|\{ \alpha(x) : x \in X \}| \leq N$;
3. The global constraint Disjoint is specified by two sets of variables $X,Y$ and requires that $\alpha(x) \neq \alpha(y)$ for each pair $x \in X$ and $y \in Y$;
4. The global constraint Uses is also specified by two sets of variables $X,Y$ and requires that for each $x \in X$ there is some $y \in Y$ such that $\alpha(x) = \alpha(y)$.
5. The global constraint EGC is specified by a set of variables $X$, a set of values $D = \text{dom}(X)$, and a finite domain $\text{dom}(v) \subseteq \mathbb{N}$ for each value $v \in D$, and it requires that for each $v \in D$ we have $|\{ \alpha(x) = v : x \in X \}| \in \text{dom}(v)$.

A global constraint $C$ is consistent if there is a legal instantiation of its variables. The constraint $C$ is hyper arc consistent (HAC) if for each variable $x \in \text{scope}(C)$ and each value $v \in \text{dom}(x)$, there is a legal instantiation $\alpha$ such that $\alpha(x) = v$ (in that case we say that $C$ supports $v$ for $x$). In the literature, HAC is also called domain consistent or generalized arc consistent. The constraint $C$ is bound consistent if when a variable $x \in \text{scope}(C)$ is assigned the minimum or maximum value of its domain, there are compatible values between the minimum and maximum domain value for all other variables in $\text{scope}(C)$. The main algorithmic problems for a global constraint $C$ are the following: Consistency, to decide whether $C$ is consistent, and Enforcing HAC, to remove from all domains the values that are not supported by the respective variable.

It is clear that if HAC can be enforced in polynomial time for a constraint $C$, then the consistency of $C$ can also be decided in polynomial time (we just need to see if any domain became empty). The reverse is true if for each $x \in \text{scope}(C)$ and $v \in \text{dom}(x)$, the consistency of $C \land (x \leftarrow v)$, requiring $x$ to be assigned the value $v$, can be decided in polynomial time (see [53] Theorem 17). This is the case for most constraints of practical use, and in particular for all constraints considered below. The same correspondence holds with respect to fixed-parameter tractability. Hence, we will focus mainly on Consistency.

For several important types $T$ of global constraints, the problem of deciding whether a constraint of type $T$ is consistent is NP-hard. This includes the 5 global constraints NValue, AtMost-NValue, Disjoint, Uses, and EGC defined above (see [5]).

Each global constraint of type $T$ and parameter par gives rise to a parameterized problem:

$T$-Cons(par)

Instance: A global constraint $C$ of type $T$.

Parameter: The integer par.

Question: Is $C$ consistent?
Bessière et al. [5] considered $dx = |\text{dom}(X)|$ as parameter for NVALUE, $dy = |\text{dom}(X) \cap \text{dom}(Y)|$ as parameter for DISJOINT, and $dy = |\text{dom}(Y)|$ as parameter for Uses. They showed that consistency checking is fixed-parameter tractable for the constraints under the respective parameterizations, i.e., the problems NVALUE-CONS($dx$), DISJOINT-CONS($dy$), and Uses-CONS($dy$) are fixed-parameter tractable.

Bessière et al. [5] also showed that polynomial time algorithms for enforcing bounds consistency imply that the corresponding consistency problem is fixed-parameter tractable parameterized by the number of holes. This is the case for the global constraints NVALUE, ATMOST-NVALUE, and EGC.

**Definition 1.** When $D$ is totally ordered, a hole in a subset $D' \subseteq D$ is a couple $(u, w) \in D' \times D'$, such that there is a $v \in D \setminus D'$ with $u < v < w$ and there is no $v' \in D'$ with $u < v' < w$.

We denote the number of holes in the domain of a variable $x \in X$ by $\#\text{holes}(x)$. The parameter of the consistency problem for ATMOST-NVALUE constraints is $\text{holes} = \sum_{x \in X} \#\text{holes}(x)$.

### 6.1 Kernel Lower Bounds

We show that it is unlikely that most of the FPT results of Bessière et al. [5] can be improved to polynomial kernels.

**Theorem 6.** The problems NVALUE-CONS($dx$), DISJOINT-CONS($dy$), Uses-CONS($dy$) do not admit polynomial kernels unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.

**Proof.** We devise a polynomial parameter transformation from SAT($\text{vars}$). We use a construction of Bessière et al. [5]. Let $F = \{C_1, \ldots, C_m\}$ be a CNF formula over variables $x_1, \ldots, x_n$. We consider the clauses and variables of $F$ as the variables of a global constraint with domains $\text{dom}(x_i) = \{-i, i\}$, and $\text{dom}(C_j) = \{i : x_i \in C_j\} \cup \{-i : \neg x_i \in C_j\}$. Now $F$ can be encoded as an NVALUE constraint with $X = \{x_1, \ldots, x_n, C_1, \ldots, C_m\}$ and $\text{dom}(N) = \{n\}$. By the pigeonhole principle, a legal instantiation $\alpha$ for this constraint has $|\{\alpha(x_i) : 1 \leq i \leq n\}| = N$. Setting $\alpha(x_i) = i$ corresponds to setting the variable $x_i$ of $F$ to 1 and setting $\alpha(x_i) = -i$ corresponds to setting the variable $x_i$ of $F$ to 0. Now, for each $C_j \in F$, $\alpha(C_j) \in \{\alpha(x_i) : 1 \leq i \leq n\}$ since only $n$ values are available for $\alpha$, and the literal corresponding to $\alpha(C_j)$ satisfies the clause $C_j$. Since $dx = 2n$ we have a polynomial parameter reduction from SAT($\text{vars}$) to NVALUE-CONS($dx$). Similarly, as observed by Bessière et al. [5], $F$ can be encoded as a DISJOINT constraint with $X = \{x_1, \ldots, x_n\}$ and $Y = \{C_1, \ldots, C_m\}$ ($\text{dy} \leq 2n$), or as a Uses constraint with $X = \{C_1, \ldots, C_m\}$ and $Y = \{x_1, \ldots, x_n\}$ ($\text{dy} = 2n$). Since the unparameterized problems are clearly NP-complete, and SAT($\text{vars}$) is known to have no polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$ (as remarked in the proof of Theorem 1), the result follows by Theorem 2.

The Consistency problem for EGC constraints is NP-hard [54]. However, if all sets $\text{dom}(\cdot)$ are intervals, then consistency can be checked in polynomial time using network flows [50]. By the result of Bessière et al. [5], the Consistency problem for EGC constraints is fixed-parameter tractable, parameterized by the number of holes in the sets $\text{dom}(\cdot)$. Thus Régis’s result generalizes to instances that are close to the interval case.

However, it is unlikely that EGC constraints admit a polynomial kernel.

**Theorem 7.** EGC-CONS(holes) does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.

**Proof.** We use the following result of Quimper et al. [54]: Given a CNF formula $F$ on $k$ variables, one can construct in polynomial time an EGC constraint $C_F$ such that

1. for each value $v$ of $C_F$, $\text{dom}(v) = \{0, i_v\}$ for an integer $i_v > 0$,
2. $i_v > 1$ for at most $2k$ values $v$, and
3. $F$ is satisfiable if and only if $C_F$ is consistent.

Thus, the number of holes in $C_F$ is at most twice the number of variables of $F$.

We observe that this result provides a polynomial parameter reduction from SAT($\text{vars}$) to EGC-CONS(holes). As remarked in the proof of Theorem 1 SAT($\text{vars}$) is known to have no polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$. Hence the theorem follows.
AtMost-NValue instance $\mathcal{I} = (X, D, \text{dom}, N)$, with $X = \{x_1, \ldots, x_{15}\}$, $N = 6$, $D = \{1, \ldots, 14\}$, and $\text{dom}(x_1) = \{1, 2\}$, $\text{dom}(x_2) = \{2, 3, 10\}$, etc.

### 6.2 A Polynomial Kernel for NValue Constraints

Beldiceanu [3] and Bessière et al. [6] decompose NVALUE constraints into two other global constraints: AtMost-NValue and AtLeast-NValue, which require that at most $N$ or at least $N$ values are used for the variables in $X$, respectively. The Consistency problem is NP-complete for NVALUE and AtMost-NValue constraints, and polynomial time solvable for AtLeast-NValue constraints.

In this subsection, we will present a polynomial kernel for AtMost-NValue-Cons(holes).

**AtMost-NValue-Cons(holes)**

**Instance:** An instance $\mathcal{I} = (X, D, \text{dom}, N)$, where $X = \{x_1, \ldots, x_n\}$ is a set of variables, $D$ is a totally ordered set of values, $\text{dom} : X \rightarrow 2^D$ is a map assigning a non-empty domain $\text{dom}(v) \subseteq D$ to each variable $x \in X$, and an integer $N$.

**Parameter:** The integer $k = \#\text{holes}(X)$.

**Question:** Is there a set $S \subseteq D$, $|S| \leq N$, such that for every variable $x \in X$, $\text{dom}(x) \cap S \neq \emptyset$?

**Theorem 8.** The problem AtMost-NValue-Cons(holes) has a polynomial kernel. In particular, an AtMost-NValue constraint with $k$ holes can be reduced in linear time to a consistency-equivalent AtMost-NValue constraint with $O(k^2)$ variables and $O(k^2)$ domain values.

The proof of the theorem is based on a kernelization algorithm that we will describe in the remaining part of this section.

We say that a subset of $D$ is an interval if it has no hole. An interval $I = [v_1, v_2]$ of a variable $x$ is an inclusion-wise maximal hole-free subset of its domain. Its left endpoint $l(I)$ and right endpoint $r(I)$ are the values $v_1$ and $v_2$, respectively. Fig. 3 gives an example of an instance and its interval representation. We assume that instances are given by a succinct description, in which the domain of a variable is a set of values in the domain of each variable, a succinct representation can be computed in linear time.

Also, in a variant of AtMost-NValue-Cons(holes) where $D$ is not part of the input, we may construct $D$ by sorting the set of all endpoints of intervals in time $O((n + k) \log(n + k))$. Since, w.l.o.g., a solution contains only endpoints of intervals, this step does not compromise the correctness.

A greedy algorithm by Beldiceanu [3] checks the consistency of an AtMost-NValue constraint in linear time when all domains are intervals (i.e., $k = 0$). Further, Bessière et al. [5] have shown that Consistency (and Enforcing HAC) is fixed-parameter tractable, parameterized by the number of holes, for all constraints for which bound consistency can be enforced in polynomial time. A simple algorithm for checking the consistency of AtMost-NValue goes over all instances obtained from restricting the domain of each variable to one of its intervals, and executes the algorithm of [3] for each of these $2^k$ instances. The running time of this algorithm is clearly bounded by $O(2^k \cdot |\mathcal{I}|)$.

Let $\mathcal{I} = (X, D, \text{dom}, N)$ be an instance for the consistency problem for AtMost-NValue constraints. The algorithm is more intuitively described using the interval representation of the instance. The friends of an interval $I$ are the other intervals of $I$’s variable. An interval is optional if it has at least one friend, and required otherwise. For a value $v \in D$, let $\text{iv}(v)$ denote the set of intervals containing $v$.

A solution for $\mathcal{I}$ is a subset $S \subseteq D$ of at most $N$ values such that there exists an instantiation assigning the values in $S$ to the variables in $X$. The algorithm may detect for some value $v \in D$, that, if
the problem has a solution, then it has a solution containing $v$. In this case, the algorithm selects $v$, i.e., it eliminates all variables whose domain contains $v$, it eliminates $v$ from $D$, and it decrements $N$ by one. The algorithm may detect for some value $v \in D$, that, if the problem has a solution, then it has a solution not containing $v$. In this case, the algorithm discards $v$, i.e., it eliminates $v$ from every domain and from $D$. (Note that no new holes are created since $D$ is replaced by $D \setminus \{v\}$.) The algorithm may detect for some variable $x$, that every solution for $(X \setminus \{x\}, D, \text{dom}|_{X \setminus \{x\}}, N)$ contains a value from $\text{dom}(x)$. In that case, it removes $x$.

The algorithm sorts the intervals by increasing right endpoint (ties are broken arbitrarily). Then, it exhaustively applies the following three reduction rules.

**Red-$\subseteq$:** If there are two intervals $I, I'$ such that $I' \subseteq I$ and $I'$ is required, then remove the variable of $I$ (and its intervals).

**Red-Dom:*** If there are two values $v, v' \in D$ such that $\text{ivl}(v') \subseteq \text{ivl}(v)$, then discard $v'$.

**Red-Unit:** If $|\text{dom}(x)| = 1$ for some variable $x$, then select the value in $\text{dom}(x)$.

In the example from Fig. 2 Red-$\subseteq$ removes the variables $x_3$ and $x_8$ because $x_{10} \subseteq x'_6$ and $x_7 \subseteq x_8$. Red-Dom removes the values 1 and 5. Red-Unit selects 2, which deletes variables $x_1$ and $x_2$, and Red-Dom removes 3 from $D$. The resulting instance is depicted in Fig. 3.

After none of the previous rules apply, the algorithm scans the remaining intervals from left to right (i.e., by increasing right endpoint). An interval that has already been scanned is either a leader or a follower of a subset of leaders. Informally, for a leader $L$, if a solution contains $r(L)$, then there is a solution containing $r(L)$ and the right endpoint of each of its followers.

The algorithm scans the first intervals up to, and including, the first required interval. All these intervals become leaders.

The algorithm then continues scanning intervals one by one. Let $I$ be the interval that is currently scanned and $I_p$ be the last interval that was scanned. The active intervals are those that have already been scanned and intersect $I_p$. A popular leader is a leader that is either active or has at least one active follower.

- If $I$ is optional, then $I$ becomes a leader, the algorithm continues scanning intervals until scanning a required interval; all these intervals become leaders.

- If $I$ is required, then it becomes a follower of all popular leaders that do not intersect $I$ and that have no follower intersecting $I$. If all popular leaders have at least two followers, then set $N := N - 1$ and merge the second-last follower of each popular leader with the last follower of the corresponding leader; i.e., for every popular leader, the right endpoint of its second-last follower is set to the right endpoint of its last follower, and then the last follower of every popular leader is removed.

After having scanned all the intervals, the algorithm exhaustively applies the reduction rules Red-$\subseteq$, Red-Dom, and Red-Unit again.

In the example from Fig. 3 the interval of variable $x_6$ is merged with $x_3$’s interval, and the interval of $x_7$ with the interval of $x_{10}$. Red-Dom then removes the values 7 and 8, resulting in the instance depicted in Fig. 4.

The correctness and performance guarantee of this kernelization algorithm are proved in A. In particular, for the correctness, we prove that a solution $S$ for an instance $I$ can be obtained from a solution
$S'$ for an instance $I'$ that is obtained from $I$ by one merge-operation by adding to $S'$ one value that is common to all second-last followers of the popular leaders that were merged. We can easily bound the number of leaders by $4k$ and we prove that each leader has at most $4k$ followers. Since each interval is a leader or a follower of at least one leader, this bounds the total number of intervals by $O(k^2)$. Using the succinct description of the domains, the size of the kernel is $O(k^2)$. We also give some details for a linear-time implementation of the algorithm.

**Remark:** Denoting $\var(v) = \{x \in X : v \in \text{dom}(x)\}$, Rule **Red-Dom** can be generalized to discard any $v' \in D$ for which there exists a $v \in D$ such that $\var(v') \subseteq \var(v)$ at the expense of a higher running time.

The kernel for AtMost-NValue-Cons(holes) can now be used to derive a kernel for NValue-Cons(holes).

**Corollary 1.** The problem NValue-Cons(holes) has a polynomial kernel. In particular, an NValue constraint with $k$ holes can be reduced in $O(|X| + |D|)^{\omega/2})$ time to a consistency-equivalent NValue constraint with $O(k^2)$ variables and $O(k^2)$ domain values, where $\omega < 2.3729$ is the exponent of matrix multiplication.

**Proof.** As in [6], we determine the largest possible value for $N$ if its domain were the set of all integers. This can be done in $O(|X| + |D|)^{\omega/2})$ time by computing a maximum matching in the graph whose vertices are $X \cup D$ with an edge between $x \in X$ and $v \in D$ iff $v \in \text{dom}(x)$. Suppose this largest possible value is $N^+$. Now, set $\text{dom}(N) := \{v \in \text{dom}(N) : v \leq N^+\}$, giving a consistency-equivalent NValue constraint. Note that if this constraint has a legal instantiation $\alpha$ with $\alpha(N) \leq \max(\text{dom}(N))$, then it has a legal instantiation $\alpha'$ with $\alpha'(N) = \max(\text{dom}(N))$. Therefore, it suffices to compute a kernel for AtMost-NValue-Cons(holes) with the same variables and domains and value $N = \max(\text{dom}(N))$, and return it. \qed

### 6.3 Improved FPT Algorithm using the Kernel

Using the kernel from Theorem 8 and the simple algorithm described in the beginning of this section, one arrives at a $O(2^k k^2 + |I|)$ time algorithm for checking the consistency of an AtMost-NValue constraint. Borrowing ideas from the kernelization algorithm, we now reduce the exponential dependency on $k$ in the running time. The speed-ups due to this branching algorithm and the kernelization algorithm lead to a speed-up for enforcing HAC for AtMost-NValue constraints (by Corollary 2) and for enforcing HAC for NValue constraints (by the decomposition of [6]).

**Theorem 9.** The Consistency problem for AtMost-NValue constraints admits a $O(\var^k k^2 + |I|)$ time algorithm, where $k$ is the number of holes in the domains of the input instance $I$, and $\var = \frac{1+\sqrt{5}}{2} < 1.6181$.

**Proof.** The first step of the algorithm invokes the kernelization algorithm and obtains an equivalent instance $I'$ with $O(k^2)$ intervals in time $O(|I|)$.

Now, we describe a branching algorithm checking the consistency of $I'$. Let $I_1$ denote the first interval of $I'$ (in the ordering by increasing right endpoint). $I_1$ is optional. Let $I_1$ denote the instance obtained from $I'$ by selecting $r(I_1)$ and exhaustively applying Reduction Rules **Red-Dom** and **Red-Unit**. Let $I_2$ denote the instance obtained from $I'$ by removing $I_1$ (if $I_1$ had exactly one friend, this friend becomes required) and exhaustively applying Reduction Rules **Red-Dom** and **Red-Unit**. Clearly, $I'$ is consistent if and only if $I_1$ or $I_2$ is consistent.

Note that both $I_1$ and $I_2$ have at most $k - 1$ holes. If either $I_1$ or $I_2$ has at most $k - 2$ holes, the algorithm recursively checks whether at least one of $I_1$ and $I_2$ is consistent. If both $I_1$ and $I_2$ have exactly $k - 1$ holes, we note that in $I'$,

1. $I_1$ has one friend,
2. no other optional interval intersects $I_1$, and
3. the first interval of both $I_1$ and $I_2$ is $I_f$, which is the third optional interval in $I'$ if the second optional interval is the friend of $I_1$, and the second optional interval otherwise.

Thus, the instance obtained from $I_1$ by removing $I_1$'s friend and applying **Red-Dom** and **Red-Unit** may differ from $I_2$ only in $N$. Let $s_1$ and $s_2$ denote the number of values smaller than $r(I_f)$ that have been selected to obtain $I_1$ and $I_2$ from $I'$, respectively. If $s_1 \leq s_2$, then the non-consistency of $I_1$ implies the non-consistency of $I_2$. Thus, the algorithm need only recursively check whether $I_1$ is consistent. On the other hand, if $s_1 > s_2$, then the non-consistency of $I_2$ implies the non-consistency of $I_1$. Thus, the algorithm need only recursively check whether $I_2$ is consistent.
The recursive calls of the algorithm may be represented by a search tree labeled with the number of holes of the instance. As the algorithm either branches into only one subproblem with at most \(k - 1\) holes, or two subproblems with at most \(k - 1\) and at most \(k - 2\) holes, respectively, the number of leaves of this search tree is \(T(k) \leq T(k - 1) + T(k - 2)\), with \(T(0) = T(1) = 1\). Using standard techniques in the analysis of exponential time algorithms (see, e.g., [32, Chapter 2] and [35, Lemma 2.3]), it suffices to find a value \(c > 1\) for the base of the exponential function bounding the running time, that we will minimize, such that
\[
c^{k-1} + c^{k-2} \leq c^k
given for all \(k \geq 0\),
or, equivalently, such that
\[
c + 1 \leq c^2
\]
It now suffices to find the unique positive real root of \(x^2 - x - 1\), which is \(\varphi = \frac{1 + \sqrt{5}}{2} < 1.6181\), to determine the optimal value of \(c\) for this analysis.

Since the size of the search tree is \(O(\varphi^k)\) and the number of operations executed at each node of the search tree is \(O(k^2)\), the running time of the branching algorithm can be upper bounded by \(O(\varphi^k k^2)\).

For the example of Fig. 3 the instances \(I_1\) and \(I_2\) are computed by selecting the value 4, and removing the interval \(x_3\), respectively. The reduction rules select the value 9 for \(I_1\) and the values 6 and 10 for \(I_2\). Both instances start with the interval \(x_{11}\), and the algorithm recursively solves \(I_1\) only, where the values 12 and 13 are selected, leading to the solution \{4, 9, 12, 13\} for the kernelized instance, which corresponds to the solution \{2, 4, 7, 9, 12, 13\} for the instance of Fig. 2.

**Corollary 2.** HAC for an AtMost-NValue constraint can be enforced in time \(O(\varphi^k k^2 \cdot |D| + |I| \cdot |D|)\), where \(k\) is the number of holes in the domains of the input instance \(I = (X, D, dom, N)\), and \(\varphi = \frac{1 + \sqrt{5}}{2} < 1.6181\).

**Proof.** We first remark that if a value \(v\) can be filtered from the domain of a variable \(x\) (i.e., \(v\) has no support for \(x\)), then \(v\) can be filtered from the domain of all variables, as for any legal instantiation \(\alpha\) with \(\alpha(x') = v, x' \in X \setminus \{x\}\), the assignment obtained from \(\alpha\) by setting \(\alpha(x) := v\) is a legal instantiation as well. Also, filtering the value \(v\) creates no new holes as the set of values can be set to \(D \setminus \{v\}\).

Now we enforce HAC by applying \(O(|D|)\) times the algorithm from Theorem 9. Assume the instance \(I = (X, D, dom, N)\) is consistent. If \((X, D, dom, N - 1)\) is consistent, then no value can be filtered. Otherwise, check, for each \(v \in D\), whether the instance obtained from selecting \(v\) is consistent and filter \(v\) if this is not the case.

Using the same reasoning as in Corollary 1 we now obtain the following corollary for NValue.

**Corollary 3.** HAC for an NValue constraint can be enforced in time \(O((\varphi^k k^2 + |X| + |D|)^{\omega/2} \cdot |D|)\), where \(k\) is the number of holes in the domains of the input instance \(I = (X, D, dom, N)\), \(\varphi = \frac{1 + \sqrt{5}}{2} < 1.6181\), and \(\omega < 2.3729\) is the exponent of matrix multiplication.

### 7 Bayesian Reasoning

Bayesian networks (BNs) have emerged as a general representation scheme for uncertain knowledge [53]. A BN models a set of stochastic variables, the independencies among these variables, and a joint probability distribution over these variables. For simplicity we consider the important special case where the stochastic variables are Boolean. The variables and independencies are modeled in the BN by a directed acyclic graph \(G = (V, A)\), the joint probability distribution is given by a table \(T_v\) for each node \(v \in V\) which defines a probability \(T_v[U]\) for each possible instantiation \(U = (d_1, \ldots, d_s)\) of the parents \(v_1, \ldots, v_s\) of \(v\) in \(G\). The probability \(Pr(U)\) of a complete instantiation \(U\) of the variables of \(G\) is given by the product of \(T_v[U]\) over all variables \(v\). We consider the problem Positive-BN-Inference which takes as input a Boolean BN \((G, T)\) and a variable \(v\), and asks whether \(Pr(v = \text{true}) > 0\). The problem is NP-complete [19] and moves from NP to \#P if we ask to compute \(Pr(v = \text{true})\) [59]. The problem can be solved in polynomial time if the BN is singly connected, i.e., if there is at most one undirected path between any two variables [52]. It is natural to parametrize the problem by the number of variables one must delete in order to make the BN singly connected (the deleted variables form a loop cutset). This yields the following parameterized problem.
**Positive-BN-Inference** (loop cutset size)

*Instance:* A Boolean BN \((G, T)\), a variable \(v\), and a loop cutset \(S\) of size \(k\).

*Parameter:* The integer \(k\).

*Question:* Is \(\Pr(v = \text{true}) > 0\)?

Again we also state a related recognition problem.

**Rec-Positive-BN-Inference** (loop cutset size)

*Instance:* A Boolean BN \((G, T)\) and an integer \(k \geq 0\).

*Parameter:* The integer \(k\).

*Question:* Does \((G, T)\) has a loop cutset of size \(\leq k\)?

Now, **Positive-BN-Inference** (loop cutset size) is easily seen to be fixed-parameter tractable as we can determine whether \(\Pr(v = \text{true}) > 0\) by taking the maximum of \(\Pr(v = \text{true} \mid U)\) over all \(2^k\) possible instantiations of the \(k\) cutset variables, each of which requires processing of a singly connected network. However, although fixed-parameter tractable, it is unlikely that the problem admits a polynomial kernel.

**Theorem 10.** **Positive-BN-Inference** (loop cutset size) does not admit a polynomial kernel unless \(\text{NP} \subseteq \text{coNP}/\text{poly}\).

*Proof.* We give a polynomial parameter transformation from \(\text{SAT}(\text{vars})\) and apply Theorem 2. The reduction is based on the reduction from \(3\text{SAT}\) given by Cooper [19]. Let \(F\) be a CNF formula on \(n\) variables. We construct a BN \((G, T)\) such that for a variable \(v\) we have \(\Pr(v = \text{true}) > 0\) if and only if \(F\) is satisfiable. Cooper uses input nodes \(u_i\) for representing variables of \(F\), clause nodes \(c_i\) for representing the clauses of \(F\), and conjunction nodes \(d_i\) for representing the conjunction of the clauses. For instance, if \(F\) has three clauses and four variables, then Cooper’s reduction produces a BN \((G, T)\) where \(G\) has the following shape:

![Diagram](image)

Clearly, the input nodes form a loop cutset of \(G\). However, in order to get a polynomial parameter transformation from \(\text{SAT}(\text{vars})\) we must allow in \(F\) that clauses contain an arbitrary number of literals, not just three. If we apply Cooper’s reduction directly, then a single clause node \(c_i\) with many parents requires a table \(T_{c_i}\) of exponential size. To overcome this difficulty we simply split clause nodes \(c_i\) containing more than 3 literals into several clause nodes, as indicated below, where the last one feeds into a conjunction node \(d_i\).

![Diagram](image)

It remains to observe that the set of input nodes \(E = \{u_1, \ldots, u_n\}\) still form a loop cutset of the constructed BN, hence we have indeed a polynomial parameter transformation from \(\text{SAT}(\text{vars})\) to **Positive-BN-Inference** (loop cutset size). The result follows by Theorem 2.

Let us now turn to the recognition problem **Rec-Positive-BN-Inference** (loop cutset size).

**Proposition 3.** **Rec-Positive-BN-Inference** (loop cutset size) admits a polynomial kernel with \(O(k^2)\) nodes.
Proof. Let \((G, T), k\) be an instance of \textbf{Rec-Positive-BN-Inference}\(\text{(loop cutset size)}\). We note that loop cutsets of \((G, T)\) are just the so-called feedback vertex sets of \(G\). Hence we can apply a known kernelization algorithm for feedback vertex sets \([19]\) to \(G\) and obtain a kernel \((G', k)\) with at most \(O(k^2)\) many vertices. We translate this into an instance \((G', T', k')\) of \textbf{Rec-Positive-BN-Inference}\(\text{(loop cutset size)}\) by taking an arbitrary table \(T'\).

\(\square\)

8 Nonmonotonic Reasoning

Logic programming with negation under the stable model semantics is a well-studied form of nonmonotonic reasoning \([90, 16]\). A (normal) logic program \(P\) is a finite set of rules \(r\) of the form

\[
h \leftarrow a_1 \land \cdots \land a_m \land \neg b_1 \land \cdots \land \neg b_n
\]

where \(h, a_i, b_i\) are atoms, where \(h\) forms the head and the \(a_i, b_i\) from the body of \(r\). We write \(H(r) = h\), \(B^+(r) = \{a_1, \ldots, a_m\}\), and \(B^-(r) = \{b_1, \ldots, b_n\}\). Let \(I\) be a finite set of atoms. The \(GF\) reduct \(P^I\) of a logic program \(P\) under \(I\) is the program obtained from \(P\) by removing all rules \(r\) with \(B^-(r) \cap I \neq \emptyset\), and removing from the body of each remaining rule \(r'\) all literals \(\neg b\) with \(b \in I\). \(I\) is a \textit{stable model} of \(P\) if \(I\) is a minimal model of \(P^I\), i.e., if (i) for each rule \(r \in P^I\) with \(B^+(r) \subseteq I\) we have \(H(r) \subseteq I\), and (ii) there is no proper subset of \(I\) with this property. The \textit{undirected dependency graph} \(U(P)\) of \(P\) is formed as follows. We take the atoms of \(P\) as vertices and add an edge \(x \rightarrow y\) between two atoms \(x, y\) if there is a rule \(r \in P\) with \(H(r) = x\) and \(y \in B^+(r)\), and we add a path \(x \leftarrow u \leftarrow y\) if \(H(r) = x\) and \(y \in B^+(r)\) \((u\) is a new vertex of degree 2). The \textit{feedback width} of \(P\) is the size of a smallest set \(V\) of atoms such that every cycle of \(U(P)\) runs through an atom in \(V\) (such a set \(V\) is called a \textit{feedback vertex set}).

A fundamental computational problems is \textit{Stable Model Existence (SME)}, which asks whether a given normal logic program has a stable model. The problem is well-known to be \textit{NP}-complete \([47]\). Gottlob et al. \([11]\) considered the following parameterization of the problem and showed fixed-parameter tractability (see \([31]\) for generalizations).

\[
\text{SME(\text{feedback width})}
\]

\textit{Instance:} A logic program \(P\) and feedback vertex set \(V\) of size \(k\).

\textit{Parameter:} The integer \(k\).

\textit{Question:} Does \(P\) have a stable model?

Again we also state a related recognition problem.

\textbf{Rec-SME(\text{feedback width})}

\textit{Instance:} A logic program \(P\) and an integer \(k \geq 0\).

\textit{Parameter:} The integer \(k\).

\textit{Question:} Does \(P\) have a feedback vertex set of size at most \(k\)?

We show that the result of Gottlob et al. \([11]\) cannot be strengthened towards a polynomial kernel.

\textbf{Theorem 11.} SME(\text{feedback width}) does not admit a polynomial kernel unless \(NP \subseteq coNP/poly\).

\textit{Proof.} We give a polynomial parameter transformation from \textbf{SAT}(\text{vars}) to SME(\text{feedback width}) using a construction of Niemelä \([49]\). Given a CNF formula \(F\) on \(n\) variables, we construct a logic program \(P\) as follows. For each variable \(x\) of \(F\) we take two atoms \(x\) and \(\bar{x}\) and include the rules \((\bar{x} \leftarrow \neg x)\) and \((x \leftarrow \neg \bar{x})\); for each clause \(C\) of \(F\) we take an atom \(c\) and include for each positive literal \(a\) of \(C\) the rule \((c \leftarrow a)\), and for each negative literal \(\neg a\) of \(C\) the rule \((c \leftarrow \bar{a})\); finally, we take two atoms \(s\) and \(f\) and include the rule \((f \leftarrow \neg f \land \neg s)\) and for each clause \(C\) of \(F\) the rule \((s \leftarrow \neg c)\). The formula \(F\) is satisfiable if and only if \(P\) has a stable model \([49]\). It remains to observe that each cycle of \(U(P)\) runs through a vertex in \(V = \{x, \bar{x} : x \in \text{vars}(F)\}\), hence the feedback width of \(P\) is at most \(2n\). Hence we have a polynomial parameter transformation from \textbf{SAT}(\text{vars}) to SME(\text{feedback width}). The result follows by Theorem 2.

\(\square\)

Using a similar approach as for Proposition 3 we can establish the following result.

\textbf{Proposition 4.} Rec-SME(\text{feedback width}) admits a polynomial kernel with \(O(k^2)\) atoms.
9 Conclusion

We have provided the first theoretical evaluation of the guarantees and limits of polynomial-time preprocessing for hard AI problems. In particular we have established super-polynomial kernel lower bounds for many problems, providing firm limitations for the power of polynomial-time preprocessing for these problems. On the positive side, we have developed an efficient linear-time kernelization algorithm for the consistency problem for AtMost-NValue constraints, and have shown how it can be used to speed up the complete propagation of NValue and related constraints.

Subsequent to our work, Fellows et al. [29] investigated the parameterized complexity and kernelization for various parameterizations of Abductive Reasoning. Their kernelization results were mostly negative, showing that many parameterizations for the Abduction problem have no polynomial kernels unless \( \text{NP} \subseteq \text{coNP}/\text{poly} \). Similarly negative are the kernelization results of Bäckström et al. [2] for planning problems, parameterized by the length of the plan.

We conclude from these results that in contrast to many optimization problems (see Section 1), typical AI problems do not admit polynomial kernels. Our results suggest the consideration of alternative approaches. For example, it might still be possible that some of the considered problems admit polynomially sized Turing kernels, i.e., a polynomial-time preprocessing to a Boolean combination of a polynomial number of polynomial kernels. In the area of optimization, parameterized problems are known that do not admit polynomial kernels but admit polynomial Turing kernels [30]. This suggests a theoretical and empirical study of Turing kernels for the AI problems considered.

A Appendix: Proof of Theorem 8

In this appendix, we prove Theorem 8 by proving the correctness of the algorithm, upper bounding the size of the kernel, and analyzing its running time.

Let \( I' = (X', D', \text{dom}', N') \) be the instance resulting from applying one operation of the kernelization algorithm to an instance \( I = (X, D, \text{dom}, N) \). An operation is an instruction which modifies the instance: \textbf{Red-} \( \subseteq \), \textbf{Red-Dom}, \textbf{Red-Unit}, and \textbf{merge}. We show that there exists a solution \( S \) for \( I \) if and only if there exists a solution \( S' \) for \( I' \). A solution is \textit{nice} if each of its elements is the right endpoint of some interval. Clearly, for every solution, a nice solution of the same size can be obtained by shifting each value to the next right endpoint of an interval. Thus, when we construct \( S' \) from \( S \) (or vice-versa), we may assume that \( S \) is nice.

Reduction Rule \textbf{Red-} \( \subseteq \) is sound because a solution for \( I \) is a solution for \( I' \) and vice-versa, because any solution \( I' \) contains a value \( v \) of \( I \subseteq I' \), as \( I \) is required. Reduction Rule \textbf{Red-Dom} is correct because if \( v' \in S \), then \( S' := (S \setminus \{v'\}) \cup \{v\} \) is a solution for \( I' \) and for \( I \). Reduction Rule \textbf{Red-Unit} is obviously correct (\( S = S' \cup \text{dom}(x) \)).

After having applied these 3 reduction rules, observe that the first interval is optional and contains only one value. Suppose the algorithm has started scanning intervals. By construction, the following properties apply to \( I' \):

\textbf{Property 1}. A follower does not intersect any of its leaders.

\textbf{Property 2}. If \( I, I' \) are two distinct followers of the same leader, then \( I \) and \( I' \) do not intersect.

Before proving the correctness of the \textbf{merge} operation, let us first show that the subset of leaders of a follower is not empty.

\textbf{Claim 1}. Every interval that has been scanned is either a leader or a follower of at least one leader.
Proof. First, note that Red-Dom ensures that each domain value in D is the left endpoint of some interval and the right endpoint of some interval. We show that when an interval I is scanned it either becomes a leader or a follower of at least one leader. By induction, assume this is the case for all previously scanned intervals. Denote by \( I_p \) the interval that was scanned prior to I. If \( I_p \) or I is optional, then I becomes a leader. Suppose I and \( I_p \) are required. We have that \( l(I) > l(I_p) \), otherwise I would have been removed by Red-\( \subseteq \). By Rule Red-Dom, there is some interval \( I_\ell \) with \( r(I_\ell) = (I_p) \). If \( I_\ell \) is a leader, I becomes a follower of \( I_\ell \); otherwise I becomes a follower of \( I_\ell \)'s leader.

We will now prove the correctness of the merge operation. Recall that \( \mathcal{I}' \) is an instance obtained from \( \mathcal{I} \) by one application of the merge operation. Let I denote the interval that is scanned when the merge operation is applied. At this computation step, each popular leader has at least two followers and the algorithm merges the last two followers of each popular leader and decrements N by one. Let \( F_2 \) denote the set of all intervals that are the second-last follower of a popular leader, and \( F_1 \) the set of all intervals that are the last follower of a popular leader before merging. Let M denote the set of merged intervals. Clearly, every interval of \( F_1 \cup F_2 \cup M \) is required as all followers are required.

**Lemma 1.** Every interval in \( F_1 \) intersects \( l(I) \).

Proof. Let \( I_1 \in F_1 \). By construction, \( r(I_1) \in I \), as I becomes a follower of every popular leader that has no follower intersecting I, and no follower has a right endpoint larger than \( r(I) \). Moreover, \( l(I_1) \leq l(I) \) as no follower is a strict subset of I with Red-\( \subseteq \) and the fact that all followers are required.

The correctness of the merge operation will follow from the next two lemmas.

**Lemma 2.** If \( S \) is a nice solution for \( \mathcal{I} \), then there exists a solution \( S' \) for \( \mathcal{I}' \) with \( S' \subseteq S \).

Proof. Let \( I^- \) be the interval of \( F_2 \) with the largest right endpoint. Let L be a leader of \( I^- \). By construction and Red-\( \subseteq \), \( L \) is a leader of I as well and is therefore popular. Let \( t_1 \in S \cap I \) be the smallest value of \( S \) that intersects I and let \( t_2 \in S \cap I^- \) be the largest value of \( S \) that intersects \( I^- \). By Property[2] we have that \( t_2 < t_1 \).

Claim 2. The set \( S \) contains no value \( t_0 \) such that \( t_2 < t_0 < t_1 \).

Proof. For the sake of contradiction, suppose \( S \) contains a value \( t_0 \) such that \( t_2 < t_0 < t_1 \). Since \( S \) is nice, \( t_0 \) is the right endpoint of some interval \( I_0 \). Since \( t_2 \) is the rightmost value intersecting \( S \) and any interval in \( F_2 \), the interval \( I_0 \) is not in \( F_2 \). Since \( I_0 \) has already been scanned, and was scanned after every interval in \( F_2 \), the interval \( I_0 \) is in \( F_1 \). However, by Lemma[1] \( I_0 \) intersects \( l(I) \). Since no scanned interval has a larger right endpoint than I, we have that \( t_0 \in S \cap I \), which contradicts the fact that \( t_1 \) is the smallest value in \( S \cap I \) and that \( t_0 < t_1 \).

Claim 3. Suppose \( I_1 \in F_1 \) and \( I_2 \in F_2 \) are the last and second-last follower of a popular leader \( L' \), respectively. Let \( M_{12} \in M \) denote the interval obtained from merging \( I_2 \) with \( I_1 \). If \( I_2 \in I_1 \), then \( t_1 \in M_{12} \).

Proof. For the sake of contradiction, assume \( t_2 \in I_2 \), but \( t_1 \notin M_{12} \). As \( t_2 < t_1 \), we have that \( t_1 > r(M_{12}) = r(I_1) \). But then \( S \) is not a solution as \( S \cap I_1 = \emptyset \) by Claim[2] and the fact that \( t_2 < l(I_1) \).

Claim 4. If \( I' \) is an interval with \( t_2 \in I' \), then \( I' \in F_2 \cup F_1 \).

Proof. First, suppose \( I' \) is a leader. As every leader has at least two followers when I is scanned, \( I' \) has two followers whose left endpoint is larger than \( r(I') \geq t_2 \) (by Property[1]) and smaller than \( l(I) \leq t_1 \) (by Red-\( \subseteq \)). Thus, at least one of them is included in the interval \((t_2, t_1)\) by Property[2] which contradicts \( S \) being a solution by Claim[2].

Similarly, if \( I' \) is a follower of a popular leader, but not among the last two followers of any popular leader, Claim[3] leads to a contradiction as well.

Finally, if \( I' \) is a follower, but has no popular leader, then it is to the left of some popular leader, and thus to the left of \( t_2 \).

Consider the set \( T_2 \) of intervals that intersect \( t_2 \). By Claim[4], we have that \( T_2 \subseteq F_2 \cup F_1 \). For every interval \( I' \in T_2 \cap F_2 \), the corresponding merged interval of \( \mathcal{I}' \) intersects \( t_1 \) by Claim[3]. For every interval \( I' \in T_2 \cap F_1 \), and every interval \( I'' \in F_2 \) with which \( I' \) is merged, \( S \) contains some value \( x \in I'' \) with \( x < t_2 \). Thus, \( S' := S \setminus \{t_2\} \) is a solution for \( \mathcal{I}' \).
Lemma 3. If \( S' \) is a nice solution for \( \mathcal{I}' \), then there exists a solution \( S \) for \( \mathcal{I} \) with \( S' \subseteq S \).

Proof. As in the previous proof, consider the step where the kernelization algorithm applies the merge operation. Recall that the currently scanned interval is \( I \). Let \( F_1 \) and \( F_2 \) denote the set of all intervals that are the second-last and last follower of a popular leader before merging, respectively. Let \( M \) denote the set of merged intervals.

By Lemma 1, every interval of \( M \) intersects \( l(I) \). On the other hand, every interval of \( \mathcal{I}' \) whose right endpoint intersects \( I \) is in \( M \), by construction. Thus, \( S' \) contains the right endpoint of some interval of \( M \). Let \( t_1 \) denote the smallest such value, and let \( I_1 \) denote the interval of \( \mathcal{I} \) with \( r(I_1) = t_1 \) (due to Red-\(<\), there is a unique such interval). Let \( I_2 \) denote the interval of \( \mathcal{I} \) with the smallest right endpoint such that there is a leader \( L \) whose second-last follower is \( I_2 \) and whose last follower is \( I_1 \), and let \( t_2 := r(I_2) \).

Claim 5. Let \( I'_1 \in F_1 \) and \( I'_2 \in F_2 \) be two intervals from \( \mathcal{I} \) that are merged into one interval \( M'_{12} \) of \( \mathcal{I}' \).

If \( t_1 \in M'_{12} \), then \( t_2 \in I'_2 \).

Proof. For the sake of contradiction, suppose \( t_1 \in M'_{12} \) but \( t_2 \notin I'_2 \). We consider two cases. In the first case, \( I'_2 \subseteq (t_2, l(I'_1)) \). But then, \( I'_2 \) would have become a follower of \( L \), which contradicts that \( I_1 \) is the last follower of \( L \). In the second case, \( r(I'_2) < t_2 \). But then, \( I_1 \) is a follower of the same leader as \( I'_2 \), as \( l(I'_1) \leq l(I_1) \), and thus \( I_1 = I'_1 \). By the definition of \( I_2 \), however, \( t_2 = r(I_2) \leq r(I'_2) \), a contradiction.

By the previous claim, a solution \( S \) for \( \mathcal{I} \) is obtained from a solution \( S' \) for \( \mathcal{I}' \) by setting \( S := S' \cup \{t_2\} \).

After having scanned all the intervals, Reduction Rules Red-\(<\), Red-Dom, and Red-Unit are applied again, and we have already proved their correctness.

Thus, the kernelization algorithm returns an equivalent instance. To bound the kernel size by a polynomial in \( k \), let \( \mathcal{I}^* = (V^*, D^*, \text{dom}^*, N^*) \) be the instance resulting from applying the kernelization algorithm to an instance \( \mathcal{I} = (V, D, \text{dom}, N) \).

Property 3. The instances \( \mathcal{I} \) and \( \mathcal{I}^* \) have at most \( 2k \) optional intervals.

Property 3 holds for \( \mathcal{I} \) as every optional interval of a variable \( x \) is adjacent to at least one hole and each hole is adjacent to two optional intervals of \( x \). It holds for \( \mathcal{I}^* \) as the kernelization algorithm introduces no holes.

Lemma 4. The instance \( \mathcal{I}^* \) has at most \( 4k \) leaders.

Proof. Consider the unique step of the algorithm that creates leaders. An optional interval is scanned, the algorithm continues scanning intervals until scanning a required interval, and all these scanned intervals become leaders. As every interval is scanned only once, we have that for every optional interval there are at most 2 leaders. By Property 3 the number of leaders is thus at most \( 4k \).

Lemma 5. Every leader has at most \( 4k \) followers.

Proof. Consider all steps where a newly scanned interval becomes a follower, but is not merged with another interval. In each of these steps, the popular leader \( L_r \) with the rightmost right endpoint either

(a) has no follower and intersects \( I \), or

(b) has no follower and does not intersect \( I \), or

(c) has one follower and intersects \( I \).

Now, let \( L \) be some leader and let us consider a period where no optional interval is scanned. Let us bound the number of intervals that become followers of \( L \) during this period without being merged with another interval. If the number of followers of \( L \) increases in Situation (a), it does not increase in Situation (a) again during this period, as no other follower of \( L \) may intersect \( I \). If situation (b) occurs, Situation (b) does not occur again during this period, as \( I \) becomes a follower of \( L_r \). Moreover, the number of followers of \( L \) does not increase during this period in Situation (c) after Situation (b) has occurred, as no other follower of \( L \) may intersect \( I \). After Situation (c) occurs, the number of followers of \( L \) does not increase in Situation (c) again during this period, as no other follower of \( L \) may intersect \( I \). Thus, at most 2 followers are added to \( L \) in each period. As the first scanned interval is optional, Property 3 bounds the number of periods by \( 2k \). Thus, \( L \) has at most \( 4k \) followers.
As, by Claim 1, every interval of $I^*$ is either a leader or a follower of at least one leader, Lemmas 4 and 5 imply that $I^*$ has $O(k^2)$ intervals, and thus $|X^*| = O(k^2)$. Because of Reduction Rule Red-Dom, every value in $D^*$ is the right endpoint and the left endpoint of some interval, and thus, $|D^*| = O(k^2)$.

This bounds the kernel size, and we will now show that the algorithm can be implemented to run in linear time. First, using a counting sort algorithm with satellite data (see, e.g., [20]), the initial sorting of the $n+k$ intervals can be done in time $O(n + |D| + k)$. To facilitate the application of Red-$\subseteq$, counting sort is used a second time to also sort by increasing left endpoint the sets of intervals with coinciding right endpoint. An optimized implementation applies Red-$\subseteq$, Red-Dom and Red-Unit simultaneously in one pass through the intervals, as one rule might trigger another. To guarantee a linear running time for the scan-and-merge phase of the algorithm, only the first follower of a leader stores a pointer to the leader; all other followers store a pointer to the previous follower. This proves Theorem 8.

Acknowledgments

Both authors acknowledge support by the European Research Council, grant reference 239962 (COMPLEX REASON). Serge Gaspers is the recipient of an Australian Research Council Discovery Early Career Researcher Award (project number DE120101761). NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References


