Polynomial Algorithm For Learning From Interpretation Transition

Tony Ribeiro[1,2,3], Maxime Folschette[4], Morgan Magnin[2,3], and Katsumi Inoue[3]

[1] Independant Researcher
[2] Univ. Nantes, CNRS, Centrale Nantes, UMR 6004 LS2N, F-44000 Nantes, France
[3] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
[4] Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Abstract. Learning from interpretation transition (LFIT) automatically constructs a model of the dynamics of a system from the observation of its state transitions. The previously proposed General Usage LFIT Algorithm (**GULA**) serves as the core block to several methods of the framework that capture different dynamics. But its exponential complexity limits the use of the whole framework to relatively small systems. In this paper, we introduce an approximated algorithm (**PRIDE**) which trades the completeness of **GULA** for a polynomial complexity. Both **GULA** and **PRIDE** source codes are available as open source at `https://github.com/Tony-sama/pylfit` under GPL-3.0 License.

Keywords: logical modeling · dynamic systems · algorithm

1 Introduction

Modeling the dynamics of systems with many interactive components is crucial in many applications such as physics, cellular automata or biological systems. Learning From Interpretation Transition (LFIT) [1] is an Inductive Logic Programming framework that addresses this challenge by automatically constructing a model of the dynamics of a system from the observation of its state transitions. Given some raw data, like time-series of gene expression, a discretization of those data in the form of state transitions is assumed. From those state transitions, according to the semantics of the system dynamics, several inference algorithms modeling the system as a logic program have been proposed. In [2], we extended this framework to learn systems dynamics independently of its update semantics. For this purpose, we proposed a modeling of discrete memory-less multi-valued systems as logic programs in which each rule represents that a variable possibly takes some value at the next state. This modeling allows us to characterize optimal programs independently of the update semantics, allowing to model the dynamics of a wide range of discrete systems. To learn such semantic-free optimal programs, we proposed **GULA**: the General Usage LFIT Algorithm that now serves as the core block to several methods of the framework. But its complexity is exponential, thus limiting its use to relatively small systems (about 13 Boolean variables). In this paper, we introduce a new algorithm (**PRIDE**) whose polynomial complexity allow LFIT to deal with more complex systems.

2 Dynamical Multi-valued Logic Program

In this section, the concepts necessary to understand the learning algorithms we propose are formalized. 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 $\mathsf{dom} : \mathcal{V} \to \wp(\mathcal{V}al)$ a function associating a domain to each variable. The atoms of *multi-valued logic* ($\mathcal{M}$VL) 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 $\mathcal{M}$VL *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 $\mathrm{body}(R)$, and all variables in the body are feature variables: $\forall i \in [\![1; m]\!], \mathrm{v}_i \in \mathcal{F}$. In the following, the body of a rule is assimilated to the set $\{\mathrm{v}_1^{val_1}, \cdots, \mathrm{v}_m^{val_m}\}$; 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* ($\mathcal{DM}$VLP) is a set of $\mathcal{M}$VL rules.

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

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). *A* discrete state *s on $\mathcal{T}$ (resp. $\mathcal{F}$) of a $\mathcal{DM}$VLP is a function from $\mathcal{T}$ (resp. $\mathcal{F}$) to $\mathbb{N}$. It can be equivalently represented by the set of atoms $\{\mathrm{v}^{s(\mathrm{v})} \mid \mathrm{v} \in \mathcal{T}$ (resp. $\mathcal{F})\}$ and thus we can use classical set operations on it. We write $\mathcal{S}^{\mathcal{T}}$ (resp. $\mathcal{S}^{\mathcal{F}}$) to denote the set of all discrete states of $\mathcal{T}$ (resp. $\mathcal{F}$), and 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 $\mathcal{M}$VL rule R* matches *s, written $R \sqcap s$, if $\mathrm{body}(R) \subseteq s$.*

The final program we want to learn 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 $\mathcal{M}$VL rule and $(s, s') \in \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. The rule R* realizes *the transition (s, s') if $R \sqcap s \wedge \text{head}(R) \in s'$. A $\mathcal{DM}$VLP P* realizes *(s, s') if $\forall \text{v} \in \mathcal{T}, \exists R \in P, \text{var}(\text{head}(R)) = \text{v} \wedge R$ realizes (s, s'). P* realizes *a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ if $\forall (s, s') \in T, P$ realizes (s, s').*

Definition 5 (Conflict and Consistency). *A $\mathcal{M}$VL rule R* conflicts *with a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$ when $\exists s \in \mathcal{S}^{\mathcal{T}}, \exists (s_1, s_2) \in T, s = s_1 \wedge \big(R \sqcap s \wedge \forall (s, s') \in T, \text{head}(R) \notin s'\big)$. Otherwise, R is said to be* consistent *with T. A $\mathcal{DM}$VLP 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 $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. A $\mathcal{DM}$VLP P* is suitable for *T if: P is consistent with T, P realizes T, and for any possible $\mathcal{M}$VL rule R consistent with T, there exists $R' \in P$ s.t. $R' \geq R$. If in addition, for all $R \in P$, all the $\mathcal{M}$VL rules R' belonging to $\mathcal{DM}$VLP suitable for T are such that $R' \geq R$ implies $R \geq R'$, then P is called* optimal *and denoted $P_{\mathcal{O}}(T)$.*

3 PRIDE

The two following properties allow to build a polynomial version of **GULA** which we name **PRIDE** for Polynomial Relational Inference of Discrete Events, and which pseudo code is given in Algorithms 1 and 2. Theorem 1 states that for any transition, we can build a rule that is consistent and realizes the transition. Theorem 2 states that if removing any atom from a rule makes it inconsistent, then this rule cannot be simplified and is part of the optimal program.

Theorem 1 (Consistent Rule Always exists). *Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, $(s, s') \in T$ and $\text{v}^{val} \in s'$. The rule $R = \text{v}^{val} \leftarrow s$ is consistent with T and realizes (s, s').*

Theorem 2 (Irreducible Rules are Optimal). *Let R be a rule consistent with a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. If $\forall R' \in \{\text{head}(R) \leftarrow \text{body}(R) \setminus \{\text{v}^{val}\} \mid \text{v}^{val} \in \text{body}(R)\}$, R' conflicts with T, then $\nexists R'' \neq R$ consistent with T such that $R'' \geq R$ and thus, $R \in P_{\mathcal{O}}(T)$.*

PRIDE learns a program $P \subseteq P_{\mathcal{O}}(T)$ sufficient to realize T, at the cost of not being complete: usually, $P \subsetneq P_{\mathcal{O}}(T)$. For each atom v^{val} of a target variable, the sets of positive ($Pos_{\text{v}^{val}}$) and negative ($Neg_{\text{v}^{val}}$) examples are extracted from T. A positive example is a state from which *at least* one transition has v^{val} in the next state, while a negative example is a state from which *no* transition has v^{val} in next state. **PRIDE** starts by arbitrary picking a positive example $pos \in Pos_{\text{v}^{val}}$, then constructs a rule of $P_{\mathcal{O}}(T)$ that matches it. For this, the (trivially) non-dominated rule $R = \text{v}^{val} \leftarrow \emptyset$ is iteratively revised for each negative example $neg \in Neg_{\text{v}^{val}}$ that R matches. Revision consists in the addition of an atom that appears in pos but not in neg ensuring both matching of pos and consistency with neg. In the worst case, R becomes the most specific rule that matches pos (see Theorem 1), ensuring termination. Then, the rule is minimized by iteratively removing non-necessary conditions. If a condition cannot be removed on the

current rule, then it cannot be removed on its simplifications, since conflict is transitive (see Theorem 2), thus each condition removal only needs to be checked once. At the end of Algorithm 2, R is still consistent with T, matches *pos* and all its conditions are necessary, thus $R \in P_{\mathcal{O}}(T)$. All positives examples matched by this rule can now be discarded (at least *pos* is removed, thus **PRIDE** is guaranteed to terminate) and the same process is repeated until all positives examples are matched. At the end, a set of optimal rules that forms a program $P \subseteq P_{\mathcal{O}}(T)$ is returned and P is guaranteed to realize T. All **PRIDE** operations have a polynomial complexity regarding T and $\mathcal{A}$ thus **PRIDE** is polynomial.

Algorithm 1 **PRIDE**($\mathcal{A}$, T, $\mathcal{F}$, $\mathcal{T}$)

INPUT : A set of atoms $\mathcal{A}$, a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, two sets of variables $\mathcal{F}$ and $\mathcal{T}$.
OUTPUT: A $\mathcal{DMVLP}$ $P \subseteq P_{\mathcal{O}}(T)$ s.t. P realizes T

for each $\mathrm{v}^{val} \in \mathcal{A}$ such that $\mathrm{v} \in \mathcal{T}$ **do**
  // 1) Extract positives and negatives examples
  $Pos_{\mathrm{v}val} := \{s \in \mathcal{S}^{\mathcal{F}} \mid \exists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
  $Neg_{\mathrm{v}val} := \{s \in \mathcal{S}^{\mathcal{F}} \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
  // 2) Generate the rules of v^{val} that are in $P_{\mathcal{O}}(T)$ and that match each state of $Pos_{\mathrm{v}val}$
  while $Pos_{\mathrm{v}val} \neq \emptyset$ **do**
    pick $pos \in Pos_{\mathrm{v}val}$
    $R := search(\mathrm{v}^{val}, pos, Neg_{\mathrm{v}val})$
    // Clean other positives examples covered
    $Pos_{\mathrm{v}val} := Pos_{\mathrm{v}val} \setminus \{s \mid s \in Pos_{\mathrm{v}val}, R \sqcap s\}$
    $P = P \cup \{R\}$
return P

Algorithm 2 **search**(v^{val}, *pos*, $Neg_{\mathrm{v}val}$)

INPUT : An atom $\mathrm{v}^{val} \in \mathcal{A}$, a state $pos \in \mathcal{S}^{\mathcal{F}}$ and a set of states $Neg_{\mathrm{v}val} \subset \mathcal{S}^{\mathcal{F}}$.
OUTPUT: A $\mathcal{MVL}$ rule $R \in P_{\mathcal{O}}(T)$ s.t. $R \sqcap pos$

$R := \mathrm{v}^{val} \leftarrow \emptyset$
// Specialize R until consistency with $Neg_{\mathrm{v}val}$
for each $neg \in Neg_{\mathrm{v}val}$ **do**
  if $R \sqcap neg$ **then**
    pick $c \in (pos \setminus neg)$
    $R := \mathrm{head}(R) \leftarrow \mathrm{body}(R) \cup \{c\}$
// Generalize R while keeping consistency
for each $c \in \mathrm{body}(R)$ **do** // Test each condition
  $R' := \mathrm{head}(R) \leftarrow \mathrm{body}(R) \setminus \{c\}$
  $conflict := false$
  for each $neg \in Neg_{\mathrm{v}val}$ **do**
    if $R' \sqcap neg$ **then** // Necessary condition
      $conflict := true$; **BREAK**
  if $conflict == false$ **then** // R' is valid
    $R := R'$
return R

4 Conclusion

The polynomiality of **PRIDE** is obtained at the cost of not ensuring that all possible optimal rules are learned. Still, the program learned by **PRIDE** is a sufficient model of the observed dynamic of T in practice: it can reproduce all observations and provides minimal explanation for each of them in the form of optimal rules. **PRIDE** performances (see Table 1 in Appendix) allows us to learn more complex systems and drastically reduce computation time of smaller ones. The source code of **GULA** and **PRIDE** is available as open source[5]. A user-friendly API allows to easily use both algorithms on different kinds of datasets and is already being used in several research collaborations.

References

1. Inoue, K., Ribeiro, T., Sakama, C.: Learning from interpretation transition. Machine Learning **94**(1), 51–79 (2014)
2. Ribeiro, T., Folschette, M., Magnin, M., Roux, O., Inoue, K.: Learning dynamics with synchronous, asynchronous and general semantics. In: International Conference on Inductive Logic Programming. pp. 118–140. Springer (2018)

[5] LFIT source code and API is available at `https://github.com/Tony-sama/pylfit`

A Appendix

Table 1: Average run time of **GULA** and **PRIDE** when learning Boolean networks of PyBoolNet[6] from at most 10,000 transitions from the 2^n possible synchronous transitions, with n the number of variables in the system, i.e., $n = |\mathcal{F}| = |\mathcal{T}|$. Average over 3 runs with a time-out (T.O.) of 1,000 seconds, conducted on one core of an Intel Core i7 (6700, 3.4 GHz) with 32 Gb of RAM.

System variables (n)	7	9	10	12	13	15	18	23
GULA run time	0.027s	0.157s	0.49s	2.62s	5.63s	T.O.	T.O.	T.O.
PRIDE run time	0.005s	0.02s	0.06s	0.37s	0.484s	1.55s	6.39s	32.43s

A.1 Proofs Of Section 3

Theorem 1 (Consistent Rule Always exists) *Let $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, $(s, s') \in T$ and $\mathrm{v}^{val} \in s'$. The rule $R = \mathrm{v}^{val} \leftarrow s$ is consistent with T and realizes (s, s').*
Proof. According to Definition 3, since $\mathrm{body}(R) = s, \mathrm{body}(R) \subseteq s$ and thus $R \sqcap s$. Since $\mathrm{head}(R) \in s'$, according to Definition 4, R realizes (s, s'). And according to Definition 5, R is consistent with T. □

Theorem 2 (Irreducible Rules are Optimal) *Let R be a rule consistent with a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$. If $\forall R' \in \{\mathrm{head}(R) \leftarrow \mathrm{body}(R) \setminus \{\mathrm{v}^{val}\} \mid \mathrm{v}^{val} \in \mathrm{body}(R)\}$, R' conflicts with T, then $\nexists R'' \neq R$ consistent with T such that $R'' \geq R$ and thus, $R \in P_{\mathcal{O}}(T)$.*
Proof. 1) According to Definition 1, if R is a $\mathcal{M}$VL rule, then $\{R''\ \mathcal{M}\text{VL rule} \mid R'' \neq R \wedge R'' \geq R\} = \{R''\ \mathcal{M}\text{VL rule} \mid R'' \geq R' \wedge R' \in \{\mathrm{head}(R) \leftarrow \mathrm{body}(R) \setminus \{\mathrm{v}^{val}\} \mid \mathrm{v}^{val} \in \mathrm{body}(R)\}\}$.

2) Conflict is transitive: if R' conflicts with T, then all rules R'' such that $R'' \geq R'$ conflict with T. If R' conflicts with a set of transitions $T \subseteq \mathcal{S}^{\mathcal{F}} \times \mathcal{S}^{\mathcal{T}}$, according to Definition 5, $\exists (s, s') \in T$, $R' \sqcap s$ and $\nexists (s, s'') \in T, \mathrm{head}(R') \in s''$. According to Definition 3, $R' \sqcap s \implies \mathrm{body}(R') \subseteq s$ and for a rule R'', $\mathrm{body}(R'') \subseteq \mathrm{body}(R') \implies \mathrm{body}(R'') \subseteq s \implies R'' \sqcap s$ and thus $\mathrm{head}(R'') = \mathrm{head}(R')$ implies that R'' conflicts with T.

Using 1) and 2) we can deduce that no consistent rule dominates R (beside itself) from the conflicts of all its direct generalizations R'. Since R is consistent with T and R is not dominated by another rule consistent with T, by Definition 6, $R \in P_{\mathcal{O}}(T)$. □

[6] Klarner, H., Streck, A., Siebert, H.: PyBoolNet: a python package for the generation, analysis and visualization of boolean networks. Bioinformatics33(5), 770–772 (2016).