Generation and revision of Delayed Biological Regulatory Systems

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

[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] National Institute of Informatics, 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan.

Abstract. The modeling of biological 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 automatically generate (resp. revise) admissible models with regard to initial (resp. additional) observations provided by biologists. Our research aims at providing a logical approach to generate (resp. to revise) biological regulatory networks thanks to time series data, i.e., gene expression data depending on time. In this paper, we propose a new methodology for models expressed through a timed extension of the Process Hitting framework (which is a restriction, well suited for biological systems, of networks of synchronized 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 the most consistent as possible with the observed data. The originality of our work relies on the integration of quantitative time delays directly in our learning approach. Finally, we show the benefits of such automatic approach on dynamical biological models. In order to exhibit the scalability of our contribution, we conduct benchmarks on the DREAM4 datasets, a popular reverse-engineering challenge, and discuss the computational performances of our algorithm.

Keywords: network construction/revision, dynamic modeling, delayed biological regulatory networks, time-varying genetic networks, Process Hitting

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 [15]. 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 [2], the authors targeted the completion of stationary Boolean networks. This method has been further refined along the years. Latest works [17] 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 [19], and the implementation has been improved through heuristics [18]. 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 [14] and molecular networks represented with the SBGN-AF language [26].

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 [20,3,4] has been proven useful for network reconstruction [8] and inference of metabolic networks [25].

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 [16], 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 [24] and hybrid automata [1]. Finally, in [7], 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 [21] 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 [11], 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 and all theoretical and practical notions are then settled to introduce our timed extension of PH. Then in Section 3 we present the related generation algorithm and we demonstrate the performance of our algorithm by a a running case study example. We propose in 4 new refinement methods applied to the generated model. Then we illustrate the merits of our approach in section 5 by first applying it on dynamical biological models, then discussing the practical results on a range of benchmarks from the DREAM4 datasets, a popular reverse-engineering challenge, and discuss the computational performances of our algorithm. 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 [21] 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 least one other process in the current state. A normal action is denoted by $a_i \rightarrow b_j \Rsh b_k$, 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: $a_i \rightarrow a_i \Rsh a_k$. The original Process Hitting framework contains only actions with one hitter but it should be noted that during these last years it was gradually enriched with new type of sorts like cooperative sorts (the same role of a multiplex) and new actions like plural actions [10] (at least 2 hitters), actions with priority [12] and now actions with delays tackled in this paper.

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 [22] and recently exhaustive dynamic analysis [5].

Definition 1 (Process Hitting). *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* 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: $\mathcal{L}_a = \{a_0, a_1\}$, $\mathcal{L}_b = \{b_0, b_1\}$, $\mathcal{L}_z = \{z_0, z_1, z_2\}$.

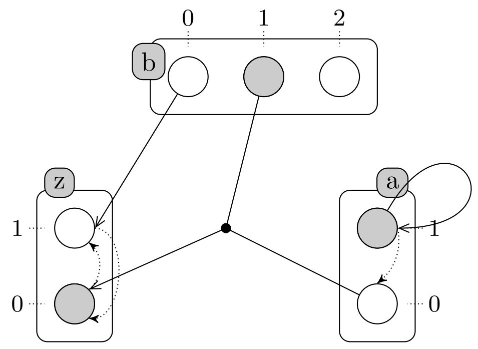

Fig. 1. A PH model example with three sorts: a, b and z (a is either at level 0 or 1, b at either level 0, 1 or 2 and z at either level 0, or 1). Boxes represent the *sorts* (network components), circles represent the *processes* (component levels), and the 3 *actions* that model the dynamics are depicted 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 a 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 $s[a] = a_i$. For each sort, it cannot have more than one active process at one state. The dynamics of the PH 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 3) at s_1. After each transition only one sort, or one component, changes its level from one process to another.

In some dynamics it is crucial to have information about the delays between two events (two states of a PH). Discret actions, described above, cannot exhibit this information: we just process chronological information, 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 *timed action* needs to be played during a specific time so that the system does not change its state until it finishes.

Definition 2 (Process Hitting with Timed Actions). *A* Process Hitting with timed 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* $a \neq b \Rightarrow \mathcal{L}_a \cap \mathcal{L}_b = \emptyset$*;*
– $\mathcal{H}_{tp} = \{A \xrightarrow{D} b_j \upharpoonright b_k \mid D \in \mathbb{R}^+, A \in \mathcal{L}^\diamond, b_j \neq b_k, b_i \in A \Rightarrow A = b_j\}$ *is the finite set of* timed actions.

To model biological networks, we used the PH framework with timed actions (Definition 2). Indeed, in the biological models that we studied we need to present not only the cooperation between the components to influence another one (by *actions*), but also the time needed by this action to make the system change. This time is called a *delay*. For instance the *delay* of this timed action $A \xrightarrow{D} b_j \upharpoonright b_k$ is equal to D. 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 0 it is equivalent to Process Hitting without delays (original PH). The way these new actions should be used is described as follows.

Definition 3 (Playable Timed Action). *Let* $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H}_{tp})$ *be a PH with timed actions and* $s \in \mathcal{L}$ *a state of* $\mathcal{PH}$*. We say that the timed action* $h = A \xrightarrow{D} b_i \upharpoonright b_j$*, with* $D \in \mathbb{R}^+$*, is* playable in a state s *if and only if* $A \subseteq s$ *and* $b_i \in s$ *(i.e.* $\forall a_i \in A, s[a] = a_i$ *and* $s[b] = b_i$*).*

The dynamics of the Process Hitting is based on asynchronous evolution (only one action can be played at a time) so we have:

$$(s \cdot h)[b] = b_j \text{ and } \forall c \in \Sigma, c \neq b \Rightarrow (s \cdot h)[c] = s[c]$$

The resulting state of the $\mathcal{PH}$ model ($\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H}_{tp})$) after playing a delayed action h at time step t and in a state s is called a *successor* of s and is denoted by $(s \cdot h)$. This succession of states can be denoted by: $\mathsf{state}(\mathcal{PH}, t) = s$ and $\mathsf{state}(\mathcal{PH}, t + D) = (s \cdot h)$ with $delay(h) = D$. The resulting state of a sort $a \in \Sigma$ by $s[a]$.

Definition 4 (Semantics of Process Hitting with Timed Actions). *Let $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H}_{tp})$ be a Process Hitting with timed actions. Let $\mathsf{state}(\mathcal{PH}, t) = s$ with $s \in \mathcal{L}$ and H^s_t be the set of playable timed actions in s at time step t. If no action is being played at t and $H^s_t \neq \emptyset$, then exactly one action $h \in H^s_t$ has to be played. Let $D \in \mathbb{R}^+$ and $h = A \xrightarrow{D} b_i \upharpoonright b_j$ and t_D be a time step such that $t_D - t = D$ so that the semantics of $\mathcal{PH}$ can be expressed as follows:*

$$\forall t' \in [t, t_D[\colon \mathsf{state}(\mathcal{PH}, t') = s \text{ and } \mathsf{state}(\mathcal{PH}, t_D) = (s \cdot h).$$

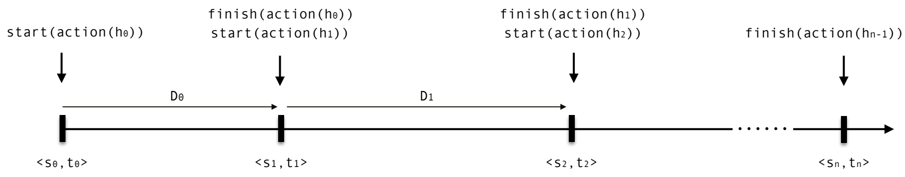

Fig. 2. Semantics of Process Hitting ($\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H}_{tp})$) with timed actions such that $\mathsf{state}(\mathcal{PH}, t_i) = s_i$, $\forall\ t_0 \leq t_i \leq t_n$, with t_0 and t_n are respectively the minimum and the maximum time step in the time series data. We mean by $start(action(h_i))$ that the timed action h_i starts playing so that the system state will not change until h_i finishes after D_i time steps with $delay(h_i) = D_i$ presented here by $finish(action(h_i))$.

We propose in Definition 4, the semantics of the timed PH in which we consider to generate the corresponding model to the given observations. Indeed it means that if we consider a PH model $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H}_{tp})$, and if there is no action in $\mathcal{H}$ that is playable in s, with $s = \mathsf{state}(\mathcal{PH}, t)$, the state of the system remains s for all time t' with $t' > t$ (i.e. steady state). If no action is playing (either the initial state or an action just finished) and $h = A \xrightarrow{D} b_i \upharpoonright b_j$ is the choosen playable action in s then the state of $\mathcal{PH}$ is s during D time steps and at the time step $t + D$ the *successor* of s denoted by $(s \cdot h)$ is obtained.

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. Indeed, a PH with timed actions is a subclass of time Petri nets [13]. 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. Other representations may have induced supplementary complexity; for instance, a labeling would be required if actions could be triggered by a variable number of processes. 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 a previous work [5]. Later we propose a new approch to resolve the generation/revision problem of PH models.

3 Generation of PH models

This algorithm takes as input a Process Hitting and observation of gene expression according to time for this model (time series data). Knowing influences of the genes (or assuming all possible influences), this algorithm will generate a model or complete the input model by adding the delayed actions that could have realized the state changes observed through the observation data.

3.1 Algorithm

In this section we propose an algorithm to construct and/or revise Process Hitting models. We assume that new time series data observations 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 (see definition below and example Figure 3).

Definition 5 (Chronogram). *A chronogram is the discretized time evolution of the time series data of each component of biological regulatory networks. It is presented by the fonction below:*

$$\mathbb{N}^+ \longmapsto \mathbb{N}^+$$

$$[0, T] \subset \mathbb{N}^+ \longmapsto \{0, 1, ..., n\}$$

with T is the maximum time point called the size of the chronogram and n is the maximum level of discretization.

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. Otherwise, if the observation data is not perfect, we refine the output model by filters taking into account almost the different possible perturbations. These filters are more detailes in Section 4.

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 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 $A \subseteq A''$.*

Algorithm 1 PH-Generation(PH,$TimeSeriesData$,$RegulationInfluences$,$Indegree$)

```
– INPUT: a Process Hitting $PH$, the time series data observations of the genes
  of $PH$ as a chronogram $C$, the set of the genes regulation influences $R$ and
  a maximal in-degree $i$ per action.
– Let $A := \emptyset$
– Step 1: For each time step where a gene $G$ changes its value from $P$ to $P'$
  in the chronogram $C$:
    - 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 $n$ of
      the genes $S_1, \dots, S_n$ having an influence on $G$ from the set $R$, with $n \leq i$:
          $a := action(S_1, P_1, \dots, S_n, P_n, G, P, P', D)$
    - Add all actions $a$ in $A$
– Step 2: Generate $S$ the set of all subsets of actions of $A$ that realizes $C$ such
  that $A$ is minimal:
          $\forall A \in S, \nexists A' \in S, A' \subset A$
– OUTPUT: $S$ a set of completed Process Hitting models that realize the
  observations.
```

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, PH'' contains all actions that can realize each gene change. Here there is no action $a \in A$ that realizes C which is not generated by the algorithm, so $a \in A''$. Then it implies that at step 2, the action a is removed from A''. But since a realizes one of the change of C and a is generated at step 1, then it will be presented in one of the minimal subset of actions. Such that a will be in one of the networks 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 bounded 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 $O(c \times i^S + (2^{2 \times T \times P \times i^S} + c \times 2^{T \times P \times i^S})$ that is bounded 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 processes that can hit p, each of those can form an action. There is P processes and at most T possible delays, so that there are $T \times P \times i^S$ possibles actions, thus at step 1, the memory of our algorithm is bounded 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 bounded by $O(T \times P^{S+1} \times 2^{T \times P^{S+1}})$.*

The complexity of this algorithm belongs to $O(c \times i^S)$. Since $0 \leq i \leq P$ and $0 \leq c \leq T$ the complexity of Algorithm 1 is bounded by $O(T \times P^S) = (T \times P^S)$.

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 $O(c \times i^S + (2^{2 \times T \times P \times i^S} + c \times 2^{T \times P \times i^S})$ that is bounded by $O(T \times P^S + (T \times P^{S+1})^2 + 2^{2 \times T \times P^{S+1}} + T \times 2^{T \times P^{S+1}})$ $\square$

3.2 Case study

In this section we demonstrate how this method generates a PH model consistent with the set of biological regulatory time series data given as an input. First, the method uses discretised observations as an input (i.e. chronogram), thus it is necessary to use an other method which transforms the analogic time series data to discretised time series data.

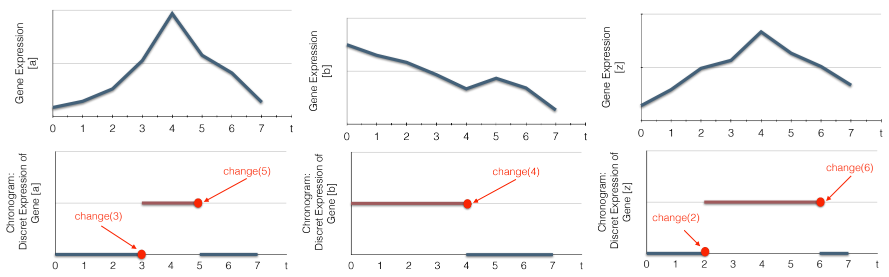

Fig. 3. Examples of the discretization of continous time series data into bi-valued chronograms. Abscisse represents time and ordinate the gene expression level. The expression level is discretized according to a threshold fixed to the half of the gene expression value in this example. A `change(t)` is a time step in which we observe a change in the gene expression level.

We can summarize our method in the following steps:

- Detection of changes
- Computation of the interactions possibly responsible of thoses changes
- Filtering of the candidates actions
- Add actions to the model: generation/revision of the model

We will now show an example of the execution of the Algorithm 1 on the chronograms of the observed data of Figure 3:

The first change occurs at $t_1 = t_{min} = 2$, which we will denote as `change(2)`. It is the gene "z" whose value changes from 0 to 1, thus the action that has realized this change is of the form $h = A \xrightarrow{D} z_0 \upharpoonright z_1$, where $A \in \mathcal{L}^{\diamond}, |A| \leq i$, (we suppose $i = 2$) and D is the delay which is equal to 2 here, since $D_{t_i} = t_i - t_{i-1}$, such that $\exists$ `change(`t_i`)` and `change(`t_{i-1}`)`, $D_{t_1} = t_1 - t_0 = 2 - 0 = 2$.

Let $R = \{b \rightarrow z, a \rightarrow z, a \rightarrow a\}$ be the set of regulation influences among the components of the system. In the first change $t_1 = 2$ that we will denotes as *change*(2), It has been caused by an action $h = A \xrightarrow{2} z_0 \upharpoonright z_1$. According to R the genes having influence "z" are $G_{R_z} = \{a, b\}$. It means that $A = \{a_?, b_?\}$ or $A = \{a_?\}$ or $A = \{b_?\}$. The expression level of the genes of GR_z between t_i and t_{i-1} is computed from the observations as follows:

- $a \in GR_z$: $[a]_t = 0 \ \forall t \in [0, 2]$
- $b \in GR_z$: $[b]_t = 1 \ \forall t \in [0, 2]$

Thus $A = \{a_0, b_1\}$ or $A = \{a_0\}$ or $A = \{b_1\}$ and the set of candidate actions is: $H_{change(2)} = \{h_1 = a_0 \xrightarrow{2} z_0 \upharpoonright z_1, h_2 = b_1 \xrightarrow{2} z_0 \upharpoonright z_1, h_3 = a_0 \wedge b_1 \xrightarrow{2} z_0 \upharpoonright z_1\}$.

The second change occurs at $t_2 = 3$ and we will denote it as *change*(3). Here its the gene "a" whose value changes from a_0 to a_1, thus the action that has realized this change is of the form $h = A \xrightarrow{D} a_0 \upharpoonright a_1$ where $A \in \mathcal{L}^{\diamond}, |A| \leq 2$ and D is the delay that is equal to 1 here: $D_{t_2} = t_2 - t_1 = 3 - 2 = 1$. According to R the genes which influence "a" are $G_{R_a} = \{a\}$. It means that $A = \{a_?\}$ and the expression level of "a" between t_1 and t_2 is a_0. So $A = \{a_0\}$ and it is an auto-influence. Thus there is only one candidate action that is a *self-action*: $H_{change(3)} = \{h = a_0 \xrightarrow{1} a_0 \upharpoonright a_1\}$.

The third change occurs at $t_3 = 4$ and we will denote it as *change*(4). Here it is the gene "b" whose value changes from b_1 to b_0, thus the action that has realized this change is of the form $h = A \xrightarrow{D} b_1 \upharpoonright b_0$ where $A \in \mathcal{L}^{\diamond}, |A| \leq 2$ and D is the delay and it is equal to 1 ($D_{t_3} = t_3 - t_2 = 4 - 3 = 1$). According to R there is no gene that can influences "b", thus no action can realizes this change.

The fourth change occurs at $t_4 = 5$ and we will denote it as *change*(5). Here it is "a" whose value changes from a_1 to a_0, thus the action that has realized this change is of the form $h = A \xrightarrow{D} a_1 \upharpoonright a_0$
where $A \in \mathcal{L}^{\diamond}, |A| \leq 2$ and D is the delay that is equal to 1 ($D_{t_4} = t_4 - t_3 = 5 - 4 = 1$). According to R the genes which influence "a" are $G_{R_a} = \{a\}$. Again $A = \{a_?\}$ and since the expression level of "a" between t_3 and t_4 is a_1, we have $A = \{a_1\}$ and there is only one candidate action that is a *self-action* $H_{change(5)} = \{h = a_1 \xrightarrow{1} a_1 \upharpoonright a_0\}$.

The fifth change occurs at $t_5 = 6$ and we will denote it as *change*(6). Here it is "z" whose value changes from z_1 to z_0, thus the action that has realized this change has the form of: $h = A \xrightarrow{D} z_1 \upharpoonright z_0$
where $A \in \mathcal{L}^{\diamond}, |A| \leq 2$ and D is equal to 1 here: $D_{t_5} = t_5 - t_4 = 6 - 5 = 1$
According to R, $G_{R_z} = \{a, b\}$. It means that $A = \{a_?, b_?\}$ or $A = \{a_?\}$ or

$A = \{b_?\}$ The expression level of "a" and "b" between t_4 and t_5 is respectively a_0 and b_0. Thus $A = \{a_0, b_1\}$ or $A = \{a_0\}$ or $A = \{b_0\}$
The candidates action are: $H_{change(6)} = \{h_1 = a_0 \xrightarrow{1} z_1 \Rsh z_0,\ h_2 = b_0 \xrightarrow{1} z_1 \Rsh z_0,$ $h_3 = a_0 \wedge b_0 \xrightarrow{1} z_1 \Rsh z_0\}$.

After proccessing all chronograms, the candidate actions are:
$H_{change(2)} = \{h_1 = a_0 \xrightarrow{2} z_0 \Rsh z_1, h_2 = b_1 \xrightarrow{2} z_0 \Rsh z_1, h_3 = a_0 \wedge b_1 \xrightarrow{2} z_0 \Rsh z_1\}$.
$H_{change(3)} = \{h_4 = a_0 \xrightarrow{1} a_0 \Rsh a_1\}$.
$H_{change(5)} = \{h_5 = a_1 \xrightarrow{1} a_1 \Rsh a_0\}$.
$H_{change(6)} = \{h_6 = a_0 \xrightarrow{1} z_1 \Rsh z_0\ , h_7 = b_0 \xrightarrow{1} z_1 \Rsh z_0, h_8 = a_0 \wedge b_0 \xrightarrow{1} z_1 \Rsh z_0\}$.

At this stage of the process, all candidates actions are consistent with all observations and given regulation influences. Until now the method used ensured completeness: here we have the complete set of consistent action that can explain the observations.

4 Refinement of generated PH model

The Algorithm 1 returns a PH model with a set of actions compeletely coherent with the observations as well as the regulatory genes influences. But in practice this set of actions can be further refined using a background knowledge. The following filtering operation can help to reduce the complexity of the model learned/revised.

4.1 1st Filter: priority to given model

If we want to minimize the number of actions added to the input PH model (PH_{ini})we can consider that the actions of this PH are more trustable than the generated one. Thus generated action(s) explaining a change can already be explained by a given action in PH_{ini}. So that the generated action(s) can be discarded:

Definition 6 (Filter: Priority to given model). *Let T be the set of time points of the given time series data, $\forall t \in T$ such that $\exists$ `change(t)`, we have:*

- *H_c: the set of candidate actions generated to explain the* `change(t)`
- *H_{ini}: the set of actions of PH_{ini} that can realize the* `change(t)`
- *H_{final}: the set of actions to keep to model the* `change(t)`

We have either:

- *If $H_c \cap H_{ini} \neq \emptyset$ then $H_{final} = H_c \cap H_{ini}$*
- *If $H_c \cap H_{ini} = \emptyset$ then $H_{final} = H_c$*

In practice, the change that already can be explained by the input PH can be detected before computing the candidates actions, allowing us to discard them without generating them.

If in our running example (Section 3.2), we start with $H_{ini} = \{h_{ini} = a_0 \xrightarrow{1} z_1 \upharpoonright z_0\}$, since the action h_{ini} can realize *change*(6) there is no need to generate new ones. Here, $H_{change(6)}$ will only consist of $h_{ini} = h_6$. So $h_7 = b_0 \xrightarrow{1} z_1 \upharpoonright z_0$ and $h_8 = a_0 \wedge b_0 \xrightarrow{1} z_1 \upharpoonright z_0$ will be discarded.

4.2 2nd Filter: strict influences (activator or inhibitor)

If knowledge about strict influences is given it can be used to discard actions that are conflicting with those influences. For example, if we know that a gene "a" influences a gene "b" and that "a" can only inhibit "b", then the actions using "a" as an active hitter to increase the value of "b" are inconsistent and can be discarded.

Definition 7 (Filter: Strict influences). *Let $PH = (\Sigma, \mathcal{L}, \mathcal{H}_{tp})$ be the input model, $\forall h \in \mathcal{H}_{tp}$ such $h = A \xrightarrow{D} b_n \upharpoonright b_m$, with $A \in \mathcal{L}^{\diamond}, |A| \leq i \wedge b \in \Sigma \wedge\ b_j \neq b_k \wedge D \in \mathbb{R}^+$ (i: indegree of the model). Let $G \in \Sigma$ we have :*
If $\exists G_k \in A$ such that $G \xrightarrow{(-)} b \wedge \nexists G \xrightarrow{(+)} b$ (resp. $G \xrightarrow{(+)} b$ and $\nexists G \xrightarrow{(-)} b$) $\wedge$ $k \neq 0 \wedge n < m$ (resp. $n > m$), then h can be discarded.

In our running example (Section 3.2), if we know that "b" is an inhibitor of "z", all actions where "b" is used to activate "z" can be discarded. Here, we have an observation where "z" is activated in *change*(2) where its value switches from z_0 to z_1. We know that "a" has an influence on "z" but as an inhibitor not as an activator thus the actions $h_2 = b_1 \xrightarrow{2} z_0 \upharpoonright z_1$ and $h_3 = a_0 \wedge b_1 \xrightarrow{2} z_0 \upharpoonright z_1$ will be discarded.

4.3 3rd Filter: delay merging

When the time information of observation is not perfect, the same regulation interaction may append with different delay. One simple solution to deal with such input can be to simply agregate action that differ only by their delays. We can merge each action with the same hitters, $S_1, P_1, \ldots, S_n, P_n$ the same target, G, P and the same bound G, P' into one action where the delay is the average.

Definition 8 (Filter: Delay merging). *$\forall h_1, h_2, ..., h_k \in H$ such that $h_1 = A \xrightarrow{D_1} a_n \upharpoonright a_m$, $h_2 = A \xrightarrow{D_2} a_n \upharpoonright a_m$, ..., $h_k = A \xrightarrow{D_k} a_n \upharpoonright a_m$ with $A \in \mathcal{L}^{\diamond}$, $a_n, a_m \in \mathcal{L}_a$ and $D_1 \neq D_2 \neq ... \neq D_k \in \mathbb{R}^+$ then :*
***`fusion`** all actions $h_1, h_2, ..., h_k$ into one action h:*

$$h = A \xrightarrow{D_{average}} a_n \upharpoonright a_m, \qquad \text{such that,} \qquad D_{average} = \frac{\sum_{i=1}^{k} D_i}{k}$$

For example, let us suppose that we came out with two actions $h_1 = a_0 \xrightarrow{2} z_0 \Rsh z_1$ and $h_2 = a_0 \wedge b_1 \xrightarrow{4} z_0 \Rsh z_1$, they will be merged into $h = b_1 \xrightarrow{3} z_0 \Rsh z_1$. If input data are perfect, usage of this merging is totally unsafe and can lead to a set of actions that cannot produce any of the observed changes in the worst case. The intuition behind this method is to give a first idea of how to cope with big amount of real data which are not perfect. The idea is that if enough observations are provided, the delay of the action will be more precise.

5 Evaluation

In this section we provide two evaluation of the proposed mehtod. First we show with the example of the circadian clock model of [6] that our method can reconstruct the correct PH from its observations. Here the model is known and we generate the corresponding chronogram given from an initial state. We evaluate the capacity of our algorithm[3] to find the correct actions and the impact of the quantity of observations on run time. In the second experiment our goal is to assess the scalability of our approach in practice. Here we process chronograms obtained from time series data of the DREAM4 challenge [23].

5.1 Circadian clock

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

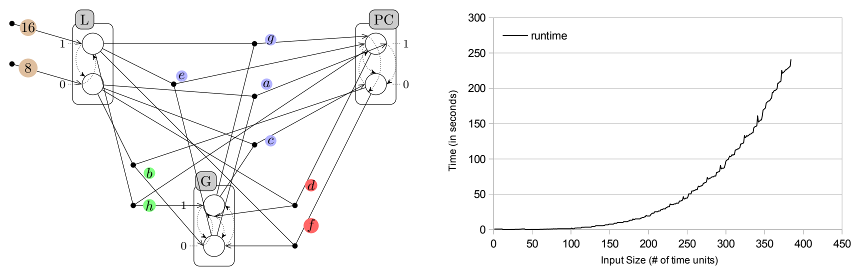

Fig. 4. Representation of circadian clock in Process Hitting (left). The value "a", "b", ..., "h" represent the delays in actions. Run time of the application of Algorithm 1 on circadian clock chronograms varying the number of time steps (right)

[3] All programs, described in this article, for PH generation and for the refinement are implemented in Answer Set Programming (ASP) and are available online at: `http://git.irccyn.ec-nantes.fr/benabdal/generate_revise_d-brn/repository/archive.zip`

Figure 5.2 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, our algorithm succeed to learn the correct actions. Using this benchmark we also analysed 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 real data from biological system with chronogram exceeding 100 points. In work like [9] 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).

5.2 DREAM4

In this section, we assess the efficiency of our algorithm through case studies coming from the DREAM4 challenge. DREAM challenges are annual reverse engineering challenges that provide biological case studies. In this paper, we focus on the datasets coming from DREAM4. The input data that we tackle here consists of the following: 5 different systems each composed of 100 genes, all coming from E. coli and yeast networks. For every such system, the available data are the following: (i) 10 time series data with 21 time points and 1000 is the duration of each time series; (ii) steady state for wild type; (iii) steady states after knocking out each gene; (iv) steady states after knocking down each gene (i.e. forcing its transcription rate at 50%); (v) steady states after some random multifactorial perturbations. We processed all the data. Here, we focus on the management of time series data.
Each time serie includes different perturbations that are maintained all time along during the first 10 time points and applied to at most 30 genes. In this setting, a perturbation means a significant increase or decrease of the gene expression. In the raw data of the time series, gene expression values are given as real numbers between 0 and 1. To apply our approach, we chose to discretize those data into two to six qualitative values. Each gene is discretized in an independent manner, with respect to the following procedure: we compute the average value of the gene expression among all data of a time series, then the values between the average and the maximal/minimal value are divided into as many levels. Discretizing the data according to the average value of expression is expected to reduce the impact of perturbation on the discretization and thus on the model learned. But in this experiment our goal is to assess the scalability of our approach in practice rather than evaluating the model learned when facing perturbated data. Figure 5.2 shows the impact of both actions indegree and discretization level on run time.

In the results obtained from the experimentations of our algorithm on the time series data of the DREAM4 we can see the exponential influence on the run

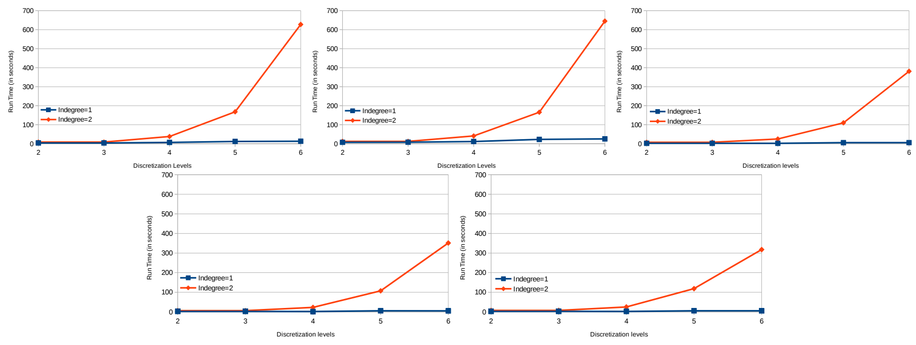

Fig. 5. Evolution of run time on processing different model time series data from DREAM4 varying indregree of actions and discretization levels. These tests were performed on processor: 3 GHz Intel Core i7 with 16 Go of RAM.

time of indegree of action considered as well as the level of discretization chosen for all the 5 different networks. But it also shows that in practice our approach can tackle big network, here 100 genes. It is important to note that here the real influences are given as background knowledge, thus it greatly reduces the quantity of action generated to explain each change. Without knowledge about those influences, the algorithm can be run considering that all genes can be influenced by all other genes. But here, the quantity of combination of influences is so huge that processing only one time series discretized into two levels of expression with an indegree of two takes more that 12 hours to process. In practice, on large networks (more than 40 genes), accessing to information about genes influences is crucial for the efficiency of this method.

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.

Acknowledgment

This paper is based upon work supported by the Agence Nationale de la Recherche for the HyClock project under Grant No. ANR-14-CE09-0011.

References

1. Jamil Ahmad, Gilles Bernot, Jean-Paul Comet, Didier Lime, and Olivier Roux. Hybrid modelling and dynamical analysis of gene regulatory networks with delays. *ComPlexUs*, 3(4):231–251, 2006.
2. Tatsuya Akutsu, Takeyuki Tamura, and Katsuhisa Horimoto. Completing networks using observed data. In *Algorithmic Learning Theory*, pages 126–140. Springer, 2009.
3. Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.
4. Chitta Baral. Using answer set programming for knowledge representation and reasoning: Future directions. In *ICLP*, pages 69–70, 2008.
5. Emna Ben Abdallah, Maxime Folschette, Olivier Roux, and Morgan Magnin. Exhaustive analysis of dynamical properties of biological regulatory networks with answer set programming. Accepted not yet published to BIBM, 2015.
6. Jean-Paul Comet, Gilles Bernot, Aparna Das, Francine Diener, Camille Massot, and Amélie Cessieux. Simplified models for the mammalian circadian clock. *Procedia Computer Science*, 11:127–138, 2012.
7. Jean-Paul Comet, Jonathan Fromentin, Gilles Bernot, and Olivier Roux. A formal model for gene regulatory networks with time delays. In *Computational Systems-Biology and Bioinformatics*, pages 1–13. Springer, 2010.
8. Markus Durzinsky, Wolfgang Marwan, Max Ostrowski, Torsten Schaub, and Annegret Wagler. Automatic network reconstruction using asp. *Theory and Practice of Logic Programming*, 11(4-5):749–766, 2011.
9. Louis Fippo Fitime, Andrea Beica, Olivier Roux, and Carito Guziolowski. Integrating time-series data on large-scale cell-based models: application to skin differentiation. In *Evry Spring school, in Advances in Systems and Synthetic Biology*, pages 57–72, 2014.
10. Maxime Folschette. *Modélisation algébrique de la dynamique multi-échelles des réseaux de régulation biologique*. PhD thesis, University of Nantes, ED STIM, Ecole Centrale de Nantes, Université de Nantes, Nantes, October 2014. Sous la direction de: Olivier Roux et Morgan Magnin. Jury : Mireille R'egnier (prés.), Jean-Paul Comet (rapp.), Anne Siegel (rapp.), Denis Thieffry.
11. Maxime Folschette, Loïc Paulevé, Morgan Magnin, and Olivier Roux. Under-approximation of reachability in multivalued asynchronous networks. *Electronic Notes in Theoretical Computer Science*, 299:33–51, 2013.
12. Maxime Folschette, Loïc Paulevé, Morgan Magnin, and Olivier Roux. Under-approximation of reachability in multivalued asynchronous networks. *Electronic Notes in Theoretical Computer Science*, 299:33–51, 2013. 4th International Workshop on Interactions between Computer Science and Biology (CS2Bio'13).
13. Paul Freedman. Time, petri nets, and robotics. *Robotics and Automation, IEEE Transactions on*, 7(4):417–433, 1991.
14. Katsumi Inoue, Andrei Doncescu, and Hidetomo Nabeshima. Completing causal networks by meta-level abduction. *Machine learning*, 91(2):239–277, 2013.
15. Vivien Marx. Biology: The big challenges of big data. *Nature*, 498(7453):255–260, 2013.
16. Hiroshi Matsuno, Atsushi Doi, Masao Nagasaki, and Satoru Miyano. Hybrid petri net representation of gene regulatory network. In *Pacific Symposium on Biocomputing*, volume 5, page 87. World Scientific Press Singapore, 2000.

17. Natsu Nakajima and Tatsuya Akutsu. Network completion for time varying genetic networks. In *Complex, Intelligent, and Software Intensive Systems (CISIS), 2013 Seventh International Conference on*, pages 553–558. IEEE, 2013.
18. Natsu Nakajima and Tatsuya Akutsu. Exact and heuristic methods for network completion for time-varying genetic networks. *BioMed research international*, 2014, 2014.
19. Natsu Nakajima and Tatsuya Akutsu. Network completion for static gene expression data. *Advances in bioinformatics*, 2014, 2014.
20. Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell.*, 25(3-4):241–273, 1999.
21. Loïc Paulevé, Morgan Magnin, and Olivier Roux. Refining dynamics of gene regulatory networks in a stochastic π-calculus framework. In *Transactions on Computational Systems Biology XIII*, volume 6575 of *Lecture Notes in Comp Sci*, pages 171–191. Springer, 2011.
22. Loïc Paulevé, Morgan Magnin, and Olivier Roux. Static analysis of biological regulatory networks dynamics using abstract interpretation. *Mathematical Structures in Computer Science*, 22(04):651–685, 2012.
23. Robert J Prill, Julio Saez-Rodriguez, Leonidas G Alexopoulos, Peter K Sorger, and Gustavo Stolovitzky. Crowdsourcing network inference: the dream predictive signaling network challenge. *Science signaling*, 4(189):mr7, 2011.
24. Heike Siebert and Alexander Bockmayr. Temporal constraints in the logical analysis of regulatory networks. *Theoretical Computer Science*, 391(3):258–275, 2008.
25. Santiago Videla, Carito Guziolowski, Federica Eduati, Sven Thiele, Martin Gebser, Jacques Nicolas, Julio Saez-Rodriguez, Torsten Schaub, and Anne Siegel. Learning boolean logic models of signaling networks with asp. *Theoretical Computer Science*, 2014.
26. Yoshitaka Yamamoto, Adrien Rougny, Hidetomo Nabeshima, Katsumi Inoue, Hisao Moriya, Christine Froidevaux, and Koji Iwanuma. Completing sbgn-af networks by logic-based hypothesis finding. In *Formal Methods in Macro-Biology*, pages 165–179. Springer, 2014.