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 in the integration of quantitative time delays directly in our learning approach. Finally, we show the benefits of such automatic approaches 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, network revision, dynamic madelisation, 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 [16]. 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 [3], the authors targeted the completion of stationary Boolean networks. This method has been further refined along the years. Latest works [18] 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 [20], and the implementation has been improved through heuristics [19]. 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 [15] 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 [21,4,5] has been proven useful for network reconstruction [9] 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 [17], 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 [2]. Finally, in [8], 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 [22] 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 [12], 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 **??** some preliminary insights about recent translation of PH into ASP presented in [6]. All theoretical and practical notions are then settled to introduce our timed extension of PH, and related completion algorithm in section 3. Then we illustrate the merits of our approach in section 5 by first applying it on a simplified model of mammalian circadian clock [7], 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 [22] 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 $a_i \to b_j \Rsh b_k$, 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: $a_i \to a_i \Rsh a_k$. 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 and new actions like plural actions [11] (at least 2 hitters), actions with priority [13] and actions with delay.

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 [23].

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: $\mathcal{L}_a = \{a_0, a_1\}$, $\mathcal{L}_b = \{b_0, b_1\}$, $\mathcal{L}_z = \{z_0, z_1, z_2\}$.

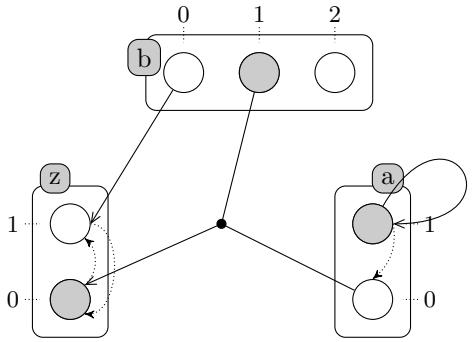

Fig. 1. A PH model example with three sorts: a, b and z (a is either at level 0 or 1, b at either level 0, 1 or 2 and z at either level 0, or 1). Boxes represent the *sorts* (network components), circles represent the *processes* (component levels), and the 3 *actions* that model the dynamic behavior 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 4) 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). 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 2).

Definition 2 (Timed 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 action of $\mathcal{PH}$ is a action with a delay D: $A \xrightarrow{D} b_i \Rsh b_j$ 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 $b_i \in A$, $A = b_i$.*

To model biological networks, we used the PH framework with timed actions (Definition 3). Indeed, in the biological models that we studied we need to present not only the cooperation between the components to influence another one (action), but also the time that this action need to take place (delays).

Definition 3 (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 $a \neq b \Rightarrow \mathcal{L}_a \cap \mathcal{L}_b = \emptyset$;*
- *$\mathcal{H}_{tp} = \{A \xrightarrow{D} b_j \Rsh b_k \mid A \in \mathcal{L}^\diamond, b_j \neq b_k, b_i \in A \Rightarrow A = b_j\}$ is the finite set of* timed 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 0 it is equivalent to Process Hitting without delays (original PH). The way these new actions should be used is described as follows.

Definition 4 (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 PH. We say that the action $h = A \xrightarrow{D} b_i \Rsh b_j$, with $D \geq 0$, 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 $s[b] = b_j$).*

We denote the resulting state of the $\mathcal{PH}$ model ($\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H}_{tp})$) after playing an action h in a state s is called a *successor* of s and is denoted by $(s \cdot h)$ as well as the resulting state of a sort $a \in \Sigma$ by $s[a]$. The dynamic of the Process Hitting is based on asynchronized evolution so we have:

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

.

Definition 5 (Semantics of a Process Hitting with Timed Actions). *Let* $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H}_{tp})$ *be a dense-time Process Hitting and* $\mathsf{state}(\mathcal{PH}, t) = s$ *be the state of* $\mathcal{PH}$ *at time* t *with* $s \in \mathcal{L}$*. Let* $h = A \xrightarrow{D} b_i \Rsh b_j$ *be a playable timed action at* t *in* s*, with* $D \geq 0$ *and* t, t' *and* $(t + D)$ *time steps. So the semantic is explained by:*

$$\forall t' \in [t, t + D] : \mathsf{state}(\mathcal{PH}, t') = s \text{ and } \mathsf{state}(\mathcal{PH}, t + D) = (s \cdot h).$$

We propose at Definition 5, the semantic of the timed PH that we base on to generate the corresponding model to the given observations. Indeed it means that if there is no action in $\mathcal{H}$ that is playable in s the state of the system remains s for all time t' with $t' > t$.If $h = A \xrightarrow{D} b_i \Rsh b_j$ is the choosen playable actions 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 at the step $t + D$.

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. COMMENT Tony: toujours un argument pour TACAS l'ASP ? 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 Revision of PH networks

3.1 Algorithm

In this section we propose an algorithm to construct and/or revise Process Hitting models. 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: Generate S the set of all subsets of actions of A that realize C such that A is minimal:

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

- 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 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 2, the action a is removed from A''. But since a realized one of the change of C and a is generated at step 1, then it will be present in one of the minimal subset of actions. Such that 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 $O(c \times i^S + (2^{2 \times T \times P \times i^S} + c \times 2^{T \times P \times i^S})$ 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 $O(T \times P^{S+1} \times 2^{T \times P^{S+1}})$.

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 bound 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 bound by $O(T \times P^S + (T \times P^{S+1})^2 + 2^{2 \times T \times P^{S+1}} + T \times 2^{T \times P^{S+1}})$ □

4 Algorithm simulation

In this section we demonstrate how our methode permits to generate a PH model coherent to the set of biological regulatory time series data given as an input. First, the method uses discritised observations as an input, thus it is necessary to use an other method which transforms the analogic time series data to discritised time series data.

We can summarrize our method in the following steps:

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

We will now show an example of the execution of algorithm on the chronogram of Figure 2:

The first change occurs at $t_1 = t_{min} = 2$, wich we will denote as `change(2)`. It has been caused by an action $h = A \xrightarrow{D} z_0 \uparrow z_1$ where $A \in \mathcal{L}^{\diamond}, |A| \leq i, i = 2$ and D is the delay wich 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}`)`, $D_{t_1} = t_1 - t_0 = 2 - 0 = 2$.

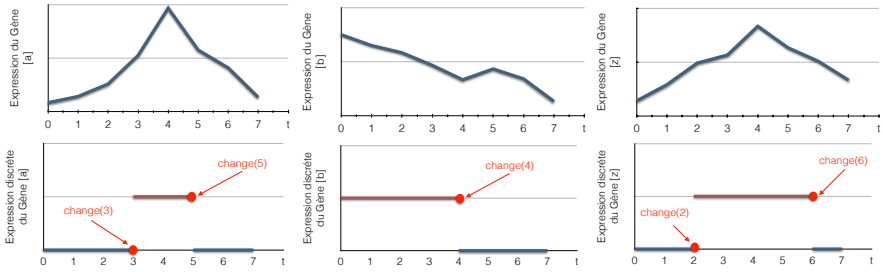

Fig. 2. Examples of the discretization of continues time series data into bi-valued chronograms. Abscisse represents time and ordinate the gene expression level. The expression level is discretized according to a treshold fixed to the half of the gene expression value in this example.

Let $R = \{b \to z, a \to z, a \to a\}$ be the set of regulation influences amoung the components of the system. In the first change $t_1 = 2$ that we will denotes as $change(2)$, its $\mathtt{z}$ whose value changes from z_0 to z_1, thus the action that has realized this change is of the form $h = A \xrightarrow{2} z_0 \nearrow z_1$. According to R the genes which influence $\mathtt{z}$ are $G_{R_z} = \{\mathtt{a}, \mathtt{b}\}$. 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 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: $H_{change(2)} = \{h_1 = a_0 \xrightarrow{2} z_0 \nearrow z_1, h_2 = b_1 \xrightarrow{2} z_0 \nearrow z_1, h_3 = a_0 \wedge b_1 \xrightarrow{2} z_0 \nearrow z_1\}$.

The second change occurs at $t_2 = 3$ and we will denotes it as $change(3)$. Here its $\mathtt{a}$ whose value changes from a_0 to a_1, thus the action that has realized this change is of the form $h = A \xrightarrow{D} a_0 \nearrow a_1$ 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 $\mathtt{a}$ are $G_{R_a} = \{\mathtt{a}\}$. It means that $A = \{a_?\}$ and the expression level of $\mathtt{a}$ between t_1 and t_2 is a_0. Thus $A = \{a_0\}$ and there is only one candidate action that is a self action: $H_{change(3)} = \{h = a_0 \xrightarrow{1} a_0 \nearrow a_1\}$.

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

The fourth change occurs at $t_4 = 5$ and we will denotes it as $change(5)$. Here its $\mathtt{a}$ whose value changes from a_1 to a_0, thus the action that has realized this change is of the form$h = A \xrightarrow{D} a_1 \nearrow a_0$
where $A \in \mathcal{L}^\diamond, |A| \leq 2$ and D is the delay that is equal to 1 here: $D_{t_4} = t_4 - t_3 = 5 - 4 = 1$. According to R the genes which influence $\mathtt{a}$ are $G_{R_a} = \{\mathtt{a}\}$.

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: $H_{change(5)} = \{h = a_1 \xrightarrow{1} a_1 \not\uparrow a_0\}$.

The fifth change occurs at $t_5 = 6$ and we will denotes it as $change(6)$. Here its `z` whose value changes from z_1 to z_0, thus the action that has realized this change is of the form $h = A \xrightarrow{D} z_1 \not\uparrow z_0$
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 the genes which influence `z` are $G_{R_z} = \{$`a, b`$\}$. It means that $A = \{a_?, b_?\}$ or $A = \{a_?\}$ or $A = \{b_?\}$ The expression level of `a` and `a` 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: mon slip $H_{change(6)} = \{h_1 = a_0 \xrightarrow{1} z_1 \not\uparrow z_0$, $h_2 = b_0 \xrightarrow{1} z_1 \not\uparrow z_0$, $h_3 = a_0 \wedge b_0 \xrightarrow{1} z_1 \not\uparrow z_0\}$.

After proccessing all chronograms, the candidates action are:
$H_{change(2)} = \{h_1 = a_0 \xrightarrow{2} z_0 \not\uparrow z_1, h_2 = b_1 \xrightarrow{2} z_0 \not\uparrow z_1$
$, h_3 = a_0 \wedge b_1 \xrightarrow{2} z_0 \not\uparrow z_1\}$.
$H_{change(3)} = \{h_4 = a_0 \xrightarrow{1} a_0 \not\uparrow a_1\}$.

$H_{change(5)} = \{h_5 = a_1 \xrightarrow{1} a_1 \not\uparrow a_0\}$.

$H_{change(6)} = \{h_6 = a_0 \xrightarrow{1} z_1 \not\uparrow z_0$, $h_7 = b_0 \xrightarrow{1} z_1 \not\uparrow z_0, h_8 = a_0 \wedge b_0 \xrightarrow{1} z_1 \not\uparrow z_0\}$.

At this stage of the process, all canditates 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. But in practice those set of action can be further refine using background knowledge. The following filtering operation can help to reduce the complexity of the model learned/revised.

4.1 1^{st} filter: priority on given model

If we want to minimize the number of actions added to the input PH we can consider that the action of this PH are more trustable than the generated one. Thus generated actions that only explain changes that can already be explain by given action can be discarded: $\forall t \in T$ such that $\exists$ `change(t)`, we have:

- H_c: the set of candidates action generated to explain $change(t)$
- H_{ini}: the set of actions of PH_{ini} that can realize the change at t
- H_{final}: the set of actions to keep as to explain the change at 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, we start with $H_{ini} = \{h_i nia_0 \xrightarrow{1} z_1 \Rsh z_0\}$, since the action $h_i ni$ can realize $change(6)$ there is no need to generate new ones. Here, $H_{change(6)}$ will only consist of $h_i ni = h_6$, $h_7 = b_0 \xrightarrow{1} z_1 \Rsh z_0$ and $h_8 = a_0 \wedge b_0 \xrightarrow{1} z_1 \Rsh z_0$ will be discarded.

4.2 2^{nd} filter: strict influences (activator or inhibitor)

If knowledge about strict influences is given it can be used to discarded 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 hitter to increase the value of b are inconsistent and can be discarded. $\forall h \in H$ such $h = A \xrightarrow{D} b_n \Rsh b_m$, with $A \in \mathcal{L}^{\diamond}, |A| \leq i$ (i: indegree of the algorithm) we have :

> If $n < m$, if $\exists G_k \in \mathcal{L}_G$ such that $G_k \xrightarrow{(-)} b$ (and $\nexists G_k \xrightarrow{(+)} b$) and $k \neq 0$ then h can be discarded.
> If $n > m$, if $\exists G_k \in \mathcal{L}_G$ such that $G_k \xrightarrow{(+)} b$ (and $\nexists G_k \xrightarrow{(-)} b$) and $k \neq 0$ then h can be discarded.

In our running example, if we now that b is an inhibitor of z, all actions where a 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 now that a as an influence on z but as an inhibitor not as an activator thus the actions $h_2 = b_1 \xrightarrow{2} z_0 \Rsh z_1$ and $h_3 = a_0 \wedge b_1 \xrightarrow{2} z_0 \Rsh z_1$ will be discarded.

4.3 $3^r d$ 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 delay. We can 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. $\forall h_1, h_2, ..., h_k \in H$ such that $h_1 = A \xrightarrow{D_1} a_n \Rsh a_m$, $h_2 = A \xrightarrow{D_2} a_n \Rsh a_m$, ..., $h_k = A \xrightarrow{D_k} a_n \Rsh a_m$ with $A \in \mathcal{L}^{\diamond}$, $a_n, a_m \in \mathcal{L}_a$ et $D_1 \neq D_2 \neq ... \neq D_k$ then :
`fusion` all actions $h_1, h_2, ..., h_k$ into one action h

$$h = A \xrightarrow{D_{average}} a_n \Rsh a_m$$

such that:

$$D_{average} = \frac{\sum_{i=1}^{k} D_i}{k}$$

For example, lets suppose that we came out with two actions $h = a_0 \xrightarrow{2} z_0 \Rsh z_1$ and $h' = a_0 \wedge b_1 \xrightarrow{4} z_0 \Rsh z_1$, they will merged into $h_2 = b_1 \xrightarrow{3} z_0 \Rsh z_1$. 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 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 10 genes, all coming from E. coli and yeast networks. For every such system, the available data are the following: (i) 5 time series data with 21 time points; (ii) steady state at 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.

5.1 Settings

Time series data provide us 20 transitions. Each of them include different perturbations that are maintained all time along during the first 10 transitions and applied to at most 3 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 number between 0 and 1. To apply our approach, we chose to discretize those data into 4 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.

5.2 Results

TODO

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.

References

1. Emna Ben Abdallah. Exhaustive search of dynamical properties in process hitting using asp. In *Extended abstract for the MOVEP student sessions, 8288. June 2014, Nantes, France*, 2014.
2. 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.
3. Tatsuya Akutsu, Takeyuki Tamura, and Katsuhisa Horimoto. Completing networks using observed data. In *Algorithmic Learning Theory*, pages 126–140. Springer, 2009.
4. Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.
5. Chitta Baral. Using answer set programming for knowledge representation and reasoning: Future directions. In *ICLP*, pages 69–70, 2008.
6. Emna Ben Abdallah, Maxime Folschette, Olivier Roux, and Morgan Magnin. Exhaustive dynamic analysis of reachability and stable states of biological regulatory networks modeled in process hitting using answer set programming. Submitted to ICLP'2015, 2015.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
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.
13. 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).

14. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080, 1988.
15. Katsumi Inoue, Andrei Doncescu, and Hidetomo Nabeshima. Completing causal networks by meta-level abduction. *Machine learning*, 91(2):239–277, 2013.
16. Vivien Marx. Biology: The big challenges of big data. *Nature*, 498(7453):255–260, 2013.
17. 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.
18. 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.
19. Natsu Nakajima and Tatsuya Akutsu. Exact and heuristic methods for network completion for time-varying genetic networks. *BioMed research international*, 2014, 2014.
20. Natsu Nakajima and Tatsuya Akutsu. Network completion for static gene expression data. *Advances in bioinformatics*, 2014, 2014.
21. Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell.*, 25(3-4):241–273, 1999.
22. 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.
23. 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.
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.