Efficient Enumeration of Fixed Points in Complex Boolean Networks Using Answer Set Programming

Van-Giang Trinh[1]
LIS, Aix-Marseille University, Marseille, France

Belaid Benhamou
LIS, Aix-Marseille University, Marseille, France

Sylvain Soliman[1]
Lifeware team, Inria Saclay, Palaiseau, France

Abstract

Boolean Networks (BNs) are an efficient modeling formalism with applications in various research fields such as mathematics, computer science, and more recently systems biology. One crucial problem in the BN research is to enumerate all fixed points, which has been proven crucial in the analysis and control of biological systems. Indeed, in that field, BNs originated from the pioneering work of R. Thomas on gene regulation and from the start were characterized by their asymptotic behavior: complex attractors and fixed points. The former being notably more difficult to compute exactly, and specific to certain biological systems, the computation of stable states (fixed points) has been the standard way to analyze those BNs for years. However, with the increase in model size and complexity of Boolean update functions, the existing methods for this problem show their limitations. To our knowledge, the most efficient state-of-the-art methods for the fixed point enumeration problem rely on Answer Set Programming (ASP). Motivated by these facts, in this work we propose two new efficient ASP-based methods to solve this problem. We evaluate them on both real-world and pseudo-random models, showing that they vastly outperform four state-of-the-art methods as well as can handle very large and complex models.

2012 ACM Subject Classification Computing methodologies → Logic programming and answer set programming; Applied computing → Computational biology; Applied computing → Systems biology

Keywords and phrases Computational systems biology, Boolean network, Fixed point, Answer set programming

Digital Object Identifier 10.4230/LIPIcs.CP.2023.35

Supplementary Material *Software (Source Code)*: `https://github.com/giang-trinh/fASP`
archived at `swh:1:dir:86a458f51d92dda88a71499b7e6bea7178377da0`

Funding *Van-Giang Trinh*: This work is supported by Institut Carnot STAR, Marseille, France.

1 Introduction

In molecular biology, the regulation of the transcription of a gene is the process by which a cell modulates the conversion of its DNA into RNA. Transcription is a vital process in all living organisms and leads to orchestrating the whole gene activity. Its regulation can take many different forms, mostly affecting the binding of the RNA polymerase on the DNA.

The lack of precise quantitative information about transcriptional regulation and the sigmoid nature of its kinetics led, about fifty years ago, to the idea to represent models of gene regulation as discrete event systems. Those gene regulation networks use thresholds or equivalently logical functions to represent the different regulations [22, 39, 41, 40]. Over the years, Boolean Network (BN) modelling has proven that it can bring powerful analyses and

[1] Corresponding authors

corresponding insight to the many cases where enough quantitative biological data is not available [48], even for modelling post-transcriptional mechanisms. This is even more true for very large models where such data is frequently missing and led to a constant increase in size of logical models *à la* Thomas [2] and more and more complex logical formulae to describe the dynamics of those models.

Besides simulation, the analysis of such models is mostly based on *attractor* computation, since those correspond roughly to observable biological phenotypes [48]. An attractor of a BN is a minimal set of states from which the dynamics of this BN cannot escape once entered [39, 48]. An attractor of size one is called a stable state or *fixed point*. Otherwise, it is called a cyclic attractor or complex attractor. To date, the analysis of the set of fixed points of a BN remains a very useful tool in understanding the behavior of those complex biological models. This is not only due to the fact that in some cases the full computation of complex attractors remains intractable, but also because for many biological systems, the expected long-term behavior is not cyclic (as in the Cell Cycle, or Circadian rhythms for instance) but rather a stabilization to an observable *phenotype* (cell differentiation, apoptosis, proliferation, signal transduction, protein transcription, etc.). See for instance [33, 15, 13, 43] for some recent publications using stable states as main validation. It is also worth noting that the fixed point computation is the crucial starting point for several state-of-the-art methods for computing complex attractors of BNs [21, 42].

Answer Set Programming (ASP) [19] has been widely applied in the field of computational systems biology [46] because of its declarative characteristics as well as strong tools' support [18]. Very early, ASP has been used to model biological networks [14, 37]. Since BNs have become a popular modeling formalism in systems biology, it is naturally that ASP has been quickly applied to modeling and analysis of BNs. One of the first connections between ASP and BNs is the theoretical work by [26], but nowadays we can find in the literature many references showing the successful application of ASP to model and reason over biological systems modeled as BNs. The notable use of ASP in the analysis of BNs ranges from enumerating fixed points [28, 1, 34], enumerating or approximating attractors [30, 28, 1, 34], and inferring BNs from biological data [35, 46, 47, 12], to controlling BNs [27, 47].

There is a rich history of research on enumerating fixed points of BNs since this modeling formalism was proposed [22, 39]. The fixed point enumeration problem has attracted researchers from various communities and many methods have been proposed [29]. We can classify the existing methods into the following main approaches: algorithmic [29], structure-based [10, 45, 25, 7], Boolean resolution-based [24, 32, 31], integer linear programming-based [5], and ASP-based [28, 1, 34]. A more detailed summary of the existing methods shall be given in Section 3. Note however that with the increase in model size and complexity of Boolean update functions, the existing methods for this problem show their limitations [29]. One reason is that they require an intermediate representation of the original BN that may be computationally expensive or even intractable to obtain, e.g., prime implicants [28], transition-based representations [1], disjunctive normal forms [34].

Inspired by the above elements along with the fact that the most recent and most efficient fixed point enumeration methods all rely on ASP, in this work we propose two new ASP-based methods for efficiently enumerating all fixed points of a BN. The first method is based on conjunctive ASP, and the second method is a modification of the first one to handle the case of a large number of source nodes. If a BN has many source nodes, its number of fixed points may be extremely large, leading to both long running time and high memory consumption. The main advantage of the two proposed methods is that they rely on negative normal forms of Boolean functions whose computation is more efficient than that of other intermediate representations used by the previous methods. After some preliminaries on BNs and the problem of enumerating their fixed points, we present the two new ASP-based methods and benchmark them against four other state-of-the-art tools. The experimental results on both real-world and pseudo-random models show that they vastly outperform the state-of-the-art and can handle very large and complex models.

2 Preliminaries

We start by recalling some classical definitions.

2.1 Boolean networks

▶ **Definition 1** (Boolean network). *A Boolean Network (BN) [40] is a pair $\mathcal{N} = (V, F)$ where:*

- $V = \{v_1, \dots, v_n\}$ *is the set of nodes. We use v_i to denote both the node v_i and its associated Boolean variable.*
- $F = \{f_1, \dots, f_n\}$ *is the set of Boolean update functions. Each function f_i is associated with node v_i and satisfies $f_i \colon \mathbb{B}^{|IN(v_i)|} \mapsto \mathbb{B}$ where $\mathbb{B} = \{0, 1\}$ and $IN(v_i)$ denotes the set of input nodes of v_i. If $v_j \in IN(v_i)$, we say that there is a* regulation *between v_j and v_i, and v_j is a regulator of v_i. Note that a node $v_i \in V$ is called a* source *node if and only if f_i is an identity function on Boolean variable v_i (i.e., $f_i = v_i$).*

▶ **Example 2.** We give a BN $\mathcal{N} = (V, F)$, where $V = \{v_1, v_2\}$ and $F = \{f_1, f_2\}$ with $f_1 = (v_1 \wedge v_2) \vee (\neg v_1 \wedge \neg v_2)$, $f_2 = (v_1 \wedge v_2) \vee (\neg v_1 \wedge \neg v_2)$.

▶ **Definition 3** (local monotonicity). *A Boolean function is* locally-monotonic *if it can be represented by a formula in Disjunctive Normal Form (DNF) in which all occurrences of any given literal are either negated or non-negated [34].*

A BN is said to be locally-monotonic if all its update functions are locally-monotonic. Otherwise, this model is said to be non-locally-monotonic.

The BN of Example 2 is non-locally-monotonic.

2.2 Fixed points

A state $x \in \mathbb{B}^n$ is as a mapping $x \colon V \mapsto \mathbb{B}$ that assigns either 0 (inactive) or 1 (active) to each node. We also write x_i to denote $x(v_i)$ for short and for simplicity we write $f_i(x)$ even when $IN(v_i) \subsetneq V$, i.e., $IN(v_i)$ does not contain some nodes of V.

▶ **Definition 4** (fixed point). *A* fixed point *of $\mathcal{N}$ is a state s such that $s_i = f_i(s)$ for every $v_i \in V$.*

The state space of the BN of Example 2 includes four states: 00, 01, 10, and 11. However, this BN has only one fixed point: 11.

Complexity

Note that the fixed points do not depend on the choice of *update scheme* of the BN, they are the same for synchronous, asynchronous or even generalized updates [20]. These *update schemes* are what precisely defines a transition relation on states from the update functions. In general they allow one (asynchronous), all (synchronous) or any number (generalized) of nodes to change their value v_i to their update value specified by f_i. However, if running a simulation in the synchronous update is a feasible way to find *the only* reachable fixed point starting from a completely known initial state, this does not scale up to big networks with many source nodes with unknown values, or to other update schemes.

From a theoretical view point, the problems of detecting a fixed point and enumerating all fixed points of a general BN have been shown to be respectively NP-hard and #P-hard [3]. In fact, for general BNs, there is no existing method that works faster than $k \times 2^n$ for any $k \geq 1$ [29].

3 Related work

The recent review of Mori and Akutsu [29] shows quite well that because of the above complexity result, a certain amount of work has been done on restricted versions of the problem, limiting the type of BN to simpler regulations like [6, 23], or simpler Boolean formulae like for instance nested canalizing functions [4]. However, when working with real-world models, built by biologists, such restrictions are often impossible to enforce.

Hence, various methods exploiting the structure of a BN have been proposed, using feedback vertex sets [3, 7], subspaces [10], graph-reductions for low connectivity [45], network decomposition [8, 25], etc. Unfortunately, these still do not scale to the size of the most recent BNs (above 1000 nodes) with average connectivity and complex logical formulae.

Other methods with broader generality are also common, using classical Boolean resolution techniques, like BDD or SAT. That is the case for instance of the `BioLQM` library [31] that is at the core of the GINsim Boolean modelling tool [24] and of the CoLoMoTo Docker images [32]. Since integer linear programming is another useful method to efficiently solve Boolean constraints, it has been applied to addressing the fixed point enumeration [5]. The evaluation in [5] shows that this method can handle well models of up to 200 nodes with small average connectivity.

However, the most recent and most efficient fixed point enumeration methods all rely on ASP [19]. This is probably due to the fact that it links the efficiency of SAT for the Boolean constraint solving, having adapted and implemented some techniques like lazy clause generation to the point of winning certain categories of the SAT competition, and the ease of enumeration of all solutions, which is crucial here, in a declarative language.

More precisely, while there is indeed a direct encoding of the fixed-point problem into SAT [29], it creates two issues. First a SAT solver needs to convert the original Boolean formula into a CNF. It is of course possible to use a polynomial transformation like our conjunctive ASP encoding (see Section 4) or Tseitin's transformation, but this introduces auxiliary variables. This in turn leads to enumerating models that encode no fixed points or other redundant models that encode the same fixed points. A step to eliminate spurious and redundant SAT models is therefore necessary to guarantee the correctness and this would add complexity to the SAT/CP approach. In contrast, the ASP approach can avoid the above issue because of the stable model semantics, i.e., only searching for minimal Herbrand models, since the set of Herbrand models one-to-one corresponds to the set of SAT/CP models, whereas the set of minimal ones one-to-one corresponds directly to the set of fixed points.

One of the first connections between ASP and BNs is the theoretical work by [26], but nowadays ASP is used for many different BN analyses, from computing fixed point as we will show, to trap-spaces [28] which are an approximation of complex attractors, and even for representing sets of BNs [12].

By constraining the number of ground atoms in a stable model, the trap-space computation method [28] was adapted to compute fixed points. Note however that this method still requires to compute prime implicants of a Boolean function, and the number of prime implicants may be exponential in the number of inputs of this function. Moreover, the computation of prime implicants of a Boolean function is also a computationally demanding task, and gets intractable when the number of source nodes exceeds 10.

It is worth noting that Paulevé *et al.* [34] have proposed a new method for computing minimal trap spaces/fixed points that can avoid computing prime implicants. This method has been implemented in the tool `mpbn`[2] demonstrated in [34] for handling medium-sized models from the literature and very large synthetic models (up to 100,000 nodes) with respect to minimal trap spaces. Although there is no benchmark designed for the fixed point computation in [34], `mpbn` should handle well very large models with respect to fixed points. However, there are two drawbacks limiting the applicability of `mpbn`. First, it requires that the original BN is locally-monotonic. The class of locally-monotonic BNs is too small as compared to the class of all possible BNs, since a BN is non-locally-monotonic if just one of its Boolean functions is non-locally-monotonic (see Definition 3). Moreover, we also found many non-locally-monotonic Boolean models in the literature (see Section 5 for some of them). Second, it requires a DNF of a Boolean function. Note that obtaining a single DNF may be exponential in the size of the Boolean function (i.e., the number of inputs $|IN|$ of this Boolean function).

Not using the concept of trap spaces, the method by [1] characterizes fixed points of a Boolean network as *dead* configurations (or deadlocks) of its corresponding Automata Network (AN). ANs are formal models similar to Petri nets with transitions representing the updates of the whole system. A transition includes the current configuration, the next configuration, and the condition for enabling this transition. A configuration is said to be dead if and only if there is no transition whose enabling condition is satisfied by this configuration. Then the above characterization is encoded as an ASP. This method has been reported to be able to handle well large-scale models [1]. However, its bottleneck lies in the construction of the corresponding AN, which in general requires to obtain two DNFs for each Boolean variable, one for the update function f_i and one for its negation $\neg f_i$.

4 Answer set programming-based methods

We will now describe the two new ASP-based encodings that we propose for the fixed point enumeration problem.

4.1 Conjunctive encoding

Let $\mathcal{N} = (V, F)$ be a BN. We intend to build an ASP encoding for $\mathcal{N}$ such that a stable model of the encoded ASP (say $\mathcal{L}$) is equivalent to a fixed point of $\mathcal{N}$. First, for each node v_i, we introduce two atoms p_i and n_i. The translation from a stable model A of $\mathcal{L}$ to a state x of $\mathcal{N}$ is that for every $v_i \in V$, $x_i = 1$ if and only if $p_i \in A$, and $x_i = 0$ if and only if $n_i \in A$. The below ASP rules ensure that a stable model of $\mathcal{L}$ corresponds to a state of $\mathcal{N}$:

$$\texttt{:-}\, p_i, n_i. \tag{1}$$

meaning $\mathit{false} \Leftarrow p_i \wedge n_i$, and

$$p_i, n_i. \tag{2}$$

meaning $p_i \vee n_i \Leftarrow \mathit{true}$, for every $v_i \in V$. Recall that state x is a fixed point of $\mathcal{N}$ if and only if the relation $x_i = f_i(x)$ holds for all $v_i \in V$. This relation can be seen as the conjunction of $x_i \leftarrow f_i(x)$ and $\neg x_i \leftarrow \neg f_i(x)$, which can be characterized by $v_i \leftarrow f_i$ and $\neg v_i \leftarrow \neg f_i$, respectively. Hereafter, we show how to encode the two parts for every $v_i \in V$ as conjunctive ASP rules.

[2] https://github.com/bnediction/mpbn

For the former part, to avoid the presence of negation, we convert f_i into its Negative Normal Form (NNF). The NNF is obtained by recursively applying De Morgan laws until all negations that remain are on literals. We now associate ASP rules to f_i as follows:

$$\gamma(v_i) \text{ :- } \gamma(NNF(f_i)).$$

where we define function γ as

$$\begin{aligned}
\gamma(v_i) &= p_i \\
\gamma(\neg v_i) &= n_i \\
\gamma(\bigwedge_{1\le j\le J} \alpha_j) &= \gamma(\alpha_1), \dots, \gamma(\alpha_J) \\
\gamma(\bigvee_{1\le j\le J} \alpha_j) &= aux_k \text{ where } aux_k \text{ is a new atom and for each } j \text{ add the rule } aux_k \text{ :- } \gamma(\alpha_j).
\end{aligned}$$

Note that k is here a global counter starting from 1 and will be increased by 1 after a new atom is created. For the latter part, we similarly apply the above process with $\neg v_i$ and $\neg f_i$ instead of respectively v_i and f_i. Note that it also requires to convert $\neg f_i$ into an NNF first.

Listing 1 shows the encoded ASP of the BN shown in Example 2 following the above encoding. Atoms p_1 and n_1 (resp. p_2 and n_2) correspond to node v_1 (resp. v_2). Lines 1 and 2 represent the rules shown in Equation (1) and Equation (2), respectively. The rules for the part $v_1 \leftarrow f_1$ (resp. $\neg v_1 \leftarrow \neg f_1$) are presented in Lines 4–5 (resp. Lines 6–8). Similarly, Lines 10–14 represent the rules for node v_2. Line 16 indicates that we omit auxiliary atoms in the resulting stable models. This ASP has only one stable model: $\{p_1, p_2\}$, which corresponds to the sole fixed point (i.e., 11) of $\mathcal{N}$.

Listing 1 Conjunctive ASP encoding for the BN shown in Example 2.

```
:- p1, n1.                      :- p2, n2.                   1
p1, n1.                         p2, n2.                      2
                                                             3
p1 :- aux1.                                                  4
aux1 :- p1, p2.                 aux1 :- n1, n2.              5
n1 :- aux2, aux3.                                            6
aux2 :- n1.                     aux2 :- n2.                  7
aux3 :- p1.                     aux3 :- p2.                  8
                                                             9
p2 :- aux4.                                                  10
aux4 :- p1, p2.                 aux4 :- n1, n2.              11
n2 :- aux5, aux6.                                            12
aux5 :- n1.                     aux5 :- n2.                  13
aux6 :- p1.                     aux6 :- p2.                  14
                                                             15
#show p1/0.  #show n1/0.        #show p2/0.  #show n2/0.     16
```

We here discuss the advantages of the above ASP encoding. First, the ASP $\mathcal{L}$ has no negation besides Equation (1) that may hinder the efficiency of ASP solvers. Note also that obtaining the NNF of a Boolean function is linear in its size and thus quite efficient. Second, except the rules of Equation (2), all the rules in $\mathcal{L}$ are conjunctive. This is the reason we name the above encoding as the *conjunctive* encoding.

The next result shows that our ASP encoding is sound and complete with respect to the fixed points of a BN.

▶ **Proposition 5.** *The set of stable models of $\mathcal{L}$ one-to-one corresponds to the set of fixed points of $\mathcal{N}$.*

Proof. Note that the only negations in the conjunctive encoding come from the state constraints of Equation (1), i.e., $:\!-\, p_i, n_i$. Since this equation is coupled with Equation (2), all stable models will contain exactly one of p_i or n_i, i.e., they will correspond with the state x where x_i is true or false depending on the atom in the stable model.

The remainder of the ASP has no negation, no disjunction in heads and is logically equivalent to $\forall v_i \in V, x_i = f_i(x)$ via the introduction of some existentially quantified aux_j. Hence there is for each state x at most a single stable model, equal to the smallest Herbrand model containing the p_i or n_i corresponding to x, the necessary aux_j, and that satisfies $x = f(x)$ by construction.

All stable models of $\mathcal{L}$ will therefore correspond one to one with states x described by the p_i or n_i that are true, and such that $\forall v_i \in V, x_i = f_i(x)$, hence they correspond one to one with fixed points of $\mathcal{N}$. ◀

4.2 Source encoding

Recall that the number of fixed points of a BN may be extremely large if it has many source nodes (see Definition 1). Specifically, that number may be exponential in the number of source nodes. In the conjunctive encoding as well as those of the state-of-the-art methods [28, 1, 34], a resulting stable model always corresponds to a single fixed point. Hence, having many source nodes is actually a bottleneck for these methods. To overcome this issue, we propose a new encoding based on the conjunctive encoding of Section 4.1.

Let $\mathcal{L}_c$ be the encoded ASP of the BN following the conjunctive encoding. Let V^s be the set of source nodes of the BN. Our main idea is to group two stable models A_1 and A_2 of $\mathcal{L}_c$ into a stable model A if they only differ in the atoms corresponding to a source node. More specifically, if there is a source node v_i such that $p_i \in A_1$, $n_i \in A_2$, and $A_1 \setminus \{p_i\} = A_2 \setminus \{n_i\}$, then we can group A_1 and A_2 into a stable model A such that $A = A_1 \cup \{n_i\} = A_2 \cup \{p_i\}$. For example, let A_1 and A_2 be the stable models respectively corresponding to fixed points 01 and 11 of the BN shown in Example 7. Herein, $A_1 = \{n_1, p_2\}$ and $A_2 = \{p_1, p_2\}$. They can be grouped into stable model $A = \{p_1, n_1, p_2\}$. Now, we add A to the set of stable models of $\mathcal{L}_c$, and then repeat the grouping process until there is no new stable model. Note that this process introduces more stable models than before, e.g., we need to consider all A, A_1, and A_2. However, the new stable model covers all the fixed points represented by the two stable models constituting it. Hence, we just need to consider the maximal set-inclusion stable models. We adjust the conjunctive encoding to make the above approach fully automated in the ASP solver. Since the new encoding aims to handle the case of many source nodes, we name it the *source* encoding.

Similar to the conjunctive encoding, for each node $v_i \in V$, we introduce two atoms p_i and n_i. For each node in V, we associate to this node the ASP rules identical to whose of the conjunctive encoding. For each $v_i \in V^s$, we remove from the encoded ASP (say $\mathcal{L}_s$) the rule of Equation (1), i.e., $:\!-\, p_i, n_i$. By releasing this condition, $\mathcal{L}_s$ can have Herbrand models that contain both p_i and n_i, $v_i \in V^s$. To make such Herbrand models to be stable models of $\mathcal{L}_s$, we add to $\mathcal{L}_s$ the choice rules $\{p_i\}$. and $\{n_i\}$. for all $v_i \in V^s$. A choice rule $\{p_i\}$. is equivalent to the rule $p_i :\!-\, \text{not not}\, p_i$. where not denotes the default negation. Finally, we add to $\mathcal{L}_s$ the rules #show $p_i/0$ and #show $n_i/0$ for all $v_i \in V$, which indicate that we omit auxiliary atoms in the resulting stable models.

Note that a stable model of $\mathcal{L}_s$ may correspond to multiple fixed points of the BN. Given a stable model A of $\mathcal{L}_s$, the set F of fixed points represented by A is specified as follows. For each node $v_i \in V^s$, it can receive value 0 if $n_i \in A$ and $p_i \notin A$, value 1 if $p_i \in A$ and $n_i \notin A$, both if $n_i \in A$ and $p_i \in A$. For each node $v_i \in V \setminus V^s$, it can receive value 0 if $n_i \in A$ and value 1 if $p_i \in A$. Then, F is equivalent to the set of all possible value combinations of all nodes.

The next result shows that our ASP encoding is correct.

▶ **Proposition 6.** *The set of maximal set-inclusion stable models of $\mathcal{L}_s$ with respect to the shown atoms exactly covers all fixed points of the BN.*

Proof. First, we see that $\mathcal{L}_s$ still contains all Herbrand models of $\mathcal{L}_c$. However, by releasing the condition of Equation (1) for all source nodes, p_i and n_i can both appear in a Herbrand model of $\mathcal{L}_s$ if $v_i \in V^s$. Assume that A_1 and A_2 are two stable models of $\mathcal{L}_s$ such that $p_i \in A_1$, $n_i \in A_2$, and $A_1 \setminus \{p_i\} = A_2 \setminus \{n_i\} = B$ for a node $v_i \in V^s$. The ASP rules corresponding to node v_i are tautology. In all the remaining rules, p_i and n_i never appear in the left hand side. Hence, $A = B \cup \{p_i, n_i\}$ is also a Herbrand model of $\mathcal{L}_s$. By introducing the choice rules for all source nodes, such Herbrand models will be stable models of $\mathcal{L}_s$. Hence, the set of all stable models of $\mathcal{L}_s$ is equivalent to the set of stable models obtained by the grouping approach on the set of stable models of $\mathcal{L}_c$. All fixed points represented by a stable model of $\mathcal{L}_s$ are also covered by a maximal set-inclusion stable model of $\mathcal{L}_s$ with respect to the shown atoms (corresponding to nodes in the BN). Hence, the set of all maximal set-inclusion stable models of $\mathcal{L}_s$ exactly covers all fixed points of the BN. ◀

For illustration, consider the BN shown in Example 7. Listing 2 shows the encoded ASP of this BN following the above encoding. Atoms p_1 and n_1 (resp. p_2 and n_2) correspond to node v_1 (resp. node v_2). Line 1 represents the rule of Equation (2). Note that the rule $:\!-\, p_1, n_1$ is removed because v_1 is a source node. Instead, two choice rules $\{p_1\}$. and $\{n_1\}$. are added in Line 2. Line 3 represents the rules shown in Equation (1) and Equation (2) of node v_2. The rules for the parts $v_1 \leftarrow f_1$ and $\neg v_1 \leftarrow \neg f_1$ are presented in Line 5. Similarly, Lines 7–9 represent the rules for node v_2. Line 11 indicates that we omit auxiliary atoms in the resulting stable models. The encoded ASP has four stable models including: $\{n_1, n_2\}$ (corresponding to fixed point 00), $\{n_1, p_2\}$ (corresponding to fixed point 01), $\{p_1, p_2\}$ (corresponding to fixed point 11), and $\{p_1, n_1, p_2\}$ (corresponding to fixed points 01 and 11). From these results, we can see that the encoded ASP has two maximal set-inclusion stable models ($\{n_1, n_2\}$ and $\{p_1, n_1, p_2\}$), which cover all the fixed points of the BN.

▶ **Example 7.** We give a BN $\mathcal{N} = (V, F)$, where $V = \{v_1, v_2\}$ and $F = \{f_1, f_2\}$ with $f_1 = v_1, f_2 = v_1 \vee v_2$. v_1 is a source node of $\mathcal{N}$. $\mathcal{N}$ has three fixed points: 00, 01, 11.

Listing 2 Source ASP encoding for the BN shown in Example 7.

```
p1, n1.                                                              1
{p1}.                               {n1}.                            2
:- p2, n2.                          p2, n2.                          3
                                                                     4
p1 :- p1.                           n1 :- n1.                        5
                                                                     6
p2 :- aux1.                                                          7
aux1 :- p1.                         aux1 :- p2.                      8
n2 :- n1, n2.                                                        9
                                                                    10
#show p1/0.       #show n1/0.       #show p2/0.     #show n2/0.     11
```

4.3 Post-processing

The set of maximal set-inclusion stable models of the encoded ASP $\mathcal{L}_s$ with respect to the shown atoms can be seen as a meta result from which we can easily retrieve the fixed points. Note that a stable model can be group-able with multiple ones, and thus the sets of fixed points of two maximal set-inclusion stable models of $\mathcal{L}_s$ can intersect. In other words, a fixed point of the BN may be included in two different maximal set-inclusion stable models of $\mathcal{L}_s$. Hence, it is not straightforward to obtain the number of fixed points, which by itself is a common difficult problem related to #SAT (*model-counting*, see [44]). Moreover, many more precise analysis questions can be answered, but not directly from the above meta result. For example, given a state x^n on $V \setminus V^s$, return the set of states x^s on V^s such that (x^n, x^s) is a fixed point of the BN. With this motivation, we propose a post-processing step as follows.

We maintain a hash table (denoted by H) with as *key* a state x^n on $V \setminus V^s$ and as associated *value* the set of states x^s on V^s such that (x^n, x^s) is a fixed point of the BN. For each stable model A of the meta result, we extract x^n from it by simply checking either p_i or n_i belongs to A for all $v_i \in V \setminus V^s$. For each $v_i \in V^s$, x_i can receive value 0 if $n_i \in A$ and value 1 if $p_i \in A$. Then we get the set of states on V^s (denoted by S^s) as the combinations of all possible values of all $v_i \in V^s$. If x^n is not a key of H, we just add the pair (x^n, S^s) to H. Otherwise, we replace the current associated value of x^n in H by the union of it and S^s because the current associated value and S^s may intersect. When there are many nodes in V^s, S^s may contain a large number of states, even exponential in the number of nodes in V^s. Binary Decision Diagrams (BDDs) [11] are an efficient data structure for representing a set of states as well as performing set operations. Hence, we store S^s as a BDD where each node $v_i \in V^s$ corresponds to a BDD variable.

Now, let us show how to answer some analysis questions from the hash table H. First, for the example analysis question mentioned in the beginning of Subsection 4.3, if x^n is not a key of H, we return the empty set. Otherwise, the set of all states on V^s can be easily retrieved from the BDD as the value of x^n in H. Specifically, we list all satisfying valuations of this BDD. Analogously, by traversing all items in H, we can also answer the question, given a combination of values on source nodes, to return the set of fixed points of the BN restricted by this combination.

Second, we can efficiently compute the number of fixed points in the BN based on H. For a given BDD, we can efficiently compute its number of satisfying valuations. Such procedure is linear in the number of nodes in this BDD [11]. Since any two keys in H are distinct, the sets of fixed points on V corresponding to them are also distinct. Hence, the number of fixed points of the BN is equivalent to the sum of all numbers of satisfying valuations for all BDDs in H. This is in contrast with most model-counting problems, where approximate methods are usually necessary to minimize the number of calls to a SAT solver.

Note that an alternative approach might be to represent the hash table as a sole BDD where every node of the BN corresponds to a BDD variable. As such, the new BDD will have $|V|$ variables, whereas each BDD in H has $|V^s|$ variables. Recall that the size of a BDD may be exponential in its number of variables [11]. Hence, splitting a BDD into multiple BDDs with smaller numbers of variables is a good strategy in most cases, especially when $|V|$ is much larger than $|V^s|$. Indeed, in our experiments shown in Section 5, we observed that the alternative approach using a sole BDD gave poorer performance than the approach using a hash table in most cases.

To conclude this section, we discuss the advantages of the source encoding. First, it inherits all the advantages of the conjunctive encoding, which can be seen as its core part. Second, the number of maximal set-inclusion stable models of the encoded ASP may be much smaller than the number of fixed points. Hence, the proposed method can have memory benefits as compared to other ASP-based methods. Third, with a much smaller number of stable models, the solving time can be much smaller. Although the proposed method needs to spend time for the post-processing, it can have running-time benefits as compared to other ASP-based methods. The second and third points shall be analyzed in our experiments shown in Section 5. Note that even if some of the extreme examples of Section 5 do not correspond to plausible biological processes, finding common patterns among them could give biological insights. Since the source method can provide a compact representation of the set of fixed points, it can be used to find such common patterns. For example, we can know that a key-value of the hash table corresponds to a set of fixed points that only differ in values of source nodes. This is a very useful result since different source nodes values usually represent different (environmental) states of a biological system. Some analyses made possible by this encoding are: listing all fixed points, counting the number of fixed points, listing/counting all fixed points under a specific value combination of source nodes, listing the *core* fixed points projected on only normal nodes (this is exactly the list of key values of the hash table).

5 Experimental results

We implemented the newly proposed methods as a Python package named `fASP`[3]. For convenience, we name the method based the conjunctive encoding presented in Section 4.1 as `fASP-conj` and the method based on the source encoding for the case of source nodes presented in Section 4.2 as `fASP-src`. To evaluate their performance, we compared them with the four state-of-the-art methods for fixed point enumeration in BNs, including `PyBoolNet` [28], `mpbn` [34], `AN-ASP` [1], and `FPCollector` [7].

We benchmarked all the compared methods on the `BBM` repository[4], a collection of real-world Boolean models from various sources used in systems biology. `BBM` consists of 211 models, peaking at 321 variables, 1100 regulations, and 133 source nodes, respectively. Furthermore, we also included a selection of 13 real-world models that are not covered by the `BBM` repository. The BNs of this selection peak at 3158 variables, 43642 regulations, and 237 source nodes, respectively. To our knowledge, the BBM repository along with this selected set is a highly representative sample of Boolean models currently available in the literature.

To solve the ASP problems, we used the same ASP solver `Clingo` [18] and the same configuration as that used in `PyBoolNet`, `mpbn`, and `AN-ASP`. Specifically for computing maximal set-inclusion stable models, we used the configuration `-heuristic=Domain -enum-mod=domRec -dom-mod=3` (subset maximality, equivalent to the deprecated `-dom-pref=32 -heuristic=domain -dom-mod=7` used by `PyBoolNet`). We ran all the benchmarks on a machine whose environment is CPU: Intel® Core™ i9-11950H 2.60GHz × 16, 16 GB DDR4 RAM, Ubuntu 20.04.5 LTS. Note that for the methods `PyBoolNet`, `mpbn`, `AN-ASP`, and `fASP-conj`, we can control the maximum number of fixed points returned because a resulting stable model corresponds to a single fixed point. In contrast, `FPCollector` requires to compute all fixed points, and `fASP-src` allows its user to set the maximum number of resulting stable models but not of resulting fixed points. Since a model can have a huge number of fixed points due to many source nodes, which might not be biologically plausible, obtaining a sample of those can prove to be very useful to invalidate the model and lead to further refinement. Hence, to obtain a relevant, reliable and fair comparison, in our

[3] The source code and benchmarks are freely accessible at `https://github.com/giang-trinh/fASP`.
[4] `https://github.com/sybila/biodivine-boolean-models`

benchmarks, we searched for both all the fixed points and the first 1000 fixed points for each model. In addition, existing analysis shown in the literature usually revolves then around fixing some source nodes to plausible values and reducing the model accordingly. Although this approach biologically makes sense, it relies on potentially arbitrary decisions, and *hides away* critical modelling choices that were actually not part of the original BN. Hence, we did not fix specific values for source nodes in all the considered models. Finally, we set a time limit of two minutes (resp. ten minutes) for each model with respect to enumerating the first 1000 fixed points (resp. all the fixed points).

5.1 BBM repository

With regards to enumerating the first 1000 fixed points, the number of BBM models solved within two minutes of each method is: PyBoolNet (198), mpbn (185), AN-ASP (210), and fASP-conj (211). Note that there are 24 non-locally-monotonic models that mpbn cannot handle. Figure 1 shows the cumulative numbers of models solved by the four compared methods with respect to enumerating the first 1000 fixed points. As it can be observed, the AN-ASP and fASP-conj methods are comparable and they vastly outperform the PyBoolNet and mpbn methods. For every time limit, their numbers of solved models are always greater than those of PyBoolNet and mpbn, especially the difference is large for the time limit of 0.5s. In particular, AN-ASP could handle all but one model and fASP-conj could handle all models within 10s.

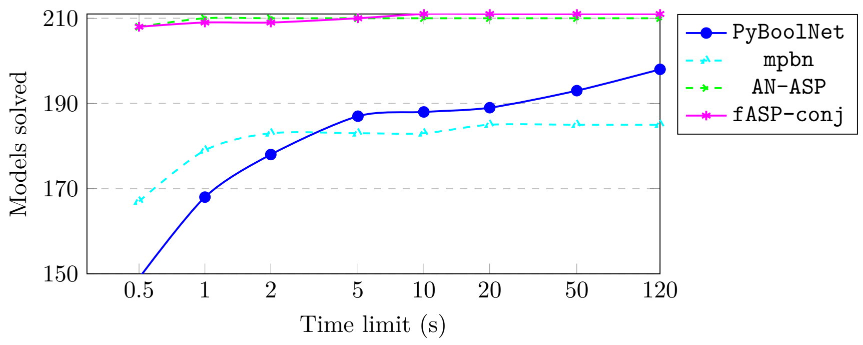

Figure 1 Cumulative numbers of BBM models solved by the four compared methods with respect to enumerating the first 1000 fixed points.

With regards to enumerating all the fixed points, the number of BBM models solved within ten minutes of each method is: FPCollector (172), PyBoolNet (170), mpbn (162), AN-ASP (179), fASP-conj (181), and fASP-src (195). We can see that fASP-src solved more models than all the other methods. One can note that the models for which fASP-src was the only successful method, have extremely large numbers of fixed points, each time because of many source nodes. This confirms our expectation when proposing fASP-src to deal with the case of many source nodes. Upon closer inspection, Figure 2 depicts the cumulative numbers of models solved by the six studied methods with respect to enumerating all the fixed points. As it can be observed, the AN-ASP and fASP-conj methods are still comparable and they outperform the FPCollector, PyBoolNet, and mpbn methods. Furthermore, for every time limit, the number of solved models using fASP-src is always the highest.

It is worth noting that, most models in the BBM repository have moderate numbers of nodes ($n < 200$) and quite simple Boolean functions. Hence, the difference in performance among the three best methods in terms of BBM models (i.e., AN-ASP, fASP-conj, and fASP-src) is not much exhibited. We shall test more in the following subsections.

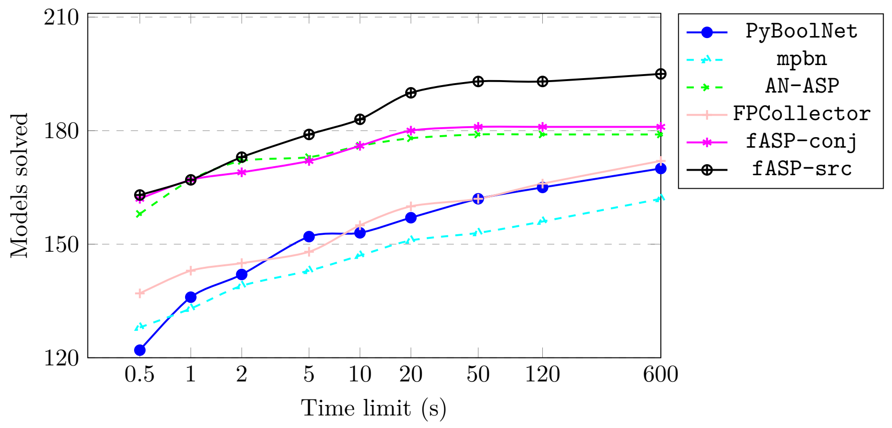

Figure 2 Cumulative numbers of `BBM` models solved by the six compared methods with respect to enumerating all the fixed points.

5.2 Selected real-world models

Table 1 shows the running time of the four compared methods on the 13 selected BNs with respect to enumerating the first 1000 fixed points. Columns n, s, and $|A|$ denote the number of nodes, the number of source nodes, and the number of fixed points, respectively. "DNF" means that the method did not finish the computation within the time limit of two minutes. "NM" indicates a non-locally-monotonic model. We can easily see that `fASP-conj` is the best method in all models as it is much faster than all the other methods on every model. In particular, it only took 35.24s to enumerate the first 1000 fixed points of the hardest model (i.e., the Cell-Cycle-Control model), whereas none of the other methods could finish the computation. Upon closer inspection, `AN-ASP` is the second best method with running time less than 1s in most models. However, it ran quite slowly on the Insulin model, could not handle the Yeast-Pheromone model, and even got an *Out of Memory* (OOM) error on the Cell-Cycle-Control model before finishing the automata network construction. We note that the above problems all come from the bottleneck of `AN-ASP`, i.e., the high number of transitions of the corresponding AN. Finally, consistent with the observations reported in the previous subsection, `PyBoolNet` and `mpbn` still performed much slower than the `AN-ASP` and `fASP-conj` methods, and there are four non-locally-monotonic models (out of the 13) that `mpbn` cannot handle.

Table 2 shows the running time of the six benchmarked methods on the selected BNs with respect to enumerating all the fixed points. Column $|A|$ denotes the number of fixed points, and a "?" denotes the case where none of the competing methods returned the result. We can first see that for 6 of the 13 models, none of the competing methods returned all the fixed points. This is clearly because the numbers of fixed points of these models are extremely large. `FPCollector` only succeeded for two models, with each less than 100 nodes. Of course, it is difficult to handle models of larger size. `PyBoolNet` and `mpbn` only succeeded for three models each. All these models are easy for the other methods. Hereafter, we shall present closer inspection on the three most efficient methods: `AN-ASP`, `fASP-conj`, and `fASP-src`.

First, `fASP-conj` is still much faster than `AN-ASP` for all the models where they both succeeded. Second, `fASP-src` is the sole method that could return all the fixed points of the T-Cell-Co-Receptor model within the time limit. Note that the number of fixed points of this model is huge, and it is apparent that none of the other methods can handle it. Moreover,

Table 1 Timing comparisons (in seconds) among `PyBoolNet` (`PBN`), `mpbn`, `AN-ASP`, and `fASP-conj` (`f-conj`) on the selected models from the literature for enumerating the first 1000 fixed points.

	model	n	s	$\|A\|$	`PBN`	`mpbn`	`AN-ASP`	`f-conj`
1	Cell-Cycle-Control [36]	3158	61	1000^+	DNF	DNF	OOM	35.24
2	EMT-Mechan [38]	136	4	82	6.02	0.17	0.22	0.05
3	EMT-Mechan-TGFbeta [38]	150	6	478	7.12	0.39	0.27	0.08
4	Alzheimer [2]	762	237	1000^+	DNF	NM	0.55	0.31
5	MAPK [2]	181	37	1000^+	8.81	0.91	0.23	0.09
6	Mast-Cell-Activation [2]	73	19	1000^+	0.07	0.30	0.14	0.03
7	Cholocystokinin [2]	383	74	1000^+	0.60	1.47	0.29	0.13
8	HOG [36]	43	5	1000^+	11.08	NM	0.17	0.07
9	Insulin [36]	82	7	1000^+	DNF	DNF	13.65	0.21
10	Leishmania [17]	342	81	1000^+	DNF	1.33	0.37	0.14
11	Pluripotency [49]	36	7	412	DNF	NM	0.22	0.02
12	T-Cell-Co-Receptor [16]	206	39	1000^+	DNF	0.79	0.28	0.09
13	Yeast-Pheromone [36]	246	17	1000^+	DNF	NM	DNF	1.01

Table 2 Timing comparisons (in seconds) among `FPCollector` (`FP`), `PyBoolNet` (`PBN`), `mpbn`, `AN-ASP`, `fASP-conj` (`f-conj`), and `fASP-src` (`f-src`) on the selected models from the literature with respect to enumerating all the fixed points.

	model	$\|A\|$	`FP`	`PBN`	`mpbn`	`AN-ASP`	`f-conj`	`f-src`
1	Cell-Cycle-Control	?	DNF	DNF	DNF	OOM	DNF	DNF
2	EMT-Mechan	82	DNF	5.82	0.15	0.30	0.05	0.08
3	EMT-Mechan-TGFbeta	478	DNF	7.05	0.38	0.26	0.08	0.09
4	Alzheimer	?	DNF	DNF	NM	OOM	OOM	DNF
5	MAPK	?	DNF	DNF	DNF	OOM	OOM	DNF
6	Mast-Cell-Activation	524288	5.45	DNF	145.87	11.84	8.96	6.34
7	Cholocystokinin	?	DNF	OOM	DNF	OOM	OOM	DNF
8	HOG	25632	149.20	17.77	NM	0.54	0.79	1.81
9	Insulin	563200	DNF	DNF	DNF	81.93	36.73	107.60
10	Leishmania	?	DNF	DNF	DNF	OOM	OOM	DNF
11	Pluripotency	412	1.30	DNF	NM	0.27	0.05	0.02
12	T-Cell-Co-Receptor	441039454208	DNF	DNF	DNF	OOM	OOM	349.15
13	Yeast-Pheromone	?	DNF	DNF	NM	DNF	DNF	DNF

both `AN-ASP` and `fASP-conj` met the OOM error, which confirms the memory advantage of `fASP-src`. For the six other models where it succeeded, `fASP-src` is comparable to `AN-ASP` and `fASP-conj`, with a bit slower running time on average. It is apparent because `fASP-src` suffers from the overhead of its post-processing based on BDDs. Indeed, we confirmed that in most of these models (also of other models considered in our experiments), the ASP solving time of `fASP-src` is negligible and most of its running time was spent for the post-processing. Note that the running time of this BDD-based post-processing depends on several factors, such as, the number of resulting stable models, the number of source nodes (the number of BDD variables), and the BDD variable ordering.

We also note that for the six failed models, the difference among the three best methods (i.e., `AN-ASP`, `fASP-conj`, and `fASP-src`) is not clear because they all did not finish or met the OOM error. Hence, we conduct new analysis on these failed models by restricting the maximum number of stable models for `fASP-src`. We then use the number of fixed points obtained by `fASP-src` as the maximum number of fixed points for the case of `AN-ASP` or `fASP-conj`. The experimental settings are the same as those used for the case of enumerating all fixed points. Table 3 shows the results of this analysis. For `fASP-src`, we consider two maximum numbers: 100 and 200. For each case, Column $|A|$ denotes the number of fixed points obtained by `fASP-src`, and the other columns denotes the running time of the corresponding methods to obtain that number of fixed points. We can see that for the four models (Cell-Cycle-Control, Alzheimer, Cholocystokinin, and Leishmania), both `AN-ASP` and `fASP-conj` met the OOM error, whereas `fASP-src` can handle them in reasonable time. For the MAPK and Yeast-Pheromone models, `fASP-src` is much faster than both `fASP-conj` and `AN-ASP`. This is also true for the Mast-Cell-Activation and Pluripotency models in the case of enumerating all fixed points (see Table 2). The above observations confirm an advantage in both memory and run-time of using `fASP-src` in the case of many source nodes.

Table 3 Timing comparisons (in seconds) among `AN-ASP`, `fASP-conj` (`f-conj`), and `fASP-src` (`f-src`) on the six failed models.

		100				200			
	model	|A|	f-src	AN-ASP	f-conj	|A|	f-src	AN-ASP	f-conj
1	Cell-Cycle-Control	159744	21.98	OOM	OOM	323584	23.73	OOM	OOM
2	Alzheimer	$> 10^{32}$	10.89	OOM	OOM	$> 10^{32}$	29.93	OOM	OOM
3	MAPK	31744	0.19	1.74	1.37	52480	0.42	2.77	2.11
4	Cholocystokinin	$> 10^{13}$	0.28	OOM	OOM	$> 10^{13}$	0.55	OOM	OOM
5	Leishmania	$> 10^{13}$	0.46	OOM	OOM	$> 10^{13}$	0.79	OOM	OOM
6	Yeast-Pheromone	1552	0.91	DNF	1.14	3152	0.91	DNF	1.45

5.3 Pseudo-random models

The results on the 224 real-world models reported in the two previous subsections draw a quite clear picture about the performance of the six compared methods. However, we observed that there is only one model with more than 1000 nodes (i.e., the Cell-Cycle-Control model). Moreover, in most of them, the Boolean functions are quite simple, even sometimes just simple conjunctions/disjunctions of literals. The reason for these facts may be that the modelers were restricted by the limited performance of the tools supported at the time they created the models. This is not the case of the Cell-Cycle-Control model, since its authors only conducted simulations instead of formal analysis [36]. Since in the present work we target large and complex BNs, we set out to test the performance of our proposed methods on larger and more complicated models than the ones available in the literature to date. Specifically, we wanted to test models with 1000 or more nodes and Boolean functions in complicated forms. Such a model is arguably not possible to achieve yet with hand-made modeling, even with a fully or semi-automated inference technique [2], but might be in the near future.

To create a benchmark set of larger and more complex models, we decided to generate pseudo-random models following the generation approach proposed by the research group who created and is maintaining the `BBM` repository. This generation approach is described in detail in [9] and its implementation is provided at `https://github.com/daemontus/artifact_cav2021`. In general, it generates Boolean models structurally similar to the real-world models in the `BBM` repository. To ensure this structural similarity, the generator uses a node-degree distribution sampled from the `BBM` repository, as opposed to other theoretical random network models. Once the regulators of a node are specified, its Boolean function is generated by randomly choosing between $\wedge$ and $\vee$ when connecting the positive/negative literals of the regulators. Note that this option does not cover the full spectrum of possible Boolean functions, but it can make the generated Boolean functions complicated enough for evaluation.

In the end, we created 400 pseudo-random models ranging from 1000 to 5000 variables, 4145 to 63507 regulations, and 127 to 1171 source nodes, respectively. We then tested all the competing methods on these models. We first reported that all the competing methods failed to obtain all the fixed points as they quickly met the OOM error. The reason is that the number of all fixed points is actually too large due to a lot of source nodes (> 100). Hence, we here only searched for the first 1000 fixed points, which might also be more biologically relevant. The time limit for each model was set to two minutes. The other settings are the same as those used in the two previous subsections.

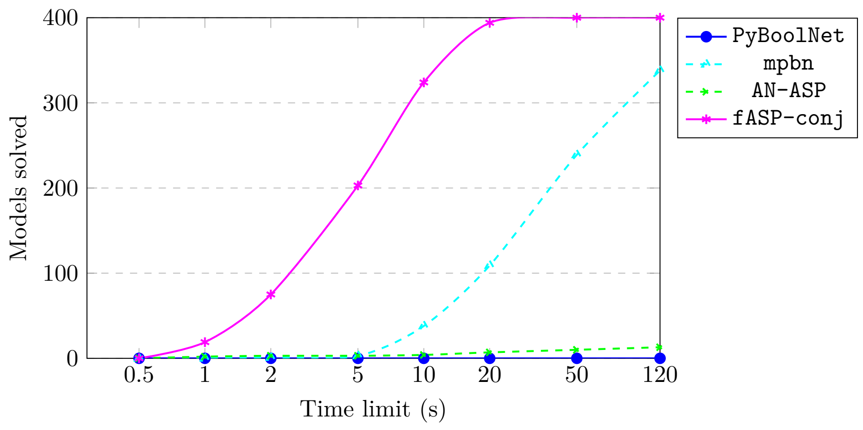

Figure 3 Cumulative numbers of pseudo-random models solved by the competing methods with respect to enumerating the first 1000 fixed points.

With regards to enumerating the first 1000 fixed points, the number of pseudo-random models solved within two minutes of each competing method is: `PyBoolNet` (0), `mpbn` (338), `AN-ASP` (13), and `fASP-conj` (400). `PyBoolNet` could not handle any model, and it failed at the phase of computing prime implicants in most cases. This is not surprising since the models are large in size and the formulae quite complex. Interestingly, `mpbn` could handle far more models than `AN-ASP`. This can be explained by the fact that the number of transitions in the corresponding AN is very large in most models, whereas the size of the DNF for each Boolean function is still moderate. Moreover, the models are all locally-monotonic, which is an assumption of the generator [9]. Upon closer inspection, Figure 3 depicts the cumulative numbers of pseudo-random models solved by the four competing methods with respect to enumerating the first 1000 fixed points. We can see that `fASP-conj` is the best method as it vastly outperforms the three other methods for every time limit except the time limit of 0.5s where all the methods could handle no model. In particular, it could handle all the 400 pseudo-random models within 50s.

6 Conclusion and future work

In this work we have proposed two new ASP-based methods called `fASP-conj` and `fASP-src` for efficiently enumerating fixed points of Boolean networks, which are crucial in modeling and analysis of biological systems. `fASP-conj` is based on conjunctive ASP and `fASP-src` is a modification of `fASP-conj` to handle the case of a large number of source nodes. The main advantage of these methods is that they only rely on NNFs of Boolean functions, which are much more efficient to obtain than other representations used by previous methods (e.g., prime-implicants, disjunctive normal forms, automata networks). The main advantage of `fASP-src` is that it provides a more compact representation of the results based on BDDs, which can give both memory and run-time benefits. We have also formally proved the correctness of the above new methods.

We have then benchmarked their performance against the four other state-of-the-art tools: `FPCollector`, `PyBoolNet`, `mpbn`, and `AN-ASP`. The experimental results on both real-world and pseudo-random models show that the new methods vastly outperform the state-of-the-art as they can robustly handle various types of large and complex models, whereas the other methods cannot. In particular, they can handle models of up to 5000 nodes with very complicated Boolean update functions. In terms of enumerating the first 1000 fixed points (resp. all fixed points), the experimental results show that `fASP-conj` is the best (resp. second best) method. `fASP-src`, which is based on `fASP-conj`, shows its superiority to all the other methods in enumerating all the fixed points of models with many source nodes.

Boolean network models of biological systems usually contain many source nodes, which might be hard to avoid in the modeling process [2]. Currently, there are many such models that `fASP-src` cannot handle. Hence, improving `fASP-src` is necessary. Note that, in some cases, the number of auxiliary atoms in the core encoding of `fASP-conj` and `fASP-src` can be reduced. Such optimization will be looked into in the future. Furthermore, we also plan to extend the methods proposed in this present paper to those for computing trap spaces of Boolean networks, which are more general than fixed points and useful approximations for complex attractors in Boolean networks. It is crucial because the state-of-the-art methods for the trap space computation are all unable to robustly handle large and complex models, for instance, the models used in our experiments here.

References

1 Emna Ben Abdallah, Maxime Folschette, Olivier F. Roux, and Morgan Magnin. ASP-based method for the enumeration of attractors in non-deterministic synchronous and asynchronous multi-valued networks. *Algorithms Mol. Biol.*, 12(1):20:1–20:23, 2017. `doi:10.1186/s13015-017-0111-2`.

2 Sara Sadat Aghamiri, Vidisha Singh, Aurélien Naldi, Tomás Helikar, Sylvain Soliman, Anna Niarakis, and Jinbo Xu. Automated inference of Boolean models from molecular interaction maps using CaSQ. *Bioinform.*, 36(16):4473–4482, 2020. `doi:10.1093/bioinformatics/btaa484`.

3 Tatsuya Akutsu, Satoru Kuhara, Osamu Maruyama, and Satoru Miyano. A system for identifying genetic networks from gene expression patterns produced by gene disruptions and overexpressions. *Genome Informatics*, 9:151–160, 1998. `doi:10.11234/gi1990.9.151`.

4 Tatsuya Akutsu, Avraham A. Melkman, Takeyuki Tamura, and Masaki Yamamoto. Determining a singleton attractor of a Boolean network with nested canalyzing functions. *J. Comput. Biol.*, 18(10):1275–1290, 2011. `doi:10.1089/cmb.2010.0281`.

5 Tatsuya Akutsu, Yang Zhao, Morihiro Hayashida, and Takeyuki Tamura. Integer programming-based approach to attractor detection and control of Boolean networks. *IEICE Trans. Inf. Syst.*, 95-D(12):2960–2970, 2012. `doi:10.1587/transinf.E95.D.2960`.

6 Julio Aracena. Maximum number of fixed points in regulatory Boolean networks. *Bull. Math. Biol.*, 70:1398–1409, 2008. `doi:10.1007/s11538-008-9304-7`.

7 Julio Aracena, Luis Cabrera-Crot, and Lilian Salinas. Finding the fixed points of a Boolean network from a positive feedback vertex set. *Bioinform.*, 37(8):1148–1155, 2021. `doi:10.1093/bioinformatics/btaa922`.

8 Ferhat Ay, Günhan Gülsoy, and Tamer Kahveci. Finding steady states of large scale regulatory networks through partitioning. In *International Workshop on Genomic Signal Processing and Statistics*, pages 1–4. IEEE, 2010. doi:10.1109/GENSIPS.2010.5719669.

9 Nikola Benes, Lubos Brim, Samuel Pastva, and David Safránek. Computing bottom SCCs symbolically using transition guided reduction. In *International Conference on Computer Aided Verification*, pages 505–528. Springer, 2021. doi:10.1007/978-3-030-81685-8_24.

10 Nikolaos Berntenis and Martin Ebeling. Detection of attractors of large Boolean networks via exhaustive enumeration of appropriate subspaces of the state space. *BMC Bioinform.*, 14(1):1–10, 2013. doi:10.1186/1471-2105-14-361.

11 R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, 1986.

12 Stéphanie Chevalier, Vincent Noël, Laurence Calzone, Andrei Yu. Zinovyev, and Loïc Paulevé. Synthesis and simulation of ensembles of Boolean networks for cell fate decision. In *International Conference on Computational Methods in Systems Biology*, pages 193–209. Springer, 2020. doi:10.1007/978-3-030-60327-4_11.

13 Karla Fabiola Corral-Jara, Camille Chauvin, Wassim Abou-Jaoudé, Maximilien Grandclaudon, Aurélien Naldi, Vassili Soumelis, and Denis Thieffry. Interplay between SMAD2 and STAT5A is a critical determinant of IL-17A/IL-17F differential expression. *Mol. Biomed.*, 2(1):9, 2021. doi:10.1186/s43556-021-00034-3.

14 Steve Dworschak, Susanne Grell, Victoria J. Nikiforova, Torsten Schaub, and Joachim Selbig. Modeling biological networks by action languages via answer set programming. *Constraints An Int. J.*, 13(1-2):21–65, 2008. doi:10.1007/s10601-007-9031-y.

15 Swann Floc'Hlay, Maria Dolores Molina, Céline Hernandez, Emmanuel Haillot, Morgane Thomas-Chollier, Thierry Lepage, and Denis Thieffry. Deciphering and modelling the TGF-β signalling interplays specifying the dorsal-ventral axis of the sea urchin embryo. *Dev.*, 148(2):dev189944, 2021. doi:10.1242/dev.189944.

16 Piyali Ganguli, Saikat Chowdhury, Rupa Bhowmick, and Ram Rup Sarkar. Temporal protein expression pattern in intracellular signalling cascade during T-cell activation: A computational study. *J. Biosci.*, 40(4):769–789, September 2015. doi:10.1007/s12038-015-9561-1.

17 Piyali Ganguli, Saikat Chowdhury, Shomeek Chowdhury, and Ram Rup Sarkar. Identification of Th1/Th2 regulatory switch to promote healing response during leishmaniasis: a computational approach. *EURASIP J. Bioinform. Syst. Biol.*, 2015:13, 2015. doi:10.1186/s13637-015-0032-7.

18 Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. Potassco: The Potsdam answer set solving collection. *AI Commun.*, 24(2):107–124, 2011. doi:10.3233/AIC-2011-0491.

19 Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *International Conference and Symposium on Logic Programming*, pages 1070–1080. MIT Press, 1988.

20 Carlos Gershenson. Classification of random Boolean networks. In *Artificial Life VIII, Proceedings of the Eighth International Conference on Artificial Life*, pages 1–8, 2003.

21 Trinh Van Giang, Tatsuya Akutsu, and Kunihiko Hiraishi. An FVS-based approach to attractor detection in asynchronous random Boolean networks. *IEEE ACM Trans. Comput. Biol. Bioinform.*, 19(2):806–818, 2022. doi:10.1109/TCBB.2020.3028862.

22 Leon Glass and Stuart A Kauffman. The logical analysis of continuous, non-linear biochemical control networks. *J. Theor. Biol.*, 39(1):103–129, 1973. doi:10.1016/0022-5193(73)90208-7.

23 Eric Goles and Lilian Salinas. Sequential operator for filtering cycles in Boolean networks. *Adv. Appl. Math.*, 45(3):346–358, 2010. doi:10.1016/j.aam.2010.03.002.

24 A Gonzalez Gonzalez, Aurélien Naldi, Lucas Sanchez, Denis Thieffry, and Claudine Chaouiya. GINsim: a software suite for the qualitative modelling, simulation and analysis of regulatory networks. *Biosyst.*, 84(2):91–100, 2006. doi:10.1016/j.biosystems.2005.10.003.

25 Changki Hong, Jeewon Hwang, Kwang-Hyun Cho, and Insik Shin. An efficient steady-state analysis method for large Boolean networks with high maximum node connectivity. *PLoS One*, 10(12):e0145734, December 2015. `doi:10.1371/journal.pone.0145734`.

26 Katsumi Inoue. Logic programming for Boolean networks. In *International Joint Conference on Artificial Intelligence*, pages 924–930. IJCAI/AAAI, 2011. `doi:10.5591/978-1-57735-516-8/IJCAI11-160`.

27 Roland Kaminski, Torsten Schaub, Anne Siegel, and Santiago Videla. Minimal intervention strategies in logical signaling networks with ASP. *Theory Pract. Log. Program.*, 13(4-5):675–690, 2013. `doi:10.1017/S1471068413000422`.

28 Hannes Klarner, Adam Streck, and Heike Siebert. PyBoolNet: a python package for the generation, analysis and visualization of Boolean networks. *Bioinform.*, 33(5):770–772, 2017. `doi:10.1093/bioinformatics/btw682`.

29 Tomoya Mori and Tatsuya Akutsu. Attractor detection and enumeration algorithms for Boolean networks. *Comput. Struct. Biotechnol. J.*, 20:2512–2520, 2022. `doi:10.1016/j.csbj.2022.05.027`.

30 Mushthofa Mushthofa, Gustavo Torres, Yves Van de Peer, Kathleen Marchal, and Martine De Cock. ASP-G: an ASP-based method for finding attractors in genetic regulatory networks. *Bioinform.*, 30(21):3086–3092, 2014. `doi:10.1093/bioinformatics/btu481`.

31 Aurélien Naldi. BioLQM: a Java toolkit for the manipulation and conversion of logical qualitative models of biological networks. *Front. Physiol.*, 9:1605, 2018. `doi:10.3389/fphys.2018.01605`.

32 Aurélien Naldi, Pedro T. Monteiro, Christoph Müssel, Hans A. Kestler, Denis Thieffry, Ioannis Xenarios, Julio Saez-Rodriguez, Tomás Helikar, and Claudine Chaouiya. Cooperative development of logical modelling standards and tools with CoLoMoTo. *Bioinform.*, 31(7):1154–1159, 2015. `doi:10.1093/bioinformatics/btv013`.

33 Karen J Nuñez-Reza, Aurélien Naldi, Arantza Sánchez-Jiménez, Ana V Leon-Apodaca, M Angélica Santana, Morgane Thomas-Chollier, Denis Thieffry, and Alejandra Medina-Rivera. Logical modelling of in vitro differentiation of human monocytes into dendritic cells unravels novel transcriptional regulatory interactions. *Interface Focus*, 11(4):20200061, 2021. `doi:10.1098/rsfs.2020.0061`.

34 Loïc Paulevé, Juraj Kolčák, Thomas Chatain, and Stefan Haar. Reconciling qualitative, abstract, and scalable modeling of biological networks. *Nat. Commun.*, 11(1), August 2020. `doi:10.1038/s41467-020-18112-5`.

35 Alexandre Rocca, Nicolas Mobilia, Eric Fanchon, Tony Ribeiro, Laurent Trilling, and Katsumi Inoue. ASP for construction and validation of regulatory biological networks. *Logical Modeling of Biological Systems*, pages 167–206, 2014.

36 Jesper C Romers and Marcus Krantz. rxncon 2.0: a language for executable molecular systems biology. *BioRxiv*, page 107136, 2017. `doi:10.1101/107136`.

37 Torsten Schaub and Sven Thiele. Metabolic network expansion with answer set programming. In *International Conference on Logic Programming*, pages 312–326. Springer, 2009. `doi:10.1007/978-3-642-02846-5_27`.

38 Emmalee Sullivan, Marlayna Harris, Arnav Bhatnagar, Eric Guberman, Ian Zonfa, and Erzsébet Ravasz Regan. Boolean modeling of mechanosensitive Epithelial to Mesenchymal Transition and its reversal. *bioRxiv*, September 2022. `doi:10.1101/2022.08.29.505701`.

39 René Thomas. Boolean formalisation of genetic control circuits. *J. Theor. Biol.*, 42:565–583, 1973. `doi:10.1016/0022-5193(73)90247-6`.

40 René Thomas. Regulatory networks seen as asynchronous automata: a logical description. *J. Theor. Biol.*, 153(1):1–23, 1991. `doi:10.1016/S0022-5193(05)80350-9`.

41 René Thomas and Richard d'Ari. *Biological feedback.* CRC press, 1990.

42 Van-Giang Trinh, Kunihiko Hiraishi, and Belaid Benhamou. Computing attractors of large-scale asynchronous Boolean networks using minimal trap spaces. In *ACM International Conference*

on Bioinformatics, Computational Biology and Health Informatics, pages 13:1–13:10. ACM, 2022. doi:10.1145/3535508.3545520.

43 Eirini Tsirvouli, Felicity Ashcroft, Berit Johansen, and Martin Kuiper. Logical and experimental modeling of cytokine and eicosanoid signaling in psoriatic keratinocytes. *iScience*, 24(12):103451, 2021. doi:10.1016/j.isci.2021.103451.

44 Leslie G Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

45 Alan Veliz-Cuba, Boris Aguilar, Franziska Hinkelmann, and Reinhard C. Laubenbacher. Steady state analysis of Boolean molecular network models via model reduction and computational algebra. *BMC Bioinform.*, 15(1):1–8, 2014. doi:10.1186/1471-2105-15-221.

46 Santiago Videla, Carito Guziolowski, Federica Eduati, Sven Thiele, Martin Gebser, Jacques Nicolas, Julio Saez-Rodriguez, Torsten Schaub, and Anne Siegel. Learning Boolean logic models of signaling networks with ASP. *Theor. Comput. Sci.*, 599:79–101, 2015. doi:10.1016/j.tcs.2014.06.022.

47 Santiago Videla, Julio Saez-Rodriguez, Carito Guziolowski, and Anne Siegel. caspo: a toolbox for automated reasoning on the response of logical signaling networks families. *Bioinform.*, 33(6):947–950, 2017. doi:10.1093/bioinformatics/btw738.

48 Rui-Sheng Wang, Assieh Saadatpour, and Reka Albert. Boolean modeling in systems biology: an overview of methodology and applications. *Phys. Biol.*, 9(5):055001, 2012. doi:10.1088/1478-3975/9/5/055001.

49 Ayako Yachie-Kinoshita, Kento Onishi, Joel Ostblom, Matthew A Langley, Eszter Posfai, Janet Rossant, and Peter W Zandstra. Modeling signaling-dependent pluripotency with Boolean logic to predict cell fate transitions. *Mol. Syst. Biol.*, 14(1):e7952, 2018. doi:10.15252/msb.20177952.