Building model prototypes from time-course data

Alan Veliz-Cuba
University of Dayton

Stephen Randal Voss
University of Kentucky

David Murrugarra
University of Kentucky

January 27, 2022

Abstract

A primary challenge in building predictive models from temporal data is selecting the appropriate network and the regulatory functions that describe the data. Software packages are available for equation learning of continuous models, but not for discrete models. In this paper we introduce a method for building model prototypes that consist of a network and a set of discrete functions that can explain the time course data. The method takes as input a collection of time course data or discretized measurements over time. After model inference, we use our toolbox to simulate the prototype model as a stochastic Boolean network. Our method provides a model that can qualitatively reproduce the patterns of the original data and can further be used for model analysis, making predictions, and designing interventions. We applied our method to a time-course, gene expression data that were collected during salamander tail regeneration. The inferred model captures important regulations that were previously validated in the research literature. The toolbox for inference and simulations is freely available at `github.com/alanavc/prototype-model`.

1 Introduction

The process of constructing discrete models from experimental data has several steps that have been studied in parallel. The main steps involved in this process are discretization, network inference and network selection, model interpolation, and stochastic simulations. Although these steps have been studied independently [3, 5, 6, 8, 10, 16, 17, 20], few tools exist that provide an automated and easily customizable pipeline to quickly create model prototypes. Equation learning (EQ) methods for differential equation (DE) models start with a collection of time course data and then "recovers" the governing equations using a library of functions [2, 7]. Many methods for EQ for DE models are based on formulating the inference problem as a parameter estimation problem that can be solved

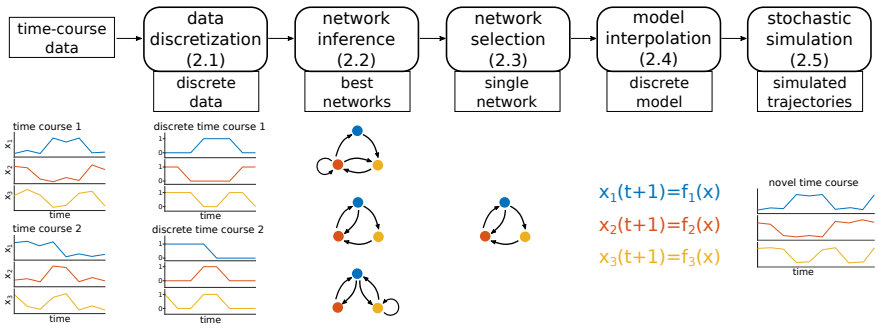


Figure 1: Flowchart showing the steps in model creation from data and the sections where each step is described. Starting from experimental time courses, we first transform the data into discrete values (in this case Boolean). Using algebraic techniques, we find the best networks that explain the data. Each network found will be consistent with all discrete time courses. We select the best network from the networks found and then find a discrete model that fits all the discrete data. This will result in a discrete model that can be simulated and compared with the original data. The model can also be run with new initial conditions or for longer time to create novel time courses that can be used to make predictions.

via optimization techniques [2, 7]. Analogue methods for equation learning of discrete models that can learn both the network and the functions are still under development. Some of these existing methods can provide network candidates (i.e., possible wiring diagrams) that can explain the data. Other methods can provide candidate functions based on interpolating the data.

In this paper we present an implementation of several algorithms, which together, form a pipeline for prototyping models from time-course data. Our approach is modular so that each module can be modified or even replaced as the user sees fit. It is written in Matlab/Octave and does need any external toolboxes or libraries.

The starting point of our method is real numerical time-course data that is internally discretized. Our focus is the construction of Boolean models, but we show with a toy model how our method also works for mixed-state models where variables can have different number of states. As an application, we construct a model prototype using gene expression data for several time points which was collected during tail regeneration experiments in axolotls.

2 Methods

Here we describe the methods for model selection (i.e., network and regulatory functions) and the framework for simulations.

We assume that we are given time courses of the form $s^1 \to s^2 \to \ldots \to s^r$,

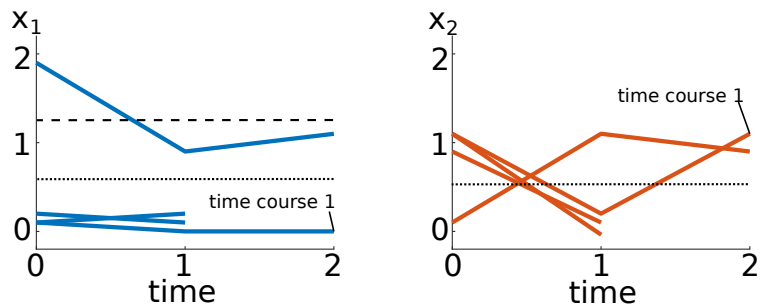


Figure 2: Values of x_1 and x_2 for the time courses. Variable x_1 can be considered as having 3 levels, whereas variable x_2 has 2 levels. The dashed lines show how the range of the data can be divided into regions (3 regions for x_1 and 2 for x_2), which will determine the discretization.

where $s^i = (s^i_1, \ldots, s^i_n) \in S = S_1 \times \cdots \times S_n$. Here S_i is a finite set of all the values that the i-th variable can take. Note that if $S_i = \{0, 1\}$, then we have a Boolean network.

Example 2.1. *To illustrate the methods, we use an example with the following four time courses.*

(1) $(0.1, 1.1, 1.9, 0.9, 0.2) \to (0.0, 0.2, 0.2, 0.1, 0.1) \to (0.0, 1.1, 0.1, 1.9, 2.1)$
(2) $(1.9, 0.1, 0.9, 0.1, 0.0) \to (0.9, 1.1, 0.1, 1.9, 2.1) \to (1.1, 0.9, 0.1, 1.9, 2.0)$
(3) $(0.2, 1.1, 1.9, 0.9, 1.1) \to (0.1, 0.0, 0.2, 0.1, 0.1)$
(4) $(0.1, 0.9, 2.1, 1.1, 2.1) \to (0.2, 0.1, 0.2, 0.1, 1.1)$

2.1 Discretization

We implemented a simple discretization method based on binning data by dividing the range of the data into equally spaced regions. The time courses suggest that the number of levels for variables x_1, x_2, x_3, x_4, x_5, are 3, 2, 3, 3, 3, respectively. For example, by plotting the values of x_1 and x_2 for each trajectory (Fig. 2), we see that x_1 has 3 distinctive levels and x_2 has 2 distinctive levels. For x_1, all values below the dotted line will be mapped to 0 (low); all values between the dotted and dashed lines will get mapped to 1 (medium); and all values above the dashed line will get mapped to 2 (high). For x_2, all values below the dotted line will be mapped to 0 (low); and all values above the dotted line will get mapped to 1 (high).

Then, the discrete time courses are given below.
(1) $01210 \to 00000 \to 01022$
(2) $20100 \to 11022 \to 11022$
(3) $01211 \to 00000$
(4) $01212 \to 00001$
In this case $S = \{0, 1, 2\} \times \{0, 1\} \times \{0, 1, 2\} \times \{0, 1, 2\} \times \{0, 1, 2\}$.

2.2 Network Inference

To find the wiring diagrams that are consistent with a collection of time courses of the form $s^1 \to s^2 \to \ldots \to s^r$ we use the algebraic framework introduced in [17]. This framework takes partial information about the evolution of a network $s \to f(s)$ and returns all the minimal wiring diagrams that are consistent with the data. This approach guarantees that for each minimal wiring diagram there exists a network that fits the data such that each interaction is activation or inhibition.

To use the framework in [17], we first note that each time course $s^1 \to s^2 \to \ldots \to s^r$ implies that $s^{j+1} = f(s^j)$ for $j = 1, \ldots, r-1$, where f is the network one is trying to infer. This results in a set $D \subseteq S$ such that $f(s)$ is known for every $s \in D$.

Example 2.2. *In Example 2.1,* $D = \{01210, 00000, 20100, 11022, 01211, 01212\}$. *Then, the partial information we have is given by the table*

x	$f(x)$
01210	00000
00000	01022
20100	11022
11022	11022
01211	00000
01212	00001

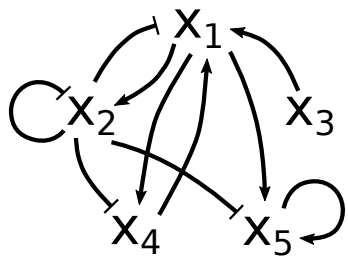
Table 1: Partial information for example.

Then, using the algebraic techniques in [17] results in all minimal wiring diagrams that are consistent with the data. For each variable x_i in the network, the algebraic framework returns $W_1, \ldots, W_k$, where each W_j is a minimal wiring diagram for variable i. For our example we obtain Table 2.

x_i	minimal wiring diagrams for x_i (+/− indicate activation/inhibition)
x_1	$\{x_1^+\}$, $\{x_2^-, x_3^+, x_4^+\}$
x_2	$\{x_3^-\}$, $\{x_2^-, x_4^+\}$, $\{x_1^+, x_4^-\}$, $\{x_1^+, x_2^-\}$
x_3	$\{\}$ (no variable affects x_3, constant function)
x_4	$\{x_3^-\}$, $\{x_2^-, x_4^+\}$, $\{x_1^+, x_4^-\}$, $\{x_1^+, x_2^-\}$
x_5	$\{x_3^-, x_5^+\}$, $\{x_2^-, x_4^+, x_5^+\}$, $\{x_1^+, x_4^-, x_5^+\}$, $\{x_1^+, x_2^-, x_5^+\}$

Table 2: Minimal wiring diagrams.

By selecting one wiring diagram for each x_i, we obtain a (global) wiring diagram that is consistent with the data. For example, if we select $\{x_2^-, x_3^+, x_4^+\}$ for x_1, $\{x_1^+, x_2^-\}$ for x_2, $\{\ \}$ for x_3, $\{x_1^+, x_2^-\}$ for x_4, and $\{x_1^+, x_2^-, x_5^+\}$ for x_5, we obtain the wiring diagram shown in Fig. 3. To compare different wiring diagrams we can use the adjacency matrix representation.



$$\begin{pmatrix} 0 & -1 & 1 & 1 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 1 \end{pmatrix}$$

Figure 3: Example of wiring diagram consistent with the data. Left: wiring diagram. Right: Adjacency matrix representation.

2.3 Wiring diagram selection

The network inference described in Section 2.2 could return several minimal network candidates for each variable. That is, for a given time course data, there might be several networks that explain the data and that are minimal. The method will return all candidate wiring diagrams. In order to select one model out of all possible options, we calculate the "best network" by including only the most frequent interactions from the network candidates. For each variable, x_i, we quantified the frequency q^+_{ji} of positive interactions $x_j \longrightarrow x_i$ across all possible network candidates and the frequency q^-_{ji} of negative interaction $x_j \dashv x_i$ across all possible network candidates for all $j = 1, \ldots, n$. Then we construct an adjacency matrix W^* by considering only the most frequent interactions. If conflicts arise (that is, when $q^-_{ji} = q^+_{ji}$ for some j), then we discard those interactions. Subsequently, for each row of W^*, say W^*_i, we calculate the distance with each possible wiring diagram of x_i (these are represented as rows). Finally, we construct an adjacency matrix W with rows corresponding to the rows with minimum distances.

Example 2.3. *For the network in Example 2.1, we calculated the frequencies which are given in Table 3 (only nonzero frequencies shown). Then, we compute*

x_i	Frequencies q^+_{ji}/q^-_{ji} of activations/inhibitions	Total
x_1	$q^+_{11} = 1$, $q^-_{21} = 1$, $q^+_{31} = 1$, $q^+_{41} = 1$	4
x_2	$q^+_{12} = 2$, $q^-_{22} = 2$, $q^+_{32} = 1$, $q^+_{42} = 1$, $q^-_{42} = 1$	7
x_3	NA	0
x_4	$q^+_{14} = 2$, $q^-_{24} = 2$, $q^-_{34} = 1$, $q^-_{44} = 1$, $q^+_{44} = 1$	7
x_5	$q^+_{15} = 2$, $q^-_{25} = 2$, $q^-_{35} = 1$, $q^+_{45} = 1$, $q^-_{45} = 1$, $q^+_{55} = 4$	11

Table 3: Frequencies of interactions on minimal wiring diagrams.

an adjacency matrix W^* from the table of frequencies that contains the most frequent interactions and discards conflicting interactions (i.e., the cases where

$q_{ji}^- = q_{ji}^+$).

$$W^* = \begin{pmatrix} 1 & -1 & 1 & 1 & 0 \\ 1 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 \\ 1 & -1 & 1 & 0 & 1 \end{pmatrix}$$

Then, for each row of W^*, say W_i^*, we calculate the distance with each possible wiring diagram of x_i (these are represented as rows). Then we construct an adjacency matrix with rows corresponding to the rows with minimum distances. Then matrix after the distance calculations is:

$$W = \begin{pmatrix} 0 & -1 & 1 & 1 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 1 \end{pmatrix}$$

The reason for why we take a distance approach is because there might not be a truth table satisfying the for W^* but there is certainly one for W as shown in Example 2.2.

2.4 Fitting Model to Data

After one wiring diagram has been selected from the family of minimal wiring diagrams, we proceed to construct a function that fits the data. Although there are known formulas for interpolation, we are interested in *monotone* interpolation, that is, we need to find a network that not only fits the data, but one whose signs of interaction match the wiring diagram selected.

We illustrate our approach with wiring diagram $\{x_1^+, x_2^-\}$ for variable x_4. Since it is guaranteed that there is a monotone function $h(x_1, x_2)$ that fits the data for variable x_4, then we consider the table 1 with only x_1 and x_2 in the first column (inputs) and only x_4 in the second column (output).

(x_1, x_2)	$h(x_1, x_2)$
01	0
00	2
20	2
11	2
01	0
01	0

Table 4: Partial information for variable x_4 with wiring diagram $\{x_1^+, x_2^-\}$.

We now rewrite this table as a truth table, where some entries are unknown.

To fill in the table, we use the fact that the function increases with respect to x_1 and decreases with respect to x_2. For example, since $h(2,1) \geq h(1,1) = 2$,

(x_1, x_2)	$h(x_1, x_2)$
00	2
01	0
10	?
11	2
20	2
21	?

Table 5: Incomplete truth table for variable x_4 with wiring diagram $\{x_1^+, x_2^-\}$.

it follows that $h(2,1) = 2$. Similarly, since $2 = h(0,0) \leq h(1,0) \leq h(2,0) = 2$, it follows that $h(1,0) = 2$. In this way, we obtain the value of the missing entries. This process can be done for all wiring diagrams and for all variables.

(x_1, x_2)	$h(x_1, x_2)$
00	2
01	0
10	2
11	2
20	2
21	2

Table 6: Complete truth table for variable x_4 with wiring diagram $\{x_1^+, x_2^-\}$.

2.5 Stochastic Framework

For the simulations we will use the stochastic framework introduced in [11] referred to as Stochastic Discrete Dynamical Systems (SDDS). This framework is a natural extension of Boolean networks and is an appropriate setup to model the effect of intrinsic noise on network dynamics. Consider the discrete variables $x_1, \ldots, x_n$ that can take values in finite sets $S_1, \ldots, S_n$, respectively. Let $S = S_1 \times \cdots \times S_n$ be the Cartesian product. A *SDDS* in the variables $x_1, \ldots, x_n$ is a collection of n triplets

$$F = \{f_i, p_i^{\uparrow}, p_i^{\downarrow}\}_{i=1}^n$$

where

- $f_i : S \to S_i$ is the update function for x_i, for all $i = 1, \ldots, n$.
- $p_i^{\uparrow} \in [0,1]$ is the activation propensity.
- $p_i^{\downarrow} \in [0,1]$ is the degradation propensity.

The stochasticity originates from the propensity parameters $p_i^{\uparrow}$ and $p_i^{\downarrow}$, which should be interpreted as follows: If there would be an activation of x_k at

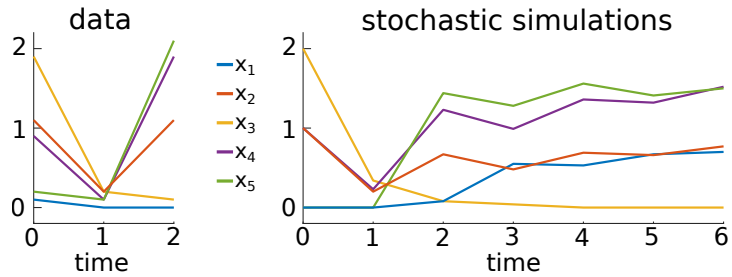


Figure 4: Comparison between data (only first time course shown) and stochastic simulations. Using the discretization of the initial condition of the data, 01210, we can use the model obtained to simulate the system for any artibtrary number of steps.

the next time step, i.e., if $s_1, s_2 \in S_k$ with $s_1 < s_2$ and $x_k(t) = s_1$, and $f_k(x_1(t), \ldots, x_n(t)) = s_2$, then $x_k(t+1) = s_2$ with probability $p_i^{\uparrow}$. The degradation probability $p_i^{\downarrow}$ is defined similarly. SDDS can be represented as a Markov chain by specifying its transition matrix in the following way. For each variable x_i, $i = 1, \ldots, n$, the probability of changing its value is given by

$$Prob(x_i \to f_i(x)) = \begin{cases} p_i^{\uparrow}, & \text{if } x_i < f_i(x), \\ p_i^{\downarrow}, & \text{if } x_i > f_i(x), \\ 1, & \text{if } x_i = f_i(x), \end{cases}$$

and the probability of maintaining its current value is given by

$$Prob(x_i \to x_i) = \begin{cases} 1 - p_i^{\uparrow}, & \text{if } x_i < f_i(x), \\ 1 - p_i^{\downarrow}, & \text{if } x_i > f_i(x), \\ 1, & \text{if } x_i = f_i(x). \end{cases}$$

Let $x, y \in S$. The transition from x to y is given by

$$a_{xy} = \prod_{i=1}^{n} Prob(x_i \to y_i). \tag{1}$$

Notice that $Prob(x_i \to y_i) = 0$ for all $y_i \notin \{x_i, f_i(x)\}$.

3 Applications

In this section we apply our method to a time-course, gene expression data that were collected during salamander (axolotls – Ambystoma mexicanum) tail regeneration. Modeling gene interactions can provide confirmatory and novel information for developing hypotheses about the actions of cell-signaling molecules and transcription factors that orchestrate tissue regeneration.

3.1 Gene expression data from experiments in axolotls

Using our method, we generated a Boolean network model for a set of 10 genes that were expressed differently during axolotl tail regeneration under control conditions [15]. Seven of the genes are ligands (AREG, FGF9, BMP2, INHBB, and WNT5A) or negative feedback regulators (DUSP6, NRADD) of cell signaling pathways, Sp7 is a bone-specific transcription factor, Hapln3 is a cell adhesion molecule and Phlda2 is an intracellular protein. We label these genes using the following variables:

$$\begin{array}{ll} x_1 = \mathit{AREG}, & x_2 = \mathit{PHLDA2}, \\ x_3 = \mathit{FGF9}, & x_4 = \mathit{BMP2}, \\ x_5 = \mathit{NRADD}, & x_6 = \mathit{HAPLN3}, \\ x_7 = \mathit{SP7}, & x_8 = \mathit{Wnt5\text{-}a}, \\ x_9 = \mathit{INHBB}, & x_{10} = \mathit{DUSP6}. \end{array} \tag{2}$$

In Figure 5 we show the wiring diagram obtained using our method. This network presents gene-by-gene interactions for the given gene expression data set. We note that the well-established inhibitory effect of Dusp6 on FGF signaling is captured by this network [9], and that the network implicates Inhbb as a key activator/integrator of BMP, WNT, and FGF signaling.

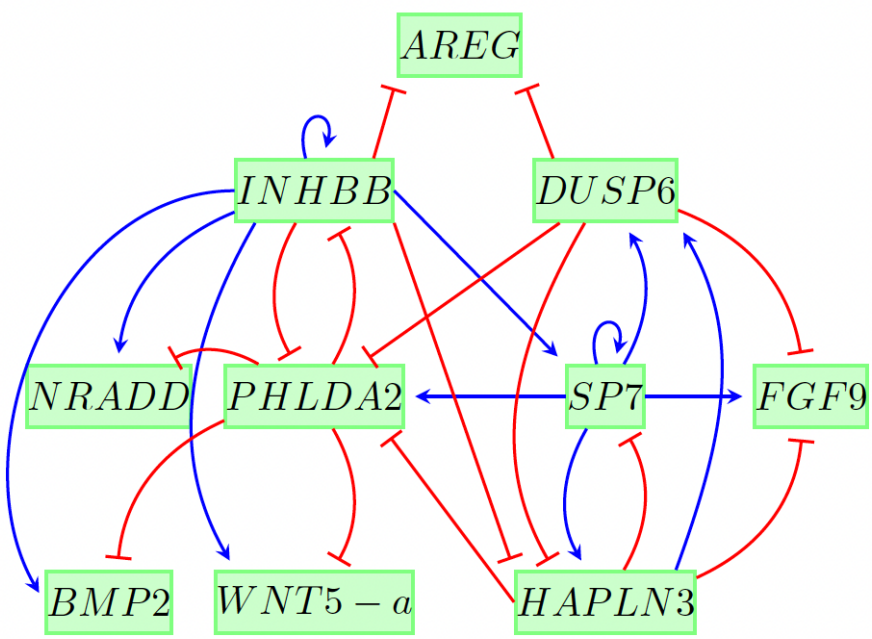


Figure 5: Wiring diagram for the genes in Equation 2. Blue edges represent activation while red edges inhibition.

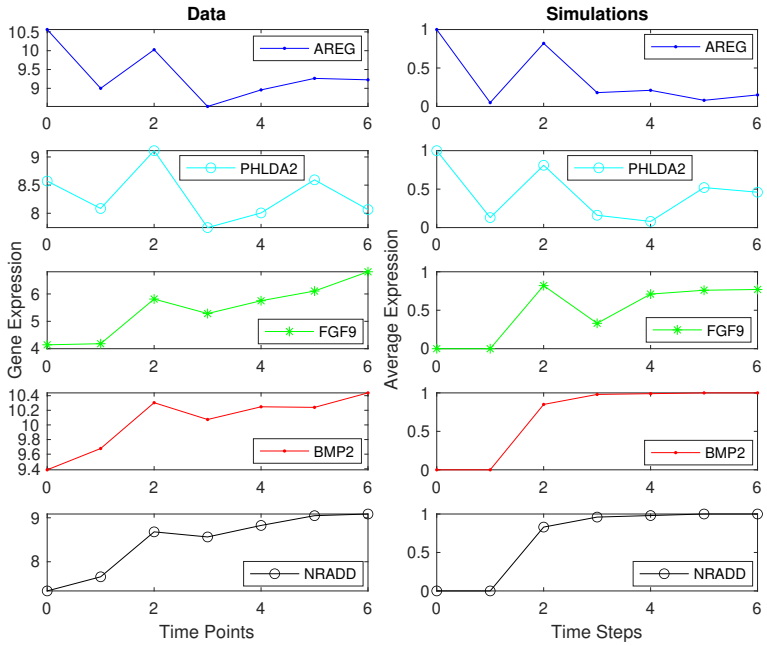


Figure 6: Data vs simulations of the first five genes in Equation 2. The plots in the left panel are experimental data while the ones in the right are simulations, 100 runs all initialized at 1100000001.

The Boolean network corresponding to the wiring diagram in Figure 5 is giving by a function $\mathbf{F} = (f_1, \ldots, f_{10}) : \{0,1\}^{10} \to \{0,1\}^{10}$ in the variables $x_1, \ldots, x_{10}$, where each coordinate function $f_i : \{0,1\}^{10} \to \{0,1\}$ represents how the future value of the i-th variable depends on the present values of the other variables. We provide the function in Appendix B as a collection of truth tables.

Simulations using the framework SDDS [11] was performed initializing the system at the initial state 1100000001. This initialization represents a discretized version of the actual data at time 0.

To validate this model we compare the experimental data versus the simulations that are shown in Figures 6-7. These figures were obtained from 100 runs.

From Figures 6-7, one can see that the simulated data captures the main patterns of the original data. This model can further be used to attractor analysis, control, modularity, etc.

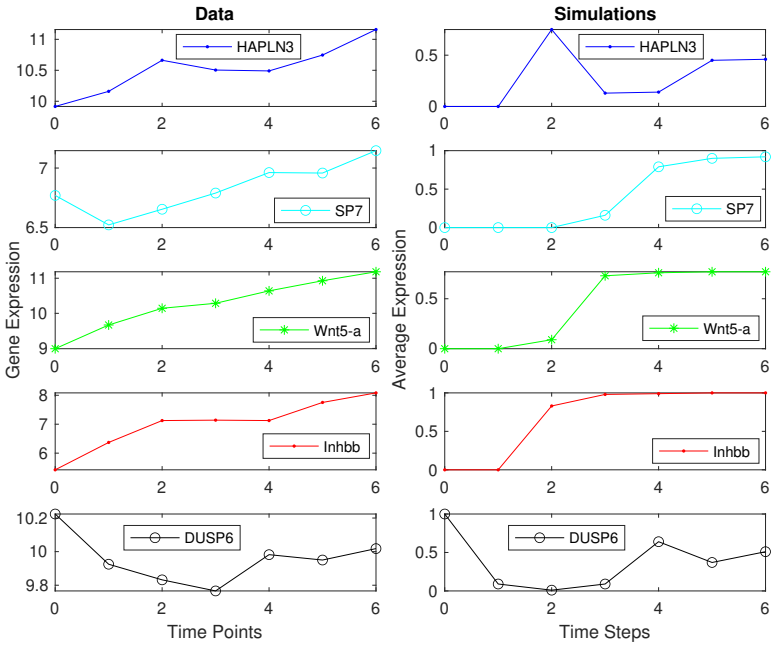


Figure 7: Data vs simulations of the last five genes in Equation 2. The plots in the left panel are experimental data while the ones in the right are simulations, 100 runs all initialized at 1100000001.

4 Discussion

Discrete models have been successfully to model biological systems [19, 20]. Although several discrete modeling packages exist for their analysis (e.g., Plantsimlab [4], Boolnet [12], BNReduction [18], Ginsim [13], casQ [1], WebMaBoSS [14]), they require an existing model or the wiring diagram to be created by the user. Few tools exist that provide an automated and easily customizable pipeline to quickly create model prototypes. Our toolbox allows the creation of model prototypes easily, which can then be used by existing modeling packages for validation, modification, or extension.

Equation learning methods in general require large amounts of data which might not be feasible in practice. Our method can be used even with a limited number of time points (e.g., 3 time points). Admittedly, in terms of recovery of a given model, the more data you provide to the algorithm the better [17].

We note that the expensive part of the method presented here is the network inference task which for our application with ten genes took only a few seconds but it might not be as efficient for larger networks (e.g., hundreds of genes).

For the purpose of reproducibility, we provide all the data and the code that

we use in our toy example and application which can be accessed through this link: `github.com/alanavc/prototype-model`.

References

[1] Sara Sadat Aghamiri, Vidisha Singh, Aurélien Naldi, Tomáš Helikar, Sylvain Soliman, and Anna Niarakis. Automated inference of Boolean models from molecular interaction maps using casq. *Bioinformatics*, 36(16):4473–4482, 2020.

[2] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

[3] Elena S Dimitrova, M Paola Vera Licona, John McGee, and Reinhard Laubenbacher. Discretization of time series data. *Journal of Computational Biology*, 17(6):853–868, 2010.

[4] S Ha, E. Dimitrova, D. Hoops, S.and Altarawy, M. Ansariola, D. Deb, J. Glazebrook, R. Hillmer, H. Shahin, F. Katagiri, J. McDowell, M. Megraw, J. Setubal, B. M. Tyler, and R. Laubenbacher. PlantSimLab - a modeling and simulation web tool for plant biologists. *BMC Bioinformatics*, 20(1):508, 2019.

[5] Franziska Hinkelmann and Abdul Salam Jarrah. Inferring biologically relevant models: nested canalyzing functions. *International Scholarly Research Notices*, 2012, 2012.

[6] Abdul Salam Jarrah, Reinhard Laubenbacher, Brandilyn Stigler, and Michael Stillman. Reverse-engineering of polynomial dynamical systems. *Advances in Applied Mathematics*, 39(4):477–489, 2007.

[7] John H Lagergren, John T Nardini, G Michael Lavigne, Erica M Rutter, and Kevin B Flores. Learning partial differential equations for biological transport models from noisy spatio-temporal data. *Proceedings of the Royal Society A*, 476(2234):20190800, 2020.

[8] Reinhard Laubenbacher and Brandilyn Stigler. A computational algebra approach to the reverse engineering of gene regulatory networks. *J Theor Biol*, 229(4):523–37, Aug 2004.

[9] Chaoying Li, Daryl A Scott, Ekaterina Hatch, Xiaoyan Tian, and Suzanne L Mansour. Dusp6 (mkp3) is a negative feedback regulator of fgf-stimulated erk signaling during mouse development. 2007.

[10] David Murrugarra and Reinhard Laubenbacher. The number of multistate nested canalyzing functions. *Physica D: Nonlinear Phenomena*, 241(10):929–938, 5 2012.

[11] David Murrugarra, Alan Veliz-Cuba, Boris Aguilar, Seda Arat, and Reinhard Laubenbacher. Modeling stochasticity and variability in gene regulatory networks. *EURASIP Journal on Bioinformatics and Systems Biology*, 2012(1):5, 2012.

[12] Christoph M ussel, Martin Hopfensitz, and Hans A Kestler. BoolNet - an R package for generation, reconstruction and analysis of Boolean networks. *Bioinformatics*, 26(10):1378–1380, 2010.

[13] A. Naldi, D. Berenguier, A. Fauré, F. Lopez, D. Thieffry, and C. Chaouiya. Logical modelling of regulatory networks with ginsim 2.3. *Biosystems*, 97(2):134–139, 2009.

[14] Vincent Noël, Marco Ruscone, Gautier Stoll, Eric Viara, Andrei Zinovyev, Emmanuel Barillot, and Laurence Calzone. Webmaboss: A web interface for simulating boolean models stochastically. *Frontiers in molecular biosciences*, 8, 2021.

[15] Larissa V Ponomareva, Antony Athippozhy, Jon S Thorson, and S Randal Voss. Using ambystoma mexicanum (mexican axolotl) embryos, chemical genetics, and microarray analysis to identify signaling pathways associated with tissue regeneration. *Comparative Biochemistry and Physiology Part C: Toxicology & Pharmacology*, 178:128–135, 2015.

[16] Brandy Stigler, Abdul Jarrah, Michael Stillman, and Reinhard Laubenbacher. Reverse engineering of dynamic networks. *Annals of the New York Academy of Sciences*, 1115(1):168–177, 2007.

[17] A. Veliz-Cuba. An algebraic approach to reverse engineering finite dynamical systems arising from biology. *SIAM Journal on Applied Dynamical Systems*, 11(1):31–48, 2012.

[18] Alan Veliz-Cuba, Boris Aguilar, Franziska Hinkelmann, and Reinhard Laubenbacher. Steady state analysis of boolean molecular network models via model reduction and computational algebra. *BMC Bioinformatics*, 15:221, 2014.

[19] Alan Veliz-Cuba and Brandilyn Stigler. Boolean models can explain bistability in the *lac* operon. *Journal of Computational Biology*, 18(6):783–794, 2011.

[20] David J Wooten, Jorge Gómez Tejeda Zañudo, David Murrugarra, Austin M Perry, Anna Dongari-Bagtzoglou, Reinhard Laubenbacher, Clarissa J Nobile, and Réka Albert. Mathematical modeling of the candida albicans yeast to hyphal transition reveals novel control strategies. *PLoS computational biology*, 17(3):e1008690, 2021.

A Code Usage

Here we show how the Matlab code is used. All code files and examples can be found at Github `github.com/alanavc/prototype-model`. The toy example and application are included.

The input files must be in the form of trajectories (one line per state of the system). Different trajectories must be in different files and all trajectories files must have the same prefix in the name. For example, `timecourse1.txt`, `timecourse2.txt`, `timecourse3.txt`, `timecourse4.txt`, etc. For our example in Methods, the content of timecourse1.txt would be the following.

```
0.1,1.1,1.9,0.9,0.2
0.0,0.2,0.2,0.1,0.1
0.0,1.1,0.1,1.9,2.1
```

Once we are in the folder `codes`, we can discretize the data. Prior to this we need to specify the prefix for the time course files and the file where the data will be saved.

```
> clear all
> time_course_prefix='timecourse';
> num_levels=[3,2,3,3,3];
> file_for_disc_data='data0.mat';
> % discretize data
> addpath('discretize_data')
> discretize(time_course_prefix,num_levels,file_for_disc_data)
```

Then, we generate all minimal wiring diagrams using the Matlab function `generate_wiring_diagrams`. Prior to this, we need to specify the file where we will save all wiring diagrams.

```
> %create wiring diagrams
> file_for_wiring_diagrams='WD0.mat';
> addpath('create_WD')
> generate_wiring_diagrams(file_for_disc_data,file_for_wiring_diagrams)
```

To select the best wiring diagram we use the command `select_best_wiring_diagram` as follows.

```
> %select wiring diagram
> addpath('select_WD')
> W=select_best_wiring_diagram(file_for_wiring_diagrams);
```

The variable `W` is a matrix that contains the best wiring diagram that fits the data. The user can modify this to account for prior knowledge or to try any other of the wiring diagrams (found in file `WD0.mat`) for exploration.

We use the function `generate_monotone_functions` to create the model that fits the data and has the wiring diagram given. The model will be saved as a truth table for each variable.

```
> %create model
> file_for_model='model0.mat';
> addpath('create_Model')
> generate_monotone_functions(time_course_prefix,W,file_for_model)
```

To simulate the model we use the following code.

```
> %simulate model
> addpath('simulate_model')
> init_state = [0 1 2 1 0];
> num_simulations = 100;
> num_steps = 6;
> num_vars = 5;
> % Propensity matrix
> propensity_matrix = 0.8*ones(2,num_vars);
> mean_trajectories = simulate(file_for_model,propensity_matrix,init_state,num_steps,num_sim
```

The mean trajectories are saved in `mean_trajectories` and can be plotted using standard Matlab commands.

B Inferred Model

Here we describe the inferred model for the data in the Results section. We used $n = 10$ genes and the number of input variables in each Boolean function is given by the following vector:

$$nv = [2, 4, 3, 2, 2, 3, 3, 2, 2, 2].$$

The maximum number of inputs is $m = 4$. In Table 7 we list the input variables for each gene. The table is a m-by-n matrix usually referred as $varF$. Since the number of variables may vary between different functions, only the first $nv(i)$ elements are relevant in the ith column of $varF$. The remaining spots in each column are set to -1 so that the whole table can be encoded as a matrix.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
9	6	6	2	2	7	6	2	2	6
10	7	7	9	9	9	7	9	9	7
-1	9	10	-1	-1	10	9	-1	-1	-1
-1	10	-1	-1	-1	-1	-1	-1	-1	-1

Table 7: Input variables for each gene in Table 8.

In Table 8 we put the truth table in compact form usually referred as F. This table has size 2^m-by-n. Since the length of the truth tables may vary between different functions, only the first $2^{nv(i)}$ bits are relevant in the ith column of F. The remaining spots in each column are set to -1 so that the whole table

can be encoded as a matrix. The entries of the columns of F are ordered in lexicographic order is the binary input arrays.

The propensities that we used for the simulations are all equal to 0.9. That is, $p_i^{\uparrow} = p_i^{\downarrow} = 0.9$ for all $i = 1, \ldots, 10$.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
1	1	1	1	1	1	0	1	1	0
0	0	0	1	1	0	1	1	1	0
0	0	1	0	0	0	0	0	0	0
0	0	0	1	1	0	1	1	1	1
-1	1	0	-1	-1	1	0	-1	-1	-1
-1	0	0	-1	-1	0	0	-1	-1	-1
-1	1	1	-1	-1	1	0	-1	-1	-1
-1	0	0	-1	-1	0	1	-1	-1	-1
-1	0	-1	-1	-1	-1	-1	-1	-1	-1
-1	0	-1	-1	-1	-1	-1	-1	-1	-1
-1	0	-1	-1	-1	-1	-1	-1	-1	-1
-1	0	-1	-1	-1	-1	-1	-1	-1	-1
-1	0	-1	-1	-1	-1	-1	-1	-1	-1
-1	0	-1	-1	-1	-1	-1	-1	-1	-1
-1	0	-1	-1	-1	-1	-1	-1	-1	-1
-1	0	-1	-1	-1	-1	-1	-1	-1	-1

Table 8: Truth tables (in compact form) for the inferred functions for the data in the Results section.