Counterfactual Explanations Under Learning From Interpretation Transition

Tony Ribeiro[1,3,4], Maxime Folschette[2], Morgan Magnin[1], Katsumi Inoue[3], Tuan Nguyen[7], Kotaro Okazaki[4], Kuo-Yen Lo[4], Jérémie Poschmann[6], and Antoine Roquilly[5,6]

[1] Centrale Nantes, CNRS, Laboratoire des Sciences du Numérique de Nantes, LS2N, UMR 6004, F-44000 Nantes, France
[2] Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
[3] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
[4] Steelous Protocol, 8-20-32, Ginza, Chuo-ku, Tokyo 104-0061, Japan
[5] Nantes Université, CHU Nantes, INSERM, Anesthesie Réanimation, F-44000 Nantes, France
[6] Inserm, Nantes Université, CHU Nantes, Center for Research in Transplantation and Translational Immunology, UMR 1064, ITUN2, F-44000, Nantes, France
[7] Hanoi University of Science and Technology, No. 1 Dai Co Viet, Hai Ba Trung, Ha Noi, Vietnam

Abstract. Counterfactual explanations are instrumental in helping humans gain insight into the decision-making processes of artificial intelligence systems by illustrating the effects of altering specific input variables. By presenting hypothetical scenarios, they foster transparency in artificial intelligence, enabling us to comprehend its operations more deeply and cultivate confidence in its dependability. This transparency is essential for the development of ethical artificial intelligence systems that are both equitable and accountable. In this paper, we expand upon the Learning From Interpretation Transition framework by proposing a theoretical modeling of counterfactual explanations for dynamic multi-valued logic programs. Furthermore, we introduce an efficient algorithm called **CELOS** that leverages properties over logic rules to compute all minimal counterfactual explanations. We show through theoretical results the correctness of our approaches. Practical evaluation is performed on benchmarks from biological literature and synthetic instances.

Keywords: Explainable artificial intelligence · inductive logic programming · logical modeling · dynamic systems · counterfactual explanations.

1 Introduction

Counterfactual explanations [26] provide valuable insights into how artificial intelligence (AI) systems arrive at their decisions, illustrating the impact of chang-

The 5th International Joint Conference on Learning & Reasoning, University of Surrey, Guildford, United Kingdom.

ing specific input variables. As demonstrated in the intuitive example by [14], consider an applicant who was rejected for a loan by a financial institution's algorithm. While the institution might simply state, "Your credit score was insufficient," such a response offers little practical guidance for improvement. However, counterfactual explanations can provide concrete, actionable feedback. For instance, they could reveal that a higher income or a lower debt-to-income ratio might have led to a different outcome. These hypothetical scenarios empower individuals with the knowledge needed to adjust their circumstances and potentially achieve a more favorable result in the future.

This concept holds significant relevance in numerous real-world applications where decisions have a profound impact on individuals' lives. Consider university admissions [21], hiring processes [27], government aid distribution [1], and identifying high-risk patients for diseases [2]. In each of these scenarios, merely knowing why something went wrong is insufficient. As highlighted by [12], people seek contrastive explanations – they want to know why event P occurred instead of event Q. What they truly need is insight into what actions they could take differently to achieve a better outcome, assuming the algorithm remains unchanged. Counterfactual explanations excel in providing this kind of insight by demonstrating how minor adjustments in a person's attributes could lead to different results. For instance, they might reveal that a higher income or a lower debt-to-income ratio might have led to a different loan outcome. These explanations are akin to a roadmap, offering a clear understanding of the algorithm's focus. Unlike other methods that attempt to approximate the algorithm's behavior, counterfactual explanations are inherently accurate as they derive directly from the model's output.

Counterfactual reasoning, when applied to dynamic systems, has some links with model verification and control theory. In counterfactual approaches, one aims to identify minimal changes needed to the input of a model to reach a different outcome. In classical model-checking approaches, one considers all the possible outcomes of the model before stating whether yes or no a given property can be verified. Finally, when such a desired property is not verified, control theory can be from help to design some modifications to the global model that would guarantee the truthfulness of the property. One limit to model verification is the reliability one can have onto the model, with regard to the real system. Distance between the model and the real system has a major impact on the quality of the knowledge one can derive from the verification process. Some approaches, typically statistical model-checking, have thus emerged to insert some uncertainty into the model. Counterfactual approaches focus on the step before confirming the model properties. It is somehow a way of proposing minimum levers of action to ensure the accessibility of certain states. Some state-of-the-art works relate to such challenges, in the context of different properties. For example, in [23], the authors propose a software tool designed to control asynchronous Boolean networks in some specific configurations. The approach consists in identifying key nodes whose perturbations can drive the network from a source attractor

(representing an initial cell state) to a target attractor (representing a desired cell state).

In this paper, we address the challenge of generating and modeling all minimal counterfactual explanations from models learned by the Learning From Interpretation Transition (LFIT) framework [7,18]. This framework represents an inductive logic programming paradigm that automatically builds a model of system dynamics from observed state transitions. In this paper, we propose a novel theoretical modeling and an efficient algorithm to produce counterfactual explanations from dynamic multi-valued logic programs produced by LFIT.

This paper is organized as follows. Section 2 provides a formalization of discrete memory-less dynamics systems as multi-valued logic programs. Section 3 models counterfactual explanations for multi-valued logic programs and introduces a simple naïve enumeration algorithm to find them. Section 4 formalizes properties and presents an efficient algorithm called **CELOS** that leverages those properties to compute counterfactual explanations more efficiently, directly utilizing the logic rules. Section 5 offers experimental evaluations to assess the scalability of the method on Boolean networks benchmarks from the literature and randomly generated programs. Section 7 concludes the paper. All proofs of theorems and propositions are provided in the appendix.

2 Dynamical Multi-Valued Logic Program

This section presents the fundamental concepts required to understand our modeling approach. The definitions provided here are largely adapted from [18], with minor modifications to align with our specific context and notation.

Let $\mathcal{V} = \{\mathrm{v}_1, \cdots, \mathrm{v}_n\}$ be a finite set of $n \in \mathbb{N}$ variables, $\mathcal{V}al$ the set in which variables take their values and $\mathsf{dom} : \mathcal{V} \to \{d \in \wp(\mathcal{V}al) \mid |d| \geq 2\}$ a function associating a domain (with at least two values) to each variable, with $\wp$ the power set. The atoms of multi-valued logic (MVL) are of the form v^{val} where $\mathrm{v} \in \mathcal{V}$ and $val \in \mathsf{dom}(\mathrm{v})$. The set of such atoms is denoted by $\mathcal{A} = \{\mathrm{v}^{val} \in \mathcal{V} \times \mathcal{V}al \mid val \in \mathsf{dom}(\mathrm{v})\}$. Let $\mathcal{F}$ and $\mathcal{T}$ be a partition of $\mathcal{V}$, that is: $\mathcal{V} = \mathcal{F} \cup \mathcal{T}$ and $\mathcal{F} \cap \mathcal{T} = \emptyset$. $\mathcal{F}$ is called the set of feature variables, which values represent the state of the system at the previous time step $(t-1)$, and $\mathcal{T}$ is called the set of target variables, which values represent the state of the system at the current time step (t). A MVL rule R is defined by:

$$R \quad = \quad \mathrm{v}_0^{val_0} \leftarrow \mathrm{v}_1^{val_1} \wedge \cdots \wedge \mathrm{v}_m^{val_m}$$

where $m \in \mathbb{N}$, and $\forall i \in [\![0; m]\!], \mathrm{v}_i^{val_i} \in \mathcal{A}$; furthermore, every variable is mentioned at most once in the right-hand part: $\forall j, k \in [\![1; m]\!], j \neq k \Rightarrow \mathrm{v}_j \neq \mathrm{v}_k$. The rule R has the following meaning: the variable v_0 can take the value val_0 in the next dynamical step if for each $i \in [\![1; m]\!]$, variable v_i has value val_i in the current dynamical step. The atom on the left side of the arrow is called the head of R, denoted $\mathrm{head}(R) := \mathrm{v}_0^{val_0}$, and is made of a target variable: $\mathrm{v}_0 \in \mathcal{T}$. The notation $\mathrm{var}(\mathrm{head}(R)) := \mathrm{v}_0$ denotes the variable that occurs in $\mathrm{head}(R)$. The

conjunction on the right-hand side of the arrow is called the body of R, written $\text{body}(R)$, and all variables in the body are feature variables: $\forall i \in [\![1; m]\!], \text{v}_i \in \mathcal{F}$. In the following, the body of a rule is assimilated to the set $\{\text{v}_1^{val_1}, \cdots, \text{v}_m^{val_m}\}$; we thus use set operations such as $\in$ and $\cap$ on it, and we denote $\emptyset$ an empty body. A dynamical multi-valued logic program (DMVLP) is a set of MVL rules.

Definition 1 (Rule Domination). *Let R_1, R_2 be* MVL *rules. R_1 dominates R_2, written $R_1 \geq R_2$ if* $\text{head}(R_1) = \text{head}(R_2)$ *and* $\text{body}(R_1) \subseteq \text{body}(R_2)$.

The dynamical system we want to learn the rules of, is represented by a succession of states as formally given by Definition 2. We also define the "compatibility" of a rule with a state in Definition 3, and with a transition in Definition 4.

Definition 2 (Discrete state and transition). *A discrete state s on a set of variables $\mathcal{X} \in \{\mathcal{F}, \mathcal{T}\}$ of a* DMVLP *is a function from $\mathcal{X}$ to $(\mathsf{dom}(\text{v}))_{\text{v}\in\mathcal{X}}$. It can be equivalently represented by the set of atoms $\{\text{v}^{s(\text{v})} \mid \text{v} \in \mathcal{X}\}$ and thus we can use classical set operations on it. We write $\mathcal{S}^{\mathcal{X}}$ to denote the set of all discrete states of $\mathcal{X}$. A state $s \in \mathcal{S}^{\mathcal{F}}$ is called a feature state and a state $s' \in \mathcal{S}^{\mathcal{T}}$ is called a target state. A couple of states $(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ is called a transition.*

Definition 3 (Rule-state matching). *Let $s \in \mathcal{S}^{\mathcal{F}}$. The* MVL *rule R matches s, written $R \sqcap s$, if* $\text{body}(R) \subseteq s$.

The objective of LFIT is to learn a program that should both: (1) match the observations in a complete (all transitions are learned) and correct (no spurious transition) way; (2) represent only minimal necessary interactions (no overly-complex rules). The following definitions formalize these desired properties.

Definition 4 (Rule and program realization). *Let R be a* MVL *rule and $(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. The rule R realizes the transition (s, s') if R matches s and* $\text{head}(R) \in s'$. *A* DMVLP *P realizes (s, s') if for any v^{val} of s', there exists $R \in P$ with* $\text{head}(R) = \text{v}^{val}$ *and R realizes (s, s'). P realizes a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ if for any (s, s') of T, P realizes (s, s').*

Definition 5 (Conflict and Consistency). *A* MVL *rule R conflicts with a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ when there is a transition (s, s') of T such that, R matches s and for any transition (s, s'') of T,* $\text{head}(R) \notin s''$. *Otherwise, R is said to be consistent with T. A* DMVLP *P is consistent with a set of transitions T if P does not contain any rule R conflicting with T.*

Definition 6 (Suitable and optimal program). *Let $T \subseteq \mathcal{S}_m^{\mathcal{F}} \times \mathcal{S}_m^{\mathcal{T}}$ a set of transitions. A* DMVLP *P is suitable for T if: P is consistent with T, P realizes T, and for any possible* MVL *rule R' consistent with T, there exists $R \in P$ such that R' dominates R. If, in addition, for all $R \in P$, all the* MVL *rules R' belonging to* DMVLP *suitable for T are such that R' dominates R implies RdominatesR', then P is called optimal and denoted $P_{\mathcal{O}}(T)$.*

In [18, Theorem 5], we proposed the General Usage LFIT Algorithm (**GULA**) that guarantees to learn the optimal program of a set of transitions: let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, $\mathbf{GULA}(\mathcal{A}, T, \mathcal{F}, \mathcal{T}) = P_{\mathcal{O}}(T)$.

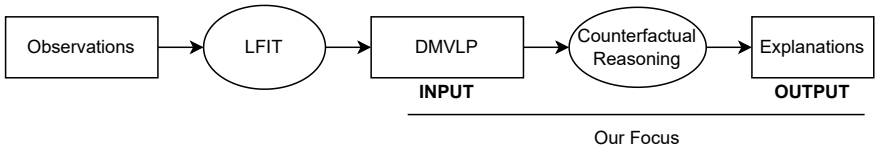

Fig. 1: This paper focuses on modeling/computing counterfactual explanations over DMVLP. This process thus happens on the output of the LFIT approach.

3 Counterfactual Reasoning with DMVLP

The present work builds on the definitions introduced earlier. As shown in Figure 1, this paper focuses on modeling and generating counterfactual explanations for a DMVLP. Since the method is independent of how the DMVLP is learned, we refer interested readers to [18] for details on learning such models using LFIT.

Definition 7 (Counterfactual Explanation Problem). *Let P be a* DMVLP*, $s \in \mathcal{S}^{\mathcal{F}}$ be a feature state, $\mathrm{v} \in \mathcal{T}$ be a target variable. A counterfactual explanation problem is a tuple $CP := (P, s, \mathrm{v}, Val_{out}, Val_{in})$ where $Val_{out} \subset \mathsf{dom}(\mathrm{v})$ and $Val_{in} \subset \mathsf{dom}(\mathrm{v})$ such that $Val_{out} \cap Val_{in} = \emptyset$. A solution to CP is a set of atoms $X = s' \setminus s$ for some feature state s' such that:*
- *no rule R of P with $\mathrm{head}(R) = \mathrm{v}^{val}, val \in Val_{out}$ matches s' and*
- *there is a rule R' of P with $\mathrm{head}(R') = \mathrm{v}^{val'}, val' \in Val_{in}$ that matches s'.*

A solution to CP is minimal if no subset of the solution is also a solution.

Proposition 1 (Solution existence). *A counterfactual explanation problem $CP = (P, s, \mathrm{v}, Val_{out}, Val_{in})$ has no solution iff there is no $s' \in \mathcal{S}^{\mathcal{F}}$ such that:*
- *there is no rule R of P with $\mathrm{head}(R) = \mathrm{v}^{val}, val \in Val_{out}$ such that R matches s' and*
- *there is a rule R' of P with $\mathrm{head}(R') = \mathrm{v}^{val'}, val' \in Val_{in}$ that matches s'.*

Given a DMVLP and a feature state, the aim of counterfactual reasoning is adjusting the feature state to prevent the program from producing unwanted outcomes (Val_{out}) while still allowing for the realization of acceptable ones (Val_{in}).

Example 1. Consider the following DMVLP P such that $\mathcal{F} = \{a, b, c\}, \mathcal{T} = \{y\}, \mathsf{dom}(a) = \mathsf{dom}(b) = \{0, 1\}, \mathsf{dom}(c) = \mathsf{dom}(y) = \{0, 1, 2\}$:

$$y^0 \leftarrow a^0 \qquad y^1 \leftarrow b^1 \qquad y^2 \leftarrow a^1 \wedge b^1 \wedge c^2$$
$$y^0 \leftarrow b^0 \qquad y^1 \leftarrow a^0 \wedge c^1$$
$$y^0 \leftarrow c^0 \qquad y^1 \leftarrow c^2$$

Let $CP := (P, s, y, \{y^1\}, \{y^0, y^2\})$ where $s = \{a^0, b^1, c^1\}$. According to the program, the values of y that can be realized from s are y^0 (via rule $y^0 \leftarrow a^0$) and y^1 (via rules $y^1 \leftarrow a^0 \wedge c^1$ and $y^1 \leftarrow b^1$). Counterfactual reasoning can be used to find changes to s that seeks to eliminate y^1 while allowing y^0 or y^2. The minimal solutions for y^0 are $\{\{a^1, b^0\}, \{b^0, c^0\}\}$. The first solution $X_1 = \{a^1, b^0\}$ produces $s' = \{a^1, b^0, c^1\}$, where s' is not matched by a rule producing y^1 and

Algorithm 1 Naïve enumeration

- **INPUT:** a counterfactual explanation problem $CP := (P, s, \mathrm{v}, \mathit{Val}_{out}, \mathit{Val}_{in})$.
- $\mathit{out\_rules} := \{R \in P \mid \mathrm{head}(R) = \mathrm{v}^{val}, \mathit{val} \in \mathit{Val}_{out}\}$
- For each val of $\mathit{Val}_{in}, \mathit{in\_rules}_{val} := \{R \in P \mid \mathrm{head}(R) = \mathrm{v}^{val}\}$
- For each val of $\mathit{Val}_{in}, \mathit{solutions}_{val} := \emptyset$
- For each complete state $s' \in \mathcal{S}^{\mathcal{F}}$:
  - If a rule of $\mathit{out\_rules}$ matches s': // Discard invalid state
    - Continue
  - For each val of Val_{in}: // Extract valid changes to get v^{val}
    - If a rule of $\mathit{in\_rules}_{val}$ matches s':
      - $\mathit{solutions}_{val} := \mathit{solutions}_{val} \cup \{\{x \in s' \mid x \notin s\}\}$
- For each val of Val_{in}: // Keep only minimal sets
  - $\mathit{solutions}_{val} := \{x \in \mathit{solutions}_{val} \mid \nexists x' \in \mathit{solutions}_{val}, x' \subset x\}$
- **OUTPUT:** $\{\mathit{solutions}_{val} \mid \mathit{val} \in \mathit{Val}_{in}\}$

s' is matched by the rule $y^0 \leftarrow b^0$. The second solution, $X_2 = \{b^0, c^0\}$ produces $s'' = \{a^0, b^0, c^0\}$, where s'' similarly avoids the rules producing y^1 while matching $y^0 \leftarrow a^0$, $y^0 \leftarrow b^0$ and $y^0 \leftarrow c^0$.

However, there is no solution for y^2 for CP because for any feature state s_1 there is $R = (y^1 \leftarrow b^1)$ in P, R does not match s_1 thus $b^1 \notin s_1$. Since the only rule for y^2 is $R' = (y^2 \leftarrow a^1 \wedge b^1 \wedge c^2)$, b^1 must be in s_1 for R' to match s_1.

Following Definition 7, we can find solutions by employing a straightforward naïve enumeration algorithm. It generates all feature states that are not matched by any rules in Val_{out} but still allow for the realization of at least one rule in Val_{in}. The differences between these generated states and the original state s yield the counterfactual solutions. Only minimal sets of changes are retained as valid solutions. The pseudo code for this approach is provided in Algorithm 1.

Theorem 1 (Naïve enumeration Complexity).
Let $CP := (P, s, \mathrm{v}, \mathit{Val}_{out}, \mathit{Val}_{in})$ be a counterfactual explanation problem. Let $n := |\mathcal{F}|$ and $d := \max(\{|\mathsf{dom}(\mathrm{v})|) \in \mathbb{N} \mid \mathrm{v} \in \mathcal{F} \cup \mathcal{T}\}$. There are d^{n+1} possible rules R with $\mathrm{head}(R) = \mathrm{v}$. The worst-case time complexity of the **Naïve enumeration** *algorithm when solving CP belongs to $\mathcal{O}(|\mathcal{S}^{\mathcal{F}}| \times nd^{n+1})$. The worst-case memory use belongs to $\mathcal{O}(|\mathcal{S}^{\mathcal{F}}| \times d)$.*

Proof sketch. Generate all possible feature states: $\mathcal{O}(\mathcal{S}^{\mathcal{F}})$. Compare each state to rules in Val_{out}: $\mathcal{O}(nd^{n+1})$. Same with rules in Val_{in}: $\mathcal{O}(nd^{n+1})$. Compute difference from original state s: $\mathcal{O}(n)$. Total time complexity: $\mathcal{O}(|\mathcal{S}^{\mathcal{F}}| \times (nd^{n+1} + nd^{n+1} + n))$ wich belongs to $\mathcal{O}(|\mathcal{S}^{\mathcal{F}}| \times nd^{n+1})$. Memory complexity: store all states for each Val_{in} $\mathcal{O}(\mathcal{S}^{\mathcal{F}} \times d)$. □

With the naïve approach established, we now turn our attention to developing a more efficient algorithm for solving counterfactual explanation problems.

4 CELOS

We now introduce a more efficient approach for solving counterfactual explanation problems over DMVLP. By leveraging properties inherent to inductive logic programming, we have developed an algorithm, which we refer to as **CELOS**. This method exploits these characteristics to find solutions more efficiently compared to the naïve enumeration approach.

4.1 Learning operations

Before presenting **CELOS**, we introduce two key definitions: Definition 8 and Proposition 2 enable us to determine if two rules share any common matching states, avoiding the need to enumerate all possibilities. Definition 9 and Proposition 3 allow us to modify a rule to ensure it does not match the same state as another rule, which is needed in finding counterfactual solutions for a DMVLP.

Definition 8 (Rule cross-matching). *Let R, R' be two* MVL *rules. The rules cross-match, denoted $R \sqcap R'$, if there exists a feature state $s \in \mathcal{S}^{\mathcal{F}}$ such that both R and R' match s.*

Proposition 2. *Let R, R' be two* MVL *rules, they cross-match if and only if: for any v^{val} of $\mathrm{body}(R)$, if there exists $\mathrm{v}^{val'} \in \mathrm{body}(R')$, it implies that $val = val'$.*
Proof sketch. Sufficient condition: since R and R' have no conflicting conditions, there exists a state s that contains all the atoms in $\mathrm{body}(R) \cup \mathrm{body}(R')$. This state s is matched by both rules, meaning they cross-match. Necessary condition: if R and R' have conflicting conditions but still cross-match, this would imply the existence of a state s containing conflicting values for the same variable. Such a state s is invalid by Definition 2, leading to a contradiction. □

Following our exploration of rule cross-matching, we now introduce a novel concept in rule specialization. While our previous work [18] focused on the least specialization of a rule against a feature state, Definition 9 presents a different approach that is tailored for counterfactual reasoning. This definition allows us to specialize a rule in the minimal manner required to ensure that only the common states with a second rule are not matched, as outlined in Proposition 3.

Definition 9 (Anti cross-matching least specialization). *Let R, R' be two* MVL *rules that cross-match. The anti cross-matching least specialization of R by R' according to $\mathcal{A}$ is:*

$$ACML_{\mathrm{spe}}(R, R', \mathcal{A}) := \{\mathrm{head}(R) \leftarrow \mathrm{body}(R) \cup \{\mathrm{v}^{val}\} \mid \mathrm{v}^{val'} \in \mathrm{body}(R'), \mathrm{v}^{val} \in \mathcal{A}, val \neq val'\}.$$

Proposition 3. *Let R, R' be two* MVL *rules. It holds that:*

1. *For any R'' of $ACML_{\mathrm{spe}}(R, R', \mathcal{A})$, R'' and R' do not cross-match.*

2. *The feature states matched by* $ACML_{\text{spe}}(R, R', \mathcal{A})$ *are all the states matched by* R *but not by* R''*:*

$$\{s \in \mathcal{S}^{\mathcal{F}} \mid R'' \in ACML_{\text{spe}}(R, R', \mathcal{A}), R'' \sqcap s\} = \{s \in \mathcal{S}^{\mathcal{F}} \mid R \sqcap s, R' \not\sqcap s\}.$$

Proof sketch. First point: The introduction of an atom conflicting with R' in each rule of $ACML_{\text{spe}}(R, R', \mathcal{A})$ ensures that no rule in $ACML_{\text{spe}}(R, R', \mathcal{A})$ can cross-match with R'. Second point: ($\subseteq$) By considering the fact that the rules R'' produced are dominated by R, and by construction. ($\supseteq$) By construction of the rules R''. □

4.2 Algorithm

In this section, we introduce **CELOS**: Counterfactual Explanation for LFIT with Optimized Search, an algorithm inspired by the principles of **GULA** [18]. **CELOS** is designed to compute all optimal solutions for a counterfactual explanation problem over a DMVLP. Unlike naïve enumeration, **CELOS** relies on the rules to derive solutions, making it more scalable in terms of the number of feature variables. Furthermore, it builds upon the proven efficiency of **GULA**'s search operations. The core search loop of **CELOS** mirrors that of **GULA**, with the primary difference being the nature of specialization: **GULA** specializes rules against states, while **CELOS** specializes rules against other rules.

Algorithm 2 CELOS

- **INPUT:** a counterfactual explanation problem $CP := (P, s, \text{v}, Val_{out}, Val_{in})$.
- $out\_rules := \{R \in P \mid \text{head}(R) = \text{v}^{val}, val \in Val_{out}\}$
- For each val of Val_{in}, $in\_rules_{val} := \{R \in P \mid \text{head}(R) = \text{v}^{val}\}$
- For each val of Val_{in}, $solutions_{val} := \emptyset$
- Initialize $P_{in} := \{\_ \leftarrow \emptyset\}$ // Rule head does not matter, only body is used
- For each rule R' of out_rules: // 1) Search necessary changes to avoid all Val_{out}
  - Extract and remove the rules of P_{in} that cross-match R':
    $M := \{R \in P_{in} \mid \forall \text{v}^{val} \in \text{body}(R), \text{v}^{val'} \in R' \implies val = val'\}$
  - $P_{in} := P_{in} \setminus M$
  - $LS := \emptyset$
  - For each rule R of M:
    - Compute its rule least specialization $P'_{in} = ACML_{\text{spe}}(R, R', \mathcal{A})$
    - Remove rules in P'_{in} dominated by a rule in P_{in}
    - $LS := LS \cup P'_{in}$
  - Add all remaining rules of LS to P_{in}: $P_{in} := P_{in} \cup LS$
- For each val of Val_{in}: // 2) Compute necessary changes to produce a Val_{in}
  - For each R of in_rules_{val}, for each R' of in_rules_{val}:
    - If R, R' cross-match: // Combine and extract changes with state s
      - $candidate := \{x \in (\text{body}(R) \cup \text{body}(R')) \mid x \notin s\}$
      - $solutions_{val} := solutions_{val} \cup \{candidate\}$
  - $solutions_{val} := \{S \in solutions_{val} \mid \nexists S' \in solutions_{val}, S' \subset S\}$
- **OUTPUT:** $\{solutions_{val} \mid val \in Val_{in}\}$

CELOS accepts as input a counterfactual explanation problem CP such that $CP := (P, s, \mathrm{v}, Val_{out}, Val_{in})$, where P is a DMVLP, s is a feature state, v is a target variable, and Val_{out}, Val_{in} are sets of integers. The algorithm begins similarly to the naïve approach by isolating the rules of Val_{out} and Val_{in} into separate sets. It then iteratively constructs a set of rules that do not cross-match any rules of Val_{out}, leveraging a modified version of the operations used in **GULA** but incorporating our novel anti cross-matching least specialization method (Definition 9). The body of these generated rules represents the necessary conditions for a feature state to evade matching by any rule of Val_{out}. Next, **CELOS** unifies these obtained rule sets with each rule of Val_{in} to produce sets of atoms that ensure no overlap with out_rules while maintaining compatibility with in_rules. By computing the difference between these sets and the given feature state s, and retaining the minimal set of changes, **CELOS** efficiently derives minimal solutions to the counterfactual explanation problem. The algorithm's pseudocode is provided in Algorithm 2.

Theorem 2 outlines the key properties of **CELOS**: the algorithm always terminates and produces all and only the minimal solutions to the given counterfactual explanation problem. Finally, Theorem 3 characterizes the algorithm's complexity in terms of both time and memory.

Theorem 2 (CELOS Termination, Soundness, Completeness, Optimality). *Let CP be a Counterfactual explanation problem.*
(1) *Any call to* **CELOS**(CP) *terminates,*
(2) **CELOS**(CP) *is the set of all minimal solution to CP*

Proof sketch. (1) Termination: The algorithm iterates over finite sets, ensuring it always terminates. (2) Starting with empty rules and iteratively applying anti-cross-matching least specialization guarantees a set of rules that match all states except those matched by Val_{out} rules. By unifying these rules with the conditions from Val_{in} rules and retaining only the minimal sets of changes, we obtain the necessary conditions to avoid every Val_{out} while having at least one Val_{in}. These resulting sets of conditions are precisely the minimal solutions to CP, and no minimal solution is missed, otherwise it would contradict with Proposition 3. □

Theorem 3 (CELOS Complexity). *Let $CP := (P, s, \mathrm{v}, Val_{out}, Val_{in})$ be a counterfactual explanation problem. Let $n := |\mathcal{F}|$ and $d := \max(\{|\mathsf{dom}(\mathrm{v})| \mid \mathrm{v} \in \mathcal{F} \cup \mathcal{T}\}) \in \mathbb{N}$. There are d^{n+1} possible rules R with $\mathrm{var}(\mathrm{head}(R)) = \mathrm{v}$. The worst-case time complexity of* **CELOS** *when solving CP belongs to $\mathcal{O}(n^2 d^n)$. Its worst-case memory use belongs to $\mathcal{O}(nd^{n+1})$.*

Proof sketch. Extracting out_rules and in_rules sets: $\mathcal{O}(nd^n)$. Iterating over P_{in} rules and out_rules applying anti-cross-match least specialization: $\mathcal{O}(n^2 nd^n)$. Fusion with rules of in_rules and computing differences: $\mathcal{O}(n^2 d^n)$. Total: $\mathcal{O}(nd^n + n^2 d^n + n^2 d^n)$ i.e., time complexity is $\mathcal{O}(n^2 d^n)$. Storing all possible rules and least specializations: $\mathcal{O}(nd^n + nd^{n+1})$, i.e., memory complexity is $\mathcal{O}(nd^{n+1})$. □

Example 2. Let $CP := (P, s, y, \{y^1\}, \{y^0, y^2\})$ be the counterfactual explanation problem of Example 1. Table 1 shows the evolution of P_{in} over out_rules and the solution found through union of cross-matching rules with in_rules during a

call to **CELOS**(CP). The rule with empty body is first revised against $y^1 \leftarrow b^1$ into $\_ \leftarrow b^0$. Then revised again by $y^1 \leftarrow a^1 \wedge c^1$ into three rules. Two of them (in red) are revised against the final rule $y^1 \leftarrow c^2$. The least specialization of $\_ \leftarrow b^0 \wedge c^2$ by $y^1 \leftarrow c^2$ produces no rule. $\_ \leftarrow a^1 \wedge b^0$ produces two rules but one is dominated by $\_ \leftarrow b^0 \wedge c^0$ (in blue) and thus discarded. Remain only two rules: $\_ \leftarrow b^0 \wedge c^0$ and $\_ \leftarrow a^1 \wedge b^0 \wedge c^1$. By making the union of those rules with the one of *in_rules* that cross-match, we obtain the solutions to CP that are $\{a^1 \wedge b^0, b^0 \wedge c^0\}$.

• $P_{in} = \{\_ \leftarrow \emptyset\}$

$R' \in out\_rules$	M	Least specializations	P_{in}
$y^1 \leftarrow b^1$	$\{\emptyset\}$	$\{b^0\}$	$\{b^0\}$
$y^1 \leftarrow a^0 \wedge c^1$	$\{b^0\}$	$\{a^1 \wedge b^0, b^0 \wedge c^0, b^0 \wedge c^2\}$	$\{a^1 \wedge b^0, b^0 \wedge c^0, b^0 \wedge c^2\}$
$y^1 \leftarrow c^2$	$\{a^1 \wedge b^0, b^0 \wedge c^2\}$	$\{a^1 \wedge b^0 \wedge c^0, a^1 \wedge b^0 \wedge c^1\}$	$\{b^0 \wedge c^0, a^1 \wedge b^0 \wedge c^1\}$

• $s = \{a^0, b^1, c^1\}$

$R \in in\_rules$	body(R'), $R' \in P_{in}, R \sqcap R'$	Candidates	Diff with s
$y^0 \leftarrow a^0$	$\{b^0 \wedge c^0\}$	$\{a^0 \wedge b^0 \wedge c^0\}$	$\{b^0 \wedge c^0\}$
$y^0 \leftarrow b^0$	$\{b^0 \wedge c^0, a^1 \wedge b^0 \wedge c^1\}$	$\{b^0 \wedge c^0, a^1 \wedge b^0 \wedge c^1\}$	$\{b^0 \wedge c^0, a^1 \wedge b^0\}$
$y^0 \leftarrow c^0$	$\{b^0 \wedge c^0\}$	$\{b^0 \wedge c^0\}$	$\{b^0 \wedge c^0\}$
$y^2 \leftarrow a^1 \wedge b^1 \wedge c^1$	$\emptyset$	$\emptyset$	$\emptyset$

Table 1: Iterative evolution of P_{in} over each element of *out_rules* during the execution of **CELOS**(CP) over the counterfactual explanation problem CP of Example 1. Head of rules are omitted for M, Least specializations and P_{in}.

5 Evaluation

In this section, we evaluate the scalability of **CELOS** using Boolean network benchmarks from the biological literature (Boolenet [4] and Pyboolnet [9]) and synthetic data. All experiments[8] were performed on a single core of an AMD Ryzen 9 (7950X, 4.5 GHz) with 64 Gb of RAM.

5.1 Biological literature benchmarks

In this section, we convert Boolean networks into DMVLPs where each variable v of $\mathcal{V}$ has a Boolean domain ($\mathsf{dom}(v) = \{0, 1\}$). The file formats described in [4,9] provide Boolean functions for each variable in disjunctive normal form (DNF), a disjunction of conjunction clauses that can be considered as a set of Boolean atoms of the form v or $\neg$v. Each clause c of the DNF of a variable v is directly converted into a rule R with $\text{head}(R) = \text{v}_t^1$ and $\text{v}'^1_{t-1} \in \text{body}(R) \iff \text{v}' \in c$

[8] Source code is available at: `https://github.com/Tony-sama/pylfit` in the folder tests/evaluations/ijclr2025. All experiments were run with pylfit release version 0.5.1.

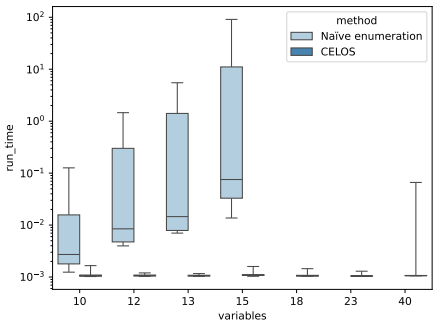

Fig. 2: Runtime (in seconds, logarithmic scale) comparison of **CELOS** and **Naïve enumeration** when applied to random counterfactual explanation problems derived from Boolean networks with 10 to 40 variables. The experiments used 100 runs per target variable value, with a timeout of 1,000 seconds.

and $\mathrm{v}'^{0}_{t-1} \in \mathrm{body}(R) \iff \neg\mathrm{v}' \in c$. To obtain the rules of v^0_t, we compute the negation of the Boolean function and derive the minimal set of atoms that make the resulting formula true. For each target variable v of $\mathcal{T}$, we generate 100 random counterfactual explanation problems as follows. (1) Randomly select a feature state $s \in \mathcal{S}^{\mathcal{F}}$. (2) Determine Val_{out}, as the head atom of the rules of v that match s (all benchmarks are deterministic programs, thus $|Val_{out}| = 1$). (3) Select Val_{in}, the other value from the target's domain than the one in Val_{out} i.e., either $Val_{out} = \{0\}$ and $Val_{in} = \{1\}$, or $Val_{out} = \{1\}$ and $Val_{in} = \{0\}$. It ensures changes must be find to solve the counterfactual problem.

As shown in Figure 2, the runtime (in seconds) of Algorithm 1 and Algorithm 2 increases as the number of variables in the Boolean network benchmarks grows. The naïve method requires enumerating all possible feature states, leading to rapid exponential explosion and timeouts (1000 seconds) on benchmarks with more than 15 variables. In contrast, **CELOS** only processes the rules of the DMVLP, allowing it to solve all benchmarks in under a second. It's important to note that the rules in these benchmarks are relatively simple, typically containing fewer than 6 conditions per rule and rarely more than 10 rules per variable. This simplicity enables **CELOS** to achieve such fast computation times.

5.2 Synthetic DMVLPs

While the biological benchmarks were relatively easy for **CELOS** due to their simplicity at the rules level, we also conducted experiments on synthetic DMVLPs. We consider DMVLPs with only one target variable where all features and the target variable v have the same number of domain values. We first generate a random state feature s and randomly select one target value to avoid and one to obtain: $Val_{out} = \{\mathrm{v}^{val}\}$ and $Val_{in} = \{\mathrm{v}^{val'}\}$ with $\mathrm{v}^{val} \neq \mathrm{v}^{val'}$. We then generate a fixed number of rules for each target value by randomly adding conditions, ensuring that all rules of v^{val} match the state s while none of the rules of $\mathrm{v}^{val'}$

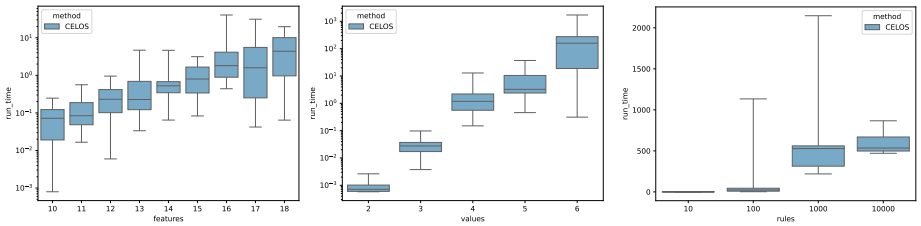

Fig. 3: Runtime (in seconds, logarithmic scale) of Algorithm 2 when applied to random counterfactual explanation problems from synthetic DMVLP with a single target variable. The experiments used 10 runs for each problem setting. Left: Increasing the number of variables from 10 to 18 while keeping the domain size fixed at 3 and 20 rules per target value. Center: Increasing the domain size from 2 to 6 while keeping the number of variables fixed at 10 and 20 rules per target value. Right: Increasing the number of rules per target value while the number of variables is fixed at 14 and the domain size is fixed at 3.

match s. This setup guarantees that changes are necessary to solve the counterfactual problem. As shown in Figure 3, the runtime (in seconds) for synthetic DMVLPs depends on the number of variables, domain values, and rules per target value. Using a base setting of 3 domain values and 20 rules per target value, we performed three experiments. (1) Varying the number of variables from 10 to 18. (2) Increasing the domain size from 2 to 6. (3) Scaling the number of rules per target from 10 to $10,000$. The results align with the complexity analysis in Theorem 3: Runtime grows exponentially with the number of variables and domain values. It grows polynomially with the number of rules per target. Random instances are particularly challenging due to the lack of useful patterns in the rules, which prevent search space reduction. Despite these challenges, **CELOS** can handle up to 18 variables with 3 domain values, even with thousands of rules per target, which is promising for practical applications.

6 Related Works

Counterfactual reasoning has been extensively studied within the framework of causal inference [10,13], particularly in Pearl's structural equation models [16]. There is growing interest in applying counterfactual explanations to interpret black box models [26,24]. Our work focuses on logic programs, which are human-readable models (white box model) produced by LFIT, allowing us to directly leverage the rules for efficient counterfactual search. In [17], the authors propose an approach for generating contrastive explanations in inductive logic programming, which are equivalent to finding counterfactual explanations. Their work focuses on first-order logic programs, whereas DMVLPs are based on propositional logic. While they enumerate candidate solutions directly, our method **CELOS** leverages the rules of the program to efficiently search for minimal counterfactual

explanations. Regarding probabilistic logic programs, recent work by [8] extends ProbLog, with the capability to process counterfactual queries. Empirical evaluations show that their top-down compilation method outperforms bottom-up approaches in handling evidence and interventions, demonstrating scalability for moderate program sizes.

There are many approaches for finding counterfactual explanations [6,5]. The simplest method involves a brute force grid search over features (like our baseline naïve enumeration algorithm), which is inefficient and scales poorly due to combinatorial explosion. Many existing methods use optimization techniques [26,3], where a cost function measures the distance between original and counterfactual instances, and optimization algorithms minimize this cost. Another approach employs heuristic search strategies [11,25], which iteratively modify the original instance until a valid counterfactual is found. These methods may be faster than brute force and sometimes optimization methods, but can still suffer from local optima and may not guarantee finding the best counterfactual explanations.

Generating meaningful counterfactual examples that aid comprehension remains a challenge. To foster a deeper understanding of machine learning models, we need a comprehensive set of such examples [22,14]. These examples should cover a wide range of possible changes (diversity), highlight feasible changes (proximity), and align with domain specific constraints. In this paper, we established the theoretical foundations for finding counterfactual explanations in DMVLP models and propose **CELOS**, a complete algorithm for finding all minimal solutions. Using heuristic methods to select interesting subsets of minimal counterfactual explanations for LFIT is outside the scope of this work and left for future research.

7 Conclusion

In this paper, we introduced a novel modeling approach to address the challenge of generating minimal counterfactual explanations from DMVLP. We formalized theoretical properties and guarantees that enable the discovery of all minimal counterfactual explanations for a given DMVLP regarding specified desired and undesired outcomes. To implement this approach, we proposed **CELOS**, an algorithm designed to find these minimal solutions efficiently. Experimental results demonstrated the effectiveness and scalability of our method, marking a significant advancement in logical modeling techniques with potential real-world applications, particularly in fields like Systems Biology.

In our previous work [19], we introduced **PRIDE**, an algorithm for finding a subset of the optimal DMVLP in polynomial time. The principles behind **PRIDE** could potentially be adapted to efficiently extract a meaningful subset of the counterfactual explanations generated by **CELOS**. This adaptation would allow us to address larger problem instances in practice, which is a promising direction for future research. In [20], we extended the LFIT framework to model partially observable systems, i.e., learning DMVLP from data with unknown values. Our method for generating counterfactual explanations can be applied

to these models as well, broadening the applicability of our work to datasets with partially observed information.

In the future, we envision leveraging the features involved in counterfactual explanations as a novel metric to assess the sensitivity of rules learned by LFIT. This metric could provide valuable insights into how robust the learned model is to changes, helping us gauge its confidence and reliability. By integrating this approach, we aim to enhance our understanding of the model's behavior and improve its applicability in real-world scenarios, particularly in medical and biological contexts where precision is paramount.

8 Acknowledgements

This research has been funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or ERCEA. Neither the European Union nor the granting authority can be held responsible for them. This work has also been supported by JSPS KAKENHI Grant Number JP25K03190 and JST CREST Grant Number JPMJCR22D3, Japan. The English quality of this manuscript was enhanced by Yumi, a virtual assistant hosted locally, using the large language model "Nemotron-15b-Thinker" [15] from Nitral-AI on a Nvidia RTX4090 using Koboldcpp (`https://github.com/LostRuins/koboldcpp`) and SillyTavern (`https://github.com/SillyTavern/SillyTavern`).

References

1. Andini, M., Ciani, E., De Blasio, G., D'Ignazio, A., Salvestrini, V.: Targeting policy-compliers with machine learning: an application to a tax rebate programme in italy. Bank of Italy Temi di Discussione (Working Paper) No **1158** (2017)
2. Dai, W., Brisimi, T.S., Adams, W.G., Mela, T., Saligrama, V., Paschalidis, I.C.: Prediction of hospitalization due to heart diseases by supervised learning methods. International journal of medical informatics **84**(3), 189–197 (2015)
3. Dhurandhar, A., Chen, P.Y., Luss, R., Tu, C.C., Ting, P., Shanmugam, K., Das, P.: Explanations based on the missing: Towards contrastive explanations with pertinent negatives. Advances in neural information processing systems **31** (2018)
4. Dubrova, E., Teslenko, M.: A SAT-based algorithm for finding attractors in synchronous boolean networks. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB) **8**(5), 1393–1399 (2011)
5. Guidotti, R.: Counterfactual explanations and how to find them: literature review and benchmarking. Data Mining and Knowledge Discovery **38**(5), 2770–2824 (2024)
6. Ignatiev, A., Narodytska, N., Asher, N., Marques-Silva, J.: From contrastive to abductive explanations and back again. In: International Conference of the Italian Association for Artificial Intelligence. pp. 335–355. Springer (2020)
7. Inoue, K., Ribeiro, T., Sakama, C.: Learning from interpretation transition. Machine Learning **94**(1), 51–79 (2014)

8. Kiesel, R., RÜCKSCHLO, K., Weitkämper, F.: "what if?" in probabilistic logic programming. Theory and Practice of Logic Programming **23**(4), 884–899 (2023)
9. Klarner, H., Streck, A., Siebert, H.: PyBoolNet: a python package for the generation, analysis and visualization of boolean networks. Bioinformatics **33**(5), 770–772 (12 2016)
10. Kusner, M.J., Loftus, J., Russell, C., Silva, R.: Counterfactual fairness. Advances in neural information processing systems **30** (2017)
11. Martens, D., Provost, F.: Explaining data-driven document classifications. MIS quarterly **38**(1), 73–100 (2014)
12. Miller, T.: Explanation in artificial intelligence: Insights from the social sciences. Artificial Intelligence **267**, 1–38 (2019)
13. Mittelstadt, B., Russell, C., Wachter, S.: Explaining explanations in ai. In: Proceedings of the conference on fairness, accountability, and transparency. pp. 279–288 (2019)
14. Mothilal, R.K., Sharma, A., Tan, C.: Explaining machine learning classifiers through diverse counterfactual explanations. In: Proceedings of the 2020 conference on fairness, accountability, and transparency. pp. 607–617 (2020)
15. Nitral-AI: Nemotron-15b-thinker-v0.1, a finetuned llm model based on apriel-nemotron-15b-thinker (May 2025), `https://huggingface.co/Nitral-AI/Nemotron-15b-Thinker-v0.1`
16. Pearl, J.: Causality. Cambridge university press (2009)
17. Rabold, J., Siebers, M., Schmid, U.: Generating contrastive explanations for inductive logic programming based on a near miss approach. Machine Learning **111**(5), 1799–1820 (2022)
18. Ribeiro, T., Folschette, M., Magnin, M., Inoue, K.: Learning any memory-less discrete semantics for dynamical systems represented by logic programs. Machine Learning (2021)
19. Ribeiro, T., Folschette, M., Magnin, M., Inoue, K.: Polynomial algorithm for learning from interpretation transition. In: 1st International Joint Conference on Learning & Reasoning. pp. 1–5 (2021)
20. Ribeiro, T., Folschette, M., Magnin, M., Okazaki, K., Lo, K.Y., Roquilly, A., Poschmann, J., Inoue, K.: Learning From Interpretation Transitions with Unknowns. In: 4th International Joint Conference on Learning & Reasoning. Nanjing, China (Sep 2024)
21. Rockoff, J.E., Jacob, B.A., Kane, T.J., Staiger, D.O.: Can you recognize an effective teacher when you recruit one? Education finance and Policy **6**(1), 43–74 (2011)
22. Russell, C.: Efficient search for diverse coherent explanations. In: Proceedings of the conference on fairness, accountability, and transparency. pp. 20–28 (2019)
23. Su, C., Pang, J.: Cabean: a software for the control of asynchronous boolean networks. Bioinformatics **37**(6), 879–881 (2021)
24. Verma, S., Boonsanong, V., Hoang, M., Hines, K., Dickerson, J., Shah, C.: Counterfactual explanations and algorithmic recourses for machine learning: A review. ACM Computing Surveys **56**(12), 1–42 (2024)
25. Vermeire, T., Brughmans, D., Goethals, S., De Oliveira, R.M.B., Martens, D.: Explainable image classification with evidence counterfactual. Pattern Analysis and Applications **25**(2), 315–335 (2022)
26. Wachter, S., Mittelstadt, B., Russell, C.: Counterfactual explanations without opening the black box: Automated decisions and the gdpr. Harv. JL & Tech. **31**, 841 (2017)
27. Waters, A., Miikkulainen, R.: Grade: Machine learning support for graduate admissions. Ai Magazine **35**(1), 64–64 (2014)

A Appendix: Proofs of Section 3

Theorem 1*:*
Let $CP := (P, s, \mathrm{v}, Val_{out}, Val_{in})$ *be a counterfactual explanation problem. Let* $n := |\mathcal{F}|$ *and* $d := \max(\{|\mathsf{dom}(\mathrm{v})|) \in \mathbb{N} \mid \mathrm{v} \in \mathcal{F} \cup \mathcal{T}\}$*. There are* d^{n+1} *possible rules* R *with* $\mathrm{head}(R) = \mathrm{v}$*. The worst-case time complexity of the* **Naïve enumeration** *algorithm when solving* CP *belongs to* $\mathcal{O}(|\mathcal{S}^{\mathcal{F}}| \times nd^{n+1})$*. The worst-case memory use belongs to* $\mathcal{O}(|\mathcal{S}^{\mathcal{F}}| \times d)$*.*

Proof. Each possible feature state is first compared to all rules in $\mathcal{V}al_{out}$. With at most d values in $\mathcal{V}al_{out}$, d^n rules per value and at-most n conditions per rules, this step has a complexity of $\mathcal{O}(nd^{n+1})$. Matching is then checked over the rules of $\mathcal{V}al_{in}$, which has the same complexity. Finally, computing the difference between the state and the given state s takes $\mathcal{O}(n)$ time Thus, the complete process of the algorithm has a time complexity of $\mathcal{O}(|\mathcal{S}^{\mathcal{F}}| \times (nd^{n+1} + nd^{n+1} + n))$ i.e., $\mathcal{O}(nd^{n+1})$.

Regarding memory, in the worst case, all possible states could be solutions, and all possible values of v could be in $\mathcal{V}al_{in}$. Therefore, the memory complexity is $\mathcal{O}(|\mathcal{S}^{\mathcal{F}}| \times d)$. □

B Appendix: Proofs of Section 4

Proposition 2*: Let* R, R' *be two* MVL *rules, they cross-match if and only if: for any* v^{val} *of* $\mathrm{body}(R)$*, if there exists* $\mathrm{v}^{val'} \in \mathrm{body}(R')$*, it implies that* $val = val'$*.*

Proof. Sufficient Condition: assume that for all $\mathrm{v}^{val} \in \mathrm{body}(R)$ and $\mathrm{v}^{val'} \in \mathrm{body}(R')$, $val = val'$. Then, there exists a state $s \in \mathcal{S}^{\mathcal{F}}$ such that $(\mathrm{body}(R) \cup \mathrm{body}(R')) \subseteq s$. Since $\mathrm{body}(R) \subseteq s$ and $\mathrm{body}(R') \subseteq s$, both rules R and R' match s. Therefore, R and R' cross-match.

Necessary Condition: now suppose that R and R' cross-match but the property does not hold. Then, there exists a state $s \in \mathcal{S}^{\mathcal{F}}$ such that $R \sqcap s$ and $R' \sqcap s$. This implies $\mathrm{body}(R) \subseteq s$ and $\mathrm{body}(R') \subseteq s$. However, since the property does not hold, there exist $\mathrm{v}^{val} \in \mathrm{body}(R)$ and $\mathrm{v}^{val'} \in \mathrm{body}(R')$ such that $val \neq val'$. This means that s contains conflicting values for the same variable, which is not allowed in a valid discrete state. This contradiction shows that the property is indeed necessary. □

Proposition 3*: Let* R, R' *be two* MVL *rules. It holds that:*

1. *For any* R'' *of* $ACML_{\mathrm{spe}}(R, R', \mathcal{A})$*,* R'' *and* R' *do not cross-match.*
2. *The feature states matched by* $ACML_{\mathrm{spe}}(R, R', \mathcal{A})$ *are all the states matched by* R *but not by* R''*:*

$$\{s \in \mathcal{S}^{\mathcal{F}} \mid R'' \in ACML_{\mathrm{spe}}(R, R', \mathcal{A}), R'' \sqcap s\} = \{s \in \mathcal{S}^{\mathcal{F}} \mid R \sqcap s, R' \not\sqcap s\}.$$

Proof. First point: according to Definition 9, for any rule $R'' \in ACML_{\text{spe}}(R, R', \mathcal{A})$, there exists a pair of atoms $\text{v}^{val'} \in \text{body}(R')$ and $\text{v}^{val''} \in \text{body}(R'')$ such that $val' \neq val''$. By the Proposition 2, this ensures that R' and R'' do not cross-match, i.e., $R' \not\upharpoonright R''$.

Second point: let $M := \{s \in \mathcal{S}^{\mathcal{F}} \mid R'' \in ACML_{\text{spe}}(R, R', \mathcal{A}), R'' \sqcap s\}$ be the set of states matched by the least specialization of R against R', and $N := \{s \in \mathcal{S}^{\mathcal{F}} \mid R \sqcap s, R' \not\upharpoonright s\}$ the set of states matched by R but not by R'.

($\subseteq$) Consider $s \in M$. First, suppose by contradiction that $R \not\upharpoonright s$. By definition of matching, this means there exists an atom $\text{v}^{val} \in s$ and $\text{v}^{val'} \in \text{body}(R)$ with $val \neq val'$. However, for all $R'' \in ACML_{\text{spe}}(R, R', \mathcal{A})$, there is: $\text{body}(R) \subset \text{body}(R'')$, therefore $R'' \not\upharpoonright s$. This contradicts the assumption that $s \in M$. Therefore, M can only contain states matched by R. Second, by the first point of the proof, since $R'' \sqcap s$, it comes: $R' \not\upharpoonright s$. Therefore, $S \in N$.

($\supseteq$) Consider $s \in N$. Since $R \sqcap s$, this means that $\text{body}(R) \subseteq s$. Moreover, since $R' \not\upharpoonright s$, this means that there exists $\text{v}^{val'} \in \text{body}(R')$ and $\text{v}^{val} \in \mathcal{A}$ with $val \neq val'$ so that $\text{v}^{val} \in s$. Consider the rule $R'' := \text{head}(R) \leftarrow \text{body}(R) \cup \{\text{v}^{val}\}$. We thus have: $R'' \sqcap s$, and by Definition 9: $s \in M$. □

Theorem 2*: Let CP be a Counterfactual explanation problem.*

(1) *Any call to* **CELOS**(CP) *terminates,*

(2) **CELOS**(CP) *is the set of all minimal solution to CP*

Proof. (1) The algorithm operates on finite sets, ensuring that it always terminates after a finite number of iterations.

(2) The main loop of **CELOS** begins with an initial rule whose empty body matches all possible feature states. Iteratively, the algorithm applies the least specialization operation to each rule in *out_rules*, ensuring that no rule in P_{in} cross-matches with any rule in *out_rules* while still matching all states that were previously matched. This guarantees that the rules in P_{in} only match states that are not covered by the rules in *out_rules*.

For each $val \in Val_{in}$, the algorithm combines the body of the rules in P_{in} with the body of the rules in $in\_rules(val)$. The resulting sets of atoms, when present in a state, ensure the state is not matched by any rule in *out_rules* but is still matched by at least one rule in $in\_rules(val)$. Finally, the algorithm computes the minimal difference between these sets and the given feature state s, yielding the minimal sets of changes required to transform s into a solution of the counterfactual explanation problem. Thus, **CELOS** outputs precisely those minimal sets of changes that ensure the resulting state is not matched by any rule of Val_{out} but is matched by at least one rule of Val_{in}, which corresponds exactly to minimal solutions of the CP.

Now, let us suppose that **CELOS** missed some minimal solution: there exists $s' \in \mathcal{S}^{\mathcal{F}}$ such that there exists no $R \in out\_rules$ with $R \sqcap s'$ and there exists $R' \in in\_rules$ with $R' \sqcap s'$, but there exists no $sol \in$ **CELOS**(CP) with $sol \subseteq s'$. Since there exists no $R \in out\_rules$ with $R \sqcap s'$, our previous analysis shows that there exists $R'' \in P_{in}$ with $R'' \sqcap s'$. And since there exists $R' \in in\_rules$ with $R' \sqcap s'$, **CELOS** would have generated a solution $sol \subseteq \text{body}(R'') \cup \text{body}(R')$

that is a subset of s'. This contradicts the assumption that **CELOS** missed this solution. Therefore, **CELOS** outputs every possible minimal solution to the given counterfactual reasoning problem (CP). □

Theorem 3*: Let* $CP := (P, s, \mathrm{v}, Val_{out}, Val_{in})$ *be a counterfactual explanation problem. Let* $n := |\mathcal{F}|$ *and* $d := \max(\{|\mathsf{dom}(\mathrm{v})| \mid \mathrm{v} \in \mathcal{F} \cup \mathcal{T}\}) \in \mathbb{N}$*. There are* d^{n+1} *possible rules* R *with* $\mathrm{var}(\mathrm{head}(R)) = \mathrm{v}$*. The worst-case time complexity of* **CELOS** *when solving* CP *belongs to* $\mathcal{O}(n^2 d^n)$*. Its worst-case memory use belongs to* $\mathcal{O}(nd^{n+1})$*.*

Proof. Initialization Phase: The algorithm first isolates the rules belonging to *out_rules* and *in_rules* sets, which takes $\mathcal{O}(nd^n)$ time. Search Phase: The algorithm iterates over a set of rules P_{in} with the same head. There are at most d^n possible distinct rule bodies, so $|P_{in}| \leq d^n$. Since each rule has at most n conditions, the memory usage for storing P_{in} is $\mathcal{O}(nd^n)$. For each rule $R' \in$ *out_rules*, the algorithm identifies rules in P_{in} that cross-match R' and isolates them into a set M. This extraction step has a time complexity of $\mathcal{O}(nd^n)$. Each rule in M undergoes anti-cross-match least specialization, which takes $\mathcal{O}(n^2)$ time per rule. Since $|M| \leq d^n$, the total time for all revisions is $\mathcal{O}(n^2 d^n)$. The resulting least specialization rules are stored in a list LS. There are at most dn revisions per rule, so $|LS| \leq d^n \times dn$. This extends the memory usage bound to $\mathcal{O}(nd^n + nd^{n+1})$. Final Phase: The algorithm compares rules in P_{in} for cross-matching with all rules in *in_rules*. This step has a time complexity of $\mathcal{O}(nd^n)$. Additionally, the algorithm computes the difference between rule bodies and the given state s, which takes $\mathcal{O}(n)$ time per comparison. This contributes an overall time complexity of $\mathcal{O}(n^2 d^n)$ for this phase. Overall Complexity: Combining all phases, the worst-case time complexity of **CELOS** is $\mathcal{O}(nd^n + n^2 d^n + n^2 d^n)$ i.e., $\mathcal{O}n^2 d^n)$. The worst-case memory use remains $\mathcal{O}(nd^n + nd^{n+1})$ i.e., $\mathcal{O}(nd^{n+1})$.

□