Complexity of Minimum-Size Arc-Inconsistency Explanations

Christian Bessiere, Clement Carbonnel, Martin Cooper, Emmanuel Hébrard

Complexity of Minimum-Size Arc-Inconsistency Explanations

Christian Bessiere
CNRS, University of Montpellier, France

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, on binary CSPs, cases for which providing a shortest such explanation is easy: when domains are Boolean or when variables have maximum degree two. However, these polynomial cases are tight. Providing a shortest explanation is NP-hard if the maximum degree is three, even if the number of variables is bounded, or if domain size is bounded by three. It remains NP-hard on trees, despite the fact that arc consistency is a decision procedure on trees. Finally, the problem is not FPT-approximable unless the Gap-ETH is false.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Theory of computation → Fixed parameter tractability; Theory of computation → Problems, reductions and completeness

Keywords and phrases Constraint programming, constraint propagation, minimum explanations, complexity

Digital Object Identifier 10.4230/LIPIcs.CP.2022.9

Funding 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. The first two authors received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 952215.

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 thanks to the expressiveness of constraints that allow to remain close to the original problem. 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 Machine Learning model's prediction is often defined as a minimal subset of features that entails that prediction [16, 10]. 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 [11]. 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, when solving a logic puzzle, 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. 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 [8, 9] 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 [17, 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 [12], logic puzzles [17]).

Our goal is to analyze the complexity of providing the *shortest* possible explanation of arc inconsistency of a problem. For simplicity of presentation, we restrict ourselves to normalized networks of binary constraints. We show that when variables have degree two or domains are Boolean, finding a shortest explanation of arc inconsistency is polynomial. However, the problem is NP-hard in general and the two polynomial cases above are tight. Finding a shortest explanation of arc inconsistency is NP-hard as soon as variables have degree three, even if the number of variables is bounded (even though the problem is obviously polynomial to solve). It is also NP-hard 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. 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 Gap-ETH is false.

2 Background and Definitions

The *constraint satisfaction problem* (*CSP*) involves finding solutions to a constraint network. A *constraint network* (or CSP instance) is defined as a set of n variables $X = X_1, \dots, X_n$, a set of domains $D = \{D(X_1), \dots, D(X_n)\}$, where $D(X_i)$ is the finite set of values that X_i can take, and a set C of constraints. A binary constraint $c(X_i, X_j)$ is a binary relation that specifies which combinations of values (tuples) the variables (X_i, X_j) are allowed to take. A CSP is *binary* when all the constraints are binary. A binary CSP is said to be *normalized* if there is at most one constraint per pair of variables. A *degree-2* CSP does not contain any variable involved in more than two constraints. *Arc consistency* (*AC*) is the basic form of inference reasoning on constraint networks. A tuple τ of values on (X_i, X_j) is called a *support* on constraint $c(X_i, X_j)$ for a value $v \in D(X_i)$ (and $\tau[X_j]$ its support in $D(X_j)$) if and only if $\tau[X_i] = v$, $\tau[X_j] \in D(X_j)$ and $\tau \in c(X_i, X_j)$. A value v in $D(X_i)$ is arc consistent if and only if v has a support on every constraint involving X_i. A network is arc consistent if all values in all domains are arc consistent. The operation $revise(X_i, c(X_i, X_j))$, often denoted by $X_i \xleftarrow{c} X_j$ in the following, removes from $D(X_i)$ all values that do not have any support on $c(X_i, X_j)$. 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 CSP instance 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 3** (Explaining the Zebra puzzle)**.** The Zebra puzzle, which may (or may not) be due to Lewis Carroll, has a well known CSP model 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 CSP detects that "**the Kools smoker does not live in the 2nd house**". A demonstrative explanation would be: **The Norwegian lives in the first house**. Since *the Norwegian lives next to the blue house*, then **the 2nd house is blue**. Since *the 2nd house has a single color*, then **it is not yellow**. Since *Kools are smoked in the yellow house*, then **the Kools smoker does not live in the 2nd house**.

Each step corresponds to the arc consistency *revision* of some domain knowledge (in bold) with respect to a constraint (in italic), that is, it corresponds in our framework to the following sequence of $revise()$ operations: $\langle X_{\text{Blue}} \leftarrow X_{\text{Norwegian}}, X_{\text{Yellow}} \xleftarrow{\neq} X_{\text{Blue}}, X_{\text{Kools}} \xleftarrow{=} X_{\text{Yellow}} \rangle$.

3 Complexity of Explaining Arc Inconsistency: Structure

We show that 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 CSP has no more than four variables. Perhaps even more surprising, the problem is NP-hard on CSPs structured as trees, despite arc consistency being a decision procedure on trees.

3.1 Tractability on degree-2 CSPs

▶ **Theorem 4.** *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, so that for example X_{n+1} actually 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 CSPs with four variables

The result in Theorem 4 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 CSPs on three variables have degree at most 2.) We use a reduction from Clique, which is NP-complete [13], to prove hardness.

▶ **Definition 5** (Clique).
Input: *An undirected graph $G = (V, E)$ and an integer k*
Question: *Is there $S \subseteq V$ such that $|S| \leq k$ and for all $i \neq j \in S$, $\{i, j\} \in E$?*

It is noticeable that CSPs 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 6.** *Shortest Arc Inconsistency Explanation is NP-hard, even on binary normalized networks with four variables.*

▶ **Lemma 7.** *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 CSP instance. Let $G = (V, E)$ be a graph with set of vertices $V = \{1, \dots, n\}$.

We construct the CSP instance 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:

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

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:

- 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 $X_1 \overset{=}{\leftarrow} X_4$.

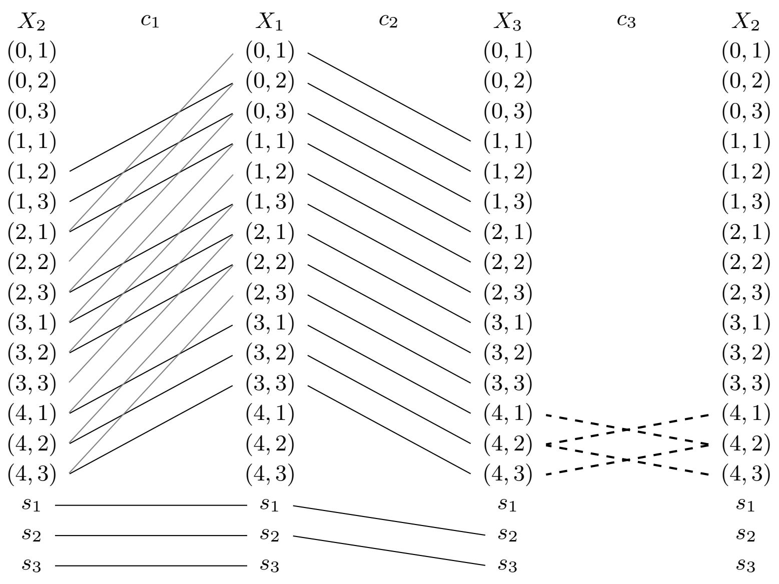


Figure 1 The CSP P_G, reduction of the graph $G = (\{1, 2, 3\}, \{(1, 2), (2, 3)\})$. Solid edges represent allowed tuples for c_1 and c_2, while dashed edges stand for forbidden tuples of c_3. The equality constraints are not represented. There are two explanations of Arc-Inconsistency of length 12. 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.

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 $i \in \{p+1,\dots,n\}$.

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, $i \in S \cup \{p,\dots,n\} \iff i \in S \cup \{p+1,\dots,n\}$ because $i < p$. Finally, by the induction hypothesis we have "$(p-1,i) \in D(X_1)$ before iteration p" iff $i \in S \cup \{p,\dots,n\}$, and hence by transitivity: "$(p,i) \in D(X_1)$ before iteration $p+1$" iff $i \in S \cup \{p+1,\dots,n\}$.

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 $i \notin S \cup \{p+1,\dots,n\}$.

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 $S \subseteq \{1,\dots,n\}$. 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 is still possible 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 [7], to derive NP-hardness of Shortest Arc-Inconsistency Explanation, even on trees.

▶ **Definition 8** (Dominating Set).
Input: *An undirected graph $G = (V,E)$ and an integer k*
Question: *Is there $S \subseteq V$ such that $|S| \le k$ 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 $f, \rho : \mathbb{N} \to \mathbb{R}_{\ge 1}$ 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* [15].

Our FPT-inapproximability result is conditional on a complexity hypothesis known as the *Gap-ETH* [6, 14], 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 9.** *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 7.

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 CSP instance.

Let $G=(V,E)$ be a graph, $V=\{v_1,\dots,v_n\}$. We construct a constraint network P_G as follows: the set of variables is $\{Y,X_1,\dots,X_n\}$, where the domain of Y is $\{v_1,\dots,v_n\}$ 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 10.** *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 9, 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. ◀

As a final remark, we note that the same inapproximability result can be established under the weaker (and more conventional) complexity hypothesis FPT $\neq$ W[2]. However, the proof is significantly more involved and has been left out for the sake of brevity.

4 Complexity of Explaining Arc Inconsistency: Domain Size

We show that finding shortest arc inconsistency explanations is polynomial on binary normalized CSPs with Boolean domains. Again, this class is tight: As soon as we allow three

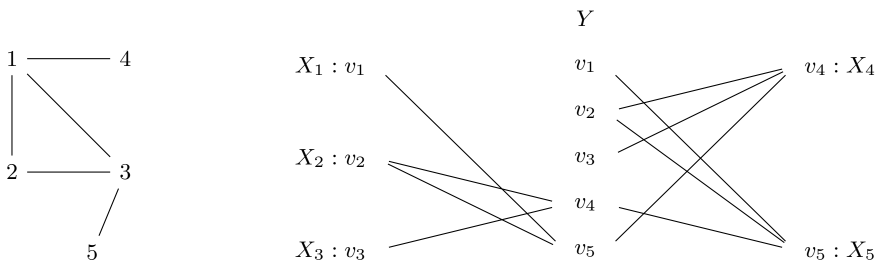


Figure 2 Left: a graph G. Right: the constraint network P_G in the proof of Lemma 9.

values per domain, the problem becomes NP-hard.

4.1 Tractability on Boolean domains

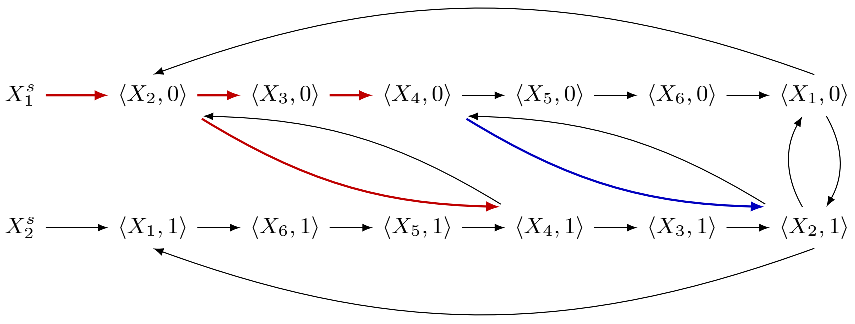
▶ **Theorem 11.** *Shortest Arc Inconsistency Explanation is solvable in polynomial time when restricted to binary normalized networks with all domains of size at most two.*

Proof. Let $P = (X, D, C)$ be a binary CSP 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 $(i = 1, \dots, n)$, and variable-value vertices $\langle X_i, a\rangle$ $(i = 1, \dots, n,\ a \in \{0, 1\})$. 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 CSP 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

$X_1 < X_2$
$X_2 \leq X_3$
$X_3 \leq X_4$
$X_4 \leq X_5$
$X_5 \leq X_6$
$X_6 \leq X_1$
$X_2 \neq X_4$



$$\begin{tikzcd}
X_1^s \arrow[r, red] & \langle X_2,0\rangle \arrow[r, red] \arrow[drrr, red, bend right] & \langle X_3,0\rangle \arrow[r, red] & \langle X_4,0\rangle \arrow[r] \arrow[drrr, blue, bend right] & \langle X_5,0\rangle \arrow[r] & \langle X_6,0\rangle \arrow[r] & \langle X_1,0\rangle \arrow[lllll, bend right] \arrow[d, bend left] \\
X_2^s \arrow[r] & \langle X_1,1\rangle \arrow[r] & \langle X_6,1\rangle \arrow[r] & \langle X_5,1\rangle \arrow[r] & \langle X_4,1\rangle \arrow[r] \arrow[ulll, bend right] & \langle X_3,1\rangle \arrow[r] & \langle X_2,1\rangle \arrow[lllll, bend left] \arrow[u, bend left] \arrow[ulll, bend right]
\end{tikzcd}$$

Figure 3 Left: a Boolean binary CSP P (the domain of every variable is $\{0,1\}$). Right: the causal graph G_P of the proof of Theorem 11. The shortest explanation involves the two paths in red originating from X_1^s and corresponds to the sequence $\langle X_2 \leftarrow X_1, X_3 \leftarrow X_2, X_4 \leftarrow X_3, X_4 \leftarrow X_2\rangle$.

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

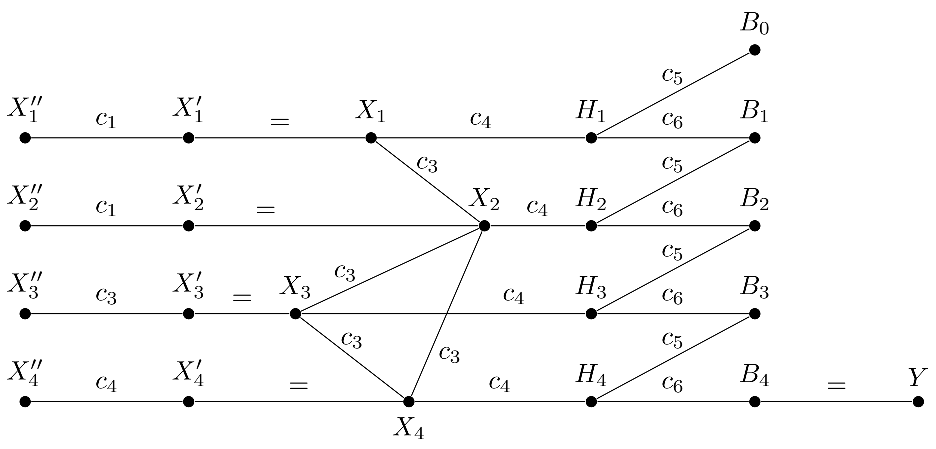


Figure 4 The constraint network P_G in the proof of Lemma 13 when looking for a dominating set in the graph $G = (\{1, 2, 3, 4\}, \{(1, 2), (2, 4), (2, 3), (3, 4)\})$.

▶ **Lemma 13.** *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 7.

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 CSP instance. Let $G = (V, E)$ be a graph with $V = \{1, \dots, n\}$.

We construct the CSP instance P_G with $5n + 2$ variables

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

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

We build the set of constraints

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

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 $X_i' \overset{c1}{\leftarrow} X_i''$ for each vertex i in S. The k next elements in R are $X_i \overset{c2}{\leftarrow} X_i'$, 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 $X_j \overset{c3}{\leftarrow} X_i$, 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 $H_i \overset{c4}{\leftarrow} X_i$, removing value 1 from $D(H_i)$ because $2 \notin D(X_i)$. The $2n$ next elements in R are $\langle H_i \overset{c5}{\leftarrow} B_{i-1}, B_i \overset{c6}{\leftarrow} H_i\rangle$ in increasing order of i from 1 to n. Each $H_i \overset{c5}{\leftarrow} B_{i-1}$ removes value 2 from $D(H_i)$ if $2 \notin D(B_{i-1})$ and $B_i \overset{c6}{\leftarrow} H_i$ 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.

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 $\langle X_i' \overset{c1}{\leftarrow} X_i'', X_i \overset{c2}{\leftarrow} X_i'\rangle$, 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 $X_j \overset{c3}{\leftarrow} X_i$ for $n - k$ variables X_j. Now, $X_j \overset{c3}{\leftarrow} X_i$ 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 $\langle X_i' \overset{c1}{\leftarrow} X_i'', X_i \overset{c2}{\leftarrow} X_i'\rangle$ 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 Conclusion

We have investigated the complexity of finding a shortest proof of inconsistency of a binary CSP 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 CSP 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 CSP 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 11.

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.

doi:10.1016/S0004-3702(01)00162-X.

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. doi:10.1016/j.artint.2021.103550.

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. doi:10.1109/FOCS.2017.74.

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. doi:10.1007/11847250_10.

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. URL: http://www.aaai.org/Library/AAAI/2008/aaai08-040.php.

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

7 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.

8 Allen Van Gelder. Verifying RUP proofs of propositional unsatisfiability. In *ISAIM*, 2008.

9 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. doi:10.1109/DATE.2003.1253718.

10 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. doi:10.1609/aaai.v33i01.33011511.

11 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.

12 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.

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

14 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. doi:10.4230/LIPIcs.ICALP.2017.78.

15 Dániel Marx. Completely inapproximable monotone and antimonotone parameterized problems. In *Proceedings of the 25th IEEE Annual Conference on Computational Complexity*, pages 181–187, 2010. doi:10.1109/CCC.2010.25.

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

17 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.