Complexity of Minimum-Size Arc-Inconsistency Explanations*

Christian Bessiere
CNRS, University of Montpellier, France
bessiere@lirmm.fr

Clément Carbonnel
CNRS, University of Montpellier, France

Martin C. Cooper
IRIT, University of Toulouse, France

Emmanuel Hebrard
LAAS CNRS, Toulouse, France

Abstract

Explaining the outcome of programs has become one of the main concerns in AI research. In constraint programming, a user may want the system to explain why a given variable assignment is not feasible or how it came to the conclusion that the problem does not have any solution. One solution to the latter is to return to the user a sequence of simple reasoning steps that lead to inconsistency. Arc consistency is a well-known form of reasoning that can be understood by a human. We consider explanations as sequences of propagation steps of a constraint on a variable (i.e. the ubiquitous revise function in arc-consistency algorithms) that lead to inconsistency. We characterize several cases for which providing a shortest such explanation is easy: For instance when constraints are binary and variables have maximum degree two. However, these polynomial cases are tight. For instance, providing a shortest explanation is NP-hard when constraints are binary and the maximum degree is three, even if the number of variables is bounded. It remains NP-hard on trees, despite the fact that arc consistency is a decision procedure on trees. The problem is not even FPT-approximable unless the FPT $\neq$ W[2] hypothesis is false.

Keywords. Constraint programming, constraint propagation, minimum explanations, complexity

1 Introduction

Constraint Programming (CP) is a technology that allows the user to solve combinatorial problems formulated as constraint networks. A constraint network is characterized by a set of variables taking values in a finite domain that are subject to constraints. Constraints restrict the combinations of values that specified subsets of variables can take. One of the advantages of using CP is that in general constraint networks represent the problem to solve much more compactly than would an integer linear program or a SAT formula. CP formulations are not only compact but also easy to understand for the user, as they remain close to a natural language formulation of the problem thanks to the expressiveness of constraints. However, nowadays, AI becomes even more demanding in terms of *explainability*. A user may want to not only understand the formulation of their problem as a constraint network but also to be provided with explanations of why this assignment is the only solution, why that value is not feasible, or why the problem does not have any solution.

An *abductive explanation* for a proposition is often defined as a *prime implicant* of that proposition, i.e. an implicant that cannot be generalized further. For instance, an explanation of a

*This is an extended version of a paper with the same title that appeared in CP 2022.

Machine Learning model's prediction is often defined as a minimal subset of features that entails that prediction [18, 12]. Similarly, a *minimal unsatisfiable core* (irreducible unsatisfiable subset of constraints) can be seen as an abductive explanation for unsatisfiability since it is a sufficient and minimal reason for unsatisfiability. At least one term of an abductive explanation must be relaxed in order to change the outcome. This is the viewpoint adopted in many existing approaches. For instance by providing explanations in the form of minimal sets of choices of the user that lead to the given value removal (e.g., product configuration [1]), or explanations in the form of minimal sets of constraints that lead to an inconsistency [13]. The purpose of such approaches is to help the user to repair the inconsistency, not to let them understand why it is an inconsistency.

Intuitively, an explanation is more than a sufficient condition. In particular, if an abductive explanation answers the "*why*" question, it does not answer the "*how*" question. An intuitive definition of an explanation also covers the *demonstration* of how the considered cause has that consequence. For instance, consider the Zebra problem, a logic puzzle whereby the house whose owner has a pet zebra is to be discovered through a set of clues. When solving this problem, we may want to let the user understand why the zebra is necessarily in the middle house, not by providing a set of constraints of the problem that rule out all other positions for the zebra, but by displaying a sequence of simple reasoning steps that lead to that conclusion (See Example 1 for more details). This notion of *demonstrative explanation* can be related to proof systems and to the notion of formal proof. A formal proof better explains unsatisfiability by making every step explicit down to axiomatic definitions. For instance, a refutation proof log using the *reverse unit propagation* (RUP) system [10, 11] allows one to formally verify the unsatisfiability of a formula, provided that one can "trust" the application of the unit propagation rule, i.e. trust that a given formula that is refutable via unit propagation is indeed unsatisfiable. This is valid in the context of formal proof verification where each unit propagation refutation can be checked efficiently. However, this may produce very long proofs in which each step might be too complex for an explanation to a non-expert.

We would therefore want to produce demonstrative explanations, allowing a trustworthy verification, however with minimal requirements on the recipient of the explanation. This is of course impossible in general. In [20, 2], the choice was made to provide explanations in the form of sequences of inferences performed by constraint propagation. We consider an even simpler, and also incomplete, proof system: Arc Consistency. Arc consistency has often been considered as a sufficiently strong inference technique on applications where the human is in the loop (configuration [14], logic puzzles [20]).

Our goal is to analyze the complexity of providing the *shortest* possible explanation of arc inconsistency of a problem. We show that on normalized networks of binary constraints, when variables have degree two or domains are Boolean, finding a shortest explanation of arc inconsistency is polynomial time. The same holds on networks of unbounded arity Boolean constraints when constraints are clauses and the network is Berge-acyclic or when constraints are bijunctive. However, the problem is NP-hard in general and the polynomial cases above are tight. Finding a shortest explanation of arc inconsistency is NP-hard on normalized networks of binary constraints as soon as variables have degree three, even if the number of variables is bounded (even though the problem can obviously be solved in polynomial time). It is also NP-hard on normalized networks of binary constraints if domain size is bounded by three. Perhaps more surprisingly, it remains NP-hard on trees, where arc consistency is known to be a decision procedure. On networks of unbounded arity Boolean constraints, finding a shortest explanation of arc inconsistency is NP-hard on non-clausal Berge-acyclic constraint networks. It is also NP-hard when constraints are Horn, anti-Horn, or affine, even when the maximum arity is three. We also show that there is little hope that we can efficiently find short (if not shortest) explanations: the problem is not FPT-approximable unless the FPT $\neq$ W[2] hypothesis is false.

2 Background and Definitions

The *constraint satisfaction problem* (*CSP*) involves finding solutions to a constraint network. A *constraint network* is defined as a set of n variables $X = X_1, \ldots, X_n$, a set of domains $D = \{D(X_1), \ldots, D(X_n)\}$, where $D(X_i)$ is the finite set of values that X_i can take, and a set C of constraints. A constraint $c \in C$ is a pair (R, S), where R is the relation that specifies which combinations of values (tuples) the variables in the set S, referred to as *scope*, of c are allowed to take. We assume without loss of generality that no variable occurs twice in the scope of a constraint. A binary constraint is a constraint whose scope contains two variables, let us say X_i and X_j. In that case, we use $c(X_i, X_j)$ to denote the relation that specifies the combinations of values that the variables (X_i, X_j) can take. A constraint network is *binary* when all the constraints are binary. A binary constraint network is *normalized* if there is at most one constraint per pair of variables. Given $k \in \mathbb{N}$, a variable has *degree* k if it is involved in exactly k constraints. A *degree-k constraint network* is a constraint network in which all variables have degree at most k.

Arc consistency (*AC*) is the basic form of inference reasoning on constraint networks. Given a constraint (R, S) with $X_i \in S$, a tuple τ of values on S is called a *support* on constraint (R, S) for a value $v \in D(X_i)$ if and only if $\tau[X_i] = v$, $\tau[X_j] \in D(X_j)$ for all variables $X_j \in S$, and $\tau \in R$. A value v in $D(X_i)$ is arc consistent if and only if v has a support on every constraint in C involving X_i. A network is arc consistent if all values in all domains are arc consistent. The $revise(X_i, c)$ operation removes from $D(X_i)$ all values that do not have a support on c. When c is binary and has scope (X_i, X_j), $revise(X_i, c)$ is often denoted by $X_i \overset{c}{\leftarrow} X_j$ in the following. If enforcing arc consistency on a network (that is, applying $revise()$ operations until a fix point is reached) leads to a domain wipe out (i.e. an empty domain), we say that the network is *arc inconsistent*.

Definition 1 (Arc-Inconsistency Explanation) *An* arc-inconsistency explanation *for a constraint network is a sequence of revise() operations such that one of the domains is wiped out by the execution of the sequence of revise() operations.*

Definition 2 (Shortest Arc-Inconsistency Explanation) *The* shortest arc-inconsistency explanation *problem consists in finding an arc-inconsistency explanation of minimum length.*

Example 1 (Explaining the Zebra puzzle) *The Zebra puzzle, which may (or may not) be due to Lewis Carroll, has a well known associated constraint network [19] whereby, for each of the 5* house colors*,* nationalities*,* beverages*,* cigarette brands*, and* pets*, we have a variable whose value is the number of the corresponding house (e.g., X_{Zebra} stands for the house where the Zebra lives). The constraints are statements such as "The Englishman lives in the red house" or "The Old Gold smoker owns snails". Moreover, each house has a unique colour, its owner has a unique nationality, drinks a unique beverage, smokes a unique brand, and has a unique pet.*

*Applying arc consistency on this constraint network detects that "**the Kools smoker does not live in the 2nd house**". A demonstrative explanation would be a list of deduction steps necessary to prove that statement, each step corresponding to an arc-consistency* revision*. For instance, consider the following demonstrative explanation, where the first line is a known fact, and each subsequent line in italic corresponds to a constraint from which we can deduce a knew fact (in bold). Next to each fact we give the corresponding domain knowledge in the CSP, and next to each constraint we give the corresponding arc-consistency revision (as well as the corresponding CSP constraint):*

The Norwegian lives in the first house.	$X_{Norwegian} \in \{1\}$
Since *the Norwegian lives next to the blue house,*	$X_{Blue} \overset{\lvert x-y\rvert=1}{\leftarrow} X_{Norwegian}$
the 2nd house is blue.	$X_{Blue} \in \{2\}$
Since *the 2nd house has a single color,*	$X_{Yellow} \overset{x \neq y}{\leftarrow} X_{Blue}$
the 2nd house is not yellow.	$X_{Yellow} \in \{1, 3, 4, 5\}$
Since *Kools are smoked in the yellow house,*	$X_{Kools} \overset{x=y}{\leftarrow} X_{Yellow}$
the Kools smoker does not live in the 2nd house	$X_{Kools} \in \{1, 3, 4, 5\}$

In other words, the sequence of revise() operations $\langle X_{Blue} \overset{|x-y|=1}{\leftarrow} X_{Norwegian}, X_{Yellow} \overset{x \neq y}{\leftarrow} X_{Blue}, X_{Kools} \overset{x=y}{\leftarrow} X_{Yellow} \rangle$ *is a demonstrative explanation of "**the Kools smoker does not live in the 2nd house**".*

3 Binary Normalized Constraint Networks: Structure

We show that on binary normalized networks, if all variables are involved in no more than two constraints, finding shortest arc-inconsistency explanations is polynomial. We then show that this class is tight. As soon as we allow a variable to be in the scope of three constraints, the problem becomes NP-hard, even if the constraint network has no more than four variables. Perhaps even more surprising, the problem is NP-hard on networks structured as trees, despite arc consistency being a decision procedure on trees. Finally, we show that there is little hope that we can efficiently find short explanations when they exist: the problem is not FPT-approximable (within any factor) on binary tree-structured constraint networks unless the Gap-ETH is false, and on binary constraint networks unless FPT = W[2].

3.1 Tractability on degree-2 constraint networks

Theorem 1 Shortest Arc-Inconsistency Explanation *is solvable in time polynomial in the number of variables and values when restricted to binary normalized networks with maximum degree two.*

Proof. A constraint network of maximum degree two is composed of unconnected cycles and paths. A shortest arc-inconsistency explanation clearly always concerns only one of the connected components of the network. An exhaustive search over all connected components only increases complexity by at most a linear factor. Since, furthermore a path can be viewed as a degenerate cycle (a cycle in which one constraint disallows no tuples), it follows that we only need consider the case of a single cycle.

Without loss of generality, we suppose that the cycle is $X_1, \ldots, X_n$, with constraints $c(X_i, X_{i+1})$, where here, and in the rest of the proof, addition within subscripts is understood to be modulo n. (For example X_{n+1} refers to X_1.) We say that *revise*() operations are clockwise (resp. anticlockwise) if they are of the form $X_{i+1} \leftarrow X_i$ (resp. $X_i \leftarrow X_{i+1}$). We say that a pair of *revise*() operations R_1, R_2 commute if the two sequences R_1R_2 and R_2R_1 produce the same result. It is easy to verify that the only *revise*() operations that may not commute are those in which the destination variable of one is the source variable of the other. Furthermore, *revise*() operations in opposite directions (clockwise and anticlockwise) always commute, even $X_i \leftarrow X_{i+1}$ and $X_{i+1} \leftarrow X_i$. Thus the only pairs of *revise*() operations that do not commute are of the form $\{X_i \leftarrow X_{i+1}, X_{i+1} \leftarrow X_{i+2}\}$. What's more, if we have the operations $X_i \leftarrow X_{i+1}, X_{i+1} \leftarrow X_{i+2}$ in this order, then the set of value-eliminations cannot decrease if we inverse the order of these two operations.

In a shortest arc-inconsistency explanation E, a *revise*() operation must be useful: it must eliminate a domain value whose elimination is essential for a subsequent *revise*() operation or for the final domain wipe-out. In the former case, the operation $X_i \leftarrow X_{i+1}$ must be followed later in the sequence by $X_{i-1} \leftarrow X_i$. Let S be the sequence of *revise*() operations in E between the operation $X_i \leftarrow X_{i+1}$ and the next subsequent occurrence of $X_{i-1} \leftarrow X_i$. By the above discussion on commutativity, we can shift the operation $X_i \leftarrow X_{i+1}$ just after S without *decreasing* the set of value-eliminations since S does not contain $X_{i-1} \leftarrow X_i$. In this way, we can group together all the anticlockwise *revise*() operations to form a sequence of anticlockwise operations on consecutive edges in the cycle. The same argument holds for clockwise operations which can be grouped together to form a sequence of clockwise operations on consecutive edges in the cycle.

An obvious observation is that a shortest arc-inconsistency explanation is necessarily of length bounded by nd, where d is the maximum domain size, since at least one elimination must occur at each operation. Moreover, there are up to n possible starting points for the sequence of clockwise

(resp. anticlockwise) operations. Hence a shortest explanation can be found in polynomial time, by exhaustive search over the starting points and lengths of the clockwise/anticlockwise sequences. □

3.2 Intractability on constraint networks with four variables

The result in Theorem 1 is tight. We show that as soon as we allow variables to have degree 3, finding a shortest explanation becomes NP-hard. This is true even if the number of variables is bounded by four. (Observe that all binary normalized constraint networks on three variables have degree at most 2.) We use a reduction from CLIQUE, which is NP-complete [15], to prove hardness: given in input an undirected graph $G = (V, E)$ and an integer k, is there $S \subseteq V$ such that $|S| \geq k$ and for all $i \neq j \in S$, $\{i, j\} \in E$?

It is noticeable that constraint networks with a bounded number of variables have a constant number of possible *revise*() operations available at each step – only 12 in the case of four variables. This is not sufficient to make the problem of finding a shortest explanation easy.

Theorem 2 SHORTEST ARC-INCONSISTENCY EXPLANATION *is NP-hard, even on binary normalized networks with four variables.*

Lemma 1 *Deciding whether there exists an arc-inconsistency explanation of length smaller than or equal to k is NP-complete, even on binary normalized networks with four variables.*

Proof. Membership. Given a sequence of *revise*() operations, we decide whether this sequence is an arc-inconsistency explanation by executing each *revise*() in the order of the sequence and checking whether one of the domains is empty after these executions. As constraints have bounded arity, executing a *revise*() operation is polynomial, so the whole process is polynomial.

Completeness. We reduce the CLIQUE problem to the problem of deciding whether there is an arc-inconsistency explanation of length at most $3n + 3$ for a constraint network. Let $G = (V, E)$ be a graph with set of vertices $V = \{1, \ldots, n\}$.

We construct the constraint network P_G with four variables $X = \{X_1, X_2, X_3, X_4\}$, all with domain $\{(p, i) : p \in 0..n + 1, i \in 1..n\} \cup \{s_t : t \in 1..k + 1\}$.

We build the set of constraints

$$C = \{c_1(X_1, X_2), c_2(X_1, X_3), c_3(X_2, X_3), X_1 = X_4, X_2 = X_4, X_3 = X_4\}$$

with:

$$\begin{aligned}
c_1(X_1, X_2) &= \{((p-1, i), (p, i)) : p \in [0, n+1], \forall i \neq p \in [1, n]\} \\
&\quad \cup \{((p-2, i), (p, i)) : p \in [0, n+1], \forall i \in [1, n]\} \\
&\quad \cup \{(s_t, s_t) : t \in [1, k+1]\} \\
c_2(X_1, X_3) &= \{(p-1, i), (p, i)) : p \in [0, n+1], \forall i \in [1, n]\} \cup \{(s_{t-1}, s_t) : t \in [1, k+1]\} \\
c_3(X_2, X_3) &= (\{(p, i) : p \in [0, n+1], i \in [1, n]\}^2 \setminus \\
&\qquad \{((n+1, i), (n+1, j)) : i = j \ \vee \ \{i, j\} \in E\}) \\
&\quad \cup \{s_t : t \in [1, k+1]\}^2
\end{aligned}$$

The constraint network for a graph with 3 vertices and the edges $\{1, 2\}$ and $\{2, 3\}$ is shown in Figure 1.

We first show that if G contains a k-clique, then, there exists an arc-inconsistency explanation of length $3n + 3$ for P_G.

Assume that the set of vertices S is a k-clique. We build the sequence $R(S)$ of *revise*() operations in the following way, and we will say that $R(S)$ *encodes* the set S, since there is a one-to-one mapping between subsets $S \subseteq V$ and this type of explanation:

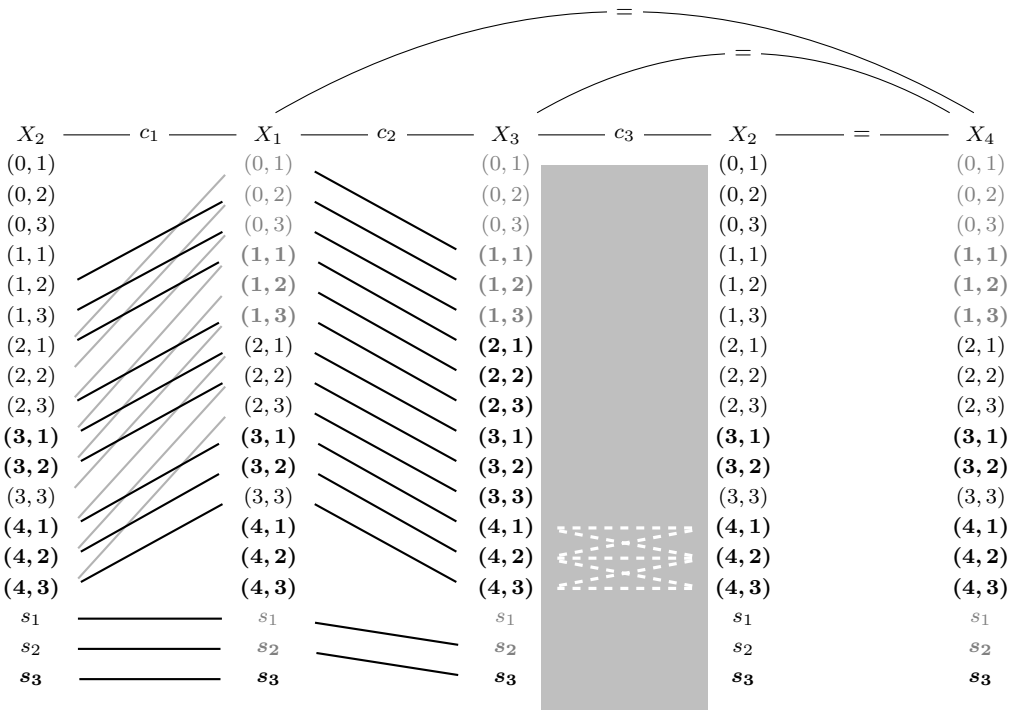

Figure 1: The constraint network P_G, reduction of the graph $G = (\{1, 2, 3\}, \{(1, 2), (2, 3)\})$. Solid edges represent allowed tuples for c_1 and c_2, while dashed edges stand for forbidden tuples of c_3 (the forbidden tuples of the form $(s_t, (p, i))$ and $((p, i), s_t)$ are not represented). The equality constraints are not represented. There are two explanations of arc inconsistency of length 12. The first *encodes* the clique $\{2, 3\}$ with the *revise*() operations $X_2 \overset{c_1}{\leftarrow} X_1$, $X_3 \overset{c_2}{\leftarrow} X_1$, $X_3 \overset{c_2}{\leftarrow} X_1$ at positions 1, 4, and 7 in the sequence. The second *encodes* the clique $\{1, 2\}$ with the revision operations $X_3 \overset{c_2}{\leftarrow} X_1$, $X_3 \overset{c_2}{\leftarrow} X_1$, $X_2 \overset{c_1}{\leftarrow} X_1$ at positions 1, 4, and 7. The latter exaplanation is illustrated through colors: gray values are removed at the first iteration, **bold gray values** are removed at the second iteration and only **bold values** remain after the third iteration. One can check that the sequence of revision $\langle X_2 \overset{c_1}{\leftarrow} X_1, X_3 \overset{c_2}{\leftarrow} X_1, X_2 \overset{c_3}{\leftarrow} X_3 \rangle$, on values in bold font, does indeed produce a wipeout.

- If $p \notin S$, the $(3p-2)$th element in the sequence $R(S)$ is $X_2 \overset{c_1}{\leftarrow} X_1$, the $(3p-1)$th element is $X_4 \overset{=}{\leftarrow} X_2$, and the $(3p)$th element is $X_1 \overset{=}{\leftarrow} X_4$.
- If $p \in S$, the $(3p-2)$th element in the sequence $R(S)$ is $X_3 \overset{c_2}{\leftarrow} X_1$, the $(3p-1)$th element is $X_4 \overset{=}{\leftarrow} X_3$, and the $(3p)$th element is $X_1 \overset{=}{\leftarrow} X_4$.

Then the last three elements in the sequence $R(S)$ are $X_2 \overset{c_1}{\leftarrow} X_1$, $X_3 \overset{c_2}{\leftarrow} X_1$, and $X_2 \overset{c_3}{\leftarrow} X_3$. In the following, the subsequence composed of the $(3p-2)$th, the $(3p-1)$th, and the $(3p)$th operations (that is, $\langle X_2 \overset{c_1}{\leftarrow} X_1, X_4 \overset{=}{\leftarrow} X_2, X_1 \overset{=}{\leftarrow} X_4\rangle$ or $\langle X_3 \overset{c_2}{\leftarrow} X_1, X_4 \overset{=}{\leftarrow} X_3, X_1 \overset{=}{\leftarrow} X_4\rangle$), is called the pth *iteration*.

Before each iteration $p \in \{1, \ldots, n\}$ of three domain revisions, the invariants are:

$$(q, i) \notin D(X_1)\ \forall q < p-1, \forall i \in [1, n] \tag{1}$$

$$s_j \in D(X_1) \iff k+1 \geq j > |S \cap \{0, \ldots, p-1\}| \tag{2}$$

$$(p-1, i) \in D(X_1) \iff i \in S \cup \{p, \ldots, n\} \tag{3}$$

All invariants are verified before entering iteration $p = 1$. For each one, we show that if it is true before entering iteration $p \geq 1$ then it remains true before entering iteration $p+1$.

Invariant 1: Notice that a value $(q, i) \in D(X_2)$ (resp. $D(X_3)$) is only supported by values $(q', i) \in D(X_1)$ such that $q' < q$. If Invariant 1 is true before iteration p, then when revising the domain of either X_2 or X_3, $D(X_1)$ contains no value (q, i) with $q < p-1$ and therefore all values $(p-1, i)$ are removed from $D(X_2)$ (resp. $D(X_3)$). The revisions w.r.t. equality constraints make sure that this is propagated back to $D(X_1)$.

Invariant 2. Notice that a value $s_t \in D(X_3)$ is only supported by value $s_{t-1} \in D(X_1)$, whereas the tuple (s_t, s_t) is a support in all other constraints. If Invariant 2 is true before iteration p, then either $p \in S$ in wich case the operation $X_3 \overset{c_2}{\leftarrow} X_1$ removes the value s_j (with $j = |S \cap \{0, \ldots, p-1\}| + 1$) from $D(X_3)$ since the value s_{j-1} was its only support and is not in $D(X_1)$; or $X_2 \overset{c_1}{\leftarrow} X_1$ removes no s value and $|S \cap \{0, \ldots, p-1\}|$ does not change.

Invariant 3. For any $i \in [1, n]$:

If $i > p$, then we have $(p-1, i) \in D(X_1)$ which is a support for (p, i) w.r.t. c_1 and c_2 hence (p, i) is not removed and the invariant holds because $i \in \{p+1, \ldots, n\}$.

If $i < p$, notice that by Invariant 1, the tuple $((p-2, i), (p, i))$ cannot be a support for $(p, i) \in D(X_2)$. Therefore, both constraints c_1 and c_2 have the same unique potential support for the value (p, i) (in $D(X_2)$ and $D(X_3)$ respectively): $((p-1, i), (p, i))$. So we have: "$(p-1, i) \in D(X_1)$ before iteration p" iff "$(p, i) \in D(X_1)$ before iteration $p+1$". In addition, $i \in S \cup \{p, \ldots, n\} \iff i \in S \cup \{p+1, \ldots, n\}$ because $i < p$. Finally, by the induction hypothesis we have "$(p-1, i) \in D(X_1)$ before iteration p" iff $i \in S \cup \{p, \ldots, n\}$, and hence by transitivity: "$(p, i) \in D(X_1)$ before iteration $p+1$" iff $i \in S \cup \{p+1, \ldots, n\}$.

If $i = p$, there are two cases: If $p \in S$, then the first operation at iteration p is $X_3 \overset{c_2}{\leftarrow} X_1$, (p, i) is not removed since it is supported by $(p-1, i)$, and the invariant is true at iteration $p+1$ since $i \in S$. If $p \notin S$, then the first operation at iteration p is $X_2 \overset{c_1}{\leftarrow} X_1$, (p, i) is removed, and the invariant is true at iteration $p+1$ since $i \notin S \cup \{p+1, \ldots, n\}$.

After n iterations, the invariants hold for $p = n+1$ (i.e. after the $3n$-th operation) and hence $D(X_1)$ is $\{(n, i) \forall i \in S\} \cup \{(n+1, i) \forall i\} \cup \{s_{k+1}\}\}$. The call to $X_2 \overset{c_1}{\leftarrow} X_1$ then yields $D(X_2) = \{(n+1, i) \forall i \in S\} \cup \{s_{k+1}\}$ and the call to $X_3 \overset{c_2}{\leftarrow} X_1$ yields $D(X_3) = \{(n+1, i) \forall i \in S\}$. Therefore, the last call to $X_2 \overset{c_3}{\leftarrow} X_3$ produces a wipe-out, since on layer $n+1$, the remaining vertices stand for a clique of G and the allowed tuples are non-edges of G.

We then prove that if G does not contain any k-clique, then the shortest arc-inconsistency explanation for P_G is of length strictly greater than $3n+3$. We first show that the shortest explanation must use constraint c_3, then we show that only explanations that encode a set $S \subseteq V$ (as defined above) such that S is a clique of size k of G can be the shortest.

Suppose first that the constraint c_3 does not appear in any *revise*() of the explanation. By construction, the values (p, i) are organized in layers, where a layer q is the set of values $(q, i), \forall i$. Wiping out the domain of a variable requires removing the $n+2$ layers 0 to $n+1$ from its domain. Moreover, removing a layer q from X_1 (resp. X_2 or X_3) requires having already removed layer $q+1$ (resp. $q-1$) from X_2 or X_3 (resp. X_1). Removing a layer q from X_4 requires having already removed layer q from X_1, X_2, or X_3. Hence, removing a layer q from a variable requires iteratively removing layers 0 to $q-1$ or $n+1$ down to $q+1$ from other variables. The only way to do that is to execute a sequence of *revise*() operations looping on a cycle of variables $\{X_1, X_2, X_4\}$, or on $\{X_1, X_3, X_4\}$, or both. Looping in the order $\langle X_1 \overset{c_1}{\leftarrow} X_2, X_4 \overset{=}{\leftarrow} X_1, X_2 \overset{=}{\leftarrow} X_4\rangle$ or $\langle X_1 \overset{c_2}{\leftarrow} X_3, X_4 \overset{=}{\leftarrow} X_1, X_3 \overset{=}{\leftarrow} X_4\rangle$ removes layers from $n+1$ down to q, whereas looping in the order $\langle X_2 \overset{c_1}{\leftarrow} X_1, X_4 \overset{=}{\leftarrow} X_2, X_1 \overset{=}{\leftarrow} X_4\rangle$ or $\langle X_3 \overset{c_2}{\leftarrow} X_1, X_4 \overset{=}{\leftarrow} X_3, X_1 \overset{=}{\leftarrow} X_4\rangle$ removes layers from 0 up to q. We can then compute the number of *revise*() operations necessary to remove a layer q from a variable given the order in which we loop. If we execute *revise*() operations in the orders $\langle X_1 \overset{c_1}{\leftarrow} X_2, X_4 \overset{=}{\leftarrow} X_1, X_2 \overset{=}{\leftarrow} X_4\rangle$ or $\langle X_1 \overset{c_2}{\leftarrow} X_3, X_4 \overset{=}{\leftarrow} X_1, X_3 \overset{=}{\leftarrow} X_4\rangle$, layer q is removed from the domain of X_1 (resp. X_2/X_3, or X_4) in $3(n+1-q)+1$ operations (resp. $3(n+1-q)+3$, or $3(n+1-q)+2$ operations). If we execute *revise*() operations in the orders $\langle X_2 \overset{c_1}{\leftarrow} X_1, X_4 \overset{=}{\leftarrow} X_2, X_1 \overset{=}{\leftarrow} X_4\rangle$ or $\langle X_3 \overset{c_2}{\leftarrow} X_1, X_4 \overset{=}{\leftarrow} X_3, X_1 \overset{=}{\leftarrow} X_4\rangle$, layer q is removed from the domain of X_1 (resp. X_2/X_3, or X_4) in $3q+3$ operations (resp. $3q+1$, or $3q+2$ operations). As wiping out a domain requires, given a value q, removing layers 0 to q from below and layers $n+1$ down to $q+1$ from above, we conclude that a domain wipe out, on either X_1, X_2, X_3, or X_4, requires at least $3n+4$ *revise*() operations. This means that there does not exist any arc-inconsistency explanation for P_G of length smaller than or equal to $3n+3$ if we do not use c_3 in the explanation.

Hence, we must use c_3. However, by construction of c_3, every value in $D(X_2)$ (resp. $D(X_3)$) is supported as long as at least one value (p, i) with $p \in [0, n]$, and any value s_t is in the domain of $D(X_3)$ (resp. $D(X_2)$). In other words, to remove a layer with a revise on c_3, the domains of X_2 and X_3 must only contain (p, i) values from layer $n+1$. This requires us to remove all layers from 0 to $n-1$ from X_1 by executing n loops by a sequence of *revise*() operations $\langle X_2/X_3 \leftarrow X_1, X_4 \leftarrow X_2/X_3, X_1 \leftarrow X_4\rangle$ for a cost of $3n$ operations, plus a $X_2 \overset{c_1}{\leftarrow} X_1$ and a $X_3 \overset{c_2}{\leftarrow} X_1$ to remove layer n from X_2 and X_3. In other words, it must be a sequence of *revise*() operations that *encodes* a set, i.e., $R(S)$ for some set $S \subseteq \{1, \ldots, n\}$. Now, suppose that S is not a clique and let i_1 and i_2 be two non-adjacent vertices in S. By Invariant 3, at iteration $n+1$, we have $(n, i_1) \in D(X_1)$ and $(n, i_2) \in D(X_1)$ and hence after operations $X_2 \overset{c_1}{\leftarrow} X_1$ and $X_3 \overset{c_2}{\leftarrow} X_1$, we have $(n+1, i_1) \in D(X_2)$ and $(n+1, i_2) \in D(X_3)$. Therefore, neither $X_2 \overset{c_3}{\leftarrow} X_3$ nor $X_3 \overset{c_3}{\leftarrow} X_2$ would fail, and at least one more operation is necessary. Finally, suppose that $|S| < k$. Then by Invariant 2, at iteration $n+1$, we have $s_k \in D(X_1)$ and hence after operations $X_2 \overset{c_1}{\leftarrow} X_1$ and $X_3 \overset{c_2}{\leftarrow} X_1$, we have $s_{k+1} \in D(X_2)$ and $s_{k+1} \in D(X_3)$. Therefore, at least one more operation is necessary. Consequently, the number of operations can be equal to $3n+3$ only if S is a clique of size k of G. □

3.3 Intractability and inapproximability on trees

We have seen in Section 3.2 that Shortest Arc-Inconsistency Explanation is already NP-hard on networks with four variables. This result does not completely settle the intractability of the problem. For example, it could be the case that a polynomial-time algorithm exists for some broad generalization of degree-2 networks that does not contain 4-cliques (for instance, networks of treewidth 2). We show that it is not the case. We use a simple reduction from Dominating Set, which is NP-complete [9], to derive NP-hardness of Shortest Arc-Inconsistency Explanation on trees: given in input an undirected graph $G = (V, E)$ and an integer k, is there $S \subseteq V$ such that $|S| \leq k$ and for all $i \in V$, there is $j \in S$ with $\{i, j\} \in E$?

The NP-hardness of Shortest Arc-Inconsistency Explanation on trees circumscribes even more tightly the degree-2 tractability island of Section 3.1. However, these NP-hardness

results do not rule out efficient approximation algorithms nor fixed-parameter tractable algorithms, which could be satisfactory for applications where only short explanations are worth computing and optimality is not strictly necessary. We again show that such desirable scenarios are not possible. We show that our reduction from Dominating Set can be used to derive (conditional) fixed-parameter inapproximability of Shortest Arc-Inconsistency Explanation.

We must briefly introduce some terminology before we can formally present the result. A minimization problem $\mathcal{P}$ is *FPT-approximable* [4] if there exist computable functions $f, \rho : \mathbb{N} \to \mathbb{R}_{\geq 1}$ such that $n \cdot \rho(n)$ is nondecreasing and an algorithm A that, given as input a non-negative integer k and an instance I of $\mathcal{P}$ that has a solution of cost at most k, computes a solution to I of cost at most $k \cdot \rho(k)$ in time $f(k) \cdot |I|^{O(1)}$. Here, ρ is the approximation ratio and f is possibly exponential. Note that if a problem is not FPT-approximable, then no such algorithm A exists for *any* computable functions f and ρ; such problems are sometimes called *completely inapproximable* [17].

Our FPT-inapproximability result is conditional on a complexity hypothesis known as the *Gap-ETH* [6, 16], which states that there exists a constant $\epsilon > 0$ such that no algorithm with runtime $2^{o(n)}$ can distinguish satisfiable 3-SAT instances from those in which no assignment satisfies a $(1 - \epsilon)$ fraction of the clauses. It has been shown recently [3] that the Minimum Dominating Set problem (which consists in finding the smallest dominating set in a graph) is not FPT-approximable unless the Gap-ETH is false.

Lemma 2 *Deciding whether there exists an arc-inconsistency explanation of length smaller than or equal to k is NP-complete, even on binary tree-structured normalized constraint networks.*

Proof. Membership. As in Lemma 1.

Completeness. We reduce the Dominating Set problem to the problem of deciding whether there is an arc-inconsistency explanation of length at most k for a constraint network.

Let $G = (V, E)$ be a graph, $V = \{v_1, \ldots, v_n\}$. We construct a constraint network P_G as follows: the set of variables is $\{Y, X_1, \ldots, X_n\}$, where the domain of Y is $\{v_1, \ldots, v_n\}$ and the domain of each X_i is $\{v_i\}$, and P_G contains a constraint $c(Y, X_i) = \{(v_j, v_i) : \{v_i, v_j\} \notin E \text{ and } v_i \neq v_j\}$ for all $i \geq 1$. An example of this reduction is shown in Figure 2. We claim that G has a dominating set of size k if and only if P_G has an arc-inconsistency explanation of length k.

If G has a dominating set S of size k, then let R be a sequence containing every operation $Y \leftarrow X_i$ such that v_i belongs to S. Since every $v_j \in V$ is dominated by some $v_k \in S$ (which is either v_j itself or one of its neighbours), by construction v_j is removed from $D(Y)$ by $Y \leftarrow X_k$. Therefore $D(Y)$ is empty at the end of the sequence and R is an arc-inconsistency explanation of length k.

Conversely, if R is a minimal arc-inconsistency explanation of P_G of length k then we can assume that it is a sequence of operations of the form $Y \leftarrow X_i$. (Since each $D(X_i)$ contains a single value, only the last operation could be $X_i \leftarrow Y$ for some i, and in that case it can be replaced with $Y \leftarrow X_i$.) Then, the set $S = \{v_i : Y \leftarrow X_i \text{ occurs in } R\}$ must be a dominating set of size k: at the end of R every $v_j \in D(Y)$ has been pruned by some operation $Y \leftarrow X_k$, and every value removed at this step is by construction dominated by v_k in G.

P_G is a tree-structured constraint network and can be constructed in polynomial time from G. Therefore, Shortest Arc-Inconsistency Explanation is NP-hard on such networks. □

Theorem 3 Shortest Arc-Inconsistency Explanation *is NP-hard and not FPT-approximable unless the Gap-ETH is false, even on binary tree-structured normalized constraint networks.*

Proof. In the reduction of the proof of Lemma 2, the size-k dominating sets of G are in one-to-one correspondence with arc-inconsistency explanations of P_G of length k. Furthermore, the dominating set corresponding to an explanation can be computed in polynomial time, so any FPT-approximation algorithm for Shortest Arc-Inconsistency Explanation translates into one for Minimum Dominating Set. By the results of [3], this would imply that the Gap-ETH is false. □

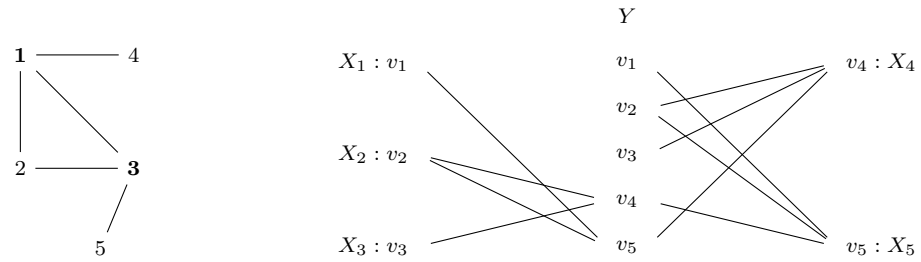

Figure 2: Left: a graph G. Right: the constraint network P_G in the proof of Lemma 2. The set $\{1, 3\}$ is a minimum dominating set of G. One can check that the sequence of revisions $\langle Y \leftarrow X_1, Y \leftarrow X_3 \rangle$ wipes Y's domain out, and that it cannot be done in a single *revise*() operation.

As a final remark, we note that the same inapproximability result can be established on general normalized binary constraint networks under the weaker (and more conventional) complexity hypothesis FPT $\neq$ W[2]. However, the proof is significantly more involved and has been moved to Appendix A for the sake of brevity.

4 Binary Normalized Constraint Networks: Domain size

We show that finding shortest arc-inconsistency explanations is polynomial on binary normalized constraint networks with Boolean domains. Again, this class is tight: As soon as we allow three values per domain, the problem becomes NP-hard.

4.1 Tractability on Boolean domains

Theorem 4 Shortest Arc-Inconsistency Explanation *is solvable in polynomial time when restricted to binary normalized networks with all domains of size at most two.*

Proof. Let $P = (X, D, C)$ be a binary constraint network with domain size at most two. We assume, without loss of generality, that all domains $D(X_i)$ are non-empty subsets of $\{0, 1\}$ and that no constraint relation is empty. Let X_r be the variable at which a domain wipe-out occurs in a shortest arc-inconsistency explanation. Complexity is only multiplied by n if we perform an exhaustive search over all possible variables X_r, so in the following we consider X_r to be fixed. We construct a directed causal graph G_P in which shortest arc-inconsistency explanations correspond to particularly simple subgraphs. In G_P there are two types of vertices: source-variable vertices X_i^s $(i = 1, \ldots, n)$, and variable-value vertices $\langle X_i, a \rangle$ $(i = 1, \ldots, n,\ a \in \{0, 1\})$. G_P has the following directed edges: $(X_i^s, \langle X_j, b \rangle)$ (for all i, j, b such that $b \in D(X_j)$ has no support in $D(X_i)$), and $(\langle X_i, a \rangle, \langle X_j, b \rangle)$ (for all i, j, a, b such that $a \in D(X_i)$ is the only support of $b \in D(X_j)$). Each arc corresponds to a possible revise operation: $(X_i^s, \langle X_j, b \rangle)$ corresponds to the elimination of b from $D(X_j)$ since it has no support in $D(X_i)$, and $(\langle X_i, a \rangle, \langle X_j, b \rangle)$ corresponds to the elimination of b from $D(X_j)$ when its unique support $a \in D(X_i)$ has been eliminated. An example of the causal graph for a simple constraint network is shown in Figure 3.

Let R be a shortest arc-inconsistency explanation, and let X_r be the variable at which a wipe-out occurs. By minimality of R, each revise operation in R eliminates a value from a domain. Indeed, each operation, except possibly the last, eliminates exactly one value otherwise there would be a domain wipe-out before the end of R. Furthermore, the only way that the final revise operation $X_r \leftarrow X_i$ of R can cause the simultaneous elimination of both 0 and 1 from $D(X_r)$ (without there already being a wipe-out at $D(X_i)$) is that (1) some value $b \in D(X_r)$ never had any support at X_i and (2) the other value $1-b$ lost its unique support a at X_i by a previous operation in R. We can deduce from (1) and (2) that just before the execution of $X_r \leftarrow X_i$, the value $1-a$ in $D(X_i)$ has no support at X_r. This implies that we can replace the last operation

$X_r \leftarrow X_i$ of R by its inverse operation $X_i \leftarrow X_r$ to produce an arc-inconsistency explanation of the same length as R but in which the final operation eliminates a single value (namely $1-a$ from $D(X_i)$ leading to a wipe-out at X_i).

For any revise operation in R, eliminating b from $D(X_j)$, there is a corresponding arc (u, v) in G_P where v is the vertex $\langle X_j, b\rangle$ and u is the cause of the elimination of b from $D(X_j)$. By the above argument, we can assume that each revise operation in R corresponds to a single elimination and hence a single arc in G_P. Let G_R be the subgraph of G_P consisting of the arcs corresponding to the operations of R. Let X_r be again the variable at which a wipe-out occurs at the end of R. For each $a \in D(X_r)$, in G_R there must be a directed path P_a from a source-variable vertex to $\langle X_r, a\rangle$. By minimality, the set of arcs of G_R is the union of the set of arcs of P_a ($a \in D(X_r)$). Since each elimination has a unique cause (given by the arc corresponding to the revise operation in R producing the elimination), the in-degree of each vertex in G_R is at most one. Furthermore, source-variable vertices have in-degree 0. It follows that P_0 and P_1 can only possibly share arcs along an initial common subpath.

If $D(X_r)$ is a singleton $\{a\}$, then G_R must be a shortest path in G_P from a source-variable vertex to $\langle X_r, a\rangle$ and hence can be found in polynomial time by a standard shortest-path algorithm. So now suppose that $D(X_r) = \{0, 1\}$. If the set of edges of P_0 and P_1 are disjoint then P_0 and P_1 must both be shortest paths in G_P from source-variable vertices to $\langle X_r, 0\rangle$ and $\langle X_r, 1\rangle$, respectively. If P_0 and P_1 have an initial common subpath, then they must diverge at some vertex v of G_P, the common initial subpath is a shortest path in G_P from a source-variable vertex to v and the remaining divergent paths P'_0 and P'_1 are shortest paths from v to $\langle X_r, 0\rangle$ and $\langle X_r, 1\rangle$, respectively. By an exhaustive search over the $O(n)$ vertices v of G_P, we can determine the paths P_0 and P_1 in polynomial time. □

It is interesting to note that in the proof of Theorem 4, one of the paths P'_0, P'_1 may actually be empty. In this case, G_R is a path (either P_0 or P_1). This occurs if the elimination of a from $D(X_r)$ triggers a sequence of revise operations that leads to the elimination of $1-a$ from $D(X_r)$. Another interesting point is that if P'_0, P'_1 are both non-empty, then the revise operations corresponding to P'_1 can all be inversed (i.e. each $X_i \leftarrow X_j$ becomes $X_j \leftarrow X_i$) and their order reversed in R to produce an alternative shortest arc-inconsistency proof $\tilde{R}$ which ends in a wipe-out at the variable X_k at which P'_0 and P'_1 diverged. For instance, the sequence $\langle X_2 \leftarrow X_1, X_3 \leftarrow X_2, X_4 \leftarrow X_3, X_2 \leftarrow X_4\rangle$ is also a shortest explanation in the example of Figure 3. In this case, $G_{\tilde{R}}$ is a path (obtained in the example by using the edge in blue ($\langle X_4, 0\rangle, \langle X_2, 1\rangle$) instead of ($\langle X_2, 0\rangle, \langle X_4, 1\rangle$)). Hence, we can optimise since the exhaustive search over vertices v is unnecessary.

4.2 Intractability on domains with three values

Theorem 5 Shortest Arc-Inconsistency Explanation *is NP-hard, even on binary normalized networks with all domains of size at most three.*

Lemma 3 *Deciding whether there exists an arc-inconsistency explanation of length smaller than or equal to k is NP-complete, even on binary normalized networks with all domains of size at most three.*

Proof. Membership. As in Lemma 1.

Completeness. We reduce the Dominating Set problem (whether a graph G has a dominating set of size at most k) to the problem of deciding whether there is an arc-inconsistency explanation of length at most $4n+k+1$ for a constraint network. Let $G = (V, E)$ be a graph with $V = \{1, \ldots, n\}$.

We construct the constraint network P_G with $5n + 2$ variables

$$X = \{X_1, \ldots, X_n, X'_1, \ldots, X'_n, X''_1, \ldots, X''_n, H_1, \ldots, H_n, B_0, \ldots, B_n, Y\}$$

all with domain $\{0, 1, 2\}$ except B_0 whose domain is $\{0\}$ and Y whose domain is $\{2\}$.

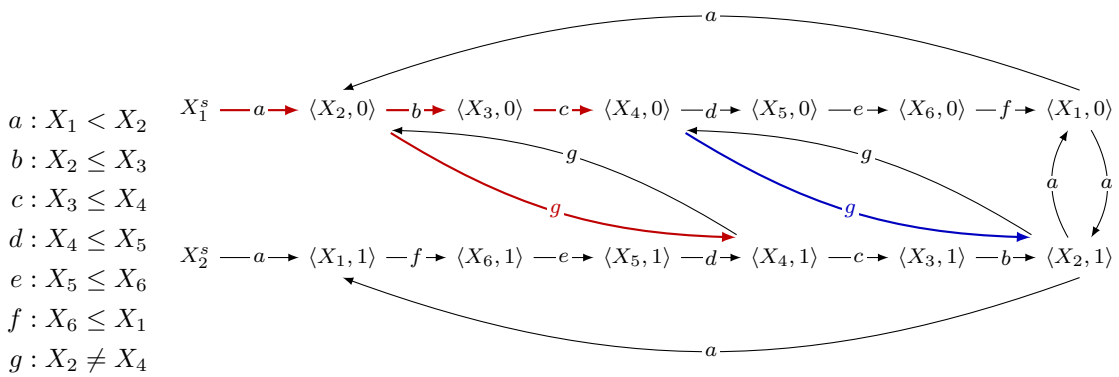

Figure 3: Left: a Boolean binary constraint network P (the domain of every variable is $\{0, 1\}$). Right: the causal graph G_P of the proof of Theorem 4. Vertices of G_P labelled with $\langle X_i, v\rangle$, or X_i^s, correspond to states where variable X_i's domain has lost value v, or is full, respectively. Any edge labelled ℓ in this graph from a vertex $\langle X_i, v\rangle$ (or X_i^s) to a vertex $\langle X_j, v\rangle$ stands for the *revise*() operation $X_j \overset{\ell}{\leftarrow} X_i$ which removes value v from $D(X_j)$. One shortest explanation involves the two paths in red originating from X_1^s and corresponds to the sequence $\langle X_2 \overset{a}{\leftarrow} X_1, X_3 \overset{b}{\leftarrow} X_2, X_4 \overset{c}{\leftarrow} X_3, X_4 \overset{g}{\leftarrow} X_2\rangle$.

We build the set of constraints

$$\begin{aligned} C \quad = \quad & \{c_1(X_i^{''}, X_i') : i \in [1, n]\} \ \cup \ \{c_2(X_i', X_i) : i \in [1, n]\} \\ & \cup \{c_3(X_i, X_j) : \{i, j\} \in E\} \ \cup \ \{c_4(X_i, H_i) : i \in [1, n]\} \\ & \cup \{c_5(B_{i-1}, H_i) : i \in [1, n]\} \ \cup \ \{c_6(H_i, B_i) : i \in [1, n]\} \ \cup \ \{B_n = Y\}\} \end{aligned}$$

where

$$c_1(X_i^{''}, X_i') = \{(0, 0)\}$$
$$c_2(X_i', X_i) = \{(0, 0), (1, 1), (2, 2)\}$$
$$c_3(X_i, X_j) = \{0, 1, 2\} \times \{0, 1, 2\} \setminus \{(0, 2), (2, 0)\}$$
$$c_4(X_i, H_i) = \{0, 1, 2\} \times \{0, 1, 2\} \setminus \{(0, 1), (1, 1)\}$$
$$c_5(B_{i-1}, H_i) = \{0, 1, 2\} \times \{0, 1, 2\} \setminus \{(0, 2), (1, 2)\}$$
$$c_6(H_i, B_i) = \{0, 1, 2\} \times \{0, 1, 2\} \setminus \{(0, 2)\}$$

The constraint network is shown in Figure 4 for a graph with $n = 4$ vertices and 4 edges.

We first prove that if G contains a k-dominating set, then there exists an arc-inconsistency explanation of length $4n + k + 1$ for P_G. Assume that the set of vertices S is a k-dominating set. We build the sequence R of *revise*() operations in the following way. The first k elements in R are $X_i' \overset{c_1}{\leftarrow} X_i^{''}$ for each vertex i in S. The k next elements in R are $X_i \overset{c_2}{\leftarrow} X_i'$, again for vertices i in S. After those $2k$ *revise*() operations, for all i in S, $D(X_i) = \{0\}$. Then, for each vertex j in $V \setminus S$, R contains $X_j \overset{c_3}{\leftarrow} X_i$, where $i \in S$ and $\{i, j\} \in E$. We know such a vertex i exists for each j because S is a dominating set. After those additional $n - k$ *revise*() operations, for all i not in S, $D(X_i) = \{0, 1\}$. The n next elements in R are $H_i \overset{c_4}{\leftarrow} X_i$, removing value 1 from $D(H_i)$ because $2 \notin D(X_i)$. The $2n$ next elements in R are $\langle H_i \overset{c_5}{\leftarrow} B_{i-1}, B_i \overset{c_6}{\leftarrow} H_i\rangle$ in increasing order of i from 1 to n. Each $H_i \overset{c_5}{\leftarrow} B_{i-1}$ removes value 2 from $D(H_i)$ if $2 \notin D(B_{i-1})$ and $B_i \overset{c_6}{\leftarrow} H_i$ removes value 2 from $D(B_i)$ if $1, 2 \notin D(H_i)$. As $B_0 = 0$ and value 1 has already been removed from all H_i's domains, those $2n$ *revise*() remove value 2 from the domain of all B_i. Finally, after these $2k + (n - k) + n + 2n = 4n + k$ *revise*() operations, the last element in R, $Y \overset{=}{\leftarrow} B_n$, wipes out the domain of Y and proves arc inconsistency.

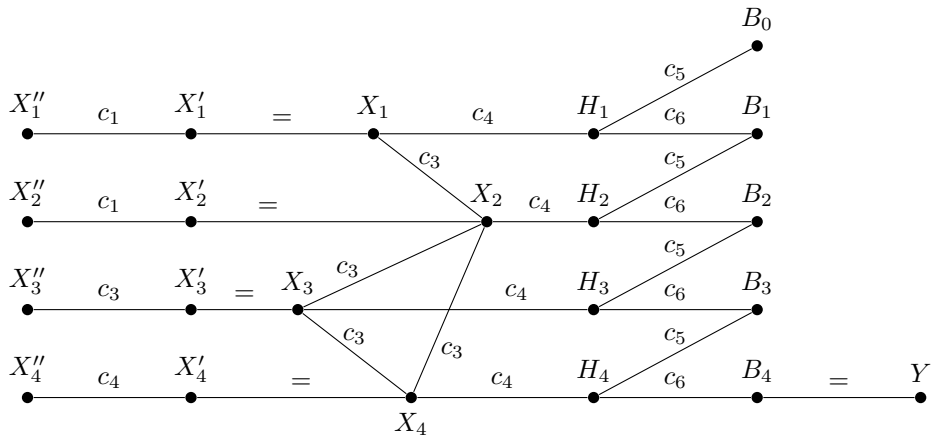

Figure 4: The constraint network P_G in the proof of Lemma 3 when looking for a dominating set in the graph $G = (\{1, 2, 3, 4\}, \{(1, 2), (2, 4), (2, 3), (3, 4)\})$

We then prove that if there exists an arc-inconsistency explanation for P_G of length $4n+k+1$, then G contains a k-dominating set. We first observe that if we remove $c_5(B_0, H_1)$ or $B_n = Y$ from P_G, the instance becomes satisfiable. (B_0 is necessary to trigger removals of value 2 from the H_is and Y to trigger removals of value 0.) Hence, no wipe out can occur without executing $2n+1$ *revise*() operations on the path from B_0 to Y. Furthermore, if a single variable H_i still has value 1 in its domain, the propagation of removals stops. As a result, value 2 needs to be removed from all X_is and a *revise*() needs to be executed on the n constraints c_4. We then have $n + k$ remaining available operations to remove value 2 from all X_is. If we do these removals thanks to the sequence $\langle X'_i \overset{c_1}{\leftarrow} X''_i, X_i \overset{c_2}{\leftarrow} X'_i \rangle$, it costs $2n$ operations, which is more than $n + k$. To reach $n + k$, we need to remove value 2 in a single operation for at least $n - k$ variables. The only way to do that is through a $X_j \overset{c_3}{\leftarrow} X_i$ for $n - k$ variables X_j. Now, $X_j \overset{c_3}{\leftarrow} X_i$ removes value 2 from $D(X_j)$ only if $D(X_i) = \{0\}$ and $c_3(X_i, X_j) \in C$. $D(X_i)$ is equal to $\{0\}$ only if X_i is one of the k variables on which $\langle X'_i \overset{c_1}{\leftarrow} X''_i, X_i \overset{c_2}{\leftarrow} X'_i \rangle$ has been executed. $c_3(X_i, X_j)$ belongs to C only if $\{i, j\} \in E$. As a result, the set of k vertices i corresponding to the k variables with $D(X_i) = \{0\}$ is a dominating set. □

5 Unbounded-Arity Boolean Constraint Networks: Structure

We now relax the condition that constraint networks are binary and normalized and we focus on Boolean constraint networks under acyclicity conditions. Defining acyclicity on the (primal) constraint graph when we refer to non-binary constraints is irrelevant because the constraint graph of a constraint network consisting of just a single k-ary constraint ($k \geq 3$) is a k-clique and thus contains a cycle. Instead, we use the more relevant Berge-acyclicity notion that corresponds to acyclicity of the *incidence graph*.

Definition 3 (Incidence graph and Berge-acyclicity) *The* incidence graph *of a constraint network consisting of constraints $c_1, \ldots, c_m$ on variables $X_1, \ldots, X_n$ is the graph $G = (V, E)$ where V is the set of constraints and variables and E is the set of edges $\{X_i, c_j\}$ such that X_i occurs in the scope of c_j. A constraint network is* Berge-acyclic *if and only if its incidence graph is acyclic.*

We show that Shortest Arc-Inconsistency Explanation is tractable on Berge-acyclic CNF formulas. (CNF formulas are constraint networks where constraints are clauses.) But

Shortest Arc-Inconsistency Explanation becomes NP-hard as soon as we relax the clausal condition, that is, on Berge-acyclic networks of general Boolean constraints.

5.1 Tractability on Berge-acyclic clausal constraint networks

Theorem 6 Shortest Arc-Inconsistency Explanation *is solvable in polynomial time when restricted to constraint networks corresponding to a Berge-acyclic CNF formula.*

Proof. Let N be a constraint network corresponding to a CNF formula ϕ, with variables $X_1, \ldots, X_n$ and clauses $c_1, \ldots, c_m$, such that the incidence graph of N is acyclic. In the following we identify a literal X_i or $\overline{X_i}$ with its corresponding assignment, respectively, $(X_i, 1)$ or $(X_i, 0)$, and we view a clause as a set of literals. For each literal (X_i, a), where $a \in \{0, 1\}$, let $L_{i,a}$ be the length of the shortest AC proof that $X_i \neq a$ (i.e. the shortest sequence of revise operations that allow us to eliminate a from $D(X_i)$). If a does not belong to the original domain $D(X_i)$, then $L_{i,a} = 0$. For each literal $(X_i, \overline{a})$ in each constraint c_j, let $M_{j,i,a}$ be the length of the shortest AC proof via c_j that $X_i \neq a$, consisting of proofs that all other literals in c_j are false, followed by a $revise(X_i, c_j)$. For each variable X_i, let W_i be the length of the shortest AC proof of a wipe-out of domain $D(X_i)$.

We give a polynomial-time algorithm. Initially we set all the L, M, W values to infinity, except for literals (X_i, a) occurring in unit clauses in ϕ, in which case $L_{i,\overline{a}}$ is initialized to 0, since we identify unit clauses with domain restrictions.

We can then apply the following update rules:

(1) $L_{i,a} = \min(L_{i,a}$, minimum of $M_{j,i,a}$ over j for which $(X_i, \overline{a})$ occurs in $c_j)$

(2) $M_{j,i,a} = \min(M_{j,i,a}$, 1+ sum of the $L_{k,b}$ over other literals $(X_k, b) \neq (X_i, \overline{a})$ of $c_j)$

(3) $W_i = \min(W_i, L_{i,0} + L_{i,1})$

Applying these rules until convergence is clearly polynomial-time. The final minimum value of the W_i's is the length of the shortest proof of inconsistency by AC. The "1+" in rule (2) corresponds to the revise operation on clause c_j which leads to the elimination of a from the domain of X_i. The sum of the $L_{k,b}$ in rule (2) follows from the fact that a clause can only imply that the literal is true if *all* other literals are false. The rules are valid, in particular (2), because the incidence graph is acyclic and hence no two sub-proofs share common steps, so 1+ the sum of the $L_{k,b}$'s is the actual length of the proof (and not just an upper bound). Similarly, in rule (3) there can be no intersection between the proof that $X_i \neq 0$ and the proof that $X_i \neq 1$. □

5.2 Intractability on Berge-acyclic non-clausal constraint networks

The algorithm in the proof of Theorem 6 relies very heavily on Berge-acyclicity and the fact that all constraints are clauses. The following proposition shows that this latter condition is essential.

Theorem 7 Shortest Arc-Inconsistency Explanation *is NP-hard for Berge-acyclic constraint networks over Boolean domains.*

Proof. We demonstrate a polynomial reduction from Hitting Set. Let $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}, k$ be an instance of Hitting Set. The question is whether there exists a subset of $\{1, \ldots, n\}$ of size k that intersects all sets S_i, $i = 1, \ldots, m$.

We build a Berge-acyclic constraint network N which has an arc-inconsistency explanation of length at most $k+1$ iff there is a hitting set of $S_1, \ldots, S_m$ of size at most k. For each $i \in \{1, \ldots, n\}$, we create variables X_i, Y_i, with domains $D(X_i) = \{0, 1\}$, $D(Y_i) = \{0\}$, and a binary constraint $X_i = Y_i$. Finally, we create an n-ary constraint c on $X_1, \ldots, X_n$ whose relation R is given by $R = \{t_j \mid j = 1, \ldots, m\}$ where t_j is the tuple associated with S_j: $t_j[X_i] = 1$ if $i \in S_j$ (0 otherwise). Suppose that $H = \{i_1, \ldots, i_q\}$ is a hitting set, with $q \leq k$. Then the q operations $X_{i_r} \leftarrow Y_{i_r}$ $(r = 1, \ldots, q)$ followed by $revise(X_{i_1}, c)$ lead to a domain wipe-out at X_{i_1}. Conversely, any length-$(q+1)$ arc-inconsistency explanation (with $q \leq k$) must use at most q operations $X_{i_r} \leftarrow Y_{i_r}$ $(r = 1, \ldots, q)$ and at least one $revise(X_{i_s}, c)$ for some $s \in \{1, \ldots, n\}$. By construction of N, it leads to a domain wipe-out only if $H = \{i_1, \ldots, i_q\}$ is a hitting set of $S_1, \ldots, S_m$. □

6 Unbounded-Arity Boolean Constraint Networks: Language

In this section we still focus on non-binary Boolean constraint networks, but we only consider restrictions on the language of the relations. We show that the polynomial-time algorithm of Theorem 4 can be generalized from binary Boolean relations to a slightly larger Boolean constraint language, which contains relations of arbitrarily large arities. This constraint language is known as the *bijunctive* (or *Krom*) language, and corresponds to the set of all relations that are logically equivalent to the conjunction of their binary projections. We will then show that this positive result does not extend to the Horn, anti-Horn and affine constraint languages, even when the constraints' arities are restricted to be at most three.

If N is a constraint network, Y is a variable and $v \in D(Y)$, we denote by $N + (Y \leftarrow v)$ the constraint network obtained from N by removing from $D(Y)$ all values other than v. For a relation R, the projection of R onto a subset of indices is the relation obtained by projection of all tuples in R onto these indices. In the case of constraint networks, the projection onto a subset of variables S discards all variables outside S and all constraints whose scope contains a discarded variable. For tuples, relations, constraints and constraint networks, we use $[\cdot]$ to denote the projection operator on a subset of indices or variables.

6.1 Tractability on networks of bijunctive constraints

Theorem 8 Shortest Arc-Inconsistency Explanation *is solvable in polynomial time when restricted to Boolean constraint networks over the bijunctive language.*

Proof. Let N be an instance of Shortest Arc-Inconsistency Explanation where N is a constraint network whose constraints are bijunctive. We assume without loss of generality that every constraint has a non-empty relation. We replace each constraint $c = (R, S)$ of arity r by its $r(r-1)/2$ binary projections $c_{i,j} = (R[i,j], S[i,j])$ and denote by N^* the resulting binary Boolean constraint network. We claim that any arc-inconsistency explanation for N^* can be transformed in polynomial time into an arc-inconsistency explanation of the same length for N, and vice versa.

The first direction is straightforward. Let E^* be an arc-inconsistency explanation for N^*. We build a sequence E of $revise()$ operations for N by replacing each $revise(Y, c_{i,j}) \in E^*$ with $revise(Y, c)$. The relation of $c_{i,j}$ is a binary projection of the relation of c, so with identical domains each value $v \in D(Y)$ with a support in c also has a support in $c_{i,j}$. It follows by induction on the number of steps that any value removed by E^* is also removed by E. In particular, E is an arc-inconsistency explanation for N whose length is the same as E^*.

The opposite direction is more delicate, as it is not obvious a priori that any value removed by a $revise()$ on a constraint with relation R can be removed with a single $revise()$ on one of its binary projections. To see that it is the case, consider a constraint $c = (R, S)$ in N where the domains of the variables in S are subsets of the initial domains. Let N_c be the constraint network whose variables are those appearing in S (with identical domains) and whose constraints are the binary projections of c.

Claim 1 *Every minimum-length arc-inconsistency explanation in N_c has length one.*

Inspecting the proof of Theorem 4, we know that such an explanation that derives a domain wipeout on a variable Z is either composed of (i) a single path of $revise()$ from a unique source variable Q to Z, or (ii) a path from a unique source variable Q to a variable Y followed by two internally disjoint paths from Y to Z, or (iii) two internally disjoint paths from source variables Q', Q'' to Z. In cases (i) and (ii), all variables involved (except for Q and possibly Z) have domain size 2. Therefore, in these cases there does not exist $t \in c$ such that $t[Q] \in D(Q)$ and $t[Z] \in D(Z)$. By construction we have $c_{i,j}(Q, Z) \cap (D(Q) \times D(Z)) = \emptyset$, which means that N_c has an arc-inconsistency explanation of length one. In case (iii), Q' and Q'' are the only variables involved in the explanation with domains of size 1, and therefore $c_{i,j}(Q', Q'') \cap (D(Q') \times D(Q'')) = \emptyset$ and we also obtain an arc-inconsistency explanation of length one.

Claim 2 *N_c has a solution if and only if its arc-consistency closure $A(N_c)$ has no empty domains.*

One direction is obvious, so let us assume that $A(N_c)$ has no empty domains. Let $S = (S_1, S_2)$ be a partition of S such that S_1 is the set of variables whose domain in $A(N_c)$ has size one. By arc consistency, $A(N_c)[S_1]$ has a solution ϕ. Furthermore, each binary constraint in $A(N_c)[S_2]$ is a projection of R, which is not empty, so $A(N_c)[S_2]$ has a solution ψ. Each constraint with one variable in S_1 and one variable in S_2 contains a support for all values so $\phi \cup \psi$ is a solution to N_c, which establishes the claim.

We can now finish the proof of Theorem 8. Let E be an arc-inconsistency explanation for N, which we assume is minimal, and $revise(Y, c)$ be a step in E with $c = (R, S)$. If this step is last in E then c is inconsistent with respect to the current domains and N_c does not have a solution. By Claims 1 and 2, N_c has an arc-inconsistency explanation that consists in a single step $revise(Z, c_{i,j})$ and we can replace $revise(Y, c)$ with $revise(Z, c_{i,j})$. Otherwise, let $Y \in S$ and v be the unique value removed from $D(Y)$ by $revise(Y, c)$. Since this step is not last and E has minimum length, N_c has a solution but $N_c + (Y \leftarrow v)$ does not. By Claims 1 and 2, this means that $N_c + (Y \leftarrow v)$ has an arc-inconsistency explanation E_c of size one that involves Y. In that case, we can assume without loss of generality that $revise(Y, c_{i,j})$ derives an empty domain on Y. This same step applied on N_c instead of $N_c + (Y \leftarrow v)$ will remove v from $D(Y)$, and therefore $revise(Y, c)$ can be replaced with $revise(Y, c_{i,j})$ in E with the same effect. After all steps have been replaced, we obtain an arc-inconsistency explanation for N^* of the same length as E.

Finally, one can compute in polynomial time a minimum arc-inconsistency explanation for N in polynomial time by computing N^*, computing a shortest arc-inconsistency explanation E^* for N^* using Theorem 4 and then replacing each step $revise(Y, c_{i,j})$ in E^* with $revise(Y, c)$. □

6.2 Intractability on networks of Horn, anti-Horn, or affine constraints

The bijunctive language is one of the four maximal Boolean constraint languages for which the corresponding restriction of the CSP (with arbitrary variable domains) is solvable in polynomial-time according to Schaefer's Theorem. The other three are the *affine language* (all relations expressible as solution sets of systems of linear equations over the 2-element field), the *Horn language* (all relations expressible as model sets of Horn formulae) and the *anti-Horn language* (all relations expressible as model sets of anti-Horn formulae). A natural question is to determine the complexity of Shortest Arc-Inconsistency Explanation for these languages. We show that Shortest Arc-Inconsistency Explanation is NP-hard in all three cases, even when constraints have arity at most three.

Theorem 9 Shortest Arc-Inconsistency Explanation *is NP-hard, even when restricted to Boolean constraint networks over the Horn (resp. anti-Horn, affine) language with maximum arity at most three.*

Proof. We start with the case where all constraints are Horn. We reduce from the NP-complete problem Set Cover. Let the universe U be $\{1, \ldots, n\}$ and $S_1, \ldots, S_m$ the family of subsets of U. The question is whether there is a subfamily represented by a set $T \subseteq \{1, \ldots, m\}$ of size at most k such that $\cup_{i \in T} S_i = U$.

We construct an instance N of Shortest Arc-Inconsistency Explanation with Horn constraints as follows. There are $2m$ variables $x_1, \ldots, x_m, x_1^*, \ldots, x_m^*$, n variables $y_1, \ldots, y_n$ and $n-3$ variables $z_1, \ldots, z_{n-3}$. The variables $x_1^*, \ldots, x_m^*$ have domain $\{1\}$ and all other variables have domain $\{0, 1\}$. For each $i \in \{1, \ldots, m\}$ there is a constraint $\overline{x_i^*} \vee x_i$, and for each set S_i such that $j \in S_i$ there is a constraint $\overline{x_i} \vee y_j$. Finally, there is a sequence of constraints $(\overline{y_1} \vee \overline{y_2} \vee z_1)$, $(\overline{y_{r+2}} \vee \overline{z_r} \vee z_{r+1})$ (for $r \in \{1, \ldots, n-4\}$), $(\overline{y_n} \vee \overline{y_{n-1}} \vee \overline{z_{n-3}})$ whose conjunction is equivalent to $\overline{y_1} \vee \cdots \vee \overline{y_n}$.

Any arc-inconsistency explanation E for N must use all the $n-2$ constraints representing $\overline{y_1} \vee \cdots \vee \overline{y_n}$ once, together with two clauses of the form $\overline{x_i^*} \vee x_i$, $\overline{x_i} \vee y_j$ for each $j \in \{1, \ldots, n\}$. Therefore, it must be the case that the set $T = \{i \in \{1, \ldots, m\} \mid revise(x_i, *) \in E\}$ (where $*$

stands for any constraint) is a set cover of U. The number of steps in E is therefore at least $|T| + 2n - 2$, and there exists one with length $k + 2n - 2$ if there exists a set cover of length k. We conclude that there exists a set cover of size at most k if and only if N has an arc-inconsistency explanation of length at most $k + 2n - 2$.

The case of the anti-Horn language is symmetrical, as replacing each literal with its negation and setting $D(x_i^*) = \{0\}$ for all $i \leq m$ in the construction above yields NP-hardness for anti-Horn constraints. The case of affine constraints is slightly different, but the same reduction from Set Cover can be adapted: if one replaces each constraint $\overline{x_i^*} \vee x_i$ with $x_i^* = x_i$, each constraint $\overline{x_i} \vee y_j$ with $x_i = y_j$, and the sequence encoding $\overline{y_1} \vee \cdots \vee \overline{y_n}$ with $y_1 + y_2 + z_1 = 0$, $y_{r+2} + z_r + z_{r+1} = 0$ (for $r \in \{1, \ldots, n-4\}$), $y_n + y_{n-1} + z_{n-3} = b$ (where b is 0 if n is odd and 1 otherwise; the conjunction of these constraints is equivalent to $y_1 + \cdots + y_n = b$ where "+" stands for addition modulo 2), then the same reasoning applies and the shortest arc-inconsistency explanation of the constraint network has length at most $k+2n-2$ if and only if there is a set cover of size at most k. □

To conclude this section, we point out that Theorems 8 and 9 do not constitute a classification over all Boolean constraint languages in any way. In particular, there exist Boolean constraint languages for which Shortest Arc-Inconsistency Explanation is solvable in polynomial time beyond the bijunctive language. A degenerate example is the language of *monotone* constraints, that is, the set of all relations R such that $t \in R$ implies $t^+ \in R$ for all t^+ such that $t^+[i] \geq t[i]$ for all indices i. Any constraint network over this language is satisfiable if and only if assigning all variables to the largest value in their domain (0 or 1) is a solution. It follows that unsatisfiable networks always have an arc-inconsistency explanation of length one: if the maximum-value assignment violates a constraint, then that constraint has no support for any value and a single *revise*() with respect to any variable in its scope will derive an empty domain. This explanation can be found in polynomial time by exhaustive search.

7 Conclusion

We have investigated the complexity of finding a shortest proof of inconsistency of a binary constraint network in the form of a sequence of arc-consistency operations. Our characterisation in terms of structure or domain size shows that this problem is polynomial when variables have degree two or domains are Boolean. The problem is NP-hard if the constraint network has four variables of degree three or if the domain size is bounded by three. It is also NP-hard on trees. In addition, the problem is not FPT-approximable unless the Gap-ETH is false. Although our initial motivation was to provide short explanations for human users, there are other possible applications. Virtual Arc Consistency (VAC) algorithms for cost-function networks use arc-inconsistency explanations in the constraint network of zero-cost tuples in order to update cost functions [5]. Our NP-hardness results can be seen as a justification for the use of minimal rather than minimum-cardinality arc-inconsistency explanations by VAC algorithms. On a final positive note, the polynomial-time algorithm for the special case of size-2 domains may prove an inspiration for heuristic methods to improve minimal arc-inconsistency explanations via the search for shortest paths in the causal graph described in the proof of Theorem 4.

Funding and Competing Interests

This work was supported by the AI Interdisciplinary Institute ANITI, funded by the French program "Investing for the Future – PIA3" under grant agreement no. ANR-19-PI3A-0004, and by the European Union's Horizon Europe Research and Innovation program under the grant agreement TUPLES No101070149. The first two authors received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 952215. The first author is on the advisory board of this journal and the fourth author is in the editorial board of this journal. They receive no compensation as members of these boards.

References

[1] Jérôme Amilhastre, Hélène Fargier, and Pierre Marquis. Consistency restoration and explanations in dynamic CSPs application to configuration. *Artif. Intell.*, 135(1-2):199–234, 2002.

[2] Bart Bogaerts, Emilio Gamba, and Tias Guns. A framework for step-wise explaining how to solve constraint satisfaction problems. *Artif. Intell.*, 300:103550, 2021.

[3] Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From Gap-ETH to FPT-inapproximability: Clique, dominating set, and more. In Chris Umans, editor, *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS'17)*, pages 743–754. IEEE Computer Society, 2017.

[4] Yijia Chen, Martin Grohe, and Magdalena Grüber. On parameterized approximability. In Hans L. Bodlaender and Michael A. Langston, editors, *Parameterized and Exact Computation, Second International Workshop, IWPEC*, volume 4169 of *Lecture Notes in Computer Science*, pages 109–120. Springer, 2006.

[5] Martin C. Cooper, Simon de Givry, Martí Sánchez-Fibla, Thomas Schiex, and Matthias Zytnicki. Virtual arc consistency for weighted CSP. In Dieter Fox and Carla P. Gomes, editors, *AAAI 2008*, pages 253–258. AAAI Press, 2008.

[6] Irit Dinur. Mildly exponential reduction from gap 3SAT to polynomial-gap label-cover. *Electronic Colloquium on Computational Complexity*, page 128, 2016.

[7] Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

[8] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, first edition edition, 1979.

[10] Allen Van Gelder. Verifying RUP proofs of propositional unsatisfiability. In *ISAIM*, 2008.

[11] E. Goldberg and Y. Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *2003 Design, Automation and Test in Europe Conference and Exhibition*, pages 886–891, 2003.

[12] Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. Abduction-based explanations for machine learning models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):1511–1519, 2019.

[13] Ulrich Junker. QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence*, pages 167–172. AAAI Press / The MIT Press, 2004.

[14] Ulrich Junker. Configuration. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 837–873. Elsevier, 2006.

[15] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[16] Pasin Manurangsi and Prasad Raghavendra. A birthday repetition theorem and complexity of approximating dense CSPs. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP'17)*, volume 80 of *LIPIcs*, pages 78:1–78:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[17] Dániel Marx. Completely inapproximable monotone and antimonotone parameterized problems. In *Proceedings of the 25th IEEE Annual Conference on Computational Complexity*, pages 181–187, 2010.

[18] Andy Shih, Arthur Choi, and Adnan Darwiche. A symbolic approach to explaining Bayesian network classifiers. In *IJCAI'18*, page 5103–5111. AAAI Press, 2018.

[19] Barbara M. Smith. How to Solve the Zebra Problem, or Path Consistency the Easy Way. In *ECAI'92*, pages 36–37. John Wiley and Sons, 1992.

[20] Mohammed H. Sqalli and Eugene C. Freuder. Inference-based constraint satisfaction supports explanation. In William J. Clancey and Daniel S. Weld, editors, *AAAI 96, IAAI 96, Volume 1*, pages 318–325. AAAI Press / The MIT Press, 1996.

A Appendix: FPT-inapproximability assuming FPT $\neq$ W[2]

In this appendix we prove that Shortest Arc-Inconsistency Explanation is not FPT-approximable on binary normalized constraint networks unless FPT = W[2]. This result is closely related to Theorem 3, but orthogonal: the underlying complexity hypothesis is significantly weaker (as the Gap-ETH implies the ETH, which in turn implies FPT $\neq$ W[2]), but the reduction does not work for tree-structured constraint networks.

A *monotone Boolean circuit* is an acyclic directed graph with a unique sink (its output), and where each non-source vertex is labelled with either AND or OR. We call source vertices *inputs*, non-source vertices *gates*, and the *fan-in* of a gate denotes its indegree. Given a gate g_j in B with fan-in p, we let $h_j^{-1} : \mathbb{N} \to \{v : (v, g_j) \in B\}$ denote an arbitrary function whose restriction to $\{1, \ldots, p\}$ is surjective. (This function can be interpreted as $h_j^{-1}(i)$ being the ith input to g_j if $i \leq p$, and some arbitrary input to g_j otherwise.) A monotone Boolean circuit is *t-normalized* [7] if its nodes are divided into $t+1$ layers $L_0, L_1, \ldots, L_t$ such that L_0 is the set of inputs, in-edges to nodes in L_i, $i \geq 1$ are out-edges of nodes in layer L_{i-1}, and the label of a node in layer L_i, $i \geq 1$ is OR if and only if i is odd. The Monotone Circuit Satisfiability problem takes as input a monotone Boolean circuit B and asks to find a 0/1-vector y accepted by B that has minimum Hamming weight.

We will derive our improved hardness result for Shortest Arc-Inconsistency Explanation from the following theorem, where FPT and W[2] are parameterized complexity classes generally believed to be distinct [8].

Theorem 10 (derived from [17], Corollary 5) *If* Monotone Circuit Satisfiability *is FPT-approximable for 4-normalized circuits where all L_3 gates have fan-in at most 2^k, then FPT = W[2].*

Note that the exact statement of this result in Marx's paper [17] is a bit different as it deals with the slightly weaker notion of *fpt cost approximability* (which is implied by FPT-approximability). It also does not explicitly mention that circuits are 4-normalized or the 2^k bound on the fan-in of L_3 gates. Both of these facts are critical for our reduction to work; they can be readily verified by inspecting the original proof.

Theorem 11 Shortest Arc-Inconsistency Explanation *is not FPT-approximable unless FPT = W[2], even on binary normalized constraint networks.*

Proof. Let (B, k) be an input to Monotone Circuit Satisfiability such that B is 4-normalized and its L_3 gates have fan-in at most 2^k. We will construct a constraint network N_B that has an arc-inconsistency explanation of length at most $k' = f(k)$ if B accepts a vector of Hamming weight at most k, and no arc-inconsistency explanation of length at most k otherwise.

The network N_B has a distinct set of variables for each layer. The input layer $L_0 = \{x_1, \ldots, x_m\}$ is represented by m variables $X_1, \ldots, X_m$ with domain $\{O\}$. The layer L_1 (resp. L_2) is represented by a single variable Y_1 (resp. Y_2) with domain $\{O\} \cup \{g_j \mid g_j \in L_1\}$ (resp. $\{O\} \cup \{g_j \mid g_j \in L_2\}$). For the layer L_3, N_B contains $2^k + 1$ variables $Y_3^1, \ldots, Y_3^{2^k}, Y_3$ with $D(Y_3^i) = \{O\} \cup \{g_j \mid g_j \in L_2\}$ for every $i \leq 2^k$ and $D(Y_3) = \{O\} \cup \{g_j \mid g_j \in L_3\}$. Finally, the output layer L_4 is represented by three variables Y_4^1, Y_4^2, Y_4^3 with domain $\{O, F\}$.

Next, we describe the network's constraints:

$$\forall i \in \{1, \ldots, m\}, c_i(X_i, Y_1) = \{(*, O)\} \cup \{(O, g_j) \mid g_j \in L_1, (x_i, g_j) \notin B\}$$

$$c_{12}(Y_1, Y_2) = \{(*, O)\} \cup \{(g_q, g_j) \mid g_j \in L_2, (g_q, g_j) \in B\}$$

$$\forall i \in \{1, \ldots, 2^k\}, c_{2i}(Y_2, Y_3^i) = \{(*, O)\} \cup \{(g_j, g_j) \mid g_j \in L_2\}$$

$$\forall i \in \{1, \ldots, 2^k\}, c_{i3}(Y_3^i, Y_3) = \{(*, O)\} \cup \{(h_j^{-1}(i), g_j) \mid g_j \in L_3\}$$

$$c_{34}(Y_3, Y_4^1) = \{(*, O)\} \cup \{(g_j, F) \mid g_j \in L_3\}$$

$$Y_4^1 = Y_4^2, Y_4^2 = Y_4^3, Y_4^3 \neq Y_4^1$$

Informally, the idea behind this construction is that setting $x_i = 1$ in B is roughly equivalent to including revise($c_i(X_i, Y_1)$, Y_1) in an arc-inconsistency explanation, in the sense that the values g_j that can be later removed from the domains will correspond exactly to the gates that would evaluate to true in B. The value F in the domain of Y_4^1 corresponds to the output gate. If it can be pruned, then a sequence of three revise() in the loop of constraints $Y_4^1 = Y_4^2, Y_4^2 = Y_4^3, Y_4^3 \neq Y_4^1$ will infer an empty domain; otherwise neither O or F can be removed from any domain and the sequence cannot be extended to an arc-inconsistency explanation.

First, we prove that a vector of weight at most k accepted B can be extracted from any arc-inconsistency explanation of N_B of length at most k. Let E be an arc-inconsistency explanation of N_B and y_E be the Boolean vector of length m whose ith entry is 1 if and only if revise($c_i(X_i, Y_1)$, Y_1) is in E. Suppose for the sake of contradiction that B does not accept y_E. We start with a general observation on the structure of E. Since the tuple (O, O) belongs to the relation of every constraint except $Y_4^3 \neq Y_4^1$, the value O cannot be removed from any domain until revise() is called on this constraint. In addition, the loop $Y_4^1 = Y_4^2, Y_4^2 = Y_4^3, Y_4^3 \neq Y_4^1$ is initially arc consistent and shares only Y_4^1 with the rest of the network. Taken together, these facts imply that Y_4^1 is the first variable in the loop whose domain is pruned by E, and that the pruned value must be F. Now, we focus on the steps of E that occur before F is pruned from $D(Y_4^1)$. Let $\mathcal{G}_1$ (resp. $\mathcal{G}_2, \mathcal{G}_3$) denote the set of gates in layer L_1 (resp. L_2, L_3) that evaluate to false on input y_E. Because $\mathcal{G}_1$ consists of OR gates, every $g_j \in D(Y_1) \cap \mathcal{G}_1$ has a support (O, g_j) in all constraints c_i such that revise($c_i(X_i, Y_1), Y_1$) occurs in E, as well as a support (g_j, O) in c_{12}. It follows that these gates remain in $D(Y_1)$ until F is pruned from $D(Y_4^1)$. Next, since the layer L_2 only contains AND gates, for every $g_j \in \mathcal{G}_2$ there exists some $g_q \in \mathcal{G}_1$ such that $(g_q, g_j) \in B$. In particular, every $g_j \in D(Y_2) \cap \mathcal{G}_2$ has a support $(g_q, g_j) \in c_{12}$ and a support (g_j, O) in every constraint $c_{2i}, i \leq 2^k$, so these gates also remain in $D(Y_2)$ and each $D(Y_3^i), i \leq 2^k$. The layer L_3 contains OR gates, and hence for every gate $g_j \in \mathcal{G}_3$ the set $\{h^{-1}(i) \mid i \leq 2^k\}$ is a subset of $\mathcal{G}_2$. Therefore, every $g_j \in D(Y_3) \cap \mathcal{G}_3$ has a support $(h_j^{-1}(i), g_j)$ for every constraint c_{i3} and a support (g_j, O) in c_{34}, so they remain in $D(Y_3)$ throughout E. Finally, since $\mathcal{G}_3 \neq \emptyset$, the value $F \in D(Y_4^1)$ has a support in all three constraints involving Y_4^1, contradicting the fact that E is an arc-inconsistency explanation of N_B.

We now prove a partial converse: if B accepts a vector y of weight at most k, then N_B has an arc-inconsistency explanation of length at most $2^{k+1} + k + 5$. Define E to be the sequence that starts with all revise(c_i,Y_1) such that $y_i = 1$ (in any order), and then progresses upwards in the layers by invoking revise(c_{12}, Y_2), revise($Y_2 = Y_3^i$, Y_3^i) for all $i \leq 2^k$, then revise(c_{i3}, Y_3) for all $i \leq 2^k$, and finally revise(c_{34}, Y_4^1). We show that this sequence removes F from $D(Y_4^1)$, and hence can be extended to derive an empty domain using three extra revise() along the final loop of constraints on Y_4^1, Y_4^2, Y_4^3; the total length will be at most $k + 1 + 2 \cdot 2^k + 1 + 3 = 2^{k+1} + k + 5$. Let $\mathcal{G}_1'$ (resp. $\mathcal{G}_2', \mathcal{G}_3'$) denote the set of gates in layer L_1 (resp. L_2, L_3) that evaluate to true on input y. Because each $g_j \in D(Y_1) \cap \mathcal{G}_1'$ is an OR gate, there exists an index $i \leq m$ such that revise(c_i,Y_1) is included in E. By construction, g_j has no support in the constraint c_i, so this step must remove g_j from $D(Y_1)$. Consequently, after all steps of this form are executed in E, $D(Y_1)$ no longer contains any gate $g_j \in \mathcal{G}_1'$. Then, since each $g_j \in D(Y_2) \cap \mathcal{G}_2'$ is an AND gate, all its input gates must belong to $\mathcal{G}_1'$. Since these gates are the only supports for g_j in c_{12}, g_j must be removed from $D(Y_2)$ by revise(c_{12}, Y_2) and after this step $D(Y_2)$ no longer contains any gate $g_j \in \mathcal{G}_2'$. In turn, the following 2^k steps revise($Y_2 = Y_3^i$, Y_3^i) will ensure $D(Y_2) = D(Y_3^i)$ for all i. Repeating the reasoning of the first layer, after all step revise(c_{i3}, Y_3) the domain of Y_3 no longer contains any gate $g_j \in \mathcal{G}_3'$, and finally revise(c_{34}, Y_4^1) will remove F from $D(Y_4^1)$, as claimed.

Suppose that Shortest Arc-Inconsistency Explanation admits an FPT-approximation algorithm A with functions ρ, f. Let A' be the algorithm that, on input (B, k), constructs N_B in polynomial time, invokes A on input $(N_B, 2^{k+1}+k+5)$ to obtain an arc-inconsistency explanation E and then returns the vector y_E. If B accepts a vector of Hamming weight at most k, then N_B has an arc-inconsistency explanation of length at most $2^{k+1}+k+5$ and hence A returns an arc-inconsistency explanation of N_B of length at most $(2^{k+1}+k+5)\cdot\rho(2^{k+1}+k+5)$. The vector y_E has Hamming weight at most $(2^{k+1}+k+5)\cdot\rho(2^{k+1}+k+5)$ and is accepted by B. Since the runtime of A' is $O(f(2^{k+1}+k+5)\cdot|B|^{O(1)}) = O(g(k)\cdot|x|^{O(1)})$, it is an FPT-approximation algorithm for Monotone Circuit Satisfiability with ratio $\rho'(k) = (2^{k+1}+k+5)\cdot\rho(2^{k+1}+k+5)$. (Note that $k\cdot\rho'(k)$ is nondecreasing because $\rho'(k)$ is nondecreasing.) By Theorem 10 this implies FPT = W[2], which concludes the proof.