Revision of Delayed Biological Systems with Answer Set Programming

Emna Ben Abdallah[1], Tony Ribeiro[2], Morgan Magnin[1,3], and Katsumi Inoue[3]

[1] LUNAM Université, École Centrale de Nantes, IRCCyN UMR CNRS 6597 (Institut de Recherche en Communications et Cybernétique de Nantes), 1 rue de la Noë, 44321 Nantes, France.
[2] The Graduate University for Advanced Studies (Sokendai), 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
[3] National Institute of Informatics, 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan.

Abstract. The modeling of biologicals systems relies on background knowledge, deriving either from literature and/or the analysis of biological observations. But with the development of high-throughput data, there is a growing need for methods that are able to automatically revise the existing models with regard to additional observations obtained by biologists. Our research aims at providing a logical approach to revise biological regulatory networks thanks to time series data, *i.e.*, gene expression data depending on time. In this paper, we propose two revision methodologies for models expressed through a timed extension of the Process Hitting framework (which is a restriction, well suited for biological systems, of networks of synchronizing timed automata). The revision methods we introduce aim to make the minimum amount of modifications (addition/deletion of actions between biological components) to a given input model so that the resulting model is most consistent with the observed data. They are implemented in Answer Set Programming, which is a paradigm proven efficient to tackle knowledge representation problems. We illustrate the merits of our methods in a twofold way: (1) we exhibit the benefits of such automatic approaches on a case study consisting of a qualitative model of circadian clock; (2) we lead computational experiments on various benchmarks to show the performances of our algorithms.

Keywords: Answer Set Programming, Process Hitting, dynamic modeling, time-varying genetic networks, network revision

1 Introduction

With both the spread of numerical tools in every part of daily life and the development of NGS methods (New Generation Sequencing methods), like DNA microarrays in biology, a large amount of time series data is now produced every day, every minute, every second [**?**]. This means that the number of experiments - and corresponding data - led on a biological system grows drastically. The

newly produced data - as long as the associated noise does not raise an issue with regard to the precision and relevance of the corresponding information - can give us some new insights on the behavior of a system. This justifies the urge to design efficient revision methods that are able to update previous knowledge on a model with regard to additional information. In other words, there is a strong need for automatic methods that update a given model so that the dynamics of the model is consistent with given observations and a set of criteria (for example, minimize the number of modifications).

Network completion has been the subject of numerous recent works. In [**?**], the authors targeted the completion of stationary Boolean networks. This method has been further refined along the years. Latest works [**?**] focus on completion in Time Varying Genetic Networks. These are networks whose topology does not change through time, but the nature of the interactions (activation, inhibition, or no interaction) between components may change at some (finite number of) time points. The completion approach (which, in these authors' papers, refers to both addition and deletion of interactions, making it a synonym of revision) has been successfully applied to biological case-studies, for example the DREAM4 Challenge [**?**], and the implementation has been improved through heuristics [**?**]. The method however is limited to acyclic networks.

Logical-based approaches may be also fruitful to network revision. It has been successfully applied to causal networks [**?**] and molecular networks represented with the SBGN-AF language [**?**].

Despite being a proper research area, revision is strongly connected to model inference. Starting from an empty model, revision may indeed be used to model from gene expression data. On the reverse, existing inference algorithms may be an inspiration for new revision methodologies. In particular, Answer Set Programming (ASP), a form of declarative programming that has been successively used in many knowledge representation and reasoning tasks [**?**,**?**,**?**] has been proven useful for network reconstruction [**?**] and inference of metabolic networks [**?**].

To our knowledge, no works have been led so far in the field of revision of timed models, without any restriction on the structure of the network.

In this paper, we aim to provide a logical approach to tackle the revision of qualitative models of biological dynamic systems, like gene regulatory networks. In our context, we assume the set of interacting components as fixed and we consider potential additions/deletions of interactions between components. The main originality of our work is that we address this problem in a timed setting, with quantitative delays potentially occurring between the moment an interaction activated and the moment its effect is visible. It allows for example to catch delays between the activation of a gene, and the moment the concentration of a gene reaches a qualitative threshold.

During the past decade, there has been a growing interest for the hybrid modeling of gene regulatory networks with delays. These hybrid approaches consider various modeling framerworks. In [**?**], the authors hybrid Petri nets: the advantage of hybrid with regard to discrete modeling lies in the possibility of capturing

biological factors, e.g., the delay for the transcription of RNA polymerase. The merits of other hybrid formalisms in biology have been studied, for instance timed automata [**?**] and hybrid automata [**?**]. Finally, in [**?**], the authors investigate a direct extension of the discrete René Thomas' modeling approach by introducing quantitative delays. These delays represent the compulsory time for a gene to turn from a discrete qualitative level to the next (or previous) one. They exhibit the advantage of such a framework for the analysis of mucus production in the bacterium Pseudomonas aeruginosa. The approach we propose in this paper inherits from this idea that some models need to capture these timing features.

In order to address the formal checking of dynamical properties within very large BRNs, we previously introduced in [**?**] a new formalism, named the *"Process Hitting"* (PH), to model concurrent systems having components with a few qualitative levels. Being a particular restriction of asynchronous automata networks or safe Petri nets, Process Hitting can be applied to complex dynamical systems with a very large number of interacting components, where each of these components can be described with a few internal states. In this paper, following recent works enriching (by adding priorities) the expressivity of PH while preserving its efficiency [**?**], we extend PH with quantitative timing features and exhibit efficient ASP-based approaches to perform network revision.

As readers may not be familiar with PH, we briefly introduce it in section 2, then give in section 3 some preliminary insights about recent translation of PH into ASP presented in [**?**]. All theoretical and practical notions are then settled to introduce our timed extension of PH, and related completion algorithm in section 4. Then we illustrate the merits of our approach in section 5 by first applying it on a simplified model of mammalian circadian clock [**?**], then discussing the practical results on a range of benchmarks from bioinformatics literature. Finally, in section 6, we summarize our contribution and give some perspectives for future works.

2 Process Hitting

Definition 1 introduces the Process Hitting (PH) framework [**?**] which allows to model a finite number of local levels, called *processes*, grouped into a finite set of components, called *sorts*. A process is noted a_i, where a is the sort's name, and i is the process identifier within sort a. At any time, exactly one process of each sort is *active*, and the set of active processes is called a *state*.

The concurrent interactions between processes are defined by a set of *actions*. Each action is responsible for the replacement of one process by another of the same sort conditioned by the presence of at most one other process in the current state. An action is denoted by $a_i \to b_j \,\restriction^{\nearrow}\, b_k$, which is read as a_i *hits* b_j to make it *bounce* to b_k, where a_i, b_j, b_k are processes of sorts a and b, called respectively *hitter*, *target* and *bounce* of the action. We also call a *self-hit* any action whose hitter and target sorts are the same, that is, of the form: $a_i \to a_i \,\restriction^{\nearrow}\, a_k$.

The PH is therefore a restriction of asynchronous automata, where each transition changes the local state of exactly one automaton, and is triggered by the local states of at most two distinct automata. This restriction in the form of the actions was chosen to permit the development of efficient static analysis methods based on abstract interpretation [**?**].

Definition 1 (Process Hitting with Plural Actions). *A* Process Hitting *is a triple* $(\Sigma, \mathcal{L}, \mathcal{H}_p)$ *where:*

- $\Sigma = \{a, b, \dots\}$ *is the finite set of* sorts*;*
- $\mathcal{L} = \prod_{a \in \Sigma} \mathcal{L}_a$ *is the set of* states *where* $\mathcal{L}_a = \{a_0, \dots, a_{l_a}\}$ *is the finite set of* processes *of sort* $a \in \Sigma$ *and* l_a *is a positive integer, with* $a \neq b \Rightarrow \mathcal{L}_a \cap \mathcal{L}_b = \emptyset$*;*
- $\mathcal{H}_p = \{\ A \rightarrow b_j \Rsh b_k$ *with* $A \in \mathcal{L}^\diamond \wedge b \in \Sigma \wedge b_j \neq b_k \wedge$ *if* $b_j \in A \Rightarrow A = b_j\}$ *is the finite set of* plural actions*. With* $\mathcal{L}^\diamond$ *the set of all the sub-states of* $\mathcal{L}$*.*

Example 1. The figure 1 represents a $\mathcal{PH}$ $(\Sigma, \mathcal{L}, \mathcal{H})$ with three sorts ($\Sigma = \{a, b, c\}$) and: $\mathcal{L}_a = \{a_0, a_1\}$, $\mathcal{L}_b = \{b_0, b_1\}$, $\mathcal{L}_z = \{z_0, z_1, z_2\}$.

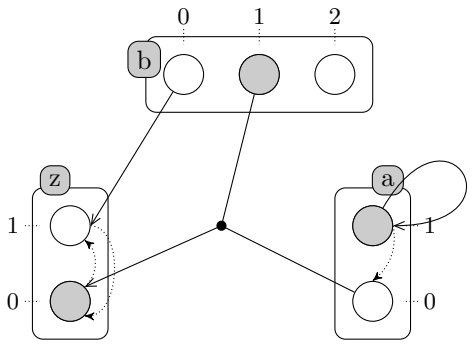


Fig. 1. A PH model example with three sorts: a, b and z (a is either at level 0 or 1, b at either level 0, 1 or 2 and z at either level 0, or 1). Boxes represent the *sorts* (network components), circles represent the *processes* (component levels), and the 3 *actions* that model the dynamic behavior are depicted by pairs of arrows in solid and dotted lines. A self-action: $a_1 \rightarrow a_1 \Rsh a_0$, a mono-action: $b_0 \rightarrow z_1 \Rsh z_0$ and plural action $a_0 \wedge b_1 \rightarrow z_0 \Rsh z_1$. The grayed processes stand for the possible initial state: $\langle a_1, b_1, z_0 \rangle$.

A state of a given PH consists in a set of active processes containing a single process of each sort. The active process of a given sort $a \in \Sigma$ in a state $s \in \mathcal{L}$ is noted $s[a]$. For any given process a_i we also note: $a_i \in s$ if and only if $s[a] = a_i$. The dynamic of the PH networks is performed thanks to the actions. Indeed, the transition from one state s_1 to its successor s_2 is done when there is a playable action (definition 5) at s_1. After each transition only one sort, or one component, changes its level from one process to another.

It should be noted that during these last years the Process Hitting framework was gradually enriched with new type of sorts like cooperative sorts and new actions like plural actions [**?**], actions with priority [**?**] and actions with delay.

In some cases it is necessary to represent a reaction of a set of components on one component. For example in the bio-chemical reactions :$X \xrightarrow{Y} Z$ or $X + Y \rightarrow Y + Z$, where X is a set of reactives, Y a set of catalysts and Z a set of products. Plural actions permit to represent this kind of reactions in PH. The plural is made up of two sets of processes of different sorts, which represent all the hitters and the bounces.

Definition 2 (Plural action). *Let $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H})$ be a Process Hitting. A* plural action *is an action noted by: $h = A \rightarrow b_j \Rsh b_k$ with $A \in \mathcal{L}^{\diamond} \wedge b_j \neq b_k \wedge$ if $b_i \in A \Rightarrow A = b_j$. With $\mathcal{L}^{\diamond}$ the set of all the sub-states of $\mathcal{L}$.*

Example 2. We give a simple example to represent a plural action by a cooperation between two biological components (x and y) in order to activate another component (z) and change its level from 0 to 1: $\{x_1, y_1, z_0\} \rightarrow \{x_1, y_1, z_1\}$. In PH, this can be translated by the following action: $\{x_1 \wedge y_1\} \rightarrow z_0 \Rsh z_1$.

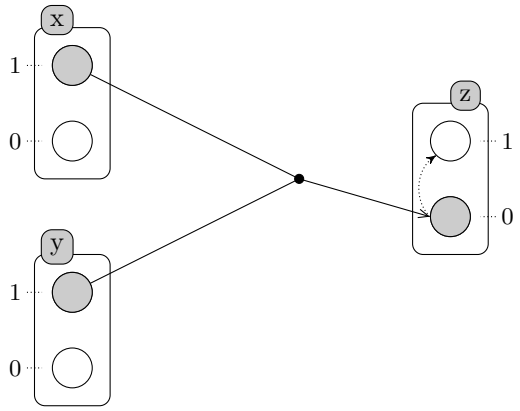


Fig. 2. Representation of a plural action in Process Hitting network: $\{x_1 \wedge y_1\} \rightarrow z_0 \Rsh z_1$.

In some dynamics it is crucial to have information about the delays between two events (two states of a PH). Classic actions cannot exhibit this information: we just know chronology, i.e., that the state s_2 will be after s_1 in the next step but it is not possible to know chronometry, i.e., how much time this transition takes to occur. We propose to add the delay in the action attributes which is responsable of the transition between the two states. That means that this action needs to be played during a specific time so that the system does not change its state (definition 3).

Definition 3 (Timed plural action). *Let $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H})$ be a process hitting and $\mathcal{L}^\diamond$ be the set of all the sub-states of $\mathcal{L}$. A timed plural action of $\mathcal{PH}$ is a plural action with a delay D: $A \xrightarrow{D} b_i \pitchfork b_j$ where $D \in \mathbb{R}^+$, $A \in \mathcal{L}^\diamond$, and b_i, b_j where $b_i \neq b_j$ are two processes of the target sort b. If $b_i \in A$, $A = b_i$.*

Definition 4 (Process Hitting with Timed Plural Actions). *A* Process Hitting with timed plural actions *is a triple $(\Sigma, \mathcal{L}, \mathcal{H}_{tp})$ where:*

- *$\Sigma = \{a, b, \dots\}$ is the finite set of* sorts*;*
- *$\mathcal{L} = \prod_{a \in \Sigma} \mathcal{L}_a$ is the set of* states *where $\mathcal{L}_a = \{a_0, \dots, a_{l_a}\}$ is the finite set of* processes *of sort $a \in \Sigma$ and l_a is a positive integer, with $a \neq b \Rightarrow \mathcal{L}_a \cap \mathcal{L}_b = \emptyset$;*
- *$\mathcal{H}_{tp} = \{A \xrightarrow{D} b_j \pitchfork b_k \mid A \in \mathcal{L}^\diamond, b_j \neq b_k, b_i \in A \Rightarrow A = b_j\}$ is the finite set of* timed plural actions*.*

Duration of actions can now be represented in a Process Hitting model thanks to timed actions. Note that if all actions delays are set to 1 it is equivalent to Process Hitting without delays. The way these new actions should be used is described as follows.

Definition 5 (Playable timed plural action). *Let $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H}_{tp})$ be a PH with timed plural actions and $s \in \mathcal{L}$ a state of PH. We say that the action $h = A \xrightarrow{D} b_i \pitchfork b_j$ is* playable in state *s if and only if $A \subseteq s$ and $b_i \in s$.*

Definition 6 (Autonomous temporized sort). *A sort a is said to be an* a*utonomous temporized sort if and only if $\forall h = A \xrightarrow{D} a_i \pitchfork a_j \in \mathcal{H}$ where a_i and a_j are processes of a, we have only $A = \{a_i\}$.*

The delays caused by those timed actions and temporized sorts have an impact on the system dynamics. Indeed, in our asynchronous model, only one action can be processing and the state of the system must not change during this time. But, clocks represented by autonomous temporized sorts can evolve in parallel of processing actions. Those new sorts are an exception to the asynchronicity behavior of our model: multiple temporized sorts can change there processes at the same time. Thus, making possible that a change of the state of the system may occurs during the processing of an action. If this case occurs, the processing of the action is stopped and a new action can be played. The dynamic of a Process Hitting with timed plural actions is formally defined as follows.

We define the semantics of *dense-time* Process Hitting as a timed transition system. In this model, two kinds of transitions may occur: *dense-time* transitions when time passes and *discrete* transitions when a transition of the net is fired.

Definition 7 (Semantics of a dense-time Process Hitting). *Let $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H}_{tp})$ be a dense-time Process Hitting and $s \in \mathcal{L}$ be the state of $\mathcal{PH}$ at time t . If there is no action in $\mathcal{H}$ that is playable in s the state of the system remain s for all $t' > t$ (i.e. steady state). Let $h = A \xrightarrow{D} b_i \pitchfork b_j$ be a playable action in s. If there is no temporized sort a that changes its process from $a_i \in A$ to $a_j \notin A$ while playing h in s then the state of $\mathcal{PH}$ at $t + D$ is called a* successor *of s and is denoted by $(s \cdot h)$, where $(s \cdot h)[b] = b_k$ and $\forall c \in \Sigma, c \neq b \Rightarrow (s \cdot h)[c] = s[c]$.*

Even if there already exists a few hybrid formalisms, we chose to propose this extension of the PH framework for several reasons. First, PH is a general framework that, although it was mainly used for biological networks, allows to represent any kind of dynamical models, and converters to several other representations are available.Although an efficient dynamical analysis already exists for this framework, based on an approximation of the dynamics, it is interesting to identify its limits (especially the fact that previous studies were focusing only on discrete, not timed, dynamics) and compare them to the approach we present later in this paper. Finally, the particular form of the actions in a PH model allows to easily represent them in ASP, with one fact per action, as described in the next section. Other representations may have required supplementary complexity; for instance, a labeling would be required if actions could be triggered by a variable number of processes. We now show how to represent the previous definitions through ASP.

3 PH through ASP

3.1 Answer Set Programming

In this section, we recapitulate the basic elements of ASP. An answer set program is a finite set of rules of the form

$$a_0 \text{ :- } a_1,\ \ldots,\ a_m,\ not\ a_{m+1},\ \ldots,\ not\ a_n \tag{1}$$

where $n \geq m \geq 0$, a_0 is a propositional atom or $\bot$, all $a_1, \ldots, a_n$ are propositional atoms and the symbol "*not*" denotes default negation. If $a_0 = \bot$, then Rule (1) is a constraint (in which case a_0 is usually omitted). The intuitive reading of a rule of form (1) is that whenever $a_1, \ldots, a_m$ are known to be true and there is no evidence for any of the default negated atoms $a_{m+1}, \ldots, a_n$ to be true, then a_0 has to be true as well. Note that $\bot$ can never become true.

In the ASP paradigm, the search of solutions consists in computing answer sets of answer set program. An answer set for a program is defined by Gelfond and Lifschitz [**?**] as follows. An interpretation I is a finite set of propositional atoms. An atom a is true under I if $a \in I$, and false otherwise. A rule r of form 1 is true under I if $\{a1,\ \ldots,\ a_m\} \subseteq I$ and $\{a_{m+1},\ \ldots,\ a_n\} \cap I = \emptyset$ implies $a_0 \in\ I$. An Interpretation I is a model of a program P if each rule $r \in P$ is true under I. Finally, I is an answer set of P if I is a subset-minimal model of P^I, where P^I is defined as the program that results from P by deleting all rules that contain a default negated atom from I, and deleting all default negated atoms from the remaining rules. Programs can yield no answer set, one answer set, or many answer sets. To compute answer sets of an answer set program, we run an ASP solver.

3.2 Translation of PH networks to ASP

PH network is easy to be formulated in ASP. Indeed we need only three predicates to define the whole network: `"sort"` to define sorts, `"process"` for the

processes and `"action"` for the network actions. In [**?**], we already showed the interest of ASP regarding Process Hitting (without plural actions, nor delays). We will see in example 3 how a PH network is defined with these predicates.

Example 3 (Representation of a PH network in ASP). The representation of the PH given in figure 1 into ASP is the following:

```
1 process("a", 0..1). process("b", 0..1). process("z", 0..2).
2 sort(X) ←process(X,I)
3 action("a",0,"b",1,0). action("a",1,"a",1,0). action("b",1,"z",0,2).
4 action("b",0,"z",1,2). action("z",0,"a",0,1).
```

In line 1, we create the list of processes corresponding to each sort, for example the sort `"z"` has 3 processes numbered from `0` to `2`; this specific predicate will in fact expand into the three following predicates: `process("z", 0)`, `process("z", 1)`, `process("z", 2)`. Line 2 enumerates every sort of the network from the previous information. Finally, all actions of the model are defined in lines 3 and 4; for example, the first predicate `action("a",0,"b",1,0)` represents the action $a_0 \rightarrow b_1 \not\uparrow b_0$.

Example 4 (Representation of PH with plural-timed actions in ASP). We can take the example given in figure 2 and consider that the action $\{x_1, y_1, z_0\} \xrightarrow{D} \{x_1, y_1, z_1\}$ has a delay "D" and in PH the action becomes: $x_1 \wedge y_1 \rightarrow z_0 \not\uparrow z_1$. As a consequence, the PH translated in ASP is the following:

```
5 process("x", 0..1). process("y", 0..1). process("z", 0..1).
6 action("x",1,"y",1,"z",0,1, D).
```

The number of indegree in this action $i = 2$, there is only 2 hitters. It is possible to have an indegree greater than 2. We will show later that the number of the maximum indegree should be an input for our algorithm.

The predicate `action` represents the ordinary actions as well as the plural actions. Indeed an ordinary action is a plural action with an indegree 1. Moreover all the ordinary actions are timed actions with a delay equal to 1 unit of time. Every action needs at least one step to be played. For example, the action `action("a",0,"b",1,0)` is equivalent to `action("a",0,"b",1,0,1)` As a result, the principle of having plural-timed actions in the modeling sounds like a generalized way to represent the actions in a Process Hitting network. That is why, in the following, we will consider only plural-timed actions.

4 Revision of PH networks

4.1 Algorithm

In this section we propose a simple algorithm to revise Process Hitting networks. We assume that new observation data are provided as a *chronogram* of size T: the value of each variable is given for each time step t, $0 \leq t \leq T$, through a time interval discretization. This algorithm takes as input a Process Hitting and a

chronogram of genes evolutions of this network. Knowing the genes' influences (or assuming all possible influences), this algorithm will complete the input network by adding the delayed actions that could have realized the changes observed in the chronogram. Algorithm 1 shows the pseudo code of our algorithm. If the observations are perfect, it will generate all possible actions that can realize each change observed. Because of the delayed semantics, it is not possible to decide whether an action is correct or not. But we can output the minimal sets of actions necessary to realize the changes observed. If the observations are not perfect, we can merge actions by replacing the one that only differs by the delay by one action whose delay is the average one. The intuition is that, in practice, if there are enough observations, the delay of those actions should tend to the real value.

Algorithm 1 PH-Completion($PH, Chronogram, Influences, indegree$)

- INPUT: a Process Hitting PH, a chronogram C of the genes evolution of PH, the genes influences and a maximal action in-degree i.
- Let $A := \emptyset$
- Step 1: For each time where a gene G changes its value from P to P' in C:
  - Let D be the delay since the last time a gene has changed.
  - Generate all actions with delay D which involve all subsets of size i of the genes $S_1, \ldots, S_n$ having an influence on G:

$$a := action(S_1, P_1, \ldots, S_n, P_n, G, P, P', D)$$

  - Add all action a in A
- Step 2 (optional): Merge each action with the same hitters, $S_1, P_1, \ldots, S_n, P_n$ and the same target, G, P, P', into one action where the delay is the average.
- Step 3: Generate S the set of all subsets of actions of A that realize C such that:
  - A is minimal:

$$\forall A \in S, \nexists A' \in S, A' \subset A$$

  - Atmost one delay per action: $\forall h \in A$, where $h = A \xrightarrow{D} b_j \barwedge b_k$, $\nexists h' \in A$, such that $h' = A \xrightarrow{D'} b_j \barwedge b_k, D \neq D'$.
- OUTPUT: S a set of completed Process Hittings that realize the chronogram.

Theorem 1 (Completeness). *Let PH be a Process Hitting, C be a chronogram of the genes of PH and A be the set of actions of PH that realized the chronogram C. Let PH' be a Process Hitting and A' be the set of actions of PH' such that $A' \subseteq A$. Given PH' and C as input, Algorithm 1 (without step 2) is complete: it will output a set of process hitting S, such that $\exists PH'' \in S$ with A'' the set of actions of PH'' such that $A'' = A' \cup A$.*

Proof. Let us suppose that the algorithm is not complete, then there is an action $a \in A$ *that realized* C *and* $a \notin A''$*. After step 1.2,* PH'' *contains all actions that can realized each gene change. Here there is no action* $a \in A$ *that realized* C *which is not generated by the algorithm, so* $a \in A''$*. Then it implies that at step 3, the action* a *is removed from* A''*. But since* a *realized one the change of* C *and* a *is generated at step 1, then it will be present in one of the minimal subset of actions. And* a *will be in one of the network outputted by the algorithm.* □

Theorem 2 (Complexity). *Let* PH *be a Process Hitting,* S *be the number of sorts of* PH *and* P *be the maximal number of processes of a sort of* PH*. Let* C *be a chronogram of the genes of* PH *over* T *units of time, such that* c *is the number of gene change of* C*. The memory use of our algorithm belongs to* $O(T \times P \times i^S \times 2^{T \times P \times i^S})$ *that is bound by* $O(T \times P^{S+1} \times 2^{T \times P^{S+1}})$*. The complexity of completing* PH *by generating actions from the observations of* C *with Algorithm 1 belongs to* $O(c \times i^S + (T \times P \times i^S)^2 + 2^{2 \times T \times P \times i^S} + c \times 2^{T \times P \times i^S})$ *that is bound by* $O(2^{3 \times T \times P^{S+1}})$*.*

Proof. Let i *be the maximal indegree of an action in* PH*,* $0 \leq i \leq P$*. Let* p *be a process of* PH *and* n *be the number of sorts that can influence* p*. There is* i^S *possible combinations of those process that can hit* p*, each of those can form an action. There is* P *process and at most* T *possibles delay, so that there are* $T \times P \times i^S$ *possibles actions, thus at step 1, the memory of our algorithm is bound by* $O(T \times P \times i^S)$*, which belongs to* $O(T \times P^{S+1})$ *since* $0 \leq i \leq P$*. Generating all minimal subsets of actions* A *of* PH' *that realize* C *can require to generate at most* $2^{T \times P \times i^S}$ *set of rules. Thus, the memory of our algorithm belongs to* $O(T \times P \times i^S \times 2^{T \times P \times i^S})$ *and is bound by* $O(T \times P^{S+1} \times 2^{T \times P^{S+1}})$*.*

Merging the actions at step 2 is polynomial in the number of actions and the complexity of this operation belongs to $O((T \times P \times i^S)^2)$*. The complexity of this algorithm belongs to* $O(c \times i^S + (T \times P \times i^S)^2)$*. Since* $0 \leq i \leq P$ *and* $0 \leq c \leq T$ *the complexity of Algorithm 1 is bound by* $O(T \times P^S + (T \times P \times P^S)^2) = (T \times P^S + T^2 \times P^{2S+2})$*.*

Generating all minimal subsets of actions A *of* PH' *that realize* C *can require to generate at most* $2^{T \times P \times i^S}$ *set of rules. Each set has to be compared with the others to keep only the minimal ones, which costs* $O(2^{2 \times T \times P \times i^S})$*. Furthermore, each set of actions has to realize each change of* C*, it requires to check* c *changes and it costs* $O(c \times 2^{T \times P \times i^S})$*. Finally, the total complexity of completing* PH *by generating actions from the observations of* C *belongs to* $O(c \times i^S + (T \times P \times i^S)^2 + 2^{2 \times T \times P \times i^S} + c \times 2^{T \times P \times i^S})$ *that is bound by* $O(T \times P^S + (T \times P^{S+1})^2 + 2^{2 \times T \times P^{S+1}} + T \times 2^{T \times P^{S+1}})$ □

5 Evaluation

5.1 Application: Circadian clock

Figure 3 shows the circadian clock model of [**?**]. The right part shows a chronogram of the gene evolution of the circadian clock. The abscisse axe represents

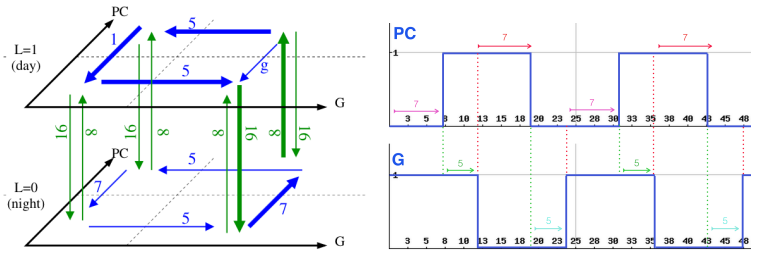


Fig. 3. The first figure is taken from [**?**], it presents the qualitative model of the mammalian circadian cycle during the summer. The second one is a chronogram corresponding to the discretization of the observed data set of a circadian clock components during the night (L=0).

time steps in hours and ordinate represents the gene value. In this example we observe the evolution of genes G and PC during the night (L is fixed to 0). Here we can see the 7 (resp. 5) hours of delay shift of PC (resp. G) from 1 to 0 and vice versa. Indeed, PC and G respectively change their state 7 hours and 5 hours after a change occurs in the system.

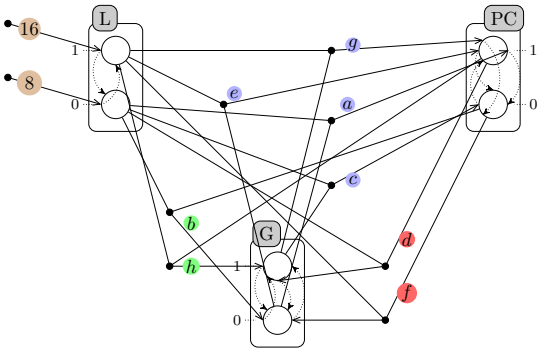
The left part of Figure 4 shows the circadian clock represented as a Process Hitting with timed plural actions. Here, the light L, the gene PC and G are represented by sorts with two processes. The light clock is represented by an autonomous temporized sort: there are only auto-actions. Actions are represented by edges as follows: when a node have incoming edge its the target of an action and the hitters of this action are the nodes where those edges are coming. Each action is labelled by its delay. Right part of the figure shows the representation of the circadian clock with the ASP formulation we presented in section 3.

5.2 Experiments

In this section, we evaluate our approach on learning the circadian clock actions. All experiments are run with a ASP implementation of Algorithm 1 on a processor Intel Xeon (X5650, 2.67GHz) with 12GB of RAM.

When the ASP representation of a chronogram like the one of Figure 3 is given as input to Algorithm 1 it will generate all the timed actions that can induce the observed changes. The delay of the action is simply determined from the time delay between two changes.

Figure 5 shows the evolution of run time of our ASP implementation of Algorithm 1 on the inference of the circadian clock actions regarding the quantity of input data. In this experiment, we analyse the scalability of our approach by varying the number of time units of the input, i.e. the size of the chronogram. Here we can see that the time needed to analyse the input data grows exponentially. The experiments stop at 384 because the memory required by the ASP solver reached the 12GB of RAM we have. It is currently quite difficult to obtain



Nodes:

```
process("L",0..1).
process("PC",0..1).
process("G",0..1).
temporised("L").
```

Actions:

```
action("L",0, "L",0,1, 8).
action("L",1, "L",1,0, 16).
action("L",0, "G",0, "PC",1,0, a).
action("L",0, "PC",0, "G",0,1, b).
action("L",0, "G",1, "PC",0,1, c).
action("L",0, "PC",1, "G",1,0, d).
action("L",1, "G",0, "PC",1,0, e).
action("L",1, "PC",0, "G",0,1, f).
action("L",1, "G",1, "PC",1,0, g).
action("L",1, "PC",1, "G",1,0, h).
```

Fig. 4. Representation of circadian clock in Process Hitting (left) and the equivalent ASP representation (right). The value "a","b",...,"h" represent the delays in actions.

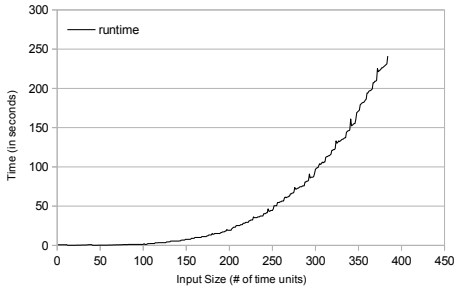


Fig. 5. Run time of the application of Algorithm 1 on circadian clock chronograms varying the number of time steps.

real data from biological system with chronogram exceeding 100 points. In work like [**?**] the considered chronograms are only composed of about 10 points. Regarding the circadian clock, a chronogram of 100 data points is about 4 days of data, that is not much information, especially in cases where we want to analyze the effect of perturbations (for example, understand the recoverability phase in case of jet-lag).

6 Conclusion and perspectives

In this paper, we proposed an approach to automatically infer timed models of Process Hitting from time series data (expressed as chronograms). To do so, we implemented our algorithm in ASP. We illustrated the applicability and limits of the method through various benchmarks. This opens the way to promising applications in the connection between biologists and computer scientists. Further works will now consist in discussing the kind of information one can get on timed Process Hitting by analyzing the associated untimed model. We also plan to improve our implementation to make it robust against noisy data.