RESEARCH ARTICLE

A Boolean network control algorithm guided by forward dynamic programming

Mohammad Moradi[1], **Sama Goliaei**[1]*, **Mohammad-Hadi Foroughmand-Araabi**[2]

1 Faculty of New Sciences & Technologies, University of Tehran, Tehran, Iran, **2** Department of Mathematical Sciences, Sharif University of Technology, Tehran, Iran

* sgoliaei@ut.ac.ir

Abstract

Control problem in a biological system is the problem of finding an interventional policy for changing the state of the biological system from an undesirable state, e.g. disease, into a desirable healthy state. Boolean networks are utilized as a mathematical model for gene regulatory networks. This paper provides an algorithm to solve the control problem in Boolean networks. The proposed algorithm is implemented and applied on two biological systems: T-cell receptor network and Drosophila melanogaster network. Results show that the proposed algorithm works faster in solving the control problem over these networks, while having similar accuracy, in comparison to previous exact methods. Source code and a simple web service of the proposed algorithm is available at http://goliaei.ir/net-control/www/.

Citation: Moradi M, Goliaei S, Foroughmand-Araabi M-H (2019) A Boolean network control algorithm guided by forward dynamic programming. PLoS ONE 14(5): e0215449. https://doi.org/10.1371/journal.pone.0215449

Editor: Gareth J. Baxter, University of Aveiro, NEW ZEALAND

Received: September 6, 2018

Accepted: April 2, 2019

Published: May 2, 2019

Data Availability Statement: All relevant data are within the manuscript and its Supporting Information files.

Funding: The authors received no specific funding for this work.

Competing interests: The authors have declared that no competing interests exist.

1 Introduction

A gene regulatory network (GRN) is a mathematical model of genes and their interaction [1]. The purpose of GRN studies is to achieve a new insight toward the important cellular processes. Examples of mathematical modeling of biological processes includes cell cycle [2, 3], oscillations in p53-mdm2 system [4–6], phage-lambda system [7–9], and T-cell large granular lymphocyte (T-LGL) leukemia network [10, 11]. There are different techniques for modeling dynamics of GRNs, including Boolean networks (BNs) [11], Bayesian networks [11], dynamic Bayesian networks [11], linear models [12] and differential equations [12]. Among the above mentioned models, the Boolean network model has received many attentions [13–16]; that is because in addition to the tractability, Boolean networks could be reconstructed by efficient biological experiments [17, 18].

Detecting a set of perturbations which cause the desirable changes in cellular behavior has many applications such as cancer treatment and drug discovery [5, 19–23]. This highlights the necessity of developing a control theory for the gene regulatory networks. Using GRN control model is considered as a key method to design the experimental control policies [23].

The control problem includes finding a sequence of interventions to be applied on the system, which changes state of the system from an undesirable state of the network to a desirable one [24–26]. The undesirable state in a gene regulatory network may express a disease such as cancer, and the desirable state can express the wellness, for example as induction of apoptosis in cancerous cells or tumors. Therefore, using the case of control and medical interventions, we can exterminate the tumor cells and achieve healthiness [26].

1.1 Related works

Up to now, numerous methods have been proposed to solve the control problem in Boolean networks. Among the vast diverse proposed methods, we name the more optimized control techniques, to which a list of possible control nodes is given as input [27–30]. Bo Gao et al proposed an algebraic method to solve the control problem and used the semi-tensor product (STP) as a state transition matrix [31]. Qiu, Yushan et al took benefit from the integer programming to solve the control problem in multiple Boolean networks, for the cancer-causing and normal cells [32]. Christopher James Langmead and Sumit Kumar Jha proposed an algorithm based on model checking to find the control strategy in Boolean networks [33]. Yang Liu et al searches for a controlling sequence to transform from a state to a desirable one, with a difference that he avoids some special and prohibited states [34].

Authors found no exact algorithm for the same network control problem in more recent years. However, an algorithm for identification of the best one-bit perturbation is provided [35]. In this problem, a Boolean network is given and the problem is to find a gene that changing its initial state 1) does not change attractors of the network, 2) maximize the size of the basin of some desired attractors. This method first computes the state graph of the network and then tries to modify the state graph after changing some node's value, efficiently.

Meanwhile, in some cases, the genetic algorithm and the greedy algorithms are used to solve the control problem in Boolean networks [36–38]. Datta et al proposed an algorithm to control the probabilistic Boolean networks (PBN) based on Markov chains and dynamic programming [24–26]. In this approach, it is supposed that states of some nodes could be controlled externally, and the goal is to find a sequence of changes to be applied as controlling policy to result in the network desirable state. In their method, they first consider the final desired state, and then, they compute a set of desired states for all the time steps in a backward manner. To compute the set of desired states for a time, given set of desired states for the next time step, they enumerate all possible states of the network and their next state. A state with its desired next state is considered as the desired state, for that time step.

Since the Boolean network is a specialization of the probabilistic Boolean network, this method is also applicable to Boolean networks. The problem with the proposed algorithm was that it lacks the necessary efficiency, because all the states within probabilistic Boolean network (or Boolean network) were required to be taken into consideration in all time steps between the initial state and the desirable state; so a state transition matrix with exponential size is produced programmatically [25].

Akutsu et al [39] showed that finding a control strategy for a Boolean network is an NP-hard problem. However, they proposed a polynomial time algorithm to find control strategies over trees, instead of general graphs. The algorithm is based on dynamic programming in which a sub-problem is to find a control sequence for a subtree of the original tree. They also expanded the algorithm for the networks with a low number of loops, but if the network has a high number of loops, or the given number of time steps between the initial state and the desirable state is high, the algorithm would not have the desirable efficiency [39]. Thus, in most of the proposed methods, if the size of the Boolean network is high, the proposed algorithm might have not the desirable efficiency.

Since most biological networks lack a tree structure, the algorithm for tree structures is not applicable for real networks [39]. Therefore, new algorithms are still needed to be more efficient for general networks with a high number of loops, a high number of time steps, and for large size networks. In this paper, we have presented a new algorithm to solve the control problem, and we investigate the applicability of the provided algorithm on two GRNs: T-cell receptor network [40] and Drosophila melanogaster network [41], and compare the efficiency with other algorithms.

In another completely different view, a mathematical formulation of the controllability problem of Boolean networks is provided. These works formulate the dynamics of Boolean networks as linear systems. They defined an extension of matrix multiplication to achieve this goal. They applied this technique on classical Boolean network control problem and showed that this formulation exactly models the desired problem. Le et al proved theorems that specify some necessary and sufficient conditions for a Boolean network to be controllable [42]. The same technique is applied to model a Boolean network with time delays. In a Boolean network with time delay, state of nodes may affect other nodes after some predefined delay. Some necessary and sufficient conditions for controllability of these networks is provided [43]. An extension to the network control problem, which has not only one initial and one desired states, but also a set of states as initial and desired ones (called trajectory), is provided recently [44]. With similar algebraic techniques, they proved theorems on controllability for trajectory problems.

1.2 General idea of proposed algorithm

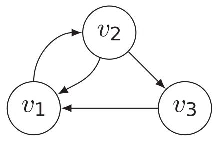
The proposed algorithm is an exact algorithm, i.e. an algorithm that always finds the correct answer, but it is using some properties of the network to enumerate a smaller number of states and perform faster than previously provided algorithms. The general idea of the proposed algorithm is to divide the network to partially-dependent parts and then consider all the possible states the network may be in. To discover these independent subnetworks, we consider strongly connected components and categorize them to find independent parts. Strongly connected components induce an order of dependency between genes. Thus, we can handle them with the induced order, one by one. For each strongly connected component, we consider all possible states. If possible states of a subnetwork (that forms a strongly connected component), is independent of some other parts, we can enumerate these two parts independently, and by this trick, the number of states that we should consider would be reduced. More precisely, if there are k independent parts having number of status $\#S_1, \ldots, \#S_k$, number of status that we consider reduces from $\#S_1 \times \ldots \times \#S_k$ to $\#S_1 + \ldots + \#S_k$. The multiplication is the number of states that the Datta algorithm enumerates. With this trick, we reduce the running time of our algorithm.

As it could be seen, size of strongly connected components of a network could be a good estimation of how faster our algorithm is, in comparison of previously provided dynamic programming algorithms (e.g. Datta algorithm). In a recent study, GRN of E. coli is analyzed. In this network, among 1807 genes, there are 202 transcription factors. This network does not contain any strongly connected component with more than 5 nodes [45]. Also, a manually curated GRN of mouse consists of 274 genes with 176 transcription factors. Its larges strongly connected component contains only 80 genes, in which only 29 are transcription factors [45]. These observations show that real regulatory networks have small strongly connected components which are the cases that our algorithm works better for them. By taking advantages of this observation we provided an algorithm in this paper.

2 Materials and methods

2.1 Background on Boolean network control

In a Boolean network, each node represents a gene, and each edge represents a regulatory effect of one gene expression on another one, which may cause an increase or decrease in the gene expression [46].



a

Time t			Time t+1		
v_1	v_2	v_3	v_1	v_2	v_3
0	0	0	0	0	1
0	0	1	0	0	1
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	0	1	0
1	1	1	1	1	0

b

Fig 1. (a) An example of a Boolean network. (b) the state transition table of this Boolean network.

https://doi.org/10.1371/journal.pone.0215449.g001

A Boolean network is modelled as a directed graph $G = (V, F)$, which includes a set of n nodes $V = \{v_1, v_2, \ldots, v_n\}$, and a set of Boolean functions $F = \{f_1, f_2, \ldots, f_n\}$. Boolean network is a discrete time model. Each node v_i has a state variable $v_i(t) \in \{0, 1\}$ representing the state of node v_i at time t, where 0 (1) indicates lack of (existence of) gene expression. Also, each node v_i has a Boolean function f_i, representing how to obtain $v_i(t + 1)$ from the state of the incoming nodes to v_i at time step t by applying basic Boolean operations (**and**, **or**, **not**). The network state at time t, is defined as vector $v^t = [v_1(t), v_2(t), \ldots, v_n(t)]$, which describes the state of the nodes in time step t.

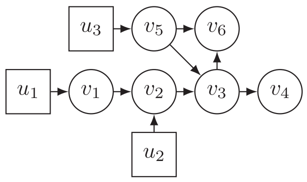
An example of a Boolean network is represented in Fig 1(a). In Fig 1(b), the state transition table of the mentioned Boolean network is represented. This table represents the next state of the network according to the current state. For example, if the network state in time step t is [0, 1, 1], then the network state in time step $t + 1$ is [1, 0, 0].

In the control problem of Boolean networks, a Boolean network $G = (V, F)$, initial state and a desirable network state v^τ is given. The set of nodes V is called internal nodes. A set of control nodes $\{u_1, \ldots, u_m\}$ are added to the network, known also as external nodes, which are used to influence internal nodes to attain the desirable state. The external nodes have no incoming edges, and their values are specified externally. The problem is to find a sequence of state values $u^0, \ldots, u^\tau$ for external nodes, which leads the network to be in desirable state v^τ in time step τ. If there exists no such control sequence, this fact should be announced as the output. In gene regulatory networks, the desirable state of the network represents a healthy state of the system, and external nodes represent potential medicines affecting network behavior. Thus, finding control strategies has applications in various medical areas, including medical protocol design for example in cancer treatment [19, 20].

An example of a control problem on a Boolean network is represented in Fig 2. In this example, $\{v_1, \ldots, v_6\}$ is the set of internal nodes and $\{u_1, u_2, u_3\}$ is the set of external nodes. The initial state of the network is $v^0 = [1, 0, 1, 1, 0, 0]$, and the desirable state is $v^3 = [0, 1, 1, 1, 0, 1]$. Thus, we are looking to find a control sequence $u^0, \ldots, u^3$ in such a way that the network would be in state v^3 at time step $t = 3$. A possible solution, which is shown in the same figure, is $u^0 = [1, 0, 0]$, $u^1 = [1, 1, 0]$, $u^2 = [0, 1, 1]$.

2.2 The proposed algorithm

The idea of our proposed algorithm is to reduce backtracking space by separating dependencies and merging information from some sub-parts of the graphs, which we call components. In addition, we categorize components to find some easily solvable components, non-branching and in some cases the branching components, to reduce the number of states to enumerate.



$$v_1(t+1) = u_1(t)$$
$$v_2(t+1) = v_1(t) \wedge u_2(t)$$
$$v_3(t+1) = v_2(t) \wedge v_5(t)$$
$$v_4(t+1) = v_3(t)$$
$$v_5(t+1) = \neg u_3(t)$$
$$v_6(t+1) = v_3(t) \vee v_5(t)$$

a **b**

	v_1	v_2	v_3	v_4	v_5	v_6
Initial $(t = 0)$	1	0	1	1	0	0
Desired $(t = 3)$	0	1	1	0	0	1

Solution

t	v_1	v_2	v_3	v_4	v_5	v_6	u_1	u_2	u_3
0	1	0	1	1	0	0	1	0	0
1	1	0	0	1	1	1	1	1	0
2	1	1	0	0	1	1	0	1	1
3	0	1	1	0	0	1			

c

Fig 2. Example of a Boolean network control problem. (a) Network, (b) its state transition rules, (c) Control problem and its solution.

https://doi.org/10.1371/journal.pone.0215449.g002

In our proposed algorithm, we compute an intermediate variable Υ to be used in the computation of the final result. For each node, at each time step, we calculate answer of two questions: is it possible for this node to be in state 1 (0), and store result of this question in an intermediate variable Υ. For this computation, we design the following steps.

For network node v_i, Boolean variable $b \in \{0, 1\}$, and time step t, we define variable $\Upsilon_b^{v_i}(t)$, which is *true* if and only if it is possible to assign values to external nodes in such a way that it cause $v_i(t)$ to get value b, and it is *false* otherwise. In the other words, $\Upsilon_1^{v_i}(t)$ and $\Upsilon_0^{v_i}(t)$ represent the possibility for node v_i in time step t to have state value 1 and 0, respectively.

Let $v_{i_1} \ldots v_{i_k}$ be the input nodes to node v_i. According to the definition of Υ we have $\Upsilon_1^{v_i}(t+1) = true$ if, and only if, there exists $[b_{i_1}, b_{i_2}, \ldots, b_{i_k}]$ such that $f_i(b_{i_1}, b_{i_2}, \ldots, b_{i_k}) = 1$ and $\Upsilon_{b_{i_j}}^{v_{i_j}}(t) = true$ for all $j = 0, \ldots, k$. Value of $\Upsilon_0^{v_i}(t+1) = true$, could be computed in the same way.

For example, for node v_6 in Fig 2, $\Upsilon_1^{v_6}(t+1)$ is true if at least one of the variables $\Upsilon_1^{v_3}(t)$ and $\Upsilon_1^{v_5}(t)$ are true. On the other hand, variable $\Upsilon_0^{v_6}(t+1)$ is true if both variables $\Upsilon_0^{v_3}(t)$ and $\Upsilon_0^{v_5}(t)$ are true.

For constant node v_i, either $\Upsilon_1^{v_i}(t) = true$ and $\Upsilon_0^{v_i}(t) = false$ or $\Upsilon_1^{v_i}(t) = false$ and $\Upsilon_0^{v_i}(t) = true$ are true for all time steps. Also, for each external node u_i, both $\Upsilon_1^{u_i}(t) = true$ and $\Upsilon_0^{u_i}(t) = true$ are true for all time steps.

To obtain the final result, we should check the existence of a control sequence which leads to $\Upsilon_{v^{\tau}[i]}^{v_i}(\tau) = true$ for all the nodes. Also, to specify and output the desired control sequence, the regression technique may be used [39].

Pseudocode of our proposed algorithm is presented in Fig 3. The source codes are available as S1 Codes.

```
function ProcessComponent(c, D)
Input: $c$ is the index of current component.
SCCs are ordered according to a topological ordering of components.
$D(v)$ is the desired state for node $v$.
    if All components are processes then
        return Current Answer
    end if
    Fill $\Upsilon_b^{v_i}(t)$ for all $v_i \in C_c$, $b \in \{0,1\}$, and time steps $t$.    ▷ By dynamic programming
    if $C_c$ is a Non-Branching Single Node Component then
        ▷ $\Upsilon$ values are already filled correctly.
        ProcessComponent(c+1, D)
    else if $C_c$ is a Branching Single Node Component then
        Let $v_i$ be the only node in $C_c$
        Let $U_t = \{b|\Upsilon_b^{v_i}(t)\}$
        Let $U = U_0 \times \cdots \times U_{\tau-1} \times (U_\tau \cap D(v_i))$
        for all $u \in U$ do
            save $\Upsilon$
            for all time step $t$ do
                Fix($v_i$, $u_t$, $t$)
            end for
            ProcessComponent(c+1, D)
            restore $\Upsilon$
        end for
    else
        if $C_c$ has no outgoing edge to other components then
            ProcessComponent(c+1, D)
        else
            Let $B$ = combination of Boolean values for component $C_c$    ▷ Calculated by Datta algorithm
            for all $b \in B$ and for all vertices $v_i \in C_c$ and time steps $t$ do
                if $b$ is compatible with $\Upsilon$ in input nodes to $C_c$ and to desired states then
                    ▷ We are fixing $\Upsilon_b^{v_i}(t)$ in accordance to $b$
                    save $\Upsilon$
                    for all $v_i \in C_c$ and $t$ do
                        Fix($v_i$, $b[v_i]$, $t$)    ▷ $b[v_i]$ is the value of $v_i$ in $b$
                    end for
                    ProcessComponent(c+1)
                    restore $\Upsilon$
                end if
            end for
        end if
    end if
end function

function Fix($v_i$, $b$, $t$)
    $\Upsilon_b^{v_i}(t) = true$
    $\Upsilon_{1-b}^{v_i}(t) = false$
end function
```

Fig 3. Pseudo-code of our proposed algorithm.

https://doi.org/10.1371/journal.pone.0215449.g003

2.2.1 Strongly connected components. In order to compute Υ values, we partition the network into strongly connected components.

A strongly connected component (SCC) of a network, is a maximal subset of network nodes, in which every node is reachable from every other one. We partition the network nodes into strongly connected components, using SCC algorithm [47]. This algorithm is one of the classical graph theory algorithms based on two depth-first searches on graph and reverse of the graph, respectively. **A topological order** on strongly connected components of a network, is an order on its components such that for every directed edge *xy* from component *x* to component *y*, *x* appears before *y* in the ordering.

We find a topological order on the components using topological sort algorithm [47]. Then, based on the size of the component and the number of outputs edges, we categorize components into three categories: 1) *non-branching single node components*, 2) *branching single node components*, 3) *multi-node components*. A branching node is a node with at least two outgoing edges and a non-branching node is a node with at most one outgoing edge.

2.2.2 Non-branching single node components. In this case, current component consists of one node v_i, and v_i has at most one outgoing edge. We simply find $\Upsilon_0^{v_i}(t)$ and $\Upsilon_1^{v_i}(t)$ for $0 \leq t \leq \tau$ from state values of incoming nodes to v_i in last time step.

For example, node v_2 in Fig 2 is a Non-Branching Single Node Component for which $\Upsilon_0^{v_2}(t)$ and $\Upsilon_1^{v_2}(t)$ for $0 \leq t \leq 2$ should be calculated based on state values of their incoming nodes v_1 and u_2 in the previous time step. Note that, interestingly, for node v_2 in time step $t = 2$ it is possible to be in state 1 and state 0. Another example with four non-branching single node components and its Υ values is present in Fig 4.

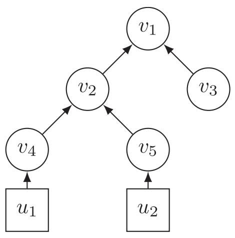
In accordance with dynamic programming arrays, we fill Υ to desirable time step τ. Since the nodes are met based on a topological sort, and all the states of the incoming nodes of the related node till the time step τ are calculated before, this calculation is possible. In time step τ, check if the related node has attained the desirable state or not. If the answer was negative, announce that there is no control sequence.

2.2.3 Branching single node components. In this case, current component consists of one node v_i, and v_i has at least two outgoing edges. Same as the previous case, we can simply find values $\Upsilon_0^{v_i}(t)$ and $\Upsilon_1^{v_i}(t)$. The difference is that since v_i has more than two outgoing edges, there are at least two other nodes in the network which their state values depend on the state value of v_i in the previous time. Thus, in some networks, the state value of v_i which is required by its outgoing neighbors may be inconsistent. An example of a branching node that is not able to attain both states 0 and 1 in a time step is present in Fig 5.

Suppose that initial state is 101100 (as it is the case in Fig 2) and we are supposed to reach to a state with $v_6 = 0$ and $v_3 = 1$ in time step $t = 4$. If we treat this node same as Non-Branching Single Node Components, we would face a problem. This algorithm announces that in time step $t = 4$, we can have $v_6 = 0$ and $v_3 = 1$, but there is no control sequence reaches to a state with $v_6 = 0$ and $v_3 = 1$ in $t = 4$. The problem is that both v_3 and v_5 in $t = 3$ are able to attain values 0 and 1, but, there is a dependency between them. This dependency does not let $v_6 = 0$ and $v_3 = 1$ in $t = 4$, simultaniously.

To deal with this problem, we fix some value for the node and recursively consider nodes for next SCCs. In this manner, we remove the dependency between the following nodes by considering different states one by one, recursively.

In other words, as we detect a dependency between states, we enumerate recursively all possible states. In the case of the branching node, if the branching node has more than one state in time *t*, we go through backtracking. Otherwise, we can branch-out some states. If the node has



a

$$v_1(t+1) = v_2(t) \wedge v_3(t)$$
$$v_2(t+1) = v_4(t) \wedge v_5(t)$$
$$v_3(t+1) = 1$$
$$v_4(t+1) = u_1(t)$$
$$v_5(t+1) = u_2(t)$$

Initial state: 01101

b

	$b = 0$	$b = 1$
t = 0:	$\Upsilon_0^{u_1}(0) = \mathit{true},$	$\Upsilon_1^{u_1}(0) = \mathit{true}$
	$\Upsilon_0^{u_2}(0) = \mathit{true},$	$\Upsilon_1^{u_2}(0) = \mathit{true}$
	$\Upsilon_0^{v_1}(0) = \mathit{true},$	$\Upsilon_1^{v_1}(0) = \mathit{false}$
	$\Upsilon_0^{v_2}(0) = \mathit{false},$	$\Upsilon_1^{v_2}(0) = \mathit{true}$
	$\Upsilon_0^{v_3}(0) = \mathit{false},$	$\Upsilon_1^{v_3}(0) = \mathit{true}$
	$\Upsilon_0^{v_4}(0) = \mathit{true},$	$\Upsilon_1^{v_4}(0) = \mathit{false}$
	$\Upsilon_0^{v_5}(0) = \mathit{false},$	$\Upsilon_1^{v_5}(0) = \mathit{true}$
t = 1:	$\Upsilon_0^{u_1}(1) = \mathit{true},$	$\Upsilon_1^{u_1}(1) = \mathit{true}$
	$\Upsilon_0^{u_2}(1) = \mathit{true},$	$\Upsilon_1^{u_2}(1) = \mathit{true}$
	$\Upsilon_0^{v_1}(1) = \mathit{false},$	$\Upsilon_1^{v_1}(1) = \mathit{true}$
	$\Upsilon_0^{v_2}(1) = \mathit{true},$	$\Upsilon_1^{v_2}(1) = \mathit{false}$
	$\Upsilon_0^{v_3}(1) = \mathit{false},$	$\Upsilon_1^{v_3}(1) = \mathit{true}$
	$\Upsilon_0^{v_4}(1) = \mathit{true},$	$\Upsilon_1^{v_4}(1) = \mathit{true}$
	$\Upsilon_0^{v_5}(1) = \mathit{true},$	$\Upsilon_1^{v_5}(1) = \mathit{true}$
t = 2:	$\Upsilon_0^{u_1}(2) = \mathit{true},$	$\Upsilon_1^{u_1}(2) = \mathit{true}$
	$\Upsilon_0^{u_2}(2) = \mathit{true},$	$\Upsilon_1^{u_2}(2) = \mathit{true}$
	$\Upsilon_0^{v_1}(2) = \mathit{true},$	$\Upsilon_1^{v_1}(2) = \mathit{false}$
	$\Upsilon_0^{v_2}(2) = \mathit{true},$	$\Upsilon_1^{v_2}(2) = \mathit{true}$
	$\Upsilon_0^{v_3}(2) = \mathit{false},$	$\Upsilon_1^{v_3}(2) = \mathit{true}$
	$\Upsilon_0^{v_4}(2) = \mathit{true},$	$\Upsilon_1^{v_4}(2) = \mathit{true}$
	$\Upsilon_0^{v_5}(2) = \mathit{true},$	$\Upsilon_1^{v_5}(2) = \mathit{true}$

c

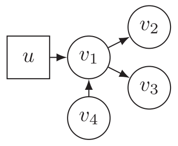
Fig 4. Example of a network with four non-branching single node components.

https://doi.org/10.1371/journal.pone.0215449.g004

one possible state, there is no need to go through backtracking. We can simply fill the next components sequentially.

2.2.4 Multi-node components. In this case, the current component consists of at least two nodes. We apply Datta algorithm in this case [24]. Datta et al fill an array $D[v_1(t), v_2(t), \ldots, v_n(t), t]$, for $t = \tau$ to $t = 0$ according to the following procedure:

$$D[v_1(t), v_2(t), \ldots, v_n(t), \tau] = \begin{cases} 1, & \text{if } [v_1(t), v_2(t), \ldots, v_n(t)] = v^\tau \\ 0, & \text{otherwise} \end{cases} \quad (1)$$



a

$$v_1(t+1) = u(t) \wedge v_4(t)$$
$$v_2(t+1) = v_1(t)$$
$$v_3(t+1) = \neg v_1(t)$$
$$v_4(t+1) = 0$$

b

Fig 5. An example of a branching node that is not able to attain both states 0 and 1 in a time step.

https://doi.org/10.1371/journal.pone.0215449.g005

$$D[v_1(t'), v_2(t'), \ldots, v_n(t'), t'] = \begin{cases} & \text{if there exists } (v^t, u) \text{such that} \\ 1, & D[v_1(t), v_2(t), \ldots, v_n(t), t] = 1 \text{ and} \\ & v^t = f(v^{t'}, u) \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where $t' = t - 1$. The table $D[v_1, \ldots, v_n, t]$ shows the possibility of reaching to the desired state, if we start from state $v_1, \ldots, v_n$ from time step t. Thus, there exists a desired solution if and only if $D[v_1(0), v_2(0), \ldots, v_n(0), 0] = 1$ holds for $v_1(0), \ldots, v_n(0)$.

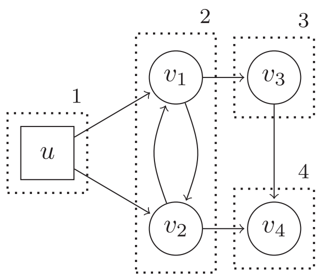
After applying Datta algorithm, we check if it is possible to attain the desirable state in time step $t = \tau$. Then, we fill Υ array accordingly. For example, node v_i in time step t may attain the desired state in time step τ with both states 0 and 1, thus, we will have $\Upsilon_0^{v_i}(t) = true$ and $\Upsilon_1^{v_i}(t) = true$. Then we partition nodes with edges going out of the component into two categories, nodes with at most one outgoing edge and nodes with at least two outgoing edges (branching node). Nodes with at most one outgoing edge are easy to handle, however, we treat branching nodes which may attain both states 0 and 1 in the same time step, like *branching single node components*. In both cases, we fix values for this component and go for the next component recursively.

2.2.5 Complexity analysis. Let $C_1, \ldots, C_{\#C}$ be the strongly connected components of G, and $\sigma_t(C_i)$ be the number of valid states of component C_i at time step t. Our algorithm enumerates all the network states, but non-branching single node components and branching single node components.

Datta algorithm finds all potentially desirable network states backward in time. To find all desirable states in time step t, it generates all potential states for the network, which takes $O(2^n)$ time, for an n node network. Then, for each potential state, it calculates next state and checks whether it leads to the desirable state by searching it within desirable states of the time step $t + 1$. Let $T(G)$ be the time required to calculate the next state of a network's state. The amount of computation for time step t, for each potential state is $T(G)$ for calculating the next step in addition to searching it, which takes $\lg \sigma_{t+1}(G)$. Datta performs these two operations for all 2^n potential states. Therefore, Datta algorithm's running time is $O(\sum_t 2^n(T(G) + \lg \sigma_t(G))) = O(\tau 2^n T(G))$. Note that, calculation of $T(G)$ requires computation of Boolean functions for each node, and if functions are given as input, their calculation asymptotic time is not more than the size of the input, which is acceptable.

On the other hand, our algorithm proceeds component by component. For each component, we consider all the $\sigma_t(C_i)$ states for all time steps t. However, in the following three cases, we do not enumerate all the states:

- For non-branching single node components, we compress states. In other words, we check further components not by considering two possible states for these components recursively. However, since network states of following components do not depend on different values of these components, we make our enumerations by half, for each of these components. Thus, if we have k such components, our running time will be reduced by a factor of $\frac{1}{2^k}$, in comparison to normal dynamic programming.
- We can check produced network states to be compatible with desired states when considering a component, and we do not delay it until enumerating all the following possible states. We cleverly do this, by not backtracking on the time step $t = M$, but only on time steps $t = 1$



a. The network structure. SCCs are shown as dotted boxes. A topological ordering is indicated by numbers beside boxes.

$$\begin{aligned} v_1(t+1) &= u \vee v_2(t) \\ v_2(t+1) &= \neg u \wedge v_1(t) \\ v_3(t+1) &= v_1(t) \\ v_4(t+1) &= v_2(t) \wedge v_3(t) \end{aligned}$$

b. The transition function of the network.

$$\begin{aligned} v_1(0) &= 0 \qquad & v_1(3) &= 1 \\ v_2(0) &= 0 \qquad & v_2(3) &= 0 \\ v_3(0) &= 0 \qquad & v_3(3) &= 0 \\ v_4(0) &= 0 \qquad & v_4(3) &= 1 \end{aligned}$$

c. Initial and desired state for the network.

Fig 6. Example of a network, its transition function, and its initial and desired states.

https://doi.org/10.1371/journal.pone.0215449.g006

through $M - 1$. Since no further network state depends on time step $t = M$ of any component, we may only check this compatibility and not backtracking on them.

- Leaf components with out-degree 0 are independent, and not required to be checked recursively, but sequentially.

Considering above notes, our running time is $O(\Sigma_t \Pi_{i:I(t)} \sigma_t(C_i) T(C_i))$. Note that since $2^n \geq \sigma_t(G) \geq \Pi_i \sigma_t(C_i)$ and $T(G) \geq \Sigma_i T(C_i)$, our running time is always better than running time of dynamic programming.

2.3 Example

Consider the network shown in Fig 6. This network consists of one external node u and four internal nodes v_1, v_2, v_3, and v_4. Transition function and initial/desired state for the network is shown in Fig 6.

Based on the proposed algorithm, we first partition network nodes to SCCs. SCCs of the network of Fig 6 are $S_1 = \{u\}$, $S_2 = \{v_1, v_2\}$, $S_3 = \{v_3\}$ and $S_4 = \{v_4\}$. SCCs are shown in Fig 6 as dotted boxes.

First we process SCC $S_1 = \{u\}$. Since node u is an external node, in every time step t, u can attain both states 0 and 1, thus, $\Upsilon_b^u(t) = true$ for all $0 \leq t \leq 3$ and $b \in \{0, 1\}$. This SCC is a branching single node component. Thus, we fix $\Upsilon_b^u(t)$ for all $0 \leq t \leq 3$ for all different combinations of assignment values, as it is represented in Table 1. In the recursion, if correct solution is found, the algorithm reports the solution and halts. Otherwise, the algorithm fixes another combination and goes for next SCC, recursively. For this example, suppose that the algorithm fixes row #2 of the Table 1 for component S_1, and then goes through next SCCs.

Next SCC, according to the topological order of SCCs, is $S_2 = \{v_1, v_2\}$. The logic behind behavior of S_2 is that, if $u(t) = 1$, then, we have $v_1(t + 1) = 1$ and $v_2(t + 1) = 0$, and if $u = 0$, v_1 and v_2 exchange their state at the next time step. S_2 is a multi-node component. Thus, the algorithm first executes a Datta algorithm on this SCC (line 28 of Fig 3). The result of Datta algorithm is shown in Table 2. As it is shown in Table 2, since we already fixed one state over all time steps for SCC S_1, there is only one possible state for the S_2. For SCC S_2, after calculating Υ table, since S_2 is a multi-node component, the algorithm fixes this state and goes through next SCCs.

Next SCC, $S_3 = \{v_3\}$, is a single node non-branching component. Thus, the algorithm calculates $\Upsilon_b^{v_3}(t)$ for all $1 \leq t \leq 3$ and $b \in \{0, 1\}$. By the definition of transition function (Fig 6), $\Upsilon_b^{v_3}(t+1) = \Upsilon_b^{v_1}(t)$. Then, we proceed to next SCC.

Table 1. State space for the SCC S_1 for the network represented in Fig 6.

	t = 0	t = 1	t = 2	t = 3
#1	0	0	0	0
#2	1	0	0	0
#3	0	1	0	0
#4	1	1	0	0
#5	0	0	1	0
#6	1	0	1	0
#7	0	1	1	0
#8	1	1	1	0
#9	0	0	0	1
#10	1	0	0	1
#11	0	1	0	1
#12	1	1	0	1
#13	0	0	1	1
#14	1	0	1	1
#15	0	1	1	1
#16	1	1	1	1

https://doi.org/10.1371/journal.pone.0215449.t001

Next SCC is $S_4 = \{v_4\}$. So far, the value of Υ table is shown in Table 3. Like SCC S_3, SCC S_4 is a single node non-branching component. Thus, the algorithm first calculates Υ table. According to the transition function (Fig 6), $\Upsilon_1^{v_4}(t+1) = \Upsilon_1^{v_2}(t) \wedge \Upsilon_1^{v_3}(t)$ and $\Upsilon_0^{v_4}(t+1) = \Upsilon_0^{v_2}(t) \vee \Upsilon_0^{v_3}(t)$. Now, the algorithm processed all the SCCs, thus, it reports the result.

Note that, in any of the recursion steps, if the algorithm does not find an answer that matches to the desired state, it traces back to the SCCs that are not single node non-branching, tries the next state from the state space. In the case of the current example, the algorithm will

Table 2. Value of Υ table for SCC S_2 for the network represented in Fig 6.

		t = 0	t = 1	t = 2	t = 3
v_1	b = 0	$\Upsilon_0^{v_1}(t) = \text{true}$	$\Upsilon_0^{v_1}(t) = \text{false}$	$\Upsilon_0^{v_1}(t) = \text{true}$	$\Upsilon_0^{v_1}(t) = \text{false}$
	b = 1	$\Upsilon_1^{v_1}(t) = \text{false}$	$\Upsilon_1^{v_1}(t) = \text{true}$	$\Upsilon_1^{v_1}(t) = \text{false}$	$\Upsilon_1^{v_1}(t) = \text{true}$
v_2	b = 0	$\Upsilon_0^{v_2}(t) = \text{true}$	$\Upsilon_0^{v_2}(t) = \text{true}$	$\Upsilon_0^{v_2}(t) = \text{false}$	$\Upsilon_0^{v_2}(t) = \text{true}$
	b = 1	$\Upsilon_1^{v_2}(t) = \text{false}$	$\Upsilon_1^{v_2}(t) = \text{false}$	$\Upsilon_1^{v_2}(t) = \text{true}$	$\Upsilon_1^{v_2}(t) = \text{false}$

https://doi.org/10.1371/journal.pone.0215449.t002

Table 3. Value of Υ table after processing SCCs S_1, S_2, and S_3 and before processing SCC S_4 for the network represented in Fig 6.

		t = 0	t = 1	t = 2	t = 3
u	b = 0	$\Upsilon_0^{u}(t) = \text{false}$	$\Upsilon_0^{u}(t) = \text{true}$	$\Upsilon_0^{u}(t) = \text{true}$	$\Upsilon_0^{u}(t) = \text{true}$
	b = 1	$\Upsilon_1^{u}(t) = \text{true}$	$\Upsilon_1^{u}(t) = \text{false}$	$\Upsilon_1^{u}(t) = \text{false}$	$\Upsilon_1^{u}(t) = \text{false}$
v_1	b = 0	$\Upsilon_0^{v_1}(t) = \text{true}$	$\Upsilon_0^{v_1}(t) = \text{false}$	$\Upsilon_0^{v_1}(t) = \text{true}$	$\Upsilon_0^{v_1}(t) = \text{false}$
	b = 1	$\Upsilon_1^{v_1}(t) = \text{false}$	$\Upsilon_1^{v_1}(t) = \text{true}$	$\Upsilon_1^{v_1}(t) = \text{false}$	$\Upsilon_1^{v_1}(t) = \text{true}$
v_2	b = 0	$\Upsilon_0^{v_2}(t) = \text{true}$	$\Upsilon_0^{v_2}(t) = \text{true}$	$\Upsilon_0^{v_2}(t) = \text{false}$	$\Upsilon_0^{v_2}(t) = \text{true}$
	b = 1	$\Upsilon_1^{v_2}(t) = \text{false}$	$\Upsilon_1^{v_2}(t) = \text{false}$	$\Upsilon_1^{v_2}(t) = \text{true}$	$\Upsilon_1^{v_2}(t) = \text{false}$
v_3	b = 0	$\Upsilon_0^{v_3}(t) = \text{true}$	$\Upsilon_0^{v_3}(t) = \text{true}$	$\Upsilon_0^{v_3}(t) = \text{false}$	$\Upsilon_0^{v_3}(t) = \text{true}$
	b = 1	$\Upsilon_1^{v_3}(t) = \text{false}$	$\Upsilon_1^{v_3}(t) = \text{false}$	$\Upsilon_1^{v_3}(t) = \text{true}$	$\Upsilon_1^{v_3}(t) = \text{false}$

https://doi.org/10.1371/journal.pone.0215449.t003 traceback for the next state for S_2, which does not have any other, and then traces back to S_1 for the next state.

2.4 Proof of the correctness of the proposed algorithm

In a general view, the proposed algorithm enumerates all possible states, but, removes some useless states. There are two extreme approaches to enumerate the states. One approach is to recursively enumerate them node-by-node. The other extreme, like Datta algorithm, is to fill an array that represents reachability of the states, time step-by-time step. Our algorithm is somewhat in-between, it fills arrays for SCCs, and fix them recursively.

We provide a stronger result for the proposed algorithm. We let logical formulas that describe states of the nodes in time step $t + 1$, to not only include variables of the time step t, but also variables of time steps 0 to t. We name these networks as extended networks. Also, we name single node non-branching SCCs as simple and the other SCCs as hard components.

Consider all the non-branching single node components that proceed an SCC S in the topological order. All easy components may have 0 or more input edges, but only one output edge. Note that, if there is a single node non-branching SCC without output edge, it could be removed from the network. After finishing the process of the network, these nodes could be checked for its consistency with the desired state at the end. These simple component nodes make a rooted tree which is directed toward the root. A schematic view is shown in Fig 7.

We can replace them in all logical formulas by the state of their input nodes in earlier time steps, or in some cases by nodes' initial states. As an example, consider the schematic view of the network shown in Fig 8. In this network, we can replace v_3 in all transition formulas by its inputs and change the network accordingly, as it is shown in Fig 8. The resulting network is a simple component-free extended network.

By this technique, we will have an extended network without any simple SCC. We provide a proof of correctness of our algorithm on these extended networks. While we proved the correctness of the provided algorithm, on these extended networks, as a direct result, the correctness of our algorithm on hard components follows immediately. Also, it is shown that simple components are derived deterministically from the state of nodes within hard components. This shows the correctness of the proposed algorithm.

Considering a simple component-free extended network, the proposed algorithm processes it SCC by SCC. The algorithm first generates all possible states for each SCC, and then enumerates them one by one and goes for the next SCC, recursively. In other words, instead of enumerating all the states for the network node by node, it makes groups of nodes (SCCs) and

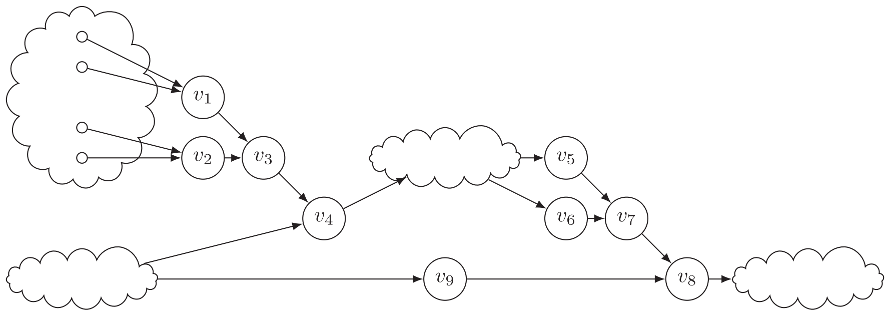
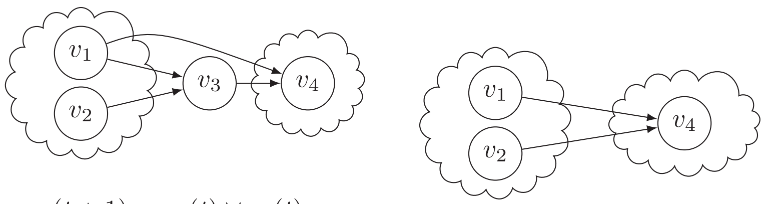


Fig 7. A schematic view of the topology of single node non-branching components between other SCCs of a network. Cloudy nodes are either branching single node or multi-node components (hard components). Normal circles represent single node non-branching components (simple components). Simple components make a rooted tree toward their roots.

https://doi.org/10.1371/journal.pone.0215449.g007



$$v_3(t+1) = v_1(t) \vee v_2(t)$$
$$v_4(t+1) = v_3(t) \wedge v_2(t)$$

$$v_4(t+1) = (v_1(t-1) \vee v_2(t)) \wedge v_2(t)$$

a. Example of a network containing one simple component.

b. Extended network obtained by removing simple components.

Fig 8. Example of applying the transformation of removing simple components from a network to obtain a simple component-free extended network.

https://doi.org/10.1371/journal.pone.0215449.g008

enumerates states for them. It is clear in this case that, the algorithm enumerates states just like a brute force algorithm. Since the brute force algorithm enumerates and checks all the states, its correctness is obvious from its definition. Note that, the proposed algorithm is not just one brute force, but, we reduced the correctness proof of our algorithm to the correctness of a brute force algorithm.

3 Application to biomolecular network control

3.1 Drosophila Melanogaster's network

In this section, we present the steps of applying the proposed algorithm on Drosophila melanogaster subnetwork [41], which contains 15 nodes. As it can be seen in Fig 9, three external nodes U1, U2 and U3 are added to this network. Evolving functions regarding the added control sequences are shown in Table 4.

First, we partitioned the network's graph into strongly connected components. As it could be seen in Fig 9, graph has only one *multi-node component*, which is shown through dotted lines. Then, nodes are ordered based on topological sort and the result is shown as numberings beside each node in Fig 9.

We treat graph nodes one by one according to their topological ordering. First, "SLP" node is met. This node is a *none-branching single node component*; therefore, we compute Υ for this node till $t = \tau$. Since this node is a constant node (a node without entering edge), it will always keep its initial state. The Υ concerning this node will be filled to this constant value. Then, for time step τ, we check that if this node has attained the desirable state or not. If not, it would be announced that there exist no desired control sequence, and the algorithm terminates.

Next, the second node in topological sort that is the "en" node will be visited. This node is also a *none-branching single node component*, thus, it is processed the same as "SLP" node.

Then the "EN" node is considered. It is a branching node, but since it has no control node before itself, it can attain only one state in every time step. Therefore this node is treated like "SLP" and "EN" nodes. For "ci" and "CI" nodes, the very procedure will go.

Now we reach a *multi-node component*. For this strongly connected component, the dynamic programming is computed, then it is checked if there is a possibility to attain the desired state in time step τ for the nodes inside this component or not. If we can attain the desirable state within the time step τ, the Υ for each node would be filled. Then, branching nodes which have outgoing edges from the component, would be handled as "CIR" node to check if these nodes are possible to attain two states of 0 and 1 in a time step. If they could, a

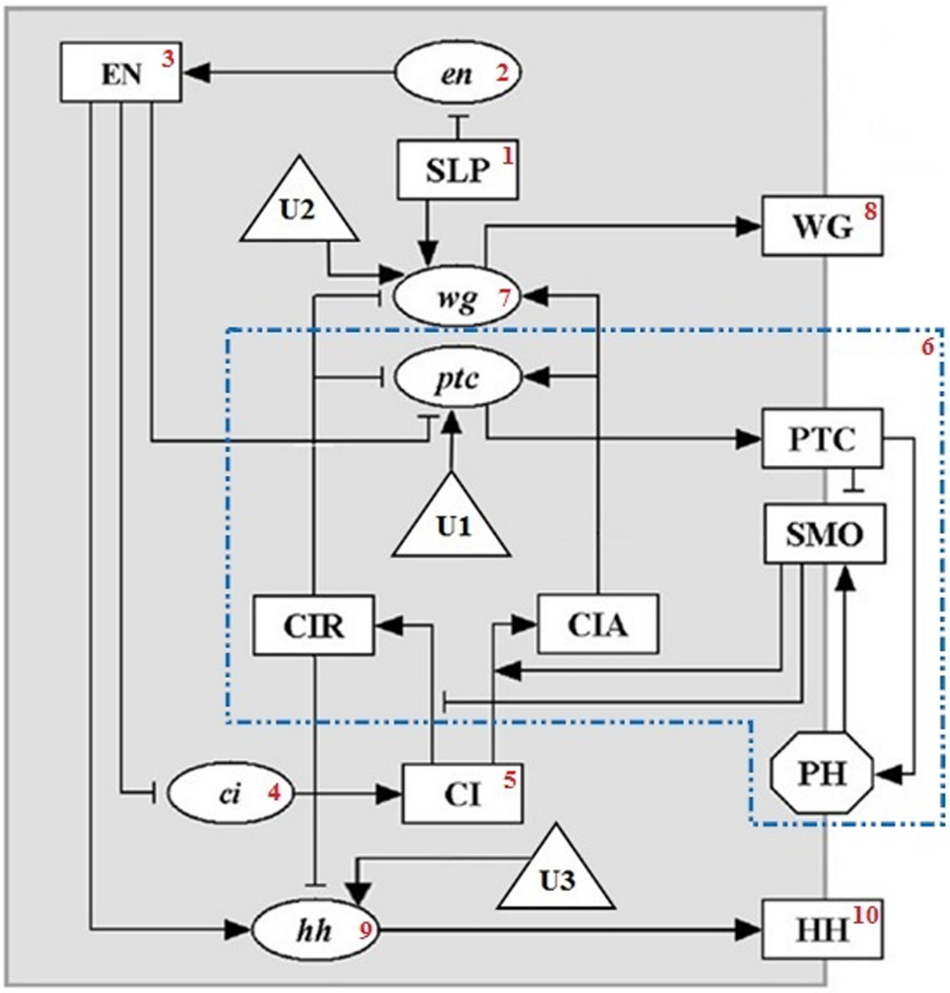


Fig 9. Boolean network model of Drosophila melanogaster. Dotted lines indicate a *multi-node component*, and the numberings beside nodes indicate topological orders. Nodes indicated as rectangles, i.e. nodes U1, U2, and U3, are external nodes.

https://doi.org/10.1371/journal.pone.0215449.g009

state sequence that we can attain the desirable state, considered and the algorithm would be recalled for the network's remaining nodes.

Then nodes "wg", "WG", "hh" and "HH" are considered. these nodes are *none-branching single node components*. Note that, if for example "CIR" with two state sequences can attain the desirable state, and for the first state sequence, it would not be possible for the next nodes (e.g. "wg") to attain the desirable state, the algorithm will return and for the sequences of the second state of "CIR", it would check the remaining nodes. In the case that all the nodes were possible to attain the desirable state, it will be announced that there exists a control sequence that causes attaining the desirable state in time step τ.

3.2 T-Cell receptor kinetics' network

We apply the proposed algorithm to the Boolean network model of T-cell receptor kinetics [40]. The *multi-node component* and the order of nodes based on a topological sort are depicted in Fig 10. As it is obvious in this figure, this model of the Boolean network has 40

Table 4. Evolution functions for the Boolean network model of Drosophila melanogaster.

Node	Boolean updating function
SLP	$SLP^{t+1} = SLP^{t}$
wg	$wg^{t+1} = ((CIA^{t} \wedge SLP^{t} \wedge \neg CIR^{t}) \vee (wg^{t} \wedge (CIA^{t} \vee SLP^{t}) \wedge \neg CIR^{t})) \wedge U2^{t}$
WG	$WG^{t+1} = wg^{t}$
en	$en^{t+1} = \neg SLP^{t}$
EN	$EN^{t+1} = en^{t}$
hh	$hh^{t+1} = EN^{t} \wedge \neg CIR^{t} \wedge U3^{t}$
HH	$HH^{t+1} = hh^{t}$
ptc	$ptc^{t+1} = CIA^{t} \wedge \neg EN^{t} \wedge \neg CIR^{t} \wedge U1^{t}$
PTC	$PTC^{t+1} = ptc^{t} \wedge PTC^{t}$
PH	$PH^{t+1} = PTC^{t}$
SMO	$SMO^{t+1} = \neg PTC^{t}$
ci	$cici^{t+1} = \neg EN^{t}$
CI	$CI^{t+1} = ci^{t}$
CIA	$CIA^{t+1} = CI^{t} \wedge SMO^{t}$
CIR	$CIR^{t+1} = CI^{t} \wedge \neg SMO^{t}$

https://doi.org/10.1371/journal.pone.0215449.t004

genes and one *multi-node component*. Three external nodes U1, U2, and U3, are added to the Boolean network.

In T-cell receptor kinetics, the network which can be seen in Fig 10, arrows with pointed heads represent activation and dashed arrows with bar heads represent inhibition (the dashed arrows represent "not" which is related to that node). The network is represented as a network of "or"s of "and"s. Thus, large filled circles representing "and" of their inputs, while when edges are going to a node, state of the node is determined according to the "or" of its incoming edges. For example

$$\mathrm{Fyn}(t+1) = (\mathrm{CD45}(t) \wedge \mathrm{TCRbind}(t)) \vee (\mathrm{CD45}(t) \wedge \mathrm{Lck}(t))$$

Our suggested algorithm on the Boolean network model of Drosophila melanogaster was implemented for an initial state (in time step $t = 0$) and a desired state (in time step $t = 6$), which is depicted in Table 5. It is noteworthy to say that there exists a control sequence to reach the desirable state in time step $t = 6$. Also, the Datta et al algorithm was implemented in this dataset, for comparison. Note that, since this dataset has 22 edges and 15 internal nodes, the algorithm presented by Akutsu et al requires a tree which has 8 fewer edges from this network ($H = 8$).

3.3 Comparison with previous methods

With initial and desired states that are mentioned in Table 6, we evaluated the proposed algorithm on the Boolean network of Drosophila melanogaster and compared it with previous algorithms. This dataset contains 22 edges and 15 internal nodes, the algorithm presented by Akutsu et al requires a tree which has 8 fewer edges from this network. Thus all combinations of 8 edges are to be removed from the graph that makes it very slow for this case.

The initial state and desired state for T-cell receptor model is shown in Fig 10. This dataset has 40 internal nodes ($n = 40$) and 3 external nodes ($m = 3$). We compared the proposed algorithm with the algorithm of Datta et al and Akutsu et al on these initial and desired conditions. Note that there is a control sequence for the above mentioned desirable state in time step $t = 5$.

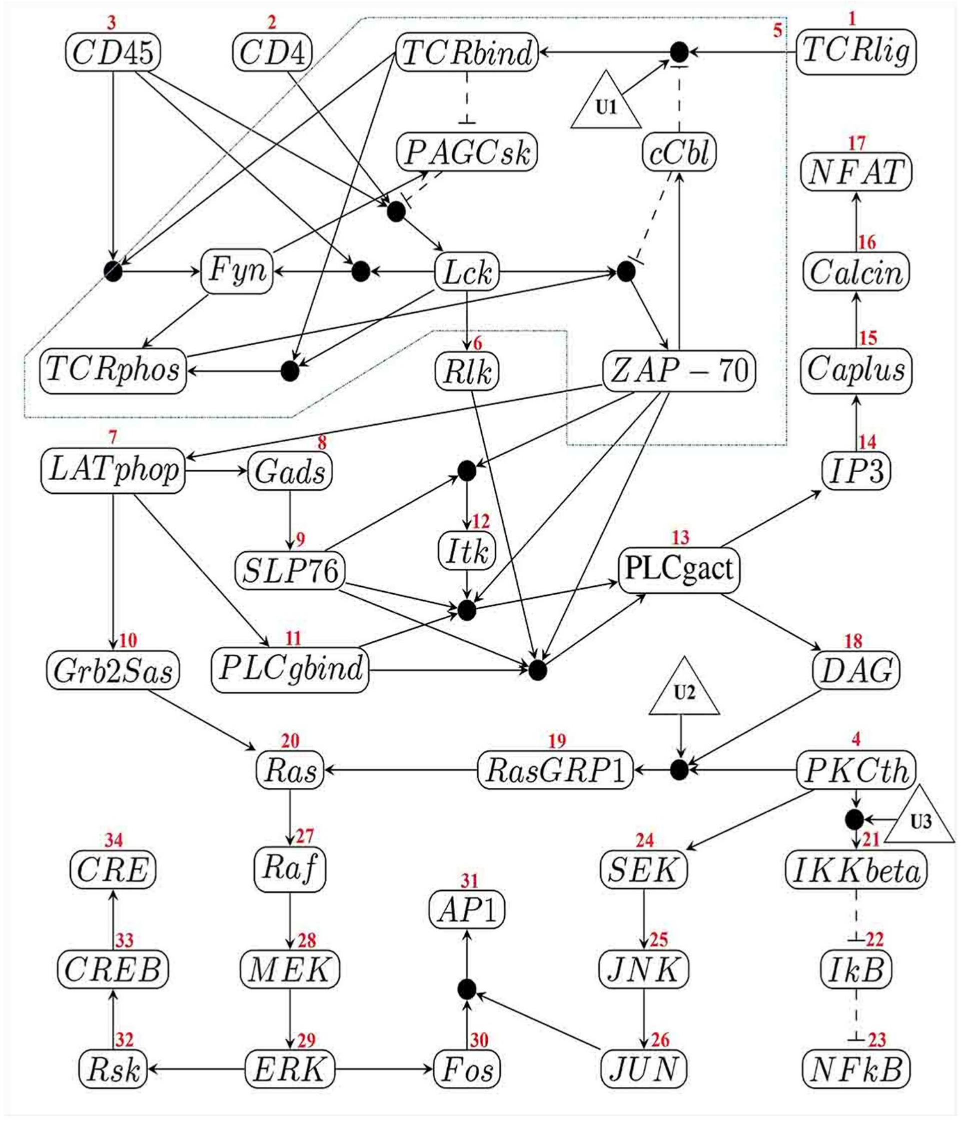


Fig 10. Boolean network model of T-cell receptor. Network's graph has one *multi-node component*. The numberings beside each node indicate their topological ordering.

https://doi.org/10.1371/journal.pone.0215449.g010

Akutsu et al [39] first provide a polynomial time algorithm for directed trees. However, their algorithm does not work on general graphs. For general graphs, they remove some edges from the graph to obtain a tree and then search for all possible solutions. Among all the solutions, if one is compatible with the removed edges, their algorithm reports the solution. Note that, since they have to obtain all solutions for the obtained trees, their algorithm on general graphs is not a polynomial time algorithm anymore.

The algorithm presented by Akutsu et al requires a tree which has 19 fewer edges from this network ($H = 19$). Time complexity of this algorithm is $O(2^{H(\tau+1)}(n+m)\tau)$, which is worse than running time of Datta et al's algorithm with time complexity $O(2^{(2n+m)}\tau)$

In some applications, not only a valid control strategy is needed, but also among valid control strategies (that lead to the desired state), the one with some optimality is preferred. To address this requirement we added the ability to compare valid control strategies for the

Table 5. Initial state and desirable state of each node for Drosophila melanogaster network's graph.

Node	Initial State ($t = 0$)	Desired State ($t = 6$)
en	0	1
EN	0	1
SLP	0	0
wg	1	0
WG	1	0
ptc	0	0
PTC	0	0
CIA	0	0
CIR	1	0
ci	0	0
CI	1	0
SMO	0	1
PH	0	0
hh	1	0
HH	0	1

https://doi.org/10.1371/journal.pone.0215449.t005

proposed algorithm. Actually, the proposed algorithm finds a valid control strategy that has minimum switching of control nodes, i.e. from 1 to 0 or 0 to 1. Adding this ability to the proposed algorithm makes it more like an optimization algorithm.

The results of the comparison of our algorithm and previous algorithms implemented on a PC with Dual-Core 2.5GHz CPU, 2G RAM are depicted in Table 7.

Table 6. Initial states and desirable states of each node in T-cell receptor kinetics network graph.

Node	Initial State ($t = 0$)	Desired State ($t = 5$)	Node	Initial State ($t = 0$)	Desired State ($t = 5$)
CD45	1	1	Grb2Sas	0	0
CD4	0	0	PLCgbind	0	0
TCRbind	0	1	DAG	0	0
TCRlig	1	1	Ras	0	0
PAGCsk	0	1	RasGRP1	0	0
cCbl	0	0	PKCth	1	1
NFAT	0	0	CRE	0	0
Fyn	0	1	Raf	0	0
Lck	1	0	SEK	0	1
Calcin	0	0	IKKbeta	0	1
TCRphos	0	1	CREB	1	0
Rlk	0	0	MEK	0	1
ZAP-70	0	0	AP1	0	0
Caplus	0	0	JNK	0	1
LATphop	0	0	IKB	1	0
Gads	0	0	Rsk	0	0
IP3	0	0	ERK	0	0
SLP76	0	0	Fos	1	0
Itk	0	0	JUN	0	1
PLCgact	1	0	NFkB	0	1

https://doi.org/10.1371/journal.pone.0215449.t006

Table 7. Comparison of algorithms for a control problem on the T-cell network's receptor kinetics (Table 6) and Drosophila melanogaster's network (Table 5).

Algorithm	T-Cell	Drosophila
Algorithm of Datta et al	Over 12 days	16.5 hours
Algorithm of transforming the graph into a rooted tree structure	Over 27 days	Over 2 days
Proposed (our) algorithm	1.83 Seconds	0.92 Seconds

https://doi.org/10.1371/journal.pone.0215449.t007

4 Conclusion

In this paper, we proposed an algorithm for network control problem that improves the running time of previously provided algorithms. The extent of improvements and efficiency of our proposed algorithm depends on the size of the *multi-node components* of the network and also on the positioning of the control nodes or more generally, on the accessibility of both states of 0 and 1 in the same time steps for branching nodes.

Since in the proposed algorithm, the nodes are met based on a topological order, the states of the entering nodes of each node are calculated before visiting that node. As a result, in each node, it would be possible to handle states from the initial time step to the desirable time. This would cause early detection of the case that if a node is not possible to attain the desirable state, thus, there be no need to check all the nodes to time step τ, and the lack of a control sequence is reported early. Although the proposed algorithm shows to be efficient on real control networks, however, if all the nodes of the network are in one strongly connected components, the proposed algorithm is not performing well anymore. Handling this case could be future work.

Supporting information

S1 Codes. Implementation of the algorithms. This file contains the implementation of the provided algorithm as well as two implementations of the Datta et al algorithm. This file also contains sample inputs.
(ZIP)

Author Contributions

Conceptualization: Mohammad Moradi, Sama Goliaei.

Data curation: Mohammad Moradi.

Formal analysis: Mohammad Moradi, Sama Goliaei, Mohammad-Hadi Foroughmand-Araabi.

Investigation: Mohammad Moradi, Sama Goliaei.

Methodology: Mohammad Moradi, Sama Goliaei.

Project administration: Sama Goliaei.

Resources: Mohammad Moradi.

Software: Mohammad Moradi, Mohammad-Hadi Foroughmand-Araabi.

Supervision: Sama Goliaei.

Validation: Mohammad Moradi, Sama Goliaei.

Writing – original draft: Mohammad Moradi.

Writing – review & editing: Mohammad Moradi, Sama Goliaei, Mohammad-Hadi Foroughmand-Araabi.

References

1. Alberts B, Johnson A, Lewis J, Morgan D, Raff M. Molecular Biology of the Cell. Garland Science; 2014.
2. Li F, Long T, Lu Y, Ouyang Q, Tang C. The yeast cell-cycle network is robustly designed. Proceedings of the National Academy of Sciences of the United States of America. 2004; 101(14):4781–4786. https://doi.org/10.1073/pnas.0305937101 PMID: 15037758
3. Tyson JJ, Chen K, Novak B. Network dynamics and cell physiology. Nature Reviews Molecular Cell Biology. 2001; 2(12):908–916. https://doi.org/10.1038/35103078 PMID: 11733770
4. Batchelor E, Loewer A, Lahav G. The ups and downs of p53: understanding protein dynamics in single cells. Nature Reviews Cancer. 2009; 9(5):371–377. https://doi.org/10.1038/nrc2604 PMID: 19360021
5. Choi M, Shi J, Jung SH, Chen X, Cho KH. Attractor landscape analysis reveals feedback loops in the p53 network that control the cellular response to DNA damage. Science Signaling. 2012; 5(251):ra83–ra83. https://doi.org/10.1126/scisignal.2003363 PMID: 23169817
6. Geva-Zatorsky N, Rosenfeld N, Itzkovitz S, Milo R, Sigal A, Dekel E, et al. Oscillations and variability in the p53 system. Molecular Systems Biology. 2006; 2(1). https://doi.org/10.1038/msb4100068 PMID: 16773083
7. Joh RI, Weitz JS. To lyse or not to lyse: transient-mediated stochastic fate determination in cells infected by bacteriophages. PLoS Computational Biology. 2011; 7(3):e1002006. https://doi.org/10.1371/journal.pcbi.1002006 PMID: 21423715
8. Murrugarra D, Veliz-Cuba A, Aguilar B, Arat S, Laubenbacher RC. Modeling stochasticity and variability in gene regulatory networks. EURASIP Journal on Bioinformatics and Systems Biology. 2012; 2012:5. https://doi.org/10.1186/1687-4153-2012-5 PMID: 22673395
9. Zeng L, Skinner SO, Zong C, Sippy J, Feiss M, Golding I. Decision making at a subcellular level determines the outcome of bacteriophage infection. Cell. 2010; 141(4):682–691. https://doi.org/10.1016/j.cell.2010.03.034 PMID: 20478257
10. Saadatpour A, Wang RS, Liao A, Liu X, Loughran TP, Albert I, et al. Dynamical and structural analysis of a T-cell survival network identifies novel candidate therapeutic targets for large granular lymphocyte leukemia. PLoS Computational Biology. 2011; 7(11):e1002267. https://doi.org/10.1371/journal.pcbi.1002267 PMID: 22102804
11. Zhang R, Shah MV, Yang J, Nyland SB, Liu X, Yun JK, et al. Network model of survival signaling in large granular lymphocyte leukemia. Proceedings of the National Academy of Sciences. 2008; 105(42):16308–16313. https://doi.org/10.1073/pnas.0806447105
12. Shih K, Chen R, Hu R, Liu F, Chen H, Tsai JJ. Prediction of gene regulatory networks using differential expression of cDNA microarray data. In: IEEE Sixth International Symposium on Multimedia Software Engineering. IEEE; 2004. p. 378–385.
13. Amaral LA, Díaz-Guilera A, Moreira AA, Goldberger AL, Lipsitz LA. Emergence of complex dynamics in a simple model of signaling networks. Proceedings of the National Academy of Sciences of the United States of America. 2004; 101(44):15551–15555. https://doi.org/10.1073/pnas.0404843101 PMID: 15505227
14. Pandey S, Wang RS, Wilson L, Li S, Zhao Z, Gookin TE, et al. Boolean modeling of transcriptome data reveals novel modes of heterotrimeric G-protein action. Molecular Systems Biology. 2010; 6(1). https://doi.org/10.1038/msb.2010.28 PMID: 20531402
15. Helikar T, Konvalina J, Heidel J, Rogers JA. Emergent decision-making in biological signal transduction networks. Proceedings of the National Academy of Sciences. 2008; 105(6):1913–1918. https://doi.org/10.1073/pnas.0705088105
16. Kim JR, Kim J, Kwon YK, Lee HY, Heslop-Harrison P, Cho KH. Reduction of complex signaling networks to a representative kernel. Science Signaling. 2011; 4(175):ra35–ra35. https://doi.org/10.1126/scisignal.2001390 PMID: 21632468
17. Helikar T, Kochi N, Konvalina J, Rogers JA. Boolean modeling of biochemical networks. The Open Bioinformatics Journal. 2011; 5:16–25. https://doi.org/10.2174/1875036201105010016
18. Bornholdt S. Boolean network models of cellular regulation: prospects and limitations. Journal of the Royal Society Interface. 2008; 5(Suppl 1):S85–S94.
19. Kitano H. Computational systems biology. Nature. 2002; 420(6912):206–210. https://doi.org/10.1038/nature01254 PMID: 12432404

20. Kitano H. Cancer as a robust system: implications for anticancer therapy. Nature Reviews Cancer. 2004; 4(3):227–235. https://doi.org/10.1038/nrc1300 PMID: 14993904

21. Erler JT, Linding R. Network medicine strikes a blow against breast cancer. Cell. 2012; 149(4):731–733. https://doi.org/10.1016/j.cell.2012.04.014 PMID: 22579276

22. Lee MJ, Albert SY, Gardino AK, Heijink AM, Sorger PK, MacBeath G, et al. Sequential application of anticancer drugs enhances cell death by rewiring apoptotic signaling networks. Cell. 2012; 149(4):780–794. https://doi.org/10.1016/j.cell.2012.03.031 PMID: 22579283

23. Wang W. Therapeutic hints from analyzing the attractor landscape of the p53 regulatory circuit. Science signaling. 2013; 6(261):pe5–pe5. https://doi.org/10.1126/scisignal.2003820 PMID: 23386744

24. Datta A, Choudhary A, Bittner ML, Dougherty ER. External control in Markovian genetic regulatory networks. Machine Learning. 2003; 52(1-2):169–191. https://doi.org/10.1023/A:1023909812213

25. Datta A, Choudhary A, Bittner ML, Dougherty ER. External control in Markovian genetic regulatory networks: the imperfect information case. Bioinformatics. 2004; 20(6):924–930. https://doi.org/10.1093/bioinformatics/bth008 PMID: 14751971

26. Pal R, Datta A, Bittner ML, Dougherty ER. Intervention in context-sensitive probabilistic Boolean networks. Bioinformatics. 2005; 21(7):1211–1218. https://doi.org/10.1093/bioinformatics/bti131 PMID: 15531600

27. Yousefi MR, Datta A, Dougherty ER. Optimal intervention strategies for therapeutic methods with fixed-length duration of drug effectiveness. IEEE Transactions on Signal Processing. 2012; 60(9):4930–4944. https://doi.org/10.1109/TSP.2012.2202114

28. Yousefi MR, Dougherty ER. Intervention in gene regulatory networks with maximal phenotype alteration. Bioinformatics. 2013; 29(14):1758–1767. https://doi.org/10.1093/bioinformatics/btt242 PMID: 23630177

29. Yousefi MR, Dougherty ER. A comparison study of optimal and suboptimal intervention policies for gene regulatory networks in the presence of uncertainty. EURASIP Journal on Bioinformatics and Systems Biology. 2014; 2014:6. https://doi.org/10.1186/1687-4153-2014-6 PMID: 24708650

30. Yousefi MR, Datta A, Dougherty E. Optimal intervention in Markovian gene regulatory networks with random-length therapeutic response to antitumor drug. IEEE Transactions on Biomedical Engineering. 2013; 60(12):3542–3552. https://doi.org/10.1109/TBME.2013.2272891 PMID: 23864151

31. Gao B, Li L, Peng H, Kurths J, Zhang W, Yang Y. Principle for performing attractor transits with single control in Boolean networks. Physical Review E. 2013; 88(6):062706. https://doi.org/10.1103/PhysRevE.88.062706

32. Qiu Y, Tamura T, Ching WK, Akutsu T. On control of singleton attractors in multiple Boolean networks: integer programming-based method. BMC Systems Biology. 2014; 8(1):1.

33. Langmead CJ, Jha SK. Symbolic approaches for finding control strategies in Boolean networks. Journal of Bioinformatics and Computational Biology. 2009; 7(02):323–338. https://doi.org/10.1142/S0219720009004084 PMID: 19340918

34. Liu Y, Chen H, Wu B. Controllability of Boolean control networks with impulsive effects and forbidden states. Mathematical Methods in the Applied Sciences. 2014; 37(1):1–9. https://doi.org/10.1002/mma.3045

35. Hu M, Shen L, Zan X, Shang X, Liu W. An efficient algorithm to identify the optimal one-bit perturbation based on the basin-of-state size of Boolean networks. Scientific Reports. 2016; 6:26247. https://doi.org/10.1038/srep26247 PMID: 27196530

36. Poret A, Boissel JP. An in-silico target identification using Boolean network attractors: avoiding pathological phenotypes. Comptes Rendus Biologies. 2014; 337(12):661–678. https://doi.org/10.1016/j.crvi.2014.10.002 PMID: 25433558

37. Vera-Licona P, Bonnet E, Barillot E, Zinovyev A. OCSANA: optimal combinations of interventions from network analysis. Bioinformatics. 2013; 29(12):1571–1573. https://doi.org/10.1093/bioinformatics/btt195 PMID: 23626000

38. Kim J, Park SM, Cho KH. Discovery of a kernel for controlling biomolecular regulatory networks. Scientific reports. 2013; 3:2223. https://doi.org/10.1038/srep02223 PMID: 23860463

39. Akutsu T, Hayashida M, Ching WK, Ng MK. Control of Boolean networks: hardness results and algorithms for tree structured networks. Journal of Theoretical Biology. 2007; 244(4):670–679. https://doi.org/10.1016/j.jtbi.2006.09.023 PMID: 17069859

40. Klamt S, Saez-Rodriguez J, Lindquist JA, Simeoni L, Gilles ED. A methodology for the structural and functional analysis of signaling and regulatory networks. BMC Bioinformatics. 2006; 7(1):56. https://doi.org/10.1186/1471-2105-7-56 PMID: 16464248

41. Albert R. Boolean modeling of genetic regulatory networks. In: Complex networks. Springer; 2004. p. 459–481.

42. Lu J, Zhong J, Huang C, Cao J. On Pinning Controllability of Boolean Control Networks. IEEE Transactions on Automatic Control. 2016; 61(6):1658–1663. https://doi.org/10.1109/TAC.2015.2478123

43. Zhong J, Lu J, Liu Y, Cao J. Synchronization in an Array of Output-Coupled Boolean Networks With Time Delay. IEEE Transactions on Neural Networks and Learning Systems. 2014; 25(12):2288–2294. https://doi.org/10.1109/TNNLS.2014.2305722 PMID: 25420249

44. Lu J, Zhong J, Ho DWC, Tang Y, Cao J. On Controllability of Delayed Boolean Control Networks. SIAM Journal on Control and Optimization. 2016; 54(2):475–494. https://doi.org/10.1137/140991820

45. Nazarieh M, Wiese A, Will T, Hamed M, Helms V. Identification of key player genes in gene regulatory networks. BMC Systems Biology. 2016; 10(1):88. https://doi.org/10.1186/s12918-016-0329-5 PMID: 27599550

46. Bar-Joseph Z. Analyzing time series gene expression data. Bioinformatics. 2004; 20(16):2493–2503. https://doi.org/10.1093/bioinformatics/bth283 PMID: 15130923

47. Cormen TH. Introduction to algorithms. MIT press; 2009.