Computing Trap Space-based Control Strategies for Boolean Networks using Answer Set Programming

Cifuentes Fontanals, Laura,[1, 2, a] Tonello, Elisa[1], and Siebert, Heike[1]

[1] *Freie Universität Berlin, Germany*
[2] *Max Planck Institute for Molecular Genetics, Berlin, Germany*

[a] *Corresponding author: l.cifuentes@fu-berlin.de*

Abstract. Control of Boolean networks enables important medical and biological applications. At the core of many approaches is value percolation, by virtue of its simplicity and ease of implementation. Methods based uniquely on percolation can however miss many control strategies. We previously introduced a new method which, using the network's trap spaces, can uncover additional sets of interventions. In this work we present a highly efficient implementation of this methodology based on Answer Set Programming, allowing for simple and fast application to biological networks, as illustrated with some cases studies of cell reprogramming.

INTRODUCTION

Control of biological systems presents many interesting applications in the fields of bioengineering and medicine, for instance in cell reprogramming or drug design [1]. Mathematical modelling can be used to identify potential targets and help reducing the usually costly and time-consuming experimental testing. Among the different modelling frameworks, the Boolean formalism stands out for its ability to capture the qualitative behaviour and main features of biological systems, and its applicability even in cases of limited availability of quantitative details.

In the context of control of biological systems, it is often useful to target a set of relevant observable components, which capture the significant features of the system attractors, for instance different phenotypes. This approach is known as target control. Identification of control strategies is in general a complex problem. The core of many approaches is value percolation, which is a straightforward technique that can be implemented in a computationally efficient way [2, 3]. However, such approaches might miss many control strategies. Extending these methods using properties of trap spaces, subspaces of the state space closed for the dynamics, can uncover previously unidentified control strategies [4]. The implementation proposed in [4], which relies on the enumeration of the possible solutions, is computationally demanding. Building on works in [3], we present a new, highly efficient implementation of the methodology developed in [4] using Answer Set Programming (ASP).

CONTROL STRATEGIES

This section provides the definitions required to describe our approach. A *Boolean network* is a function $f\colon \mathbb{B}^n \to \mathbb{B}^n$, where $\mathbb{B} = \{0,1\}$. Every $x \in \mathbb{B}^n$ is a *state* of the state space $\mathbb{B}^n$. The set of components of f is denoted by $V = \{1,...,n\}$. Given a Boolean function, different dynamics can be defined. The *asynchronous dynamics*, which considers transitions updating only one component at a time, is often considered in order to capture the possible different time scales of the system. In this dynamics, represented by the asynchronous *state transition graph (STG)*, an edge exists from x to y if and only if $f_i(x) = y_i$ for some $i \in V$ and $x_j = y_j$ for all $j \neq i$. Other dynamics might consider, for example, the update of all the components (synchronous) or subsets of them (general asynchronous) simultaneously. The long term dynamics of a system is captured by the attractors, which are terminal strongly connected components of the STG. In biological systems, *steady states* (one-state attractors) can be identified with different cell fates or cell types, and *cyclic attractors* (attractors with multiple states) with cell cycles or specific cell processes. The asynchronous dynamics of a Boolean network with two steady states is shown in Figure 1.

Given a state $c \in \mathbb{B}^n$ and a subset of variables $I \subseteq V$, we define the *subspace* induced by c and I as the set $\Sigma(I,c) = \{x \in \mathbb{B}^n \mid \forall i \in I, x_i = c_i\}$. The variables in I are called fixed variables. We denote subspaces as states, using the symbol $*$ for the unfixed variables. A subspace closed for the dynamics is called a *trap space*. Trap spaces are update-independent and, by definition, contain at least one attractor. Moreover, in biological systems, minimal trap spaces are often good approximations for attractors [5]. The system interventions considered in this work consist of fixing certain variables to certain values. Mathematically, a set of interventions can be represented as a subspace $\Theta = \Sigma(I,c)$ and the dynamics of the controlled system is identified by the restriction $f_{\restriction_\Theta}\colon \Theta \to \Theta$ of f to Θ, defined by $(f_{\restriction_\Theta})_i(x) = c_i$

$$f(x) = \begin{cases} x_1x_2 \vee x_1x_3 \vee x_2\bar{x}_3 \\ x_1x_3 \vee \bar{x}_1x_2\bar{x}_3 \\ x_1\bar{x}_4 \vee \bar{x}_2\bar{x}_4 \vee \bar{x}_3\bar{x}_4 \\ \bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4 \end{cases}$$

$$\begin{aligned} f_{\upharpoonright_{\Theta_1}}(x) &= (x_1x_2 \vee x_2, \bar{x}_1x_2, 0, 1) \\ f_{\upharpoonright_{\Theta_2}}(x) &= (x_1x_2 \vee x_1x_3 \vee x_2\bar{x}_3, x_1x_3 \vee \bar{x}_1x_2\bar{x}_3, 0, 1) \end{aligned}$$

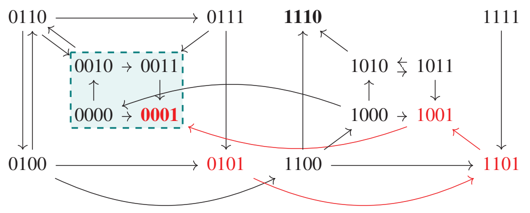

FIGURE 1. The asynchronous dynamics of the Boolean function f has two attractors $\mathscr{A}_1 = \{0001\}$ and $\mathscr{A}_2 = \{1110\}$. The trap spaces of f containing only attractors in the target subspace $P = 00{*}{*}$ (teal) are $T_1 = {*}{*}01$ (red) and $T_2 = 0001$ (the first steady state, bold red). The control strategies $\Theta_1 = {*}{*}0{*}$ and $\Theta_2 = {*}{*}{*}1$ percolate to the selected trap space T_1.

for all $i \in I$, and $(f_{\upharpoonright_\Theta})_i(x) = f_i(x)$ for $i \notin I$. A control strategy for a target subspace P is considered here as a set of interventions that fix the value of certain components so that all the attractors of the dynamics belong to P. Formally, given a Boolean function f and a subspace $P \subseteq \mathbb{B}^n$, a *control strategy for P* is a subspace $\Theta \subseteq \mathbb{B}^n$ such that, for any attractor $\mathscr{A}$ of $f_{\upharpoonright_\Theta}$, $\mathscr{A} \subseteq P$. The size of a control strategy $\Theta = \Sigma(I, c)$ is defined as the size of I, which denotes the number of control interventions. In this work, we consider optimal the control strategies with set of interventions that are minimal with respect to inclusion.

Approaches based solely on value percolation identify a set of interventions as a control strategy if the subspace obtained after iterative percolation of the interventions subspace is contained in the target subspace. The candidate set of interventions is taken from the components of the system, allowing the possibility to restrict the interventions only to a certain subset of components or candidates. Our approach goes a step further and uses trap spaces to identify new control strategies missed by techniques based solely on percolation. More precisely, it looks for subspaces that percolate to the so-called *selected trap spaces*, which are trap spaces containing only attractors belonging to the target subspace. Leading the system to a selected trap space guarantees that the system will evolve to the desired target. Moreover, since the dynamics cannot leave a trap space, the control could be released after reaching the trap space without altering the reachability of the target. This approach is applicable to the different types of updates previously described. The selected trap spaces for a certain target are identified using the attractors or their approximations via minimal trap spaces. Given a selected trap space T, any subspace that contains T and percolates to T is a control strategy for the given target [4]. The main steps of the method are presented in Algorithm 1. For the calculation of the trap spaces we use the ASP implementation described in [6].

COMPUTATION OF CONTROL STRATEGIES

Despite the efficiency of the percolation step, approaches based on percolation to the target subspace or to the selected trap spaces might require the exploration of all possible combinations of candidate interventions, whose number grows exponentially with the number of components of the network. In order to deal with such combinatorial explosion, Kaminski et al. [3] proposed the use of Answer Set Programming (ASP), a form of declarative programming suitable for hard combinatorial and optimisation problems. Here we extend the work done in [3], to identify the control strategies presented in [4].

As basis for our new implementation, we use the encoding presented in [3]. To implement the refined control strategy identification, we need to impose new conditions to the target subspace, in particular, to allow multiple targets, as well as to the interventions forming the candidate space, since only subspaces containing a selected trap space should be considered. Moreover, we also remove unnecessary parameters and redefine the constraints to percolate either to the target subspace or to one of the selected trap spaces. In the following, we describe the encoding of the problem for the example shown in Figure 1.

The Boolean function is encoded from the complete disjunctive normal form (DNF), which consists of the disjunction of its prime implicants. This encoding declares each variable of the network in the literal `var` and the literal `candidate` is instantiated only for the components that we want to consider for the control. Listing 1 shows the encoding of the Boolean function presented in Figure 1. The four variables (x_1, x_2, x_3, x_4) are instantiated (lines 1-2) and two candidates (x_3, x_4) are declared (line 3). The literal `f` represents the update function of each component and connects it to the clauses of its complete DNF, described by the literals `dnf` and `cl`. In this example, the DNF of the first variable ($x_2\bar{x}_3 \vee x_1x_3 \vee x_1x_2$) is declared in `f(x1,0)` (line 5) together with its three clauses `dnf(0,0)`, `dnf(0,1)`

Listing 1. Program instance

```
1  var(x1). var(x2).
2  var(x3). var(x4).
3  candidate(x3). candidate(x4).
4
5  f(x1,0). f(x2,1).
6  f(x3,2). f(x4,3).
7  dnf(0,0). dnf(0,1).
8  dnf(0,2). dnf(1,3).
9  dnf(1,1). dnf(2,4).
10 dnf(2,5). dnf(2,6).
11 dnf(3,7). dnf(3,8).
12 dnf(3,9). dnf(3,10).
13
14 cl(0,x2,1). cl(0,x3,-1).
15 cl(1,x1,1). cl(1,x3,1).
16 cl(2,x1,1). cl(2,x2,1).
17 cl(3,x1,-1). cl(3,x2,1).
18 cl(3,x3,-1). cl(4,x3,-1).
19 cl(4,x4,-1). cl(5,x2,-1).
20 cl(5,x4,-1). cl(6,x1,1).
21 cl(6,x4,-1). cl(7,x4,1).
22 cl(8,x3,-1). cl(9,x2,-1).
23 cl(10,x1,-1).
24
25 subs(-1). subs(0). subs(1).
26 goal(-1,x1,-1). goal(-1,x2,-1).
27 goal(0,x1,-1). goal(0,x2,-1).
28 goal(0,x3,-1). goal(0,x4,1).
29 goal(1,x3,-1). goal(1,x4,1).
30
31 #const maxsize=3.
```

Listing 2. Main program

```
1  goal(T,S) :- goal(Z,T,S), Z < 0.
2  satisfy(V,W,S) :- f(W,D); dnf(D,C); cl(C,V,S).
3  closure(V,T)   :- goal(V,T).
4  closure(V,S*T) :- closure(W,T); satisfy(V,W,S); not goal(V,-S*T).
5  { intv(V,S) : closure(V,S), candidate(V), satisfied(Z), Z < 0 }.
6  { intv(V,S) : goal(Z,V,S), candidate(V), satisfied(Z), Z >=0 }.
7  :- intv(V,1); intv(V,-1).
8  intv(V) :- intv(V,S).
9
10 eval(Z,V,S) :- subs(Z); intv(V,S).
11 free(Z,V,D) :- f(V,D); subs(Z); not intv(V).
12 eval_cl(Z,C,-1) :- cl(C,V,S); eval(Z,V,-S).
13 eval(Z,V, 1) :- free(Z,V,D); eval(Z,W,T) : cl(C,W,T); dnf(D,C).
14 eval(Z,V,-1) :- free(Z,V,D); eval_cl(Z,C,-1) : dnf(D,C).
15
16 not satisfied(Z) :- goal(Z,T,S), not eval(Z,T,S), subs(Z).
17 satisfied(Z) :- eval(Z,T,S) : goal(Z,T,S); subs(Z).
18 0 < { satisfied(Z) : subs(Z) }.
19 :- maxsize>0; maxsize + 1 { intv(X) }.
```

Algorithm 1 Control strategies for a target subspace P

```
1: function ControlStrategies($f$, $P$, $attr$)
2:     $\mathbf{T} \leftarrow$ trapSpaces($f$)
3:     $\mathbf{selTS} \leftarrow$ selectedTrapSpaces1($\mathbf{T}$, $P$)
4:     if $attr \neq \emptyset$ then:
5:         $\mathbf{selTS} \leftarrow \mathbf{selTS}$ + selectedTrapSpaces2($\mathbf{T}$, $P$, $attr$)
6:     $\mathbf{CS} \leftarrow$ createCandidatesAndPercolate($f$, $P$, $\mathbf{selTS}$, $m$)
7:     return $\mathbf{CS}$
```

FIGURE 2. Program instance for the example in Figure 1 (Listing 1). Main program for strategy identification (Listing 2). General algorithm for control strategy identification via trap spaces (Algorithm 1). The algorithm takes as inputs a Boolean function (f) and a target subspace (P) to return a set of control strategies for P. Additionally, it can also take as input the set of attractors of f or their minimal trap space approximation ($attr$) and an upper bound on the size of the control strategies (m).

and `dnf(0,2)` (lines 7-8). The first clause `dnf(0,0)` corresponds to $x_2\bar{x}_3$ and, therefore, is encoded by the literals `cl(0,x2,1)` and `cl(0,x3,-1)` (line 14). Note that the third variable of the literal `cl` denotes whether the variable is negated (-1) or not (1). The target subspace and each of the selected trap spaces are encoded in the literal `subs` (line 25), with a negative identifier for the target subspace and a positive identifier for the selected trap spaces. The fixed variables of each subspace are denoted in the variable `goal` (lines 26-29). For instance, the trap space $**01$ is declared in the literal `subs(1)` and its fixed variables are represented by the literals `goal(1,x3,-1)` and `goal(1,x4,1)`. Finally, a constant `maxsize`, which allows to set an upper bound on the size of the control strategies, is specified in the problem instance (line 31).

The main program for control strategy identification is shown in Listing 2. The first difference from the original implementation in [3] is the generation of the candidate interventions. Our approach only considers candidate subspaces containing the selected trap space they percolate to. This condition is stated in line 6 and makes sure that for every chosen `intv(V,S)` there exists a `subspace(Z)`, `Z > 1`, reached by percolation, with a `goal(Z,V,S)` that fixes the same variable `V` to the same value `S`. Interventions for direct percolation are generated in lines 1-5. Line 7 prevents that two contradictory interventions are chosen and line 8 creates an auxiliary literal to denote the variables that have been chosen in the control strategy. The effect of the interventions is represented in the literals `eval(Z,V,S)` and `free(Z,V,D)`, which contain for each target subspace the values of the variables fixed by the chosen interventions. The percolating effect is captured in lines 12-14 where each clause of the DNF is evaluated and fixed, when needed, according to the literals `eval` and `free`. Different from the original implementation, in order to ensure that a candidate subspace is a control strategy, it is required that at least one subspace constraint is satisfied (line 18). To do so, we introduce the variable `satisfied`, which captures the subspaces that have been reached by the percolation process (lines 16-17). Moreover, a limitation on the number of interventions (size of the control strategy) is added (line 19).

We compare the performance of the ASP implementation to the implementation proposed in [4] with three case studies, two of them also discussed in [4]. We chose networks in a range of sizes modelling cell fate decision systems since cell fate reprogramming is particularly interesting for applications, for example, to induce apoptosis in cancer cells. Therefore, the target chosen for the control in each network is the subspace defined by the apoptotic phenotype.

TABLE I. Benchmarks of the different implementations. *Time original* and *Time ASP* refer to the calculation of control strategies based on percolation to selected trap spaces, for the implementation presented in [4] and with the ASP encoding proposed here respectively, while *Time ASP** includes also percolation to the target subspace. Running times are expressed in seconds and obtained with a desktops 8-processor computer, Intel®Core™ i7-2600 CPU at 3.40GHz, 16GB memory. The ASP program is solved using *clingo* [10]. The columns V, E, V_i and V_r indicate the number of components, interactions among components, input components and readout components of each network respectively. All implementations use as a target the subspace corresponding to the apoptotic phenotype. The table also includes the number of selected trap spaces for each target and the time for their computation and the number of control strategies up to size 4 obtained by the complete ASP implementation (ASP*). The components fixed in the target subspace (readouts) were excluded from the candidate interventions.

Network	V	E	V_i	V_r	**Steady states**	**Cyclic attr.**	**Sel. TS**	**Time sel. TS**	**Size 1**	**Size 2**	**Size 3**	**Size 4**	**Time original**	**Time ASP**	**Time ASP***
Cell Fate [7]	28	48	3	3	27	0	27	0.05	0	16	118	165	8.90	0.07	0.27
MAPK [8]	53	108	4	3	12	6	103	0.05	3	103	151	132	373.38	0.30	9.70
T-LGL [9]	60	199	6	3	86	70	883	0.25	1	0	3	3	790.31	3.63	4.66

We investigated other target subspaces, obtaining comparable results in terms of efficiency (results not shown).

The results of the benchmarks are summarised in Table I. The ASP implementation does significantly better in all the analysed networks, independently of their size or complexity. In addition to the results shown in the table, the ASP implementation was run for all possible upper bounds on the sizes of the control strategies. The largest minimal control strategy identified has size 5, 6 and 10 for the networks Cellfate, MAPK and T-LGL respectively. The running times appear to stabilise around 0.5, 9.7 and 280 seconds respectively. This is a notable difference from the previous implementation, where the running times are always increasing with the upper bound, making the computation feasible only for small control strategies.

CONCLUSION

We presented a new, more efficient implementation in Answer Set Programming of the control strategy identification method developed in [4], that makes this more sophisticated approach accessible for application. It would be interest-ing to investigate its scalability with respect to the network size or the complexity of the update functions. We expect the latter to be a limiting factor due to the necessary computation of prime implicants.

Our results show that ASP with its flexibility and strength in solving combinatorial problems is well-suited to tackle implementation of intricate approaches to Boolean network control.

REFERENCES

1. M. Cerezo and S. Rocchi, “Cancer cell metabolic reprogramming: a keystone for the response to immunotherapy,” Cell Death and Disease (2020), 10.1038/s41419-020-03175-5.
2. R. Samaga, A. V. Kamp, and S. Klamt, “Computing combinatorial intervention strategies and failure modes in signaling networks,” Journal of Computational Biology **17**, 39–53 (2010).
3. R. Kaminski, T. Schaub, A. Siegel, and S. Videla, “Minimal intervention strategies in logical signaling networks with asp,” Theory and Practice of Logic Programming **13**, 675–690 (2013).
4. L. Cifuentes Fontanals, E. Tonello, and H. Siebert, “Control strategy identification via trap spaces in boolean networks,” in *Computational Methods in Systems Biology*, edited by A. Abate, T. Petrov, and V. Wolf (Springer International Publishing, Cham, 2020) pp. 159–175.
5. H. Klarner and H. Siebert, “Approximating attractors of boolean networks by iterative ctl model checking,” Frontiers in Bioengineering and Biotechnology **3**, 130 (2015).
6. H. Klarner, A. Bockmayr, and H. Siebert, “Computing maximal and minimal trap spaces of boolean networks,” Natural Computing **14**, 535–544 (2015).
7. L. Calzone, L. Tournier, S. Fourquet, D. Thieffry, B. Zhivotovsky, E. Barillot, and A. Zinovyev, “Mathematical modelling of cell-fate decision in response to death receptor engagement,” PLOS Computational Biology **6**, 1–15 (2010).
8. L. Grieco, L. Calzone, I. Bernard-Pierrot, F. Radvanyi, B. Kahn-Perlès, and D. Thieffry, “Integrative modelling of the influence of mapk network on cancer cell fate decision,” PLOS Computational Biology **9**, 1–15 (2013).
9. R. Zhang, M. V. Shah, J. Yang, S. B. Nyland, X. Liu, J. K. Yun, R. Albert, and T. P. Loughran, “Network model of survival signaling in large granular lymphocyte leukemia,” Proceedings of the National Academy of Sciences **105**, 16308–16313 (2008).
10. M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. Schneider, “Potassco: The potsdam answer set solving collection,” AI Commun. **24**, 107–124 (2011).