A METHODOLOGY FOR PARAMETRIZING DISCRETE MODELS OF BIOLOGICAL NETWORKS

by

Nathan Aodren Maugan Renaudie

B.S in Electrical Engineering, ENSEA (Cergy, FRANCE) 2015

Submitted to the Graduate Faculty of

Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2018

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This thesis was presented

by

Nathan Renaudie

It was defended on

April 4, 2018

and approved by

Natasa Miskov-Zivanov, PhD., Assistant Professor, Electrical and Computer Engineering

Murat Akcakaya, PhD., Assistant Professor, Electrical and Computer Engineering

Samuel Dickerson, PhD., Assistant Professor, Electrical and Computer Engineering

Cheryl Telmer, PhD., Research Biologist, Carnegie Mellon University

Thesis Advisor: Dr. Natasa Miskov-Zivanov, PhD., Assistant Professor, Electrical and Computer Engineering

A METHODOLOGY FOR PARAMETRIZING DISCRETE MODELS OF BIOLOGICAL NETWORKS

Nathan Renaudie, M.S.

University of Pittsburgh, 2018

Humans have always had the desire to understand the world that surrounds us. With the progress of science in the last decades, our knowledge has drastically increased, following the fast pace at which scientists obtain and publish new results. With this rapid increase in the volume of available information about the systems that scientists study, modeling has become crucial in the process of learning and understanding these systems. In biology, models can be developed to capture different system scales. Here, we are focusing on modeling cellular signaling networks, using a discrete modeling approach, where system components are represented as discrete variables, and their regulatory functions are approximated with logical, weighted sum, or min-max functions. We study cellular signaling networks through stochastic simulation, in which model switches from one state to another according to elements' regulatory functions. To start simulations, we define scenarios of interest (e.g., cell activation to induce fate change, or a drug added to a cancer cell). These scenarios are implemented through parametrizing the model, that is, assigning initial values to all model elements, as well as defining patterns at model inputs, or perturbations inside the model. However, the information about the initial state is most often incomplete, as experts are familiar with values in some parts of the network, but not in the whole network. We have developed several methods to initialize model elements when the knowledge about the initial values for a particular scenario is sparse. Next, we applied our initialization methods on several cancer cell models. Our results show that varying the initial values can significantly influence model behavior, and therefore, emphasize the importance of choosing a suitable initialization method. We expect that our methods and the conclusions from our studies will enable more accurate setup of future modeling experiments.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

I would like to first thanks Thomas Tang and Carine Sabouraud of the international relations team at ENSEA and Dr Mahmoud El Nokali for giving me the opportunity to study abroad at the University of Pittsburgh.

I also want to give a special thanks to my advisor Dr Natasa Miskov-Zivanov for her trust and for giving me a space as a graduate researcher in her lab. And not to forget our collaborator Cheryl Telmer that have taken time to explain her biological understanding of our models.

Then, I would like to thank every one of the MeLoDy lab: Adam Butchy, Khaled Sayed, Kara Bocan, Mickael Vimbert, Vianney Mixtur, Emily Holtzapple, Gavin Zhou, Yasmine Ahmed and Handa Ding, that have taken time to answer any problem that I faced and making of our lab a friendly place to work.

Last, but not least, I would like to thank my fellow French Valentin Paquin, Cedrine Rebrion, Blandine Russo, Lou Botherel, Lucie Broyde and Sebastien Oliver that have made those two years in the states a great experience.

1.0 INTRODUCTION

With the progress of science inside the medical field in the last decades, scientists have been able to understand better and better how biological systems work, from macroscopic level to cellular behavior. In the meantime, diseases have become more complex and harder to be treated [1]. To face these new challenges, such as cancer, scientists need powerful tools to accurately model biological systems at the cellular level.

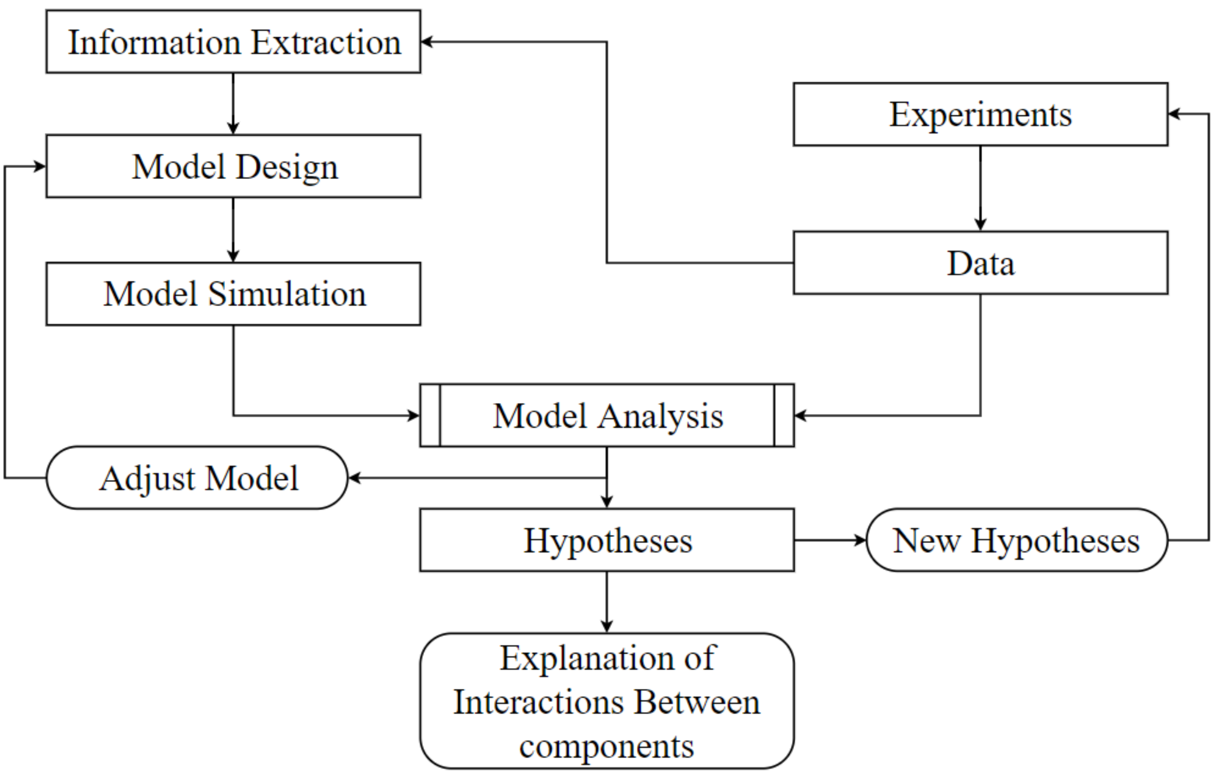


Figure 1. Diagram of the modeling approach

The modeling of biological networks aims to explain the relationships between cellular components. The models are designed based on the data collected from experiments, and according to the knowledge of experts such as biologists, chemists or physicists. A number of approaches have been used to model those biological networks including Monte-Carlo method [2], ordinary differential equations (ODEs) [3], reaction ruled-based models [4], and finally Boolean networks or logical models, which have been shown to provide both efficient and accurate analysis [5][6].

Indeed, almost all living organisms, from animals to plants, share the same basic structural biological unit: the cell. The eukaryotic cells feature membrane-bound organelles such as mitochondria, endoplasmic reticulum, and especially the nucleus that contains the genetic material (DNA) within the nuclear envelope [7]. The logical modeling approach is efficient as it provides an accurate model using a relatively simple representation: any organelle or a molecule in the cell can be represented with a Boolean variable, which has only two possible states. For example, one can represent a gene using a Boolean variable with two states, *ON* and *OFF*:

- *ON*: the gene is activated and will lead to synthesis of the corresponding mRNA and protein
- *OFF*: the gene is inhibited, it is inactive, and no proteins are synthesized.

By extending from Boolean to discrete variables that can have several discrete values, it is possible to incorporate more details in the model. Instead of modeling only active and inactive state, we can represent multiple activity levels of the system's components (e.g. we could model three levels *ZERO*, *LOW* and *HIGH*, as 0, 1, and 2, respectively).

Then, as we have defined our model elements using discrete variables, we also need rules to update their values, and thus, simulate a behavior of elements over time. In logical modeling approach, we create those rules using logic operators such as OR, AND, or NOT, based on the interconnections between elements in the model. Next, we need an update scheme for those rules: we can choose to update elements at the same time (simultaneously) or one after another (sequentially). Critical for the simulation outcomes is the choice of an initial state that the model is assigned before the simulation starts.

Without knowing the initial state of the system, and the conditions under which the simulation is initiated, the set of elements and their update rules is not sufficient to analyze the system's behavior. However, it is often the case that for many of the elements that are included in the model, initial values are neither available in experimental results, nor experts are familiar with these values. For this reason, the initialization of discrete models is an important concern that we want to tackle here.

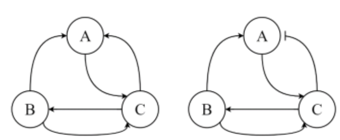
2.0 BACKGROUND

In this section, we describe the discrete model in a general way such as how we represent them and how a model can be simulated. Then, we go over the process of model extension, in order to improve model behavior and describe the importance of model initialization in such cases.

2.1 DISCRETE MODELING APPROACH

The Boolean network concept was first introduced by Kauffman in [8], and later developed by others such as [9][10][11]. Similar to our example of a gene above, which can be ON or OFF, in general, every *element* of the logical (Boolean network) model is assumed to have two possible *states*: TRUE (ON) / FALSE (OFF).

Besides having a state attribute, each element may also be connected to other elements of the model, and have a logical function assigned to it, which determines changes in its states, according to the states of other elements.



(a) (b)

Figure 2. Example of a simple Boolean network. (a) without polarity indicated in graph edges and (b) with polarity indicated with different type of edges (regular arrow represents positive regulations, and blunt arrow represents negative regulation).

The graph in Figure 2(a) is an example of a typical Boolean network that constitutes of three nodes, A, B, and C, which represent model elements, such as proteins, genes, other molecule kinds, or components of a cell. The arrows connecting the nodes in Figure 2(a) represent only the regulatory connections between the elements of the model. In logical models, we also use two opposite polarities of interactions:

- Positive: in Figure 2(b), A can be assumed to be upregulated by B (B $\rightarrow$ A).
- Negative: in Figure 2(b), A can be assumed to be downregulated by C (C —| A).

The graph representation of a model shown in Figure 2(b) still includes only the directionality and polarity of interactions, but it does not include any information about how the regulators are combined to affect the value of the regulated element. This function that combines together all the regulators of an element, is often referred to as element's *update function* or regulatory function [12]. The update function determines the behavior of an element, and is based only on the previous state of its regulators. For any node x_i of a model of N elements, its regulatory function f_i can be written as:

$$x_i^* = f_i(x_1, x_2, \dots, x_N) \quad (1)$$

where x_i^* denotes the value of the node x_i at the next step, and x_1, x_2, …, x_N are the elements of the model at the previous step before updating x_i.

Using the previous example, we can describe our model including the regulatory functions for the three elements. Together, these functions form the update rules of the model:

$$\begin{cases} A^* = f_A(B, C) = B\ AND\ NOT\ C = B \cdot \bar{C} \\ B^* = f_B(C) = C \\ C^* = f_C(A, B) = A\ OR\ B = A + B \end{cases} \quad (2)$$

These rules allow us to compute the next state of the system, and thus, simulate the behavior of our model over time.

This approach can be generalized into discrete modeling approach where the variables inside the model are still using discrete values but can be extended to give more precision about the level of activity. In this work, we will most of the time use three state levels that represent the *ZERO*, *LOW* and *HIGH* activation states of the variable.

2.2 SIMULATION METHODS

The simulations in this work rely on the DiSH simulator described in [14], that have been shown to accurately simulate biological models, some examples include T-cell differentiation, pancreatic cancer cells, and melanoma cell lines. This simulator supports several different simulation schemes, and the following sub-sections detail the two schemes that have been used in this work.

2.2.1 Simultaneous update scheme

In the simultaneous update scheme, all elements of the model are updated at the same time, that is, all the rules are executed concurrently by the simulator. Thus, this simulation scheme is deterministic, as the same model state will always lead to the same next state. To represent the evolution of the model from one state to another, we can use a state transition graph (STG) that includes all the possible state transitions following element update rules, and the deterministic update scheme. We use the model in Figure 3 as a simple example of how the simultaneous scheme works.

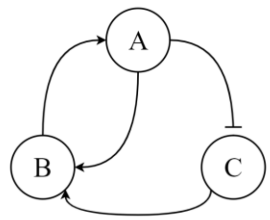


Figure 3. Graph representation of a simple model with three elements.

First, from the graph representation we can create the logical rules for every node of the model:

$$\begin{cases} A^* = B \\ B^* = A + C \\ C^* = \bar{A} \end{cases} \quad (3)$$

Then, from these logical functions, we can compute the transition table of the system by computing the new state (A^*, B^*, C^*), based on the current state (A, B, C):

Table 1. Transition table of the model with simultaneous update scheme.

Current state			**Next state**		
A	**B**	**C**	**A***	**B***	**C***
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	1	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	1	1	0

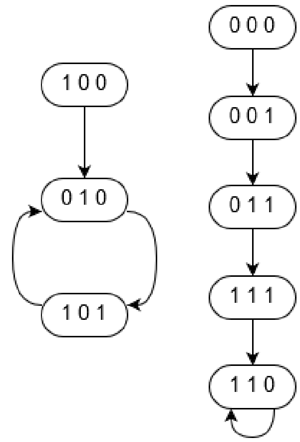


Figure 4. State Transition Graph of the model for simultaneous update

Now as we have all possible transitions for every possible state of the model, we can represent its evolution with a state transition graph using the vector of elements (A, B, C) and the previous table, as illustrated in Figure 4. For this simple model, we can see that there are two possible outcomes, depending on the initial conditions:

- A cycle between 010 and 101 if the initial state is either 100, 010 or 101.
- A steady state value 110 for any other initial state of the system.

2.2.2 Sequential update scheme

In the sequential update scheme, only one node is updated at each time step. There are different ways to choose this updated element. First, we will show how the sequential update scheme changes the behavior of the example model. In Table 2, we list the state transition now taking into account which element is updated in the current state.

Table 2. Transition table of the model with sequential update scheme.

CURRENT STATE			ELEMENT UPDATE								
A	B	C	A			B			C		
0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	1	0	1	1	0	0	1
0	1	0	1	1	0	0	0	0	0	1	1
0	1	1	1	1	1	0	1	1	0	1	1
1	0	0	0	0	0	1	1	0	1	0	0
1	0	1	0	0	1	1	1	1	1	0	0
1	1	0	1	1	0	1	1	0	1	1	0
1	1	1	1	1	1	1	1	1	1	1	0

As we have done before, using this transition table, we can draw the STG for our example model, which is different in the sequential case, compared to the one in Figure 4. The difference in the behavior stems from the fact that elements can be updated in a different order in the sequential case.

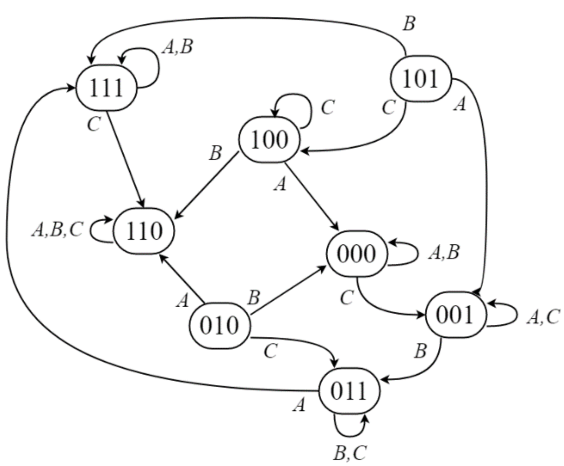


Figure 5. State Transition Graph of the model when the sequential update scheme is used.

With this update scheme, we see that our model now has a unique steady state (110). Every starting point will lead to that steady state, with more or less time, depending on the order of the model update. While this steady state is the same as in the case of the simultaneous update scheme, the trajectory to reach the steady state can be different, depending on the order of updating model elements. Therefore, besides the STGs, we are interested in studying individual model elements, as their behavior can vary on different trajectories to the same steady state.

As shown in Table 2 and in Figure 5, the order in which sequential updates occur can significantly affect element trajectories from initial to steady state. Here, we focus on a random-order update scheme, meaning that we will select at every simulation step a random element to update. With this update scheme, we can update the same element twice or more in consecutive simulation steps. By using this method, every simulation run may have different results, even with the same initial values, as the order of the element updates is different from one run to another.

By running multiple simulations starting from the same initial state, we can output the simulation results as continuous plots by averaging across different runs. For this toy example, we use a simulation length of 30 steps and compute average value of each model element at each simulation step across 100 simulation runs. We observe here the same final value as shown in the STG (110), however, we also get in this case the trajectory of each element from the initial to the final state.

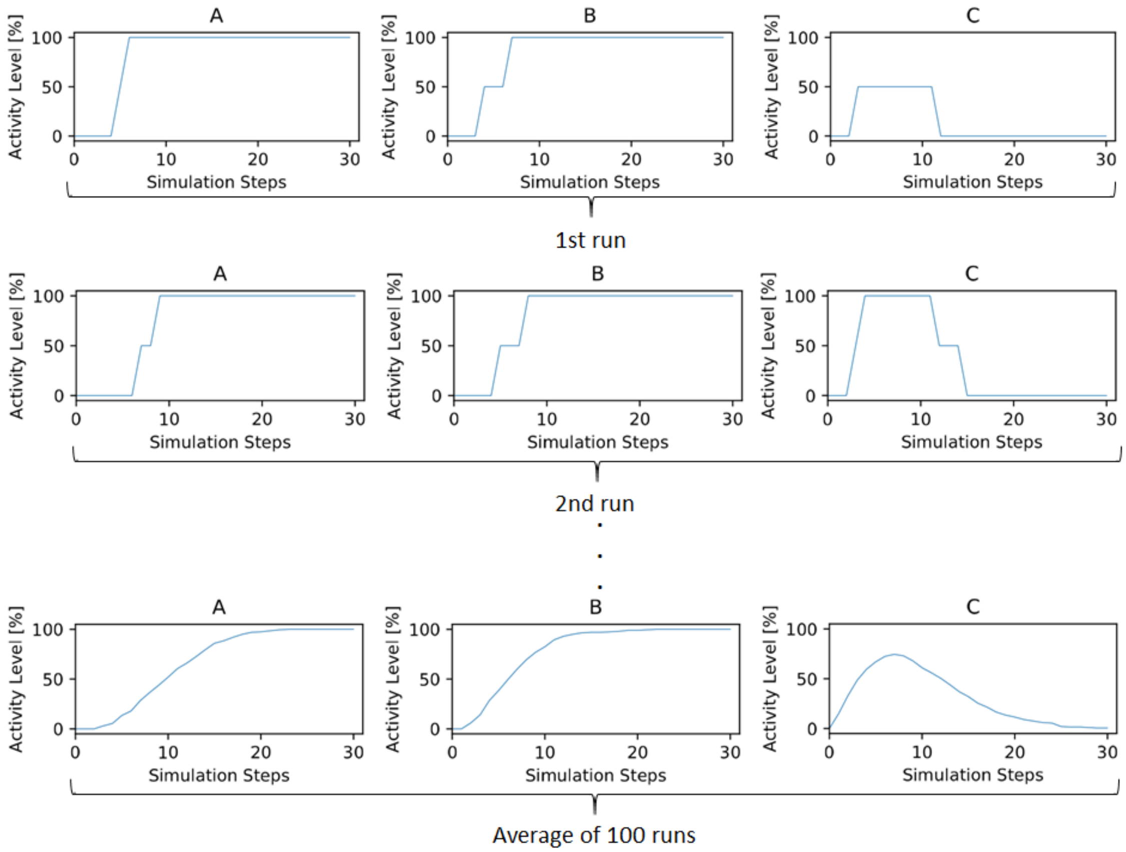


Figure 6. Simulation results for the random-order sequential update scheme for the toy model shown in Figure 3.

The random-order sequential update scheme is very suitable for simulating biological systems, as reactions in a population of cells within one organism do not all occur at the exact same time. This stochastic approach takes into account the diversity of the cells, and the average of the results can be seen as an average on population of cells: each simulation run computes one trajectory for each model element, and this can be interpreted as element behavior in a single cell. Thus, by averaging across results from 100 simulation runs, we get the general trend of elements A, B, and C in our example, in a population of 100 cells.

2.3 MODEL EXTENSION

Besides being necessary to simulate the behavior of models, initialization has also its importance in other applications such as model extension, when new information about element interactions is available. To emphasize the critical role of proper initialization in the case of model extension, we outline in Figure 7 the model extension procedure, and provide the details of the procedure following the schematic.

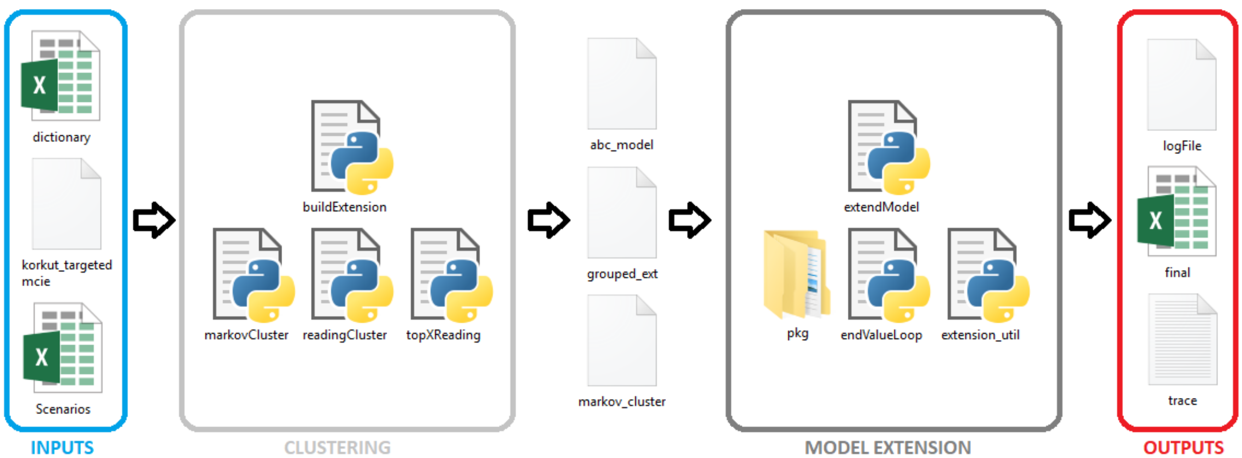


Figure 7. Schematic of the model extension process.

The inputs to model extension are the baseline model that is to be extended (called “dictionary” in Figure 7), the list of new candidate interactions that can be added to the model (“korkut_targeted_mcie” file in Figure 7), and the list of scenarios under which the model is studied (“Scenarios” in Figure 7). New element interactions can be obtained from reading the literature, from experts, or from data. After receiving inputs, the model extension process is divided into two steps. The first step, the extension clustering part, is designed to create clusters of new element interactions. The second step, the actual model extension part, consists of evaluating the performance of the model when the previously obtained clusters are added to the model. When new elements and interactions are added to the model, we need to check how this modification affects the model behavior. If we get closer to the expected behavior, then we can add the elements to the baseline model. If the extension cluster didn’t improve the model, we test other clusters until we find the set of clusters that leads to the largest improvement in model behavior.

The values of new elements that belong to the set of extensions, and that are eventually added to the model are often not known. The choice of values for these elements is not straightforward. In order to compute the trajectories for the updated model, and to accurately evaluate whether the model behavior has improved, it is important to properly determine initial values for these new elements.

3.0 METHODOLOGY

The aim of the initialization procedure is to determine initial states of elements in a model for which these values are not known or not available, such that all model elements have defined state before the start of simulation. Depending on the overall system state that is to be represented by the model, there are various ways to define element initial values. This thesis presents several approaches that utilize existing knowledge when initializing model elements. The methods are presented from the simpler one that does not use any knowledge from the model or from literature, to a mix of both for most optimal use of available information.

3.1 RANDOM INITIALIZATION

The simplest approach to initialize logical models is to perform a random initialization of all the elements in the model. There are different ways to assign random values, based on the kind of distribution used.

This method provides various initial states for the same model, as each value is randomly selected. The number of possible sets of initial values, $N_{initial\ states}$, for a model of N elements is given by:

$$N_{initial\ states} = (n_{states})^N \tag{4.1}$$

where n_{states} represents the number of states allowed for the elements inside the model. For example, in the case of the three-state model described in Section 2.1, the equation (4.1) becomes:

$$N_{initial\ states} = 3^N \tag{4.2}$$

Table 3 lists the number of possible initial states for several model sizes. As we can see, the number of possible initial states increases exponentially with the number of elements in the model.

Table 3. Evolution of possible initial set with the size of the model

Number of elements (N)	Number of possible initial states
1	3
5	243
10	59046
100	5.15×10^{47}

In the following, we describe two types of random initialization approaches. In the first one, we assume that all states are equally likely to occur, and in the second one, we assume that some states have higher probability than the other ones.

3.1.1 Uniform Random Initialization

In the case of uniform random initialization, each value that can be assigned to an element has the same probability. For example, in a logical model this means that the probability for each *ON* and *OFF* state is 50%.

It is simple to extend this method to the discrete modeling approach where elements can have more than two states. In this case, the randomization must be done with more than two values. In a discrete model, if a number of states for an element is denoted as n_{states}, the probability for each value is:

$$\begin{cases} P(X=0) = \dfrac{1}{n_{states}} \\ \vdots \\ P(X = n_{states} - 1) = \dfrac{1}{n_{states}} \end{cases} \tag{5}$$

3.1.2 Non-Uniform Random Initialization

The non-uniform distribution is used when we want to assign to each value of the model a specific probability. This is relevant for a model with more than two states, for example, we may want to assign to a non-activate state (e.g., *ZERO*) the same probability as for all the other active states together (e.g., *LOW* and *HIGH* in a three-level model). In that case, the probability for each value is given by:

$$\begin{cases} P(X=0) = \dfrac{1}{2} \\ P(X=1) = \dfrac{1}{2 \times (n_{states} - 1)} \\ \vdots \\ P(X = n_{states} - 1) = \dfrac{1}{2 \times (n_{states} - 1)} \end{cases} \tag{6}$$

To illustrate the difference between a uniform and a non-uniform random initialization, in Figure 8 we show the probabilities of the three element values in the case of our example model from Section 2.1 where elements have three states.

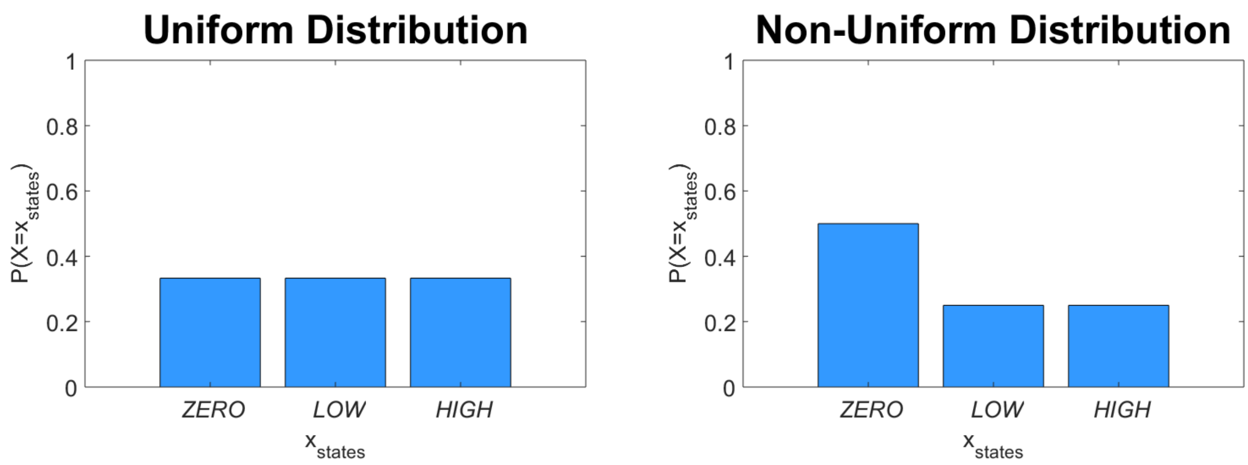


Figure 8. Representation of two different state distributions for a three states model (*ZERO*, *LOW* and *HIGH*).

3.2 USING EXTERNAL KNOWLEDGE AND DATA

Although the random initialization is a straightforward approach to testing many different initial states of the system, it also has a downside: it does not take into account the element relationships inside the model itself. Therefore, many of the initial states that are defined using this approach are not biologically meaningful, and they unnecessarily extend the time needed to reach steady state, or they cause unrealistic oscillations.

The second method that we have used in our work does not create the initial model state randomly, instead, it uses information from literature, expert knowledge, or available data to find all known initial values for model elements.

The first step when using the knowledge or data-based method is to match the names of elements that are in our model with the names of elements in databases or in literature. This step is needed as proteins usually have multiple synonyms, and the names may not exactly match between the model and the knowledge sources.

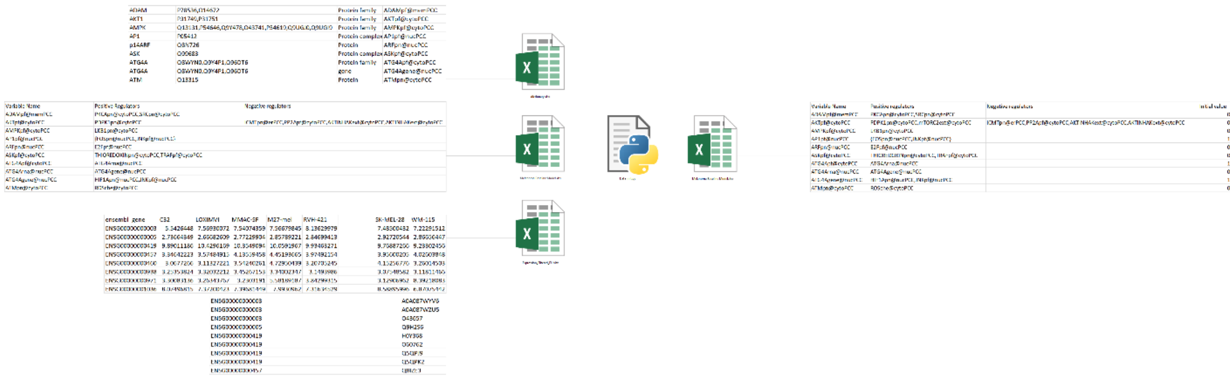


Figure 9. Schematic of the process of data extraction

As describe previously, we are using in this work models with discrete element values, and thus, for each model element we determine the smallest number of integer values needed to represent all levels of its activity. On the other hand, the raw data is usually obtained using measurement techniques that consider larger number intervals and have higher precision. Therefore, we need to develop a translation between our discrete variable levels and the raw data intervals. We convert the available data values from these external sources into the discrete values of corresponding model elements using one of the two methods described in the following sub-sections. Specifically, we will demonstrate our methods using the three-level discrete example.

3.2.1 Adaptive Threshold

In the Adaptive Threshold approach, we use the following steps to discretize the available data into the number of desired states, for example three levels (*ZERO*, *LOW*, and *HIGH*):

1. Find the minimal and maximal value inside the whole data in order to determine the range of the values we want to discretize.
2. Split this interval in as many equal parts as required to represent the desired number of states; for example, in the three-level model, we split this interval into three equal parts.
3. Map the available data to these intervals, to determine the discrete values that will be used in simulations.

This process can be summarized in the following algorithm:

Algorithm Adaptive Threshold
$MIN \leftarrow minimal\ value\ of\ the\ data$
$MAX \leftarrow maximal\ value\ of\ the\ data$
$RANGE \leftarrow MAX - MIN$
$x_{data} \leftarrow value\ of\ element\ x\ inside\ the\ data$
$n_{states} \leftarrow number\ of\ states\ of\ a\ variable\ from\ the\ model$
for $(i = 1\ to\ n_{states})$
  if $\left(x_{data} \leq MIN + \frac{i}{n_{states}} RANGE\right)$
    $x_{discrete} = i - 1$
  end if
end for

Figure 10. Adaptive Threshold mapping algorithm.

3.2.2 Quantile Threshold

While the first method assumes that the *range* of all the available values is divided into equal intervals, this method relies on the *number* of available data values when creating intervals, that is, it assumes that each interval has the same number of data values. In the three-level case, this means that the data will be split into three intervals (terciles) such that 1/3 of the values will be mapped to *ZERO* level in our discrete model, 1/3 of the values will be mapped to the *LOW* level in the model, and the last 1/3 of the values will be mapped to the *HIGH* level.

Figure 11. Comparison between the adaptive threshold method (left) and the quantile threshold method (right).

3.3 FINDING MISSING VALUES

Most often the information that is available is not sufficient to assign initial values to all model elements, and therefore a method to complete the initialization is needed. In the following sub-sections, we provide details of several methods that we have developed to derive meaningful and biologically relevant initial states.

This last way to initialize the Boolean network is based on the relationship between the elements of the model. For this method, we will need to introduce a notion from the graph theory [13] as we can represent our model as a directed graph. Also, we assume in this part that we need to find only some missing initial values and not to create a full initial state for the model.

For the future, we will use the term regulator to describe an element that has an effect on the node we are currently looking at. In our modeling approach, we assume two types of regulators:

- Activator: an element that has a positive effect on another element.
- Inhibitor: an element that has a negative effect on another element.

Using the example network from Figure 2, we can say that C is an activator of B, but also that C is an inhibitor of A.

To establish a formula to compute a node's initial value, we will use the following general representation of a node and its regulators, also outlined in Figure 12.

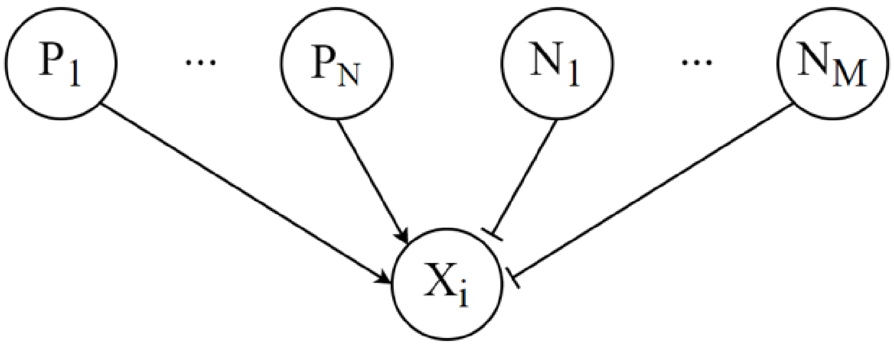


Figure 12. Representation of a node and its regulators.

We assume here that the initial values of N activators (P_1, P_2, …, P_N) of element X_i, and the initial values of M inhibitors (N_1, N_2, …, N_M) X_i are known, and that we only need to compute the initial value of element X_i.

3.3.1 Maximizing the freedom of the node

The first approach to assigning an initial value to element X_i is to allow for the largest change in the value of X_i during simulation, given the known values of its regulators. To establish a formula for the initial value of X_i, we will study three cases:

1. There is only positive regulation affecting our node X_i. This is possible either when the node doesn't have any inhibitors (M = 0) or only the activators have an initial value different from *ZERO* (so all inhibitors are inactive $\sum_1^M N_i[0] = 0$). In this case, we know that the node is

going to be up-regulated during the simulation, and thus, its value will rise to either a *LOW* or *HIGH* activation state. To allow change in the element state, we initialize it at a *ZERO* (no activation) state.

2. The second case is similar to the first one but for the negative regulation. In this case, there is negative regulation of element X_i since it does not have activators (N = 0), or because only its inhibitors have initial values different from *ZERO* (so all activators are inactive $\sum_1^N P_i[0] = 0$). Then, we know that this element value is going to decrease during the simulation. Again, to allow for larger change, we initialize it at a *HIGH* state.
3. In the last case, we have no way to predict the trend of our element: when both positive and negative regulation are affecting a node. Without more knowledge, we want to give our variable the most freedom possible: the initialization at a LOW state allows it to rise or decrease following the change in its regulators' activation.

To summarize all the above cases, we use the following formula to compute the initial value for any model element X_i (at time point t=0):

$$X_i[0] = 1 - f_{positive} + f_{negative} \tag{7}$$

where $f_{positive}$ is a Boolean function equal to 1 when at least one activator has an initial state different from *ZERO* and $f_{negative}$ is a Boolean function equal to 1 when at least one inhibitor has an initial state different from *ZERO*:

$f_{positive} = f\,(P_1, P_2, \ldots, P_N) = (P_1 > ZERO) + (P_2 > ZERO) + \ldots + (P_N > ZERO))$

$f_{negative} = f\,(N_1, N_2, \ldots, N_M) = (N_1 > ZERO) + (N_2 > ZERO) + \ldots + (N_M > ZERO))$

where "+" represents logical OR, as described in Section 2.1.

This approach to compute the missing value of a node can be describe using the following algorithm outlined in **Figure 13**.

Algorithm Initialization maximizing the freedom of the node

$P_i[0] \leftarrow \mathit{Initial\ value(s)\ of\ the\ positive\ regulator(s)}$

$N_i[0] \leftarrow \mathit{Initial\ value(s) of\ the\ negative\ regulator(s)}$

if $\sum_1^M P_i[0] = 0:$

$\quad\quad f_{positive} \leftarrow 0$

else $\quad f_{positive} \leftarrow 1$

end if

if $\sum_1^M N_i[0] = 0:$

$\quad\quad f_{negative} \leftarrow 0$

else $\quad f_{negative} \leftarrow 1$

end if

$X_{initial\ state} \leftarrow 1 - f_{Positive} + f_{Negative}$

Figure 13. Computing of the missing values using the maximization freedom of the node function.

3.3.2 Minimizing the freedom of the node

The second approach we take is to assign the missing initial value of a node to minimize the possible change of node X_i during simulations. By having the opposite reasoning as before, we can derive an equation to compute initial value for X_i using Boolean functions $f_{positive}$ and $f_{negative}$:

$$X_i[0] = 1 + f_{positive} - f_{negative} \tag{8}$$

Using equations (7) or (8), we are able to compute the initial value for any model element X_i by only knowing the initial states of its regulators. In Table 4, we illustrate the two initialization approaches on element X that has a single activator P and a single inhibitor N.

For the cases where the positive and negative regulation are both present at the same time, the two formulas, (7) and (8), will give the same initial value (*LOW*), and when there is only positive or only negative regulation, the two formulas will give opposite values (*ZERO* or *HIGH*).

Table 4. Effect of the 2 updating function on a test node

REGULATOR INITIAL VALUES		X INITIAL VALUE	
P	**N**	**MAX (7)**	**MIN (8)**
0	0	1	1
0	1	**2**	**0**
0	2	**2**	**0**
1	0	**0**	**2**
1	1	1	1
1	2	1	1
2	0	**0**	**2**
2	1	1	1
2	2	1	1

3.4 INITIALIZATION USING INPUTS

This last approach is meant to create a full set of initial values based on the model knowledge and with some of the outside data available. But first, let us classify our nodes into three possible categories:

- *Inputs*: those elements that do not have any regulators within the model, thus they will not be updated during the simulation and their regulatory function is the identity:

$$X^*_{input} = X_{input} \tag{9}$$

- *Intermediate nodes*: these are the most common elements for bigger models, they both regulate other elements and are themselves regulated by other elements.
- *Outputs*: these elements do not affect the model as they do not regulate any other elements, but they are influenced by some model elements.

Once we have classified our model's elements into those three categories, we can put our model in the following form, where the top elements are the ones that have more influence on the model, and the bottom ones are only affected by the model.

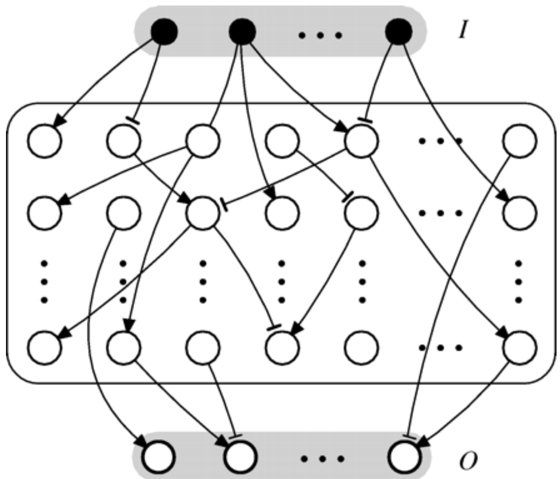


Figure 14. Classification of the nodes of a model into *INPUTS* at the top, *INTERMEDIATES* in the middle and *OUTPUTS* at the bottom.

Following this top-to-bottom hierarchy, the following are the steps of the method that we have developed to initialize the whole model:

- First, we need to initialize the inputs of the model. As the input values cannot be computed, they must be taken from the external data.
- Then, we can start to compute initial values of the intermediate nodes. To do so, we take all the intermediate elements as a sub-model of our complete network.
    - All these variables should have an initial value to be able to compute a new value. We do a random initialization here to give values to our intermediate variables.
    - We need now to consider the relationship between these variables in the model: we update the value of every variable according to our previous formulas (7) or (8) to complete the missing values.

- For bigger model, the path that goes from one element to another can be long, in order to take this into account, we do a loop of the updating process to give time to a change in an initial value to go through a long path and have influence on downstream variables.

- Finally, we now only have to deal with the outputs. This can be achieved by computing the values using the same updating formulas from (7) and (8).

This method uses all the previous methods described in this section in order to create a full initial state for the system based on both the information from the data and from our model knowledge.

3.5 ROLE OF LOGIC FUNCTIONS IN INITIALIZATION

The previous methods were focusing on the relationship between the elements inside the model, so they were based on the assumption that we have already all the knowledge in our model. In some cases, such as in the case of model extension, we are trying to integrate new elements to improve the model.

In the model extension process we are assuming to have a baseline model, with known element update functions and known initial values of all elements, and we are automatically adding elements to that baseline. In that case, we have less elements to initialize and the following method has been implemented:

- If the element added is an *INPUT*, then its initial value should be *LOW* as we want to see the effect of this variable on our model.

- For the other cases, we can determine elements' initial values from our formula (7) or (8), as we have the knowledge of the elements' regulators, as long as all the regulators are in the baseline model.

Before, the way to initialize the new elements was arbitrary chosen to add all of them at a LOW level. But now, with this method, when including extensions in the model, we assign to new elements an initial state that is coherent with the rest of the model, taking into account the new elements interactions with the baseline model.

Then, the second step after choosing initial states for those new elements added to the model is to take into account their effects in the extended model. Let us take a single element B with an activator A as an example, where we add one inhibitor Ext from outside of the model using the model extension process, as shown in Figure 15. Example of modeling extension.Figure 15.

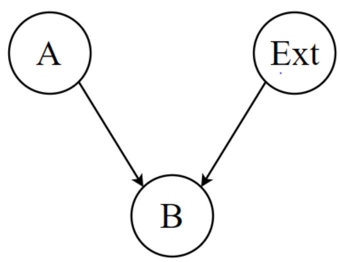


Figure 15. Example of modeling extension.

After adding the external element Ext to regulation of element B, we have changed its regulatory function, and thus, its behavior. However, there are several ways to add elements, that is, the new rule for element B can be one of the following

$$B^* = A \ AND \ Ext = A \times Ext \tag{10}$$

$$B^* = A \ OR \ Ext = A + Ext \tag{11}$$

Using those new rules and the initial values for A and Ext, we can create the truth table to determine the initial value of B:

Table 5. Initial values of *B*, based on its update rule and initial states of its regulators.

INITIALS VALUES		***B***	
A	***Ext***	**AND (10)**	**OR (11)**
0	0	0	0
0	1	**0**	**1**
0	2	**0**	**2**
1	0	0	1
1	1	1	2
1	2	2	2
2	0	**0**	**2**
2	1	1	2
2	2	2	2

From Table 5, we can observe that using an AND function when adding a new regulator to B will force B to start at *ZERO* if the two regulators are not both activated at the same time (either *LOW* or *HIGH*).

3.6 INITIALIZATION FOR DIFFERENT MAXIMUM STATES

All the previous approaches have something in common: they assume that every element within the model has the same number of states. Here, we propose a method to initialize model elements with different number of states.

The first step in this process, is to determine the maximum state for each model element. The proposed method is based on the idea to give a value for the maximal state to each element according to the number of its regulators. The aim is to give a wider range to elements that have a high number of regulators, and thus, might have more interesting regulatory functions. However, even if we assign different number of possible states to model elements, we do not want these numbers to vary too much, as this could disturb the balance during simulations. We will use the following formula to keep the balance between elements with different number of states:

$$maxStates = minStates\left(\frac{N_{regulators}}{\alpha}\right) \tag{12}$$

where $minStates$ is the minimum number of states allowed for an element and α a coefficient to limit the maximum number of states $maxStates$ that an elements can have.

Then, as we have defined the maximum number of states, *maxStates*, for each element in the model, we need to decide how to discretize our data according to that number of states. As every element has a different *maxStates* number, we need to create a thresholding function that considers this number. The following function is similar to the adaptive threshold, but gives different results based on the maximum state of the variable:

$$X_{discrete} = \left(\frac{X_{data} - MIN}{MAX - MIN}\right)(maxStates + 1) \tag{13}$$

where $X_{discrete}$ is the discrete value of element *X*, X_{data} is the numerical value of *X* in the data, *MIN* the minimal value in the data, and *MAX* the maximal value in the data. To demonstrate how this could be applied in a real example, let us assume that we have a database in which the values range from 3 to 12, and that the element we are discretizing has a value $X_{data} = 7.2$, then in Table 6 we show possible initial values for the element, based on the number of states of the element using the formula (13):

$$X_{discrete} = \left(\frac{7.2 - 3}{12 - 3}\right)(maxStates + 1)$$

Table 6. Initial values based on the maximum number of states.

MAXSTATES	3	4	5	6
$X_{DISCRETE}$	1	2	2	3

As it can be seen in this example, we can assign different initial values to element X due to the fact that the range of values found in the data can be split into different intervals, such that the same numerical value in the data does not have to imply the same discrete value in the model (Figure 16).

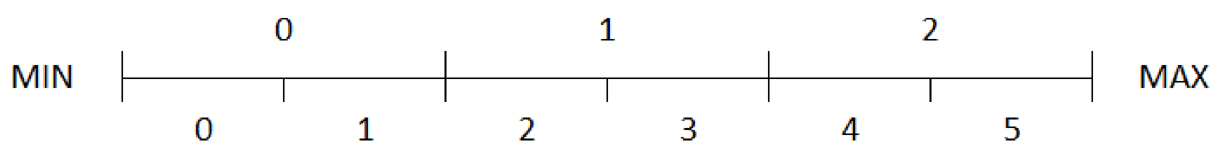


Figure 16. An example of dividing the range of value from the data into discrete intervals based on the preferred number of element states.

Finally, to complete the set of initial values for our model, if there are missing values, we need to determine those values similar to the approach shown in equations (7) and (8), now assuming different maximum number of states:

$$X_{initial} = (1 - R_N + R_P)\frac{maxStates}{2} \tag{14}$$

$$X_{initial} = (1 + R_P - R_N)\frac{maxStates}{2} \tag{15}$$

where $X_{initial}$ is the initial state, R_P is the ratio of the positive regulators (sum of the initial values over the sum of the maximum states of all the positive regulators), R_N is the ratio of negative regulators.

We can use the same example as before with element X having a single activator P and a single inhibitor N. We will assume that each of those variables has a different number of maximum states. X will have 6 states (from 0 to 5), P will have 5 states and N will have 3 states. Table 7 shows initial values for X for different combinations of P and N, and when using one of the two formulas (14) and (15) above. The trend in initialization is similar to what we saw in Table 4: if we use function (14), the only case when we can initialize element X to a 0 value is when the negative regulator is at 0, and the positive is at its highest value.

Table 7. Effect of the two updating functions on a test node

REGULATORS INITIAL VALUES		***X* INITIAL VALUE**	
P	*N*	MAX (14)	MIN (15)
0	0	3	3
0	1	4	2
0	2	5	1
1	0	2	3
1	1	3	2
1	2	4	1
2	0	1	4
2	1	2	3
2	2	3	2
3	0	1	4
3	1	2	3
3	2	3	2
4	0	0	5
4	1	1	4
4	2	2	3

4.0 APPLICATION OF DEVELOPED TECHNOLOGY ON A MELANOMA CELL

As we have now several different initialization methods, we will apply them on an existing biological model: the melanoma cell network. We will first describe this model, then apply all the initialization methods on the model, and finally, discuss the effect of different methods on the steady-state and transient responses of the system.

4.1 MELANOMA CELL NETWORK

Melanoma is a malignant proliferation of the melanocytes (pigment-containing cells). It's the most dangerous kind of skin cancer with more than 130,000 new cases every year, with 59,800 deaths in a single year (2015) [15]. It results in moles with increase in size, irregular edges, or changes in color. The main cause of Melanoma, as for most skin cancers, is due to a DNA damage resulting from an exposure to UV light [16].

The modeling of this complex process was conducted using several cell lines. In Figure 17, we illustrate the procedure for creating variable names, as this notation will be used in the following sections.

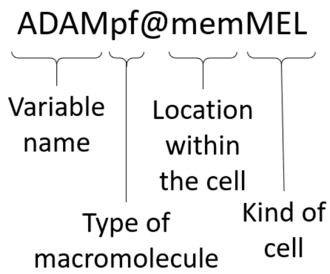


Figure 17. Schematic of a variable inside the Melanoma model

In this model, we use two types of cells: melanoma cells (MEL) and the macrophage (MAC). And for those two types of cells, we assume elements can be at five different locations, listed in **Table 8**. The full name of the variable also indicates the type of macromolecules that are classified in six different categories, listed in Table 9.

Table 8. Location of the elements within the cell

LOCATION NAME IN THE MODEL	LOCATION IN THE CELL
MEM	Membrane
CYTO	Cytoplasm
NUC	Nucleus
MITO	Mitochondria
EX	Outside of the cell

Table 9. Type of macromolecules inside the Melanoma model

NAME INSIDE THE MODEL	TYPE OF MACROMOLECULE
PN	Protein
PF	Protein family
GENE	Gene
RNA	RNA
CHE	Chemical
BP	Biological process

The melanoma model that we used has 265 variables, and those variables have 233 positive regulation and 82 negative regulations. With 315 edges, the model is pretty complex, and the pathways are not easily reckonable.

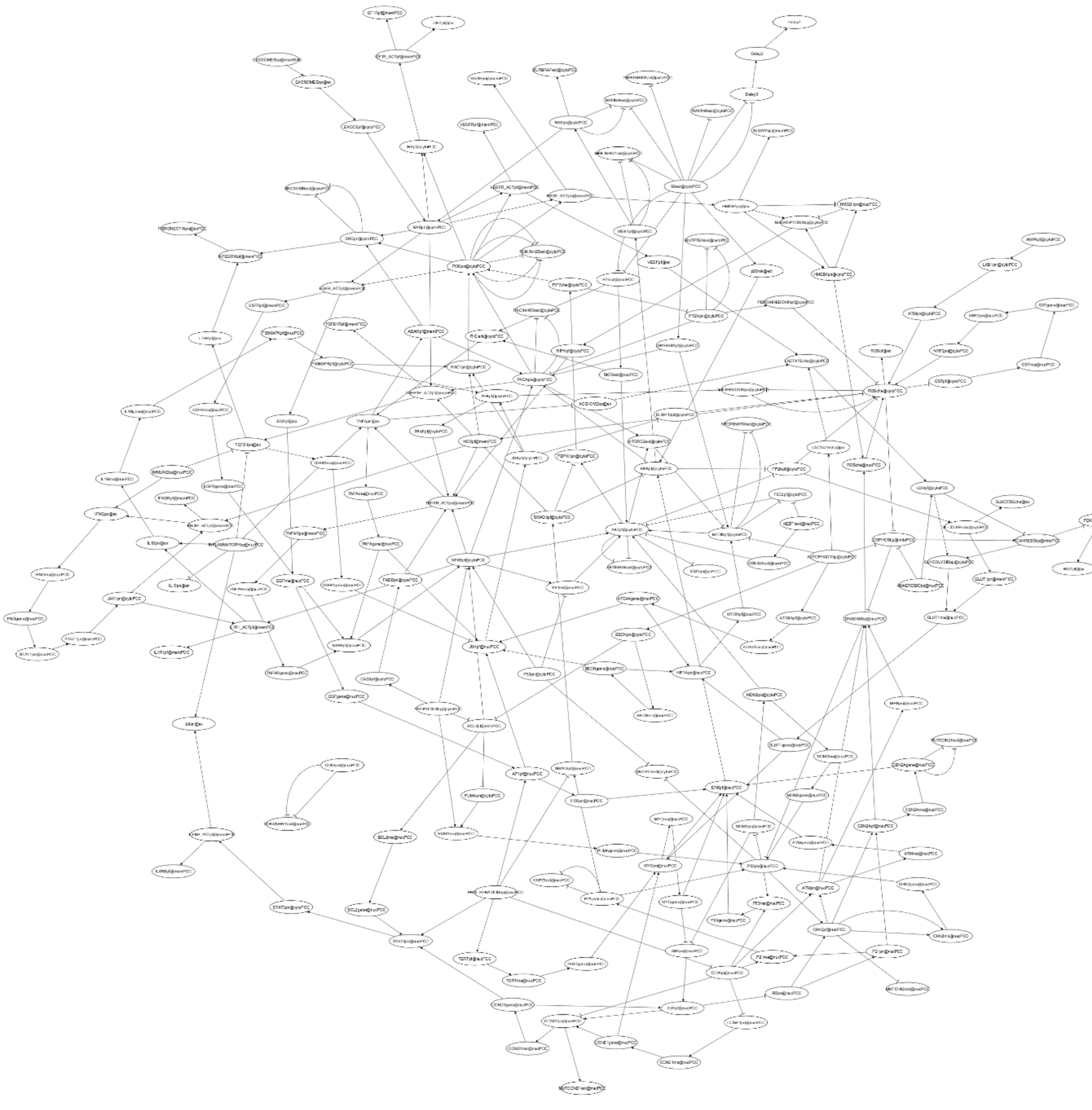


Figure 18. Interactions map of the Melanoma model

4.2 RANDOM INITIALIZATION

Using the first method for initializing model elements, that does not take any information from the literature or from the model, we will assign only the values given by the two random initializations:

- Random Uniform Initialization (RUI)
- Random Non-Uniform Initialization (RNUI)

Figure 19. Initial values using the random algorithms.

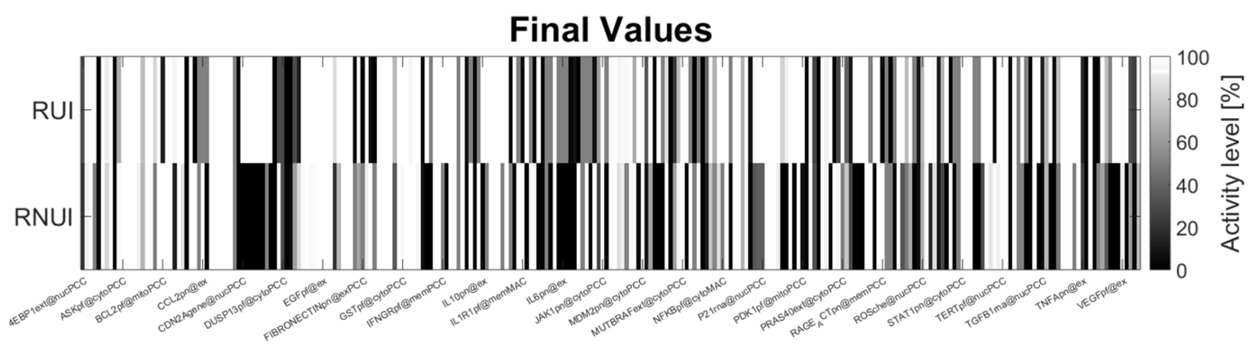


Figure 20. Final values using the random algorithms.

We can see from Figure 19 that applying the RNUI method results in a larger number of elements being initialized to value *ZERO* than in the case of the RUI method. To see how this difference in the initial states impacts our system, we will look at the final values after 200 runs of 3000 steps using the random-order sequential update scheme of the simulator. The results are shown in Figure 20. Note that the gray scale results from the fact that we are using random-order sequential approach and averaging across multiple runs, as described in Section 2.2.2 and illustrated in Figure 6.

As can be seen from Figure 20, the final values for the RNUI method seem much closer to the RUI method than the initial values. To further evaluate the difference between initial and final values, we compute the average difference for both initial and final values between the two methods, shown in Table 10.

Table 10. Average difference between the 2 random initialization methods

	INITIAL (%)	FINAL (%)
AVERAGE DIFFERENCE BETWEEN RUI AND RNUI	43.77	30.48

4.3 INITIALIZATION FROM THE INPUTS

The first step we need to do here is to classify our model into *INPUTS*, *INTERMEDIATE* and *OUTPUTS*. With the high number of elements, we will only display the *INPUTS* and the *OUTPUTS* in Table 11.

Table 11. Inputs/Outputs of the Melanoma cell network

INPUTS		
FIBRONECTINpn@exMEL	GLUCOSEche@ex	ICMTpn@erMEL
IFNGRpf@memMEL	IGF1pf@ex	IGF1Rpf@memMEL
IL1R1pf@memMEL	MUTPTENext@cytoMEL	IL6RBpf@memMEL
IL6pn@ex	RAGEpn@memMEL	ROScf@ex
TGFB1Rpf@memMEL	VEGFRpf@memMEL	WNTpf@ex
INJURYbp@memMEL	MUTBRAFext@cytoMEL	IL10pn@ex
MUTCDN2Aext@nucMEL		

Table 11. (continued)

OUTPUTS		
EXOSOMESbp@memMAC	INFLAMMATORYbp@nucMEL	AUTOPHAGYbp@cytoMEL
RPS6KB1pf@cytoMEL	PROLIFERATIONbp@nucMEL	INGR2gene@memMAC
IMMUNEbp@nucMEL	APOPTOSISbp@cytoMEL	P53pn@cytoMEL
ANAEROBICbp@nucMEL		

Then, we use the value in the data from [17] for the inputs, and we use either the formula from (7) or (8) to compute the missing values:

- Initialization from Inputs using the MAX update function (IIMAX)
- Initialization from Inputs using the MIN update function (IIMIN)

Figure 21. Initial values using the initialization from the inputs.

In that initialization process, some elements are initialized with the same values: the 19 inputs of the model and the elements with *LOW* activation value as this is the only value that is assigned in the same way in the two methods (7) and (8). The difference in the computation of the missing value can be seen on values with either *ZERO* or *HIGH* activation as they are initialized differently in those two methods.

4.4 INITIALIZATION FROM DATA

This method is similar to the previous one except that instead of using the values of the data only at the inputs, we derive element initial values from all available data, and we use the update functions only for the elements that were missing in the data:

- Initialization from Data using Adaptive Threshold and MAX to fill the missing values (IDATMAX)
- Initialization from Data using Adaptive Threshold and MIN to fill the missing values (IDATMIN)
- Initialization from Data using Quantile Threshold and MAX to fill the missing values (IDTTMAX)
- Initialization from Data using Quantile Threshold and MIN to fill the missing values (IDTTMIN)

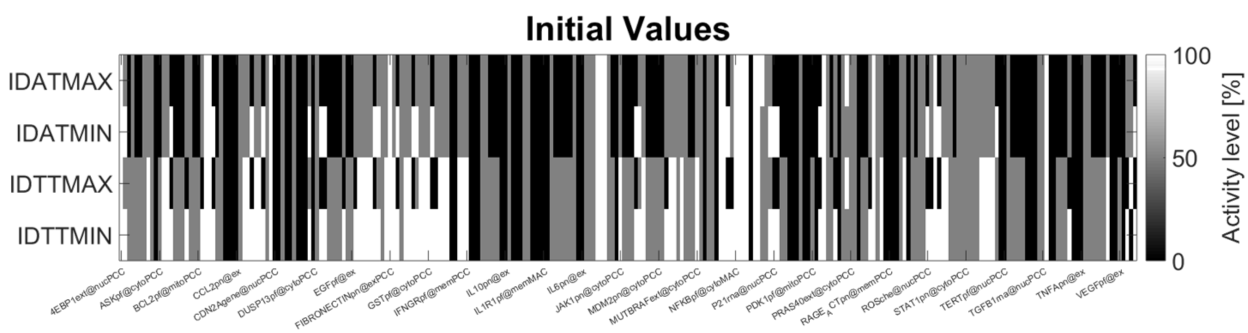


Figure 22. Initial values from the data initialization methods

By looking at the overall initial values, we can see that some elements are always initialize at the same values across the four methods (elements that were found in the data) and some elements that have different initial value either from the different thresholding or from the computation using the missing value algorithm.

The difference in the thresholding function can be seen as we have more *ZERO* value in the first two rows (Adaptive thresholding) and only a few *HIGH* values. And in the last two rows, we observe a better repartition of those three level of expression when using the quantile threshold.

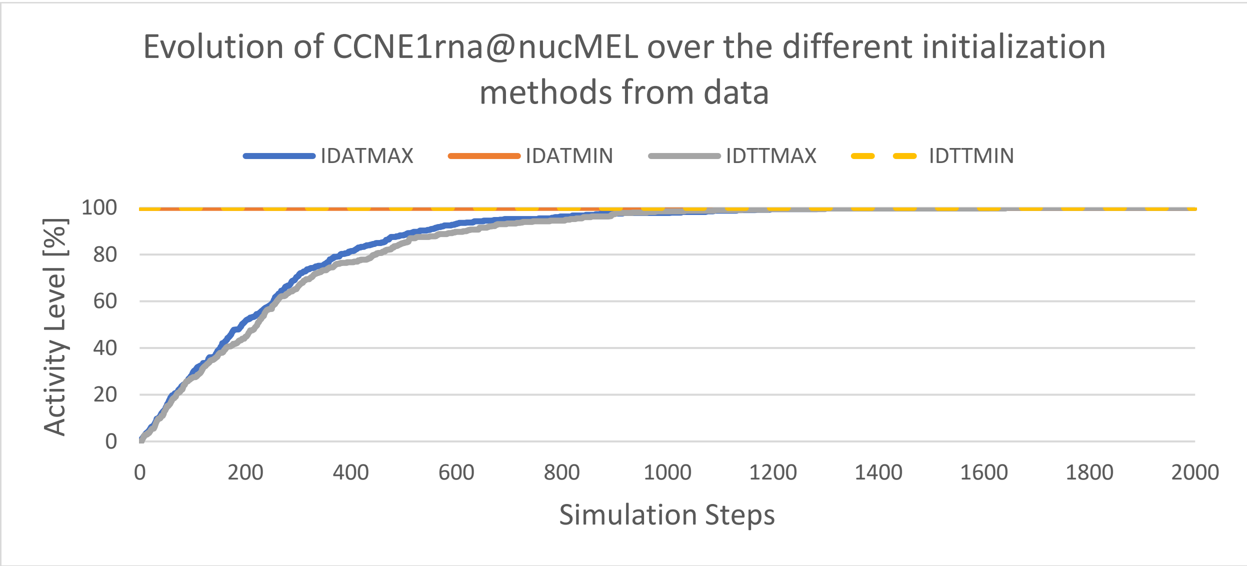


Figure 23. Comparison of the different initialization from data on the CCE1rna

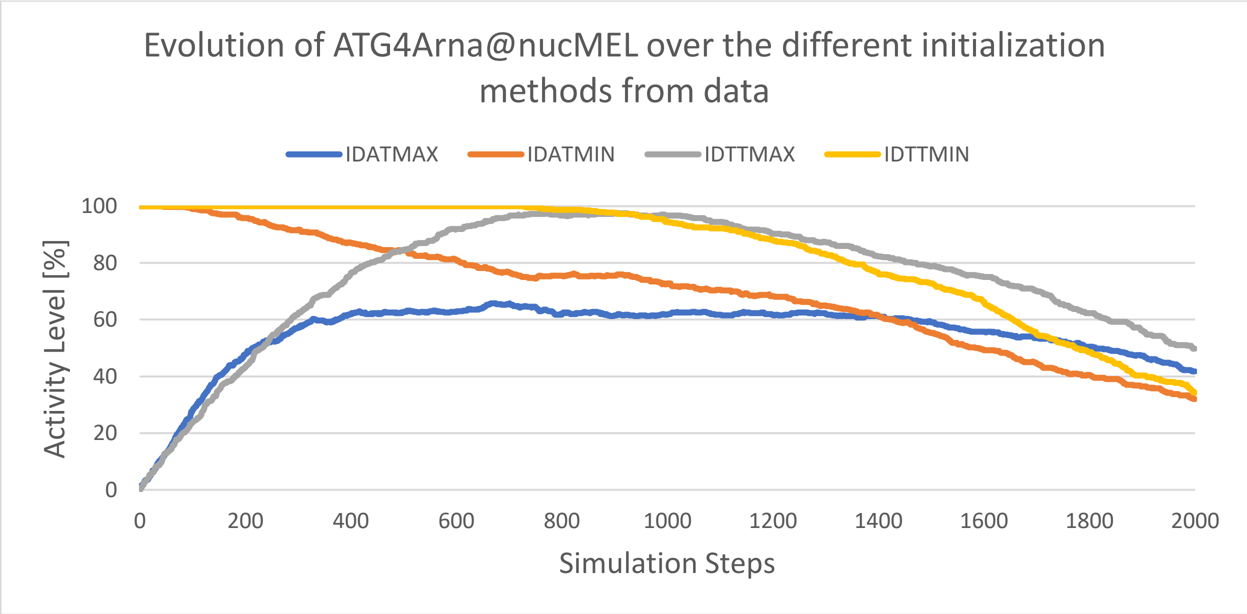


Figure 24. Comparison of the different initialization from data on the ATG4rna

To observe the effect of the computation of missing functions, there are two possible cases:

- The prediction of the element behavior has been correct such as in Figure 23 and the use of MAX function (7) enables better modeling of transient response, while the MIN function (8) only provides the steady state value.
- The prediction of the behavior has been wrong, as in the Figure 24 where initially there was a positive regulation affecting our node but during the transient period between initial and steady state the negative regulation took over. In that case, both computations of the missing value methods show a transient response.

4.5 EFFECT OF LOGICAL FUNCTION CHOICE AND INITIALIZAITON

Here, we have used the baseline model initial state updated with the elements that are added to the model, and we are also taking into account the combined impact of initialization and regulatory functions:

- Assuming that the new element is a necessary condition for the node activation/inhibition, that is, using an AND operator when extending the regulatory function.
- Assuming that the new element is only a sufficient condition for the node activation/inhibition, that is, using an OR operator when extending the regulatory function.

Figure 25. Initial values for the implication of logical functions

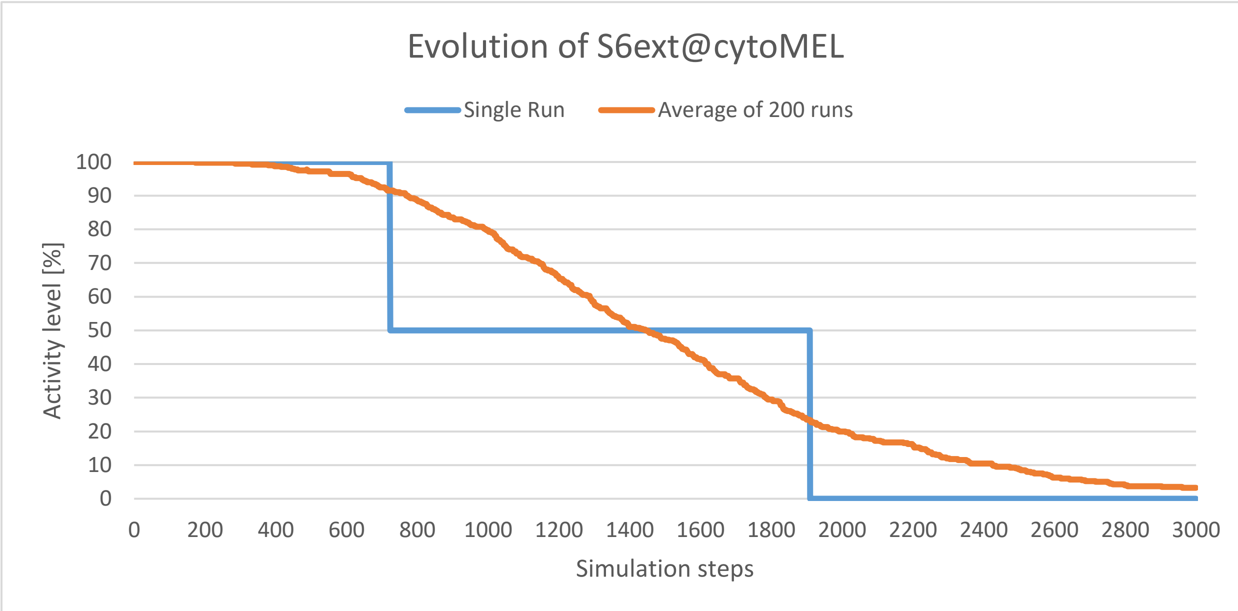
The only difference in initial values in this case stems from the elements that get the external elements as a new regulator. And then, based on the old model element's previous initial value in the baseline model and on the new external element initial value, we update the old model element value by using either AND or OR operator in the updated logical function.

4.6 INITIALIZATION WITH DIFFERENT MAXIMUM STATES

To compare the results of the model where all elements had three states and the model where every element has a different number of maximum states, we ran a simulation of the classic three states model for 3000 steps and 200 runs and compared this results with the model using elements with a different number of maximum states. We achieve a similar result by using the following parameters for the number of $maxStates$ (12):

- $\alpha = 3$
- $minStates = 3$

$$maxStates = minStates + \left(\frac{N_{regulators}}{\alpha}\right) = 3 + \frac{N_{regulators}}{3}$$



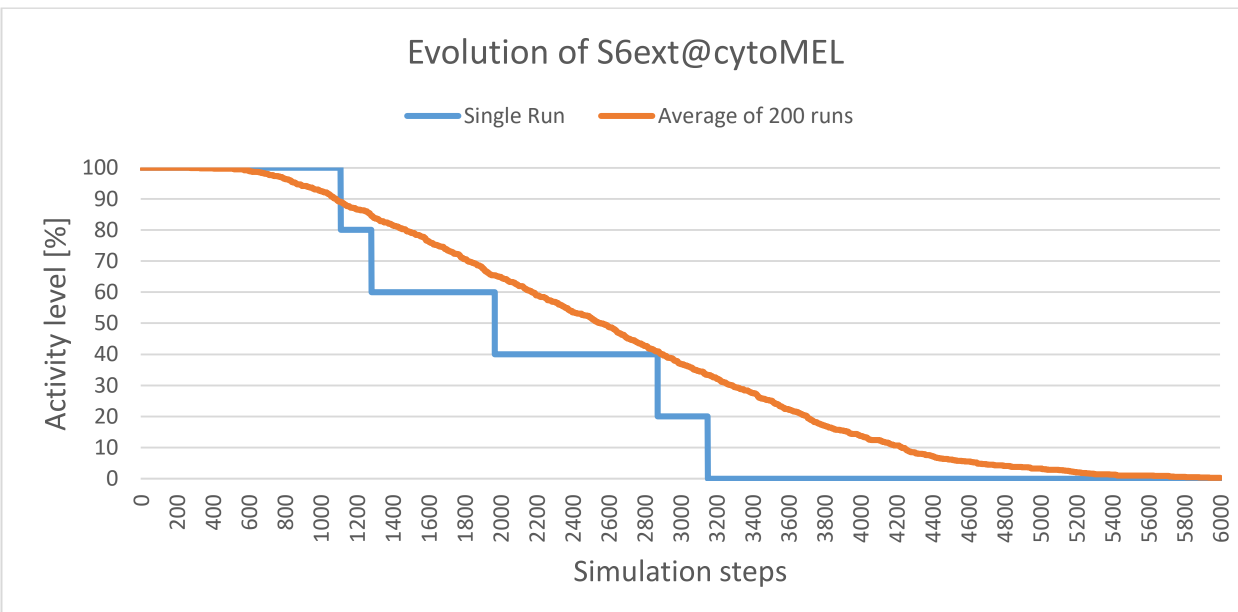


Figure 26. Simulation of the ribosomal protein S6 in the classical model with 3 states (top) and in the different maximum states model (bottom).

By looking at a single run, we can see the difference in the number of levels: in the different maximum states model, the S6 protein has six states. The only difference is on the simulation steps axis, as in the new model the element has twice the number of states than before (six instead of three), it also needs twice the time to decrease the element value by the same amount (6000 steps instead of 3000 before). Adding more states to our element with our formula does not change their behavior, it slows them but gives them more freedom in terms of changing their values.

4.7 IMPACT OF THE DIFFERENT THRESHOLDING METHODS ON THE TRANSIENT RESPONSE OF THE SYSTEM

To compare different thresholding methods on the transient response of the system, we chose a set of 10 elements that have larger number of regulators in the model. Having a higher number of regulators means a more complex regulatory function, and thus, more sensitivity to a change in the initial values.

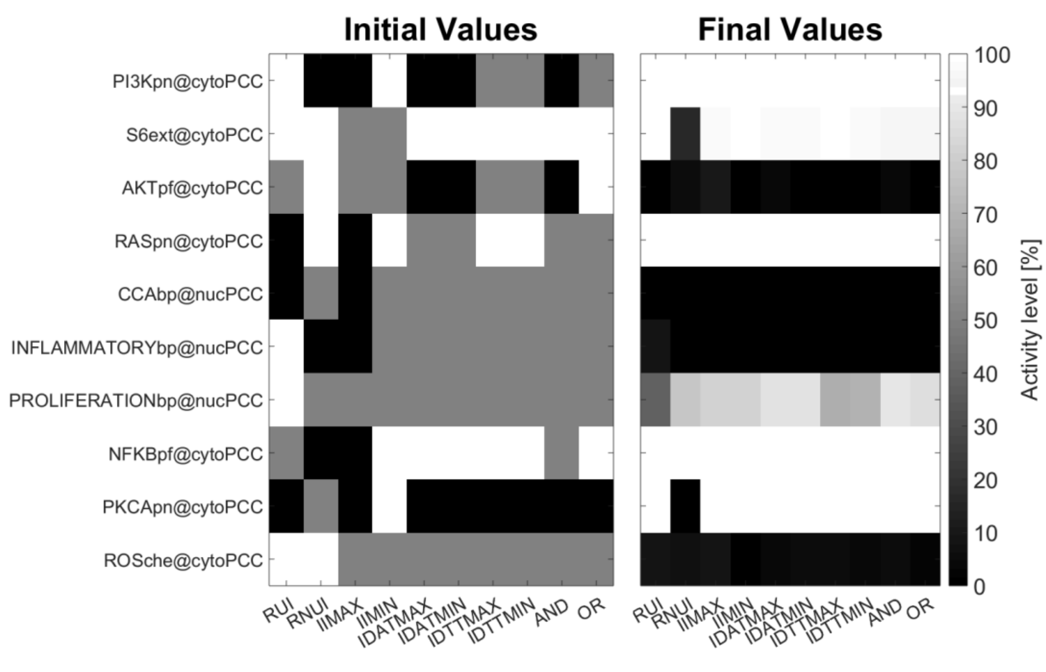


Figure 27. Set of the initial and final values for the different initialization methods

By looking at both the initial values and the final values of the selected 10 elements, we can find out some behavior change during our simulations.

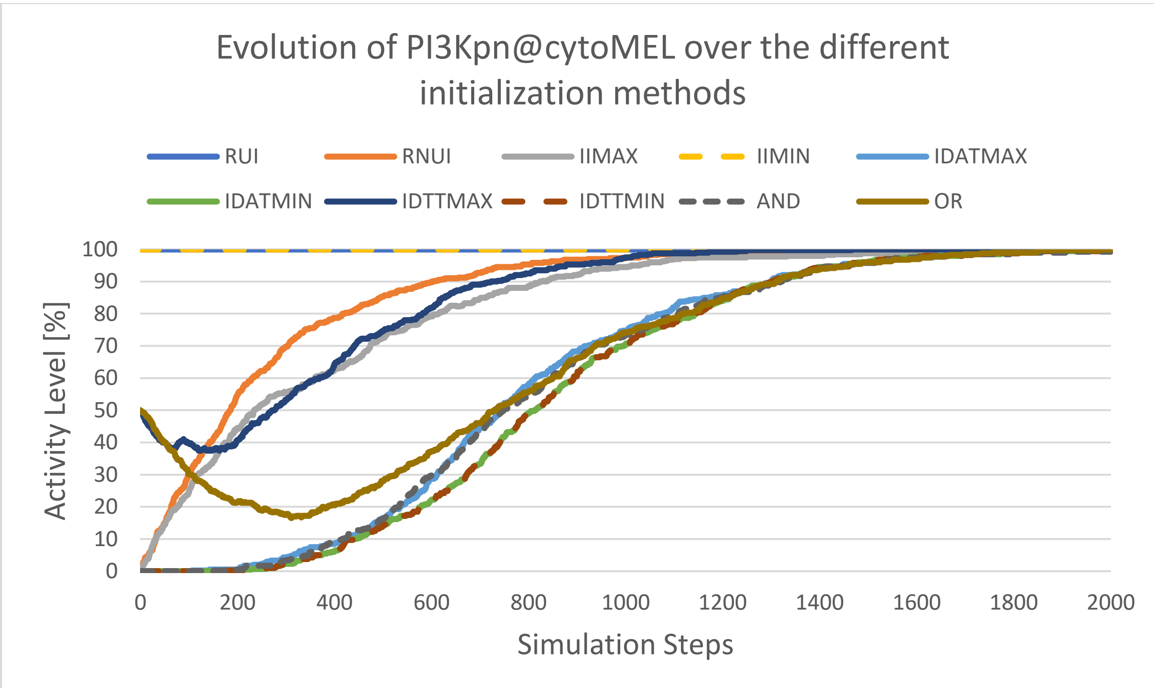


Figure 28. Evolution of the PI3K for the different initialization methods.

Even if we change the initial value as for the PI3K, the final state of the element might not change. In this case, changing the element's initial value only changes its transient response to achieve its final value. The RUI and IIMIN method that have set up the PI3K at a *HIGH* initial level (100%) do not show any changes during the intermediate simulations steps as these elements are already in their steady states. The IDTTMAX and OR method that set these elements to *LOW* activation (50%) have also a different behavior: there is first a negative regulation going on before the positive took over after a few steps, while all the others methods initialize the elements at a *ZERO* level when they have only positive regulation. This is the case for most of the elements we have selected, the final values show less changes than the initial ones.

In another case, the same initial value can lead to different final values. For the PROLIFERATION, the changes in the final values are due to a slight change in its behavior where its positive regulation has been delayed, and thus, its oscillating behavior looks slightly different and induces a change in its final values.

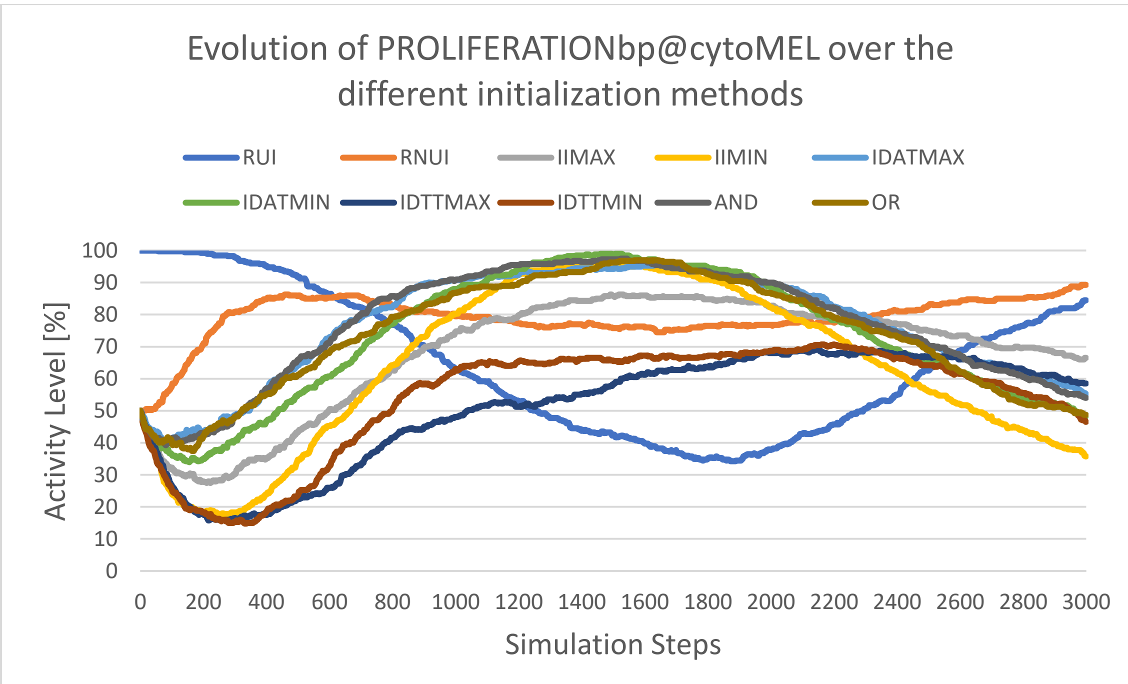


Figure 29. Evolution of the PROLIFERATION for the different initialization methods.

Thus, we can say that in the case of a stable element value, such as in Figure 28, no matter what starting point is chosen for the element, since the system is stable for that element, the final value will not be affected. Only the time needed to reach that steady state and the behavior between initial and final state is changing.

But for those elements that are not stable, or have an oscillatory behavior, such as in Figure 29, the choice of initial values induces more changes in both transient behavior and final values.

5.0 CONCLUSION

As we have seen, initialization of discrete models is the first and mandatory step in order to be able to simulate these models, and thus emulate a biological system behavior. In this thesis, we have proposed several approaches to initialize such models.

Random initialization, as the simplest one and independent from any knowledge of the system. We have shown that the use of this method does not achieve the best results. In spite of giving an easy way to initialize any model and produce a huge selection of possible initial states, its randomness is more likely to produce undesired behavior within the model.

Initialization from inputs, using the less possible knowledge from the outside and the more from the model is suitable when we already have sufficient confidence in our model. For a model that have been proven to accurately simulate a system, initialization from inputs is the best alternative, as it heavily relies on the interconnection between elements inside the model.

Initialization from data, that uses as much of the external knowledge as possible and the less from our model is, opposite from the initialization from inputs, more suitable for a new model that needs to be tested. As the information from our model are to be verified, it is recommended to rely more on outside knowledge to test our model.

In other cases, when we want to emphasize the study of our model in some particular elements, the initialization of the model with a different number of maximum states is the best choice in order to achieve a finer resolution on the targeted elements.

And finally, when we are looking at improving a model, such as in the process of model extension, the creation of new connections within this model must be reflected on the new initial state of the model. The initialization using the interplay of logical function AND/OR is in that case recommended.

BIBLIOGRAPHY

[1] IOM (Institute of Medicine). 2009. *Microbial evolution and co-adaptation: a tribute to the life and scientific legacies of Joshua Lederberg*. Washington, DC: The National Academies Press.

[2] Gillespie, D. T. Exact stochastic simulation of coupled chemical reactions. The Journal of Physical Chemistry, 81, 25 (1977), 23402361.

[3] Novak, M. and Tyson, J. J. A model for restriction point control of the mammalian cell cycle. Journal of Theoretical Biology, 230, 4 (2004), 563-579.

[4] Faeder, J. R., Blinov, M. L. and Hlavacek, W. S. Rule-based modeling of biochemical systems with BioNetGen. Methods Mol Biol, 500(2009), 113-167.

[5] M. I. Davidich and S. Bornholdt "Boolean Network Model Predicts Cell Cycle Sequence of Fission Yeast" PLoS One, vol. 3, Feb 2008.

[6] N. Miskov-Zivanov and et al. "Boolean Modeling and Analysis of Peripheral T Cell Differentiation of Work" personal communication, 2013.

[7] Molecular Biology of the Cell. 4th edition. Alberts B, Johnson A, Lewis J, et al. New York: Garland Science; 2002.

[8] Kauffman, S. A. Metabolic stability and epigenesis in randomly constructed genetic nets. J Theor Biol, 22, 3 (1969), 437-467.

[9] Thomas, R., Thieffry, D. and Kaufman, M. Dynamical behaviour of biological regulatory networks--I. Biological role of feedback loops and practical use of the concept of the loop-characteristic state. Bull Math Biol,57, 2 (1995), 247-276.

[10] I. Shmulevich, R. Dougherty, S. Kim, W. Zhang, "Probabilistic Boolean neworks: A rule-based uncertainty model for gene regulatory networks", Bioinformatics, vol. 2, no. 18, pp. 261-274, 2002.

[11] B. Samuelsson, C. Troein, "Superpolynomial growth in the number of attractots in Kauffman networks", Phys. Rev. Lett., vol. 90, pp. 90098701, 2003.

[12] Xiao Y.F. A tutorial on analysis and simulation of Boolean gene regulatory network models. Curr. Genomics 10, 511–525 (2009).

[13] T. Cormen, C. Leiserson, R Rivest, C. Stein (2001) Introduction to algorithms, second edition.

[14] Khaled Sayed, Natasa Miskov-Zivanov, Yu-Hsin Kuo and Anuva Kulkarni, DiSH Simulator: Capturing Dynamics of Cellular Signaling with Heterogeneous Knowledge.

[15] World Cancer Report 2014. World Health Organization. 2014. pp. Chapter 5.14.

[16] Wang S, Setlow R, Berwick M, Polsky D, Marghoob A, Kopf A, Bart R (2001). "Ultraviolet A and melanoma: a review". J Am Acad Dermatol. 44 (5): 837–46

[17] Mohammad Fallahi-Sichani, Nathan J Moerke, Mario Niepel, Tinghu Zhang, Nathanael S Gray, andPeter K Sorger1Systematic analysis of BRAF(V600E) melanomas reveals a role for JNK/c-Jun pathway in adaptive resistance to drug-induced apoptosis. Mol. Syst. Biol. 2015;11:797. doi: 10.15252/msb.20145877.