

# Trap spaces of Boolean networks are conflict-free siphons of their Petri net encoding

Van-Giang Trinh<sup>a</sup>, Belaid Benhamou<sup>a</sup>, Sylvain Soliman<sup>b,\*</sup>

<sup>a</sup>*LIS, Aix-Marseille University, Marseille, France*

<sup>b</sup>*Lifeware team, Inria Saclay center, Palaiseau, France*

---

## Abstract

Boolean network modeling of gene regulation but also of post-transcriptomic systems has proven over the years that it can bring powerful analyses and corresponding insight to the many cases where precise biological data is not sufficiently available to build a detailed quantitative model. Besides simulation, the analysis of such models is mostly based on attractor computation, since those correspond roughly to observable biological *phenotypes*. The recent use of trap spaces made a real breakthrough in that field allowing to consider medium-sized models that used to be out of reach. However, with the continuing increase in model size and complexity of Boolean update functions, the state-of-the-art computation of minimal trap spaces based on *prime implicants* shows its limits due to the difficulty of the prime-implicant computation.

In this article we explore and prove for the first time a connection between trap spaces of a general Boolean network and siphons of its Petri net encoding. Besides important theoretical applications in studying properties of trap spaces, the connection enables us to propose an alternative approach to compute minimal trap spaces, and hence complex attractors, of a general Boolean network. It replaces the need for *prime implicants* by a completely different technique, namely the enumeration of maximal siphons in the Petri net encoding of the original model. We then demonstrate its efficiency and compare it to the state-of-the-art methods on a large collection of real-world

---

\*Corresponding author.

*Email addresses:* `trinh.van-giang@lis-lab.fr` (Van-Giang Trinh),  
`belaid.benhamou@lis-lab.fr` (Belaid Benhamou), `Sylvain.Soliman@inria.fr`  
(Sylvain Soliman)

and randomly generated models.

*Keywords:* Logical model, Boolean network, Trap space, Attractor computation, Petri net, Siphon, Systems biology

---

## 1. Introduction

From the observation that the transcriptional regulation behaved in a sigmoid step-like way, came the original 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 [1, 2, 3, 4].

Boolean modeling made available some powerful analyses and corresponding insight for gene regulation models. Then, over the years, its use increased even for modelling post-transcriptional mechanisms, supported by the many cases where precise biological data was not sufficiently available to build a detailed quantitative model [5]. This lack of data is more frequent for large and very large models, which led to a steady increase in the size of logical models *à la* Thomas [6]. The main analysis tool for such models is the computation of its fixed and periodic attractors, since those correspond roughly to observable biological *phenotypes*. The recent use of trap spaces [7] made a real breakthrough in that field allowing to consider medium-sized models that used to be out of reach and for which only simulation was available. However, with the most recent models both being quite large and using rather complex update functions, the state-of-the-art computation of minimal trap spaces based on *prime implicants* shows its limits. More specifically, the number of prime implicants of a Boolean function is in general exponential in the number of input nodes of this function [7]. Moreover, the computation of prime implicants is a demanding task, especially for complex Boolean functions.

It is worth noting that the recent method presented in [8] for computing minimal trap spaces avoids the prime-implicant computation by relying on the *most-permissive* semantics of Boolean networks. This method has been implemented in the tool `mpbn`<sup>1</sup> demonstrated in [9] for handling medium-sized models from the literature and very large synthetic models

---

<sup>1</sup><https://github.com/bnediction/mpbn>

(up to 100,000 nodes). However, this method is only applicable for *locally-monotonic* Boolean networks, whereas the **prime implicants** based method [7] is applicable for *general* Boolean networks (i.e., including both locally-monotonic and non-locally-monotonic ones). In addition, the **bioLQM** platform also provides another method using Binary Decision Diagrams (BDDs) in <http://colomoto.org/biolqm/doc/tools-trapSpace.html>. This method avoids the prime-implicant computation as it characterizes the set of generic trap spaces of a Boolean network by a BDD, then filters this set to get the set of all minimal trap spaces. By this approach, it requires the computation of all solutions, whereas the methods [7, 9] based on Answer Set Programming (ASP) can start enumerating them as they are found. Moreover, the main issue with **this** BDD-based method is that the number of generic trap spaces of a Boolean network may be extremely larger than its number of minimal trap spaces. This issue limits the efficiency of the **current** BDD-based method. The study [10] highlights the need for non-locally-monotonic Boolean networks in both biological and theoretical aspects. Hence, it is still necessary to develop efficient methods for computing minimal trap spaces of large-scale general Boolean networks.

Petri nets were introduced in the 60s as simple formalism for describing and analyzing information-processing systems that are characterized as being concurrent, asynchronous, non-deterministic and possibly distributed [11, 12]. The use of Petri nets for representing biochemical reaction systems, by mapping molecular species to places and reactions to transitions, hinted at already in [11, 12] was used more thoroughly quite late in [13], together with some Petri net concepts and tools for the analysis of metabolic networks. Siphons are such a concept, but they have not been used a lot for the study of biochemical systems [14, 15] even if the practical cost of computing their minimal/maximal elements appear much more manageable than the theoretical complexity would indicate [16, 17].

In this article we explore and prove for the first time a connection between trap spaces of a general Boolean network and siphons of its Petri net encoding. Not only having important theoretical applications in studying properties of trap spaces in Boolean networks, the connection has important practical applications in the trap space computation. Specifically, based on the connection, we propose an alternative approach to compute minimal trap spaces, and hence complex attractors, of a general Boolean network. It replaces the need for **prime implicants** by a completely different technique, namely the enumeration of maximal siphons in the Petri net encoding of the

original model. We then demonstrate its efficiency and compare it to the state-of-the-art methods for computing minimal trap spaces of Boolean networks on many real-world models from various sources in the literature and on randomly generated models.

Herein we revise and extend our previous work in [18] as follows. First, more formal definitions are given and the existing proofs are made more detailed. In particular, an updated proof provides another way to prove the independence of trap spaces of a Boolean network with respect to its update scheme, which was originally proved in [7]. Second, we showcase a theoretical application of the connection between trap spaces in Boolean networks and conflict-free siphons in Petri nets. Third, beyond the proposed ASP method implementing the alternative approach [18], we propose several other possible methods for computing minimal trap spaces using Maximum Satisfiability (MaxSAT), Constraint Programming (CP), and Integer Linear Programming (ILP). Fourth, we discuss in detail how to compute several special types of trap spaces in a Boolean network. Besides minimal trap spaces, these special types also play crucial roles in analyzing and controlling Boolean networks [19]. Fifth, regarding the implementation, we have developed a new converter that directly reads a `.bnet` file and builds the Petri net encoding, instead of using the PNML conversion of `bioLQM` [18]. Finally, we conduct a more extensive benchmark on more real-world models from various sources and randomly generated models to evaluate all the proposed methods (the benchmark conducted in [18] considers only a few dozens of representative real-world models), therefore obtaining more comprehensive insights.

The rest of this paper is organized as follows: Section 2 recalls the basic concepts including Boolean networks, attractors, trap spaces, Petri nets, and siphons. Section 3 presents the main finding, the connection between trap spaces in Boolean networks and siphons in Petri nets. Section 4 presents the alternative approach for computing minimal trap spaces and the four possible methods implementing it. Section 5 shows an important biological case study showing the applicability of the new approach. Section 6 reports the experimental results for evaluating the efficiency of the proposed methods. Finally, Section 7 concludes the paper and draws future work.

## 2. Preliminaries

We shall briefly recall here some preliminaries on Boolean networks related to trap spaces and Petri nets.

104 2.1. Boolean networks

105 **Definition 2.1.** A Boolean Network (BN) is a pair  $\mathcal{N} = (V, F)$  where:

- 106 •  $V = \{v_1, \dots, v_n\}$  is the set of nodes. We use  $v_i$  to denote both the node  
107  $v_i$  and its associated Boolean variable.
- 108 •  $F = \{f_1, \dots, f_n\}$  is the set of update functions. Each function  $f_i$  is  
109 associated with node  $v_i$  and satisfies  $f_i: \mathbb{B}^{|IN(v_i)|} \mapsto \mathbb{B}$  where  $\mathbb{B} = \{0, 1\}$   
110 and  $IN(v_i)$  denotes the set of input nodes of  $v_i$ . Note that a node  $v_i \in V$   
111 is called a source node if and only if  $f_i = v_i$ .

112 A Boolean function is *locally-monotonic* if it can be represented by a  
113 formula in disjunctive normal form in which all occurrences of any given  
114 literal are either negated or non-negated [9]. A Boolean network is said  
115 to be locally-monotonic if all its Boolean functions are locally-monotonic.  
116 Otherwise, this model is said to be non-locally-monotonic.

A state  $s \in \mathbb{B}^n$  is as a mapping  $s: V \mapsto \mathbb{B}$  that assigns either 0 (inactive) or 1 (active) to each node. We denote the set of all possible states of a Boolean network  $\mathcal{N}$  by  $\mathcal{S}_{\mathcal{N}} = \mathbb{B}^n$ . At each time step  $t$ , node  $v_i$  can update its state by

$$s'(v_i) = f_i(s)$$

117 where  $s$  (resp.  $s'$ ) is the state of  $\mathcal{N}$  at time  $t$  (resp.  $t+1$ ). Note that for sim-  
118 plicity, we write  $f_i(s)$  even if  $IN(v_i) \subsetneq V$  (i.e.,  $IN(v_i)$  does not contain some  
119 nodes of  $V$ ). An update scheme of a Boolean network specifies the way that  
120 the nodes update their states through time evolution [20]. There are many  
121 different update schemes, but the two main types [20] are: *synchronous*,  
122 where all the nodes are updated simultaneously, and *fully asynchronous*,  
123 where only one node is selected non-deterministically to be updated. Follow-  
124 ing the update scheme, the Boolean network transits from a state to another  
125 state (possibly identical). This transition is called the *state transition* and  
126 denoted by  $\rightarrow \subseteq \mathcal{S}_{\mathcal{N}} \times \mathcal{S}_{\mathcal{N}}$ . For example, under the synchronous update  
127 scheme, we have  $x \rightarrow y$  if and only if  $y(v_i) = f_i(x), \forall v_i \in V$ , whereas under  
128 the fully asynchronous update scheme, we have  $x \rightarrow y$  if and only if there  
129 is a node  $v_i \in V$  such that  $y(v_i) = f_i(x)$  and  $y(v_j) = x(v_j), \forall v_j \in V, j \neq i$ .  
130 Then the dynamics of  $\mathcal{N}$  is captured by the directed graph  $(\mathcal{S}_{\mathcal{N}}, \rightarrow)$  called  
131 the State Transition Graph (STG).

## 132 2.2. Traps spaces

133 We recall here some definitions from [7] for the introduction of *trap spaces*.  
 134 Minimal trap spaces prove to be a very good approximation of the attractors  
 135 of a Boolean network under asynchronous update schemes and have become  
 136 the *de facto* standard way to analyze models of a few tens of *genes* [21, 22].  
 137 A non-empty set  $T \subseteq \mathcal{S}_{\mathcal{N}}$  is a *trap set* with respect to  $\rightarrow$  if for every  
 138  $x \in T$  and  $y \in \mathcal{S}_{\mathcal{N}}$  with  $x \rightarrow y$  it holds that  $y \in T$  [7]. An attractor of  
 139  $\mathcal{N}$  with respect to  $\rightarrow$  can be defined as an inclusion-wise minimal trap set  
 140 of  $(\mathcal{S}_{\mathcal{N}}, \rightarrow)$ . An attractor can be also seen as a terminal strongly connected  
 141 component of  $(\mathcal{S}_{\mathcal{N}}, \rightarrow)$  [23]. An attractor of size 1 is called a fixed point,  
 142 otherwise it is called a cyclic or complex attractor [7].

A subspace  $m$  of a Boolean network  $\mathcal{N} = (V, F)$  is a mapping  $m: V \mapsto \mathbb{B} \cup \{\star\}$ .  $m(v_i) \in \mathbb{B}$  means that the value of  $v_i$  is fixed in  $m$  and  $v_i$  is called a *fixed* variable.  $m(v_i) \in \star$  means that the value of  $v_i$  is free in  $m$  and  $v_i$  is called a *free* variable. We denote  $D_m$  the set of all fixed variables of  $m$ . A subspace  $m$  is equivalent to a set of states:

$$\mathcal{S}_{\mathcal{N}}[m] := \{s \in \mathcal{S}_{\mathcal{N}} \mid \forall v \in D_m: s(v) = m(v)\}.$$

143 For example,  $m = \star\star 1$  (for simplicity, we shall write subspaces likes states as  
 144 a sequence of values) means that  $D_m = \{v_3\}$ ,  $m(v_3) = 1$ , and it is equivalent  
 145 to the set of states  $\{001, 011, 101, 111\}$ . We denote  $\mathcal{S}_{\mathcal{N}}^{\star} = (\mathbb{B} \cup \{\star\})^n$  the set  
 146 of all possible subspaces of  $\mathcal{N}$ . Note that  $|\mathcal{S}_{\mathcal{N}}^{\star}| = 3^n$  and  $\mathcal{S}_{\mathcal{N}} \in \mathcal{S}_{\mathcal{N}}^{\star}$  [7].

147 A *trap space* is defined as a subspace that is also a trap set. It is noted  
 148 that trap spaces of a Boolean network are independent of the update scheme  
 149 of this model [7], **we provide in Corollary 3.1 another proof of this**. Then, we  
 150 define a partial order  $<$  on  $\mathcal{S}_{\mathcal{N}}^{\star}$  as:  $m < m'$  if and only if  $\mathcal{S}_{\mathcal{N}}[m] \subseteq \mathcal{S}_{\mathcal{N}}[m']$   
 151 and  $\mathcal{S}_{\mathcal{N}}[m] \neq \mathcal{S}_{\mathcal{N}}[m']$ . Consequently, a trap space  $m$  is minimal if and only  
 152 if there is no trap space  $m' \in \mathcal{S}_{\mathcal{N}}^{\star}$  such that  $m' < m$ .

153 For example, let us consider the Boolean network shown in Example 2.1.  
 154 Figure 1(a) shows the dynamics of this model under the fully asynchronous  
 155 update **scheme** (i.e., only one node is updated at each time step). The model  
 156 has all two trap spaces,  $m_1 = 11$  and  $m_2 = \star\star$ . Since  $m_1 < m_2$ ,  $m_1$  is the  
 157 only minimal trap space of the Boolean network.

158 **Example 2.1.** We give a Boolean network  $\mathcal{N} = (V, F)$ , where  $V = (x_1, x_2)$   
 159 and  $F = (f_1, f_2)$  with  $f_1 = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$ ,  $f_2 = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$ .  
 160 Herein,  $\wedge$ ,  $\vee$ , and  $\neg$  denote the logical conjunction, disjunction, and negation  
 161 operators, respectively.

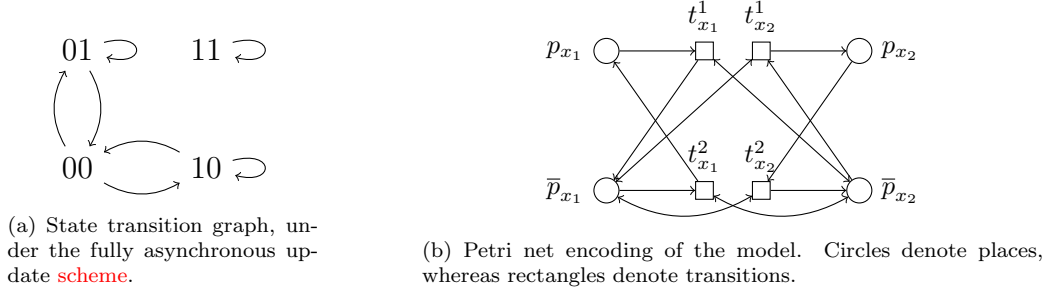


Figure 1: Dynamics and encoding of the Boolean network of Example 2.1.

### 2.3. Petri net encoding of Boolean networks

**Definition 2.2.** A Petri net is a weighted bipartite directed graph  $(P, T, W)$ , where  $P$  is a non-empty finite set of vertices called places,  $T$  is a non-empty finite set of vertices called transitions,  $P \cap T = \emptyset$ , and  $W : (P \times T) \cup (T \times P) \mapsto \mathbb{N}$  is a weight function attached to the arcs.

A marking for a Petri net is a mapping  $M : P \mapsto \mathbb{N}$  that assigns a number of tokens to each place. A place  $p$  is **marked** by a marking  $M$  if and only if  $M(p) > 0$ . We shall write  $\text{pred}(x)$  (resp.  $\text{succ}(x)$ ) to represent the set of vertices that have a (non-zero weighted) arc leading to (resp. coming from)  $x$ . In this work, we consider a class of Petri nets called 1-safe Petri nets where every place has at most 1 token and all arcs are of weight 1. **Note that in such nets we have  $M : P \mapsto \{0, 1\}$ , we might therefore represent a marking by the equivalent set of places containing a token and will use this notation for simplicity.** In this case, weights are implicitly omitted in the arcs of a Petri net. Then, a transition  $t \in T$  is **enabled** at a marking  $M$  if and only if  $\text{pred}(t) \subseteq M$ . A marking  $M$  is called a deadlock if there are no enabled transitions at  $M$ . The firing of  $t$  leads to a new marking  $M'$  specified by  $M' = (M \setminus \text{pred}(t)) \cup \text{succ}(t)$ . Note that when multiple transitions are enabled, we need to embed one firing scheme (similar to the update scheme of a Boolean network) to the Petri net. The classical firing scheme is that only one of the enabled transition is non-deterministically chosen to fire [12].

The link between Boolean networks *à la* Thomas and Petri nets was originally established in [24] in order to make available formal methods like model-checking for the analysis of such systems. The basic encoding into 1-safe (i.e., never more than one token in each place) nets only holds for purely Boolean networks but was later extended to multivalued logical models in

two ways, either in [25] with non 1-safe Petri nets or more recently in [23] with 1-safe nets but many more places.

Since our study is focused on Boolean networks, we briefly recall the original encoding here. Its basis is that every node (*gene*)  $v$  of the original model  $\mathcal{N} = (V, F)$  is represented by two separate places ( $p_v$  and  $\bar{p}_v$ ), corresponding to its two states, active, and inactive, respectively. Each conjunct of the logical function that activates the *gene* will lead to a transition  $t$ , consuming the inactive place (i.e., a directional arc from  $\bar{p}_v$  to  $t$ ), producing the active place (i.e., a directional arc from  $t$  to  $p_v$ ), and with all other literals both consumed and produced (i.e., a bidirectional arc). **Conversely a transition is added from the active place to the inactive place for each conjunct of the negation of that function.** Let  $s$  be a state of the Boolean network and  $M_s$  be its corresponding marking in the encoded Petri net. **It holds that  $\forall v \in V$ ,  $s(v) = 0$  if and only if  $M_s(\bar{p}_v) = 1$  and  $M_s(p_v) = 0$  and  $s(v) = 1$  if and only if  $M_s(p_v) = 1$  and  $M_s(\bar{p}_v) = 0$ . Note also that at any marking  $M$  of the Petri net encoding a Boolean network, it always holds that  $M(p_v) + M(\bar{p}_v) = 1$ .**

The main property of this encoding is that it is completely faithful with respect to the update scheme of the original Boolean network. For each node  $v$  of  $\mathcal{N}$ , only transitions corresponding to  $v$  can change the current marking of  $p_v$  or  $\bar{p}_v$ . **In addition, at any marking at most one of such transitions is enabled because  $M(p_v) + M(\bar{p}_v) = 1$  holds.** Hence, for any update scheme in  $\mathcal{N}$ , we have a corresponding firing scheme in  $\mathcal{P}$ , which preserves the equivalence between the dynamics of  $\mathcal{N}$  and  $\mathcal{P}$  [26].

For illustration, let us reconsider the Boolean network shown in Example 2.1. Figure 1(b) shows the Petri net encoding of this Boolean network. Place  $p_{x_1}$  (resp.  $\bar{p}_{x_1}$ ) in  $\mathcal{P}$  represents the activation (resp. the inactivation) of node  $x_1$  in  $\mathcal{N}$ . Marking  $\{p_{x_1}, \bar{p}_{x_2}\}$  in  $\mathcal{P}$  represents state 10 in  $\mathcal{N}$ . Transitions  $t_{x_1}^1$  and  $t_{x_1}^2$  represent the update of node  $x_1$ . Of course, in any marking  $t_{x_1}^1$  and  $t_{x_1}^2$  cannot be both enabled. Then, the fully asynchronous update scheme in  $\mathcal{N}$  corresponds to the classical firing scheme in  $\mathcal{P}$  where only one of the enabled transitions for a given marking will be fired [12].

Note that given a Boolean network in the standard **SBML-Qual** format [27], i.e., the package of SBML v3 [28] for such models, one can easily obtain its Petri net encoding in the Petri Net Markup Language (PNML)<sup>2</sup> standard

---

<sup>2</sup><https://www.pnml.org/>



222 using the `bioLQM`<sup>3</sup> library. This piece of software extracted from `GINsim` [29]  
 223 and part of the `CoLoMoTo`<sup>4</sup> [30] software suite allows for easy conversion  
 224 between standard formats. It also accepts many other common formats for  
 225 Boolean networks, notably the `.bnet` files of the `BoolNet` [31, 21] tools. The  
 226 conversion is executed as follows:

```
227 java -jar GINsim.jar -lqm <input.{sbml,bnet,...}> <output.pnml>
```

228 Note that transforming a Boolean network defined by its functions into its  
 229 Petri net encoding roughly relies on obtaining conditions for the activation  
 230 and inactivation of the states. In [24] this took the form of the whole truth  
 231 table of the Boolean functions, but as shown in Appendix 1 of [23] comput-  
 232 ing Disjunctive Normal Forms (DNF) of each Boolean function is enough.  
 233 Though this might appear quite computationally intensive it is important to  
 234 remark first that contrary to the **prime implicants** case, there is no need to  
 235 find *minimal* DNFs. One way to look at this is to consider that this amounts  
 236 to a similar approach as that used in [8] but with the encoding of both activa-  
 237 tion and inhibition functions as DNFs in order to take into account possible  
 238 non-local-monotonicity. This does not change the worst-case-complexity (ob-  
 239 taining a single DNF being exponential) but might matter a lot in practice.  
 240 As such, we will explore how this transformation, here using BDDs in `bioLQM`  
 241 or directly in our tool using the `pyeda`<sup>5</sup> library, and the one based on the  
 242 most-permissive semantics compare with each other in Section 6.

## 243 2.4. Siphons

244 Siphons are a static and classical property of Petri nets [11]. Note how-  
 245 ever that the use of siphons for the analysis of biological models, though it is  
 246 not new, has been mostly relevant to the ODE-based continuous semantics  
 247 of chemical reaction networks [32, 33, 34]. We recall here the basic definition  
 248 establishing that to produce something in a siphon you must consume some-  
 249 thing from the siphon. This corresponds to the idea that a siphon is a set of  
 250 places that once unmarked remains unmarked.

**Definition 2.3.** *A siphon of a Petri net  $(P, T, W)$  is a set of places  $S$  such that:*

$$\forall t \in T, S \cap \text{succ}(t) \neq \emptyset \Rightarrow S \cap \text{pred}(t) \neq \emptyset.$$

---

<sup>3</sup><http://www.colomoto.org/biolqm/>

<sup>4</sup><http://colomoto.org/>

<sup>5</sup><https://pyeda.readthedocs.io/en/latest/>

251 Note that  $\emptyset$  is trivially a siphon.

252 Let  $\text{pred}(S) := \bigcup_{s \in S} \text{pred}(s)$  and  $\text{succ}(S) := \bigcup_{s \in S} \text{succ}(s)$ . If  $S = \emptyset$ , then  
 253 conventionally  $\text{pred}(S) = \text{succ}(S) = \emptyset$ . We have an important property on  
 254 siphons [35] as follows.

255 **Proposition 2.1.** *A set  $S$  of places is a siphon of a Petri net  $(P, T, W)$  if  
 256 and only if  $\text{pred}(S) \subseteq \text{succ}(S)$ .*

### 257 3. Trap spaces as conflict-free siphons

258 First let us associate subspaces and sets of places in the Petri net encod-  
 259 ing.

**Definition 3.1.** *Let  $m$  be a subspace of Boolean network  $\mathcal{N} = (V, F)$ . A  
 mirror of  $m$  is a set of places  $S$  in the Petri net encoding  $\mathcal{P}$  of  $\mathcal{N}$  such that:*

$$\forall v \in D_m [m(v) = 0 \Leftrightarrow p_v \in S \wedge m(v) = 1 \Leftrightarrow \bar{p}_v \in S]$$

and

$$\forall v \in V \setminus D_m [p_v \notin S \wedge \bar{p}_v \notin S].$$

260 Now, we add a definition related to any set of places of a Petri net en-  
 261 coding a Boolean network, and notably a siphon of such a net.

262 **Definition 3.2.** *A set of places of Petri net  $\mathcal{P}$  encoding Boolean network  
 263  $\mathcal{N}$  is conflict-free if it does not contain any two places corresponding to the  
 264 active and inactive states of the same node of  $\mathcal{N}$ . Then, a conflict-free siphon  
 265  $S$  is said to be maximal if and only if there is no other conflict-free siphon  
 266  $S'$  such that  $S \subset S'$ .*

267 Intuitively, a siphon is a set of places that once unmarked remains so. If  
 268 it is conflict-free it is possible to associate a subspace to it, more precisely it  
 269 is the *mirror* of a subspace. Since it is a siphon, the fixed values will remain  
 270 so whatever update happens, as the unmarked places remain unmarked. The  
 271 subspace corresponding to that conflict-free siphon is therefore a trap space,  
 272 and the maximality of the siphon is equivalent to the minimality of the trap  
 273 space (as many fixed values as possible). For example, the Boolean network  
 274 given in Example 2.1 has two trap spaces,  $m_1 = 11$  and  $m_2 = \star\star$ . The  
 275 Petri net encoding of this Boolean network has five generic siphons,  $S_1 = \emptyset$ ,  
 276  $S_2 = \{p_{x_1}, \bar{p}_{x_1}\}$ ,  $S_3 = \{p_{x_2}, \bar{p}_{x_2}\}$ ,  $S_4 = \{\bar{p}_{x_1}, \bar{p}_{x_2}\}$ , and  $S_5 = \{p_{x_1}, \bar{p}_{x_1}, p_{x_2}, \bar{p}_{x_2}\}$ .

277 However, only  $S_1$  and  $S_4$  are conflict-free siphons and correspond to  $m_2$  and  
 278  $m_1$ , respectively. Since  $S_1 \subset S_4$ ,  $S_4$  is a maximal siphon corresponding to  
 279 the minimal trap space  $m_1$ . Hereafter, we formally prove that a (maximal)  
 280 conflict-free siphon is equivalent to a (minimal) trap space.

281 **Theorem 3.1.** *Let  $\mathcal{N} = (V, F)$  be a Boolean network and  $\mathcal{P}$  be its Petri net  
 282 encoding. A subspace  $m$  is a trap space of  $\mathcal{N}$  if and only if its mirror  $S$  is a  
 283 conflict-free siphon of  $\mathcal{P}$ .*

284 *Proof. First, we show that if  $m$  is a trap space of  $\mathcal{N}$ , then  $S$  is a conflict-free*  
 285 *siphon of  $\mathcal{P}$  (\*).*

286 If  $D_m = \emptyset$ , then  $S = \emptyset$  is trivially a conflict-free siphon of  $\mathcal{P}$ . Thus,  
 287 we consider the case that  $D_m \neq \emptyset$  (resp.  $S \neq \emptyset$ ). Assume that  $S$  is not a  
 288 siphon of  $\mathcal{P}$ . Then, there is a transition  $t \in T$  such that  $S \cap \text{succ}(t) \neq \emptyset$   
 289 but  $S \cap \text{pred}(t) = \emptyset$ . This implies that there is a place  $p \in S$  such that  
 290  $p \in \text{succ}(t)$  but  $p \notin \text{pred}(t)$ . Let  $v$  be the node in  $\mathcal{N}$  corresponding to  $p$ . By  
 291 the characteristics of the encoding [24], there is a directional arc from  $t$  to  $p$   
 292 and a directional arc from the complementary place of  $p$  to  $t$ . Without loss  
 293 of generality, we assume that  $p = p_v$ , then there is a directional arc from  $t$   
 294 to  $p_v$  and a directional arc from  $\bar{p}_v$  to  $t$ .

295 We follow the following procedure to find a state  $s \in \mathcal{S}_{\mathcal{N}}[m]$  such that  
 296  $M_s(p') = 1, \forall p' \in \text{pred}(t)$  where  $M_s$  is the corresponding marking in  $\mathcal{P}$  of  $s$ .  
 297 For every place  $p' \in \text{pred}(t)$ , let  $p''$  be the complementary place of  $p'$  and  $v'$   
 298 be the corresponding node in  $\mathcal{N}$  of  $p'$  and  $p''$ .

299 If  $p'' \notin S$ , then  $v' \notin D_m$  and we can always set the Boolean value to  $s(v')$   
 300 such that  $s \in \mathcal{S}_{\mathcal{N}}[m]$  and  $M_s(p') = 1$ .

301 If  $p'' \in S$ , then  $v' \in D_m$  and we set  $s(v') = m(v')$ . In this case, if  
 302  $p' = p_{v'}$  then  $s(v') = m(v') = 1$  leading to  $M_s(p') = 1$ , if  $p' = \bar{p}_{v'}$  then  
 303  $s(v') = m(v') = 0$  leading to  $M_s(p') = 0$ .

304 For the remaining nodes of  $\mathcal{N}$ , we can always set Boolean values to these  
 305 nodes to preserve that  $s \in \mathcal{S}_{\mathcal{N}}[m]$  by applying the same procedure. We also  
 306 have  $M_s(p_v) = 0$  by the characteristics of the encoding [24] (and Definition  
 307 3.1). Now,  $t$  is enabled at marking  $M_s$ . Its firing leads to a new marking  
 308  $M'_s$  such that  $M'_s(p_v) = 1$  and  $M'_s(\bar{p}_v) = 0$ . Let  $s'$  be the corresponding state  
 309 in  $\mathcal{N}$  of  $M'_s$ . We have  $s'(v) = 1$  because  $M'_s(p_v) = 1$  and  $m(v) = 0$  because  
 310  $p_v \in S$ . This implies that  $s' \notin \mathcal{S}_{\mathcal{N}}[m]$ .

311 For any firing scheme of  $\mathcal{P}$ , the firing of  $t$  always happens. Since a firing  
 312 scheme of  $\mathcal{P}$  is equivalent to an update scheme of  $\mathcal{N}$ ,  $s$  can escape from the  
 313 trap space  $m$  for any update scheme of  $\mathcal{N}$ , which contradicts to the property

314 of a trap space. Hence,  $S$  is a siphon of  $\mathcal{P}$ . By the definition of a mirror,  $S$   
 315 is also a conflict-free one.

316 *Second, we show that if  $S$  is a conflict-free siphon of  $\mathcal{P}$ , then  $m$  is a trap*  
 317 *space of  $\mathcal{N}$  (\*\*).*

318 By the definition of a mirror,  $m$  is a subspace of  $\mathcal{N}$ . Let  $s$  be an arbitrary  
 319 state in  $\mathcal{S}_{\mathcal{N}}[m]$  and  $M_s$  be its corresponding marking in  $\mathcal{P}$ . Assume that  
 320 there is a place  $p \in S$  such that  $M_s(p) = 1$ . Let  $v$  be the corresponding node  
 321 in  $\mathcal{N}$  of  $p$ . Since  $p \in S$ ,  $v \in D_m$  and  $m(v) = s(v)$ . If  $p = p_v$ , then  $M_s(p_v) = 1$   
 322 leading to  $m(v) = s(v) = 1$  by the characteristics of the encoding [24]. By the  
 323 definition of a mirror,  $m(v) = 0$  because  $p_v \in S$ , meaning that  $M_s(p_v) = 0$ ,  
 324 which is a contradiction.

325 It is symmetric for the case that  $p = \bar{p}_v$ . Hence,  $M_s(p) = 0, \forall p \in S$ . In any  
 326 marking  $M'_s$  reachable from  $M_s$  regardless of the firing scheme of  $\mathcal{P}$ , we have  
 327  $M'_s(p) = 0, \forall p \in S$  by the dynamical property on markings of a siphon [35].  
 328 Let  $s'$  be the corresponding state in  $\mathcal{N}$  of  $M'_s$ . For every node  $v \in D_m$ ,  
 329 we have all two cases as follows. Case 1:  $p_v \in S$ , then  $M'_s(p_v) = 0$ , thus  
 330  $s'(v) = 0 = m(v)$ . Case 2:  $\bar{p}_v \in S$ , then  $M'_s(\bar{p}_v) = 0$ , thus  $s'(v) = 1 = m(v)$ .  
 331 Hence,  $s'(v) = m(v)$  for every  $v \in D_m$ . Then,  $s' \in \mathcal{S}_{\mathcal{N}}[m]$ . By the definition  
 332 of a trap space and the arbitrariness of  $s$ ,  $m$  is a trap space of  $\mathcal{N}$ .

333 From (\*) and (\*\*), we can conclude the proof.  $\square$

334 Note that this proof gives us as corollary a well-known result on trap  
 335 spaces.

336 **Corollary 3.1.** *Trap spaces of a Boolean network are independent of the*  
 337 *update scheme.*

338 *Proof.* From the proof of Theorem 3.1, we can see that the theorem holds  
 339 for any update scheme associated to the Boolean network. Since the Petri  
 340 net encoding of a Boolean network is independent of its update scheme and  
 341 siphons are a static property of a Petri net, we get that trap spaces of a  
 342 Boolean network are independent of its update scheme.  $\square$

343 Note that the original proof for this property of trap spaces (see Theorem  
 344 1 of [7]) only considers the two popular update schemes (i.e., synchronous  
 345 and fully asynchronous). Theorem 3.1 exhibits the very first theoretical  
 346 application of the connection between trap spaces of Boolean networks and  
 347 siphons of Petri nets.

348 **Theorem 3.2.** *Let  $\mathcal{N}$  be a Boolean network and  $\mathcal{P}$  be its Petri net encoding.*  
 349 *A subspace  $m$  is a minimal trap space of  $\mathcal{N}$  if and only if its mirror  $S$  is a*  
 350 *maximal conflict-free siphon of  $\mathcal{P}$ .*

351 *Proof.* First, we show that if  $m$  is a minimal trap space of  $\mathcal{N}$ , then  $S$  is  
 352 a maximal conflict-free siphon of  $\mathcal{P}$  (\*). Since  $m$  is a trap space of  $\mathcal{N}$ ,  
 353  $S$  is a conflict-free siphon of  $\mathcal{P}$  by Theorem 3.1. Assume that  $S$  is not  
 354 maximal. Then, there is another conflict-free siphon  $S'$  such that  $S \subset S'$ .  
 355 By Theorem 3.1, there is a trap space  $m'$  corresponding to  $S'$ . Following the  
 356 definition of a mirror,  $D_m \subset D_{m'}$  and  $m(v) = m'(v), \forall v \in D_m$ . It follows  
 357 that  $S_{\mathcal{N}}[m'] \subset S_{\mathcal{N}}[m]$ , thus  $m' < m$ . This contradicts to the minimality of  
 358  $m$ . Hence,  $S$  is a maximal conflict-free siphon of  $\mathcal{P}$ .

359 Second, we show that if  $S$  is a maximal conflict-free siphon of  $\mathcal{P}$ , then  
 360  $m$  is a minimal trap space of  $\mathcal{N}$  (\*\*). Since  $S$  is a conflict-free siphon of  $\mathcal{P}$ ,  
 361  $m$  is a trap space of  $\mathcal{N}$  by Theorem 3.1. Assume that  $m$  is not minimal.  
 362 Then, there is another trap space  $m'$  such that  $m' < m$ . By the definition of  
 363 the partial order  $<$  on subspaces,  $S_{\mathcal{N}}[m'] \subset S_{\mathcal{N}}[m]$ . Let  $S'$  be the mirror of  
 364  $m'$ .  $S'$  is a conflict-free siphon by Theorem 3.1. Following the definition of  
 365 a mirror,  $S \subset S'$ , which contradicts to the maximality of  $S$ . Hence,  $m$  is a  
 366 minimal trap space of  $\mathcal{N}$ .

367 From (\*) and (\*\*), we can conclude the proof.  $\square$

368 We here showcase a theoretical application of the connection between  
 369 trap spaces in Boolean networks and conflict-free siphons in Petri nets. We  
 370 use it to prove a property of minimal trap spaces, which has surprisingly  
 371 not been formally proved. Specifically, all minimal trap spaces of a Boolean  
 372 network are mutually disjoint. **This property is important because it can**  
 373 **benefit attractor identification of Boolean networks.** Specifically, in [36], the  
 374 **authors use random walks inside each minimal trap space to obtain approx-**  
 375 **imations for attractors of a Boolean network under the fully asynchronous**  
 376 **update scheme, then they use CTL model checking to verify the quality of**  
 377 **the approximations.** In [37], the authors use the set of minimal trap spaces  
 378 **as a seed to speedup their previous attractor identification method that re-**  
 379 **lies on feedback vertex sets and reachability analysis.** The soundness of the  
 380 **two above approaches comes from the separation of minimal trap spaces.**  
 381 Note that it would be not difficult to obtain a direct proof on trap spaces  
 382 for this property, which follows the same structure as the proof on siphons.  
 383 However, we emphasize here the potential of using the connection between

384 Boolean networks and Petri nets to explore and prove properties of trap  
385 spaces in Boolean networks.

386 **Theorem 3.3.** *Let  $\mathcal{N} = (V, F)$  be a Boolean network. For any two distinct*  
387 *minimal trap spaces  $m_1$  and  $m_2$  of  $\mathcal{N}$ , we have that  $\mathcal{S}_{\mathcal{N}}[m_1] \cap \mathcal{S}_{\mathcal{N}}[m_2] = \emptyset$ .*

388 *Proof.* Let  $\mathcal{P}$  be the Petri net encoding of  $\mathcal{N}$ . If  $\mathcal{N}$  has only one minimal  
389 trap space, then the theorem trivially holds. Note that by Theorem 3.2,  
390  $\mathcal{N}$  always has at least one minimal trap space because  $\mathcal{P}$  has at least one  
391 maximal conflict-free siphon. Hence, we consider the case that  $\mathcal{N}$  has at least  
392 two minimal trap spaces.

393 Consider two any distinct minimal trap spaces  $m_1$  and  $m_2$ . Assume that  
394  $\mathcal{S}_{\mathcal{N}}[m_1] \cap \mathcal{S}_{\mathcal{N}}[m_2] \neq \emptyset$ . Let  $S_1$  and  $S_2$  be the mirrors of  $m_1$  and  $m_2$ , re-  
395 spectively. By Theorem 3.2,  $S_1$  and  $S_2$  are maximal conflict-free siphons  
396 of  $\mathcal{P}$ . We have that  $S = S_1 \cup S_2$  is also a siphon because of Proposi-  
397 tion 2.1. For every node  $v \in V$ , assume that  $p_v \in S$  and  $\bar{p}_v \in S$  hold.  
398 Since  $S_1$  and  $S_2$  are conflict-free, there are all two cases. Case 1:  $p_v \in S_1$   
399 and  $\bar{p}_v \in S_2$ . Case 2:  $p_v \in S_2$  and  $\bar{p}_v \in S_1$ . These two cases lead to  
400  $m_1(v) \neq m_2(v)$ ,  $m_1(v) \neq \star$ ,  $m_2(v) \neq \star$ , then  $\mathcal{S}_{\mathcal{N}}[m_1] \cap \mathcal{S}_{\mathcal{N}}[m_2] = \emptyset$ . This is a  
401 contradiction. Hence, for every node  $v \in V$ ,  $p_v \in S$  and  $\bar{p}_v \in S$  cannot hold  
402 together. Therefore,  $S$  is conflict-free. Now, we have that  $S$  is a conflict-free  
403 siphon but  $S_1 \subset S$  or  $S_2 \subset S$  holds because  $S_1 \neq S_2$ . This contradicts to the  
404 maximality of  $S_1$  and  $S_2$ . Hence,  $\mathcal{S}_{\mathcal{N}}[m_1] \cap \mathcal{S}_{\mathcal{N}}[m_2] = \emptyset$  holds.

405 □

406 A natural computational application of Theorem 3.1 is that we can effi-  
407 ciently decide whether a subspace  $m$  is a trap space. In PyBoolNet [21], this  
408 is checked by using the percolation on the **prime implicants** of the Boolean  
409 functions. As we have mentioned at the beginning of this article, the compu-  
410 tation of **prime implicants** is a demanding task for complex Boolean networks,  
411 even is sometimes intractable. Hence, the checking method in [21] shows its  
412 limitations. Instead, we can first compute the mirror  $S_m$  of  $m$  in the Petri  
413 net encoding. Then, by Proposition 2.1 and Theorem 3.1, we can check if  
414  $\text{pred}(S_m) \subseteq \text{succ}(S_m)$ . Note that the Petri net construction is less com-  
415 putationally demanding than the prime-implicant computation because it  
416 only requires computing generic (not prime) implicants of the Boolean func-  
417 tions [23]. In addition, the worst case time complexity of the above checking  
418 method is quadratic in the number of transitions of the Petri net.

419 Furthermore, by Theorem 3.2, we can reduce the problem of computing  
 420 all minimal trap spaces of a Boolean network to the problem of computing  
 421 all maximal conflict-free siphons of its Petri net encoding. Note that in  
 422 the case of special types of trap spaces (e.g., fixed points), this can be put  
 423 in regard to special types of siphons in Petri nets. See Subsection 4.5 for  
 424 more discussions about many special types of trap spaces. It might actually  
 425 be possible to generalize our result to any 1-safe place-complementary (i.e.,  
 426 places are defined by pairs such that the markings are complementary) Petri  
 427 net to define a notion of trap spaces that might be useful for the analysis of  
 428 Petri nets, but this is out of the scope of the present article. Note also that  
 429 conversely, investigating static analyses on such 1-safe place-complementary  
 430 nets might allow for a more efficient computation of their siphons and hence  
 431 of trap spaces.

432 Note that there are no existing methods specifically designed for comput-  
 433 ing maximal conflict-free siphons (even maximal generic siphons) of a Petri  
 434 net. The reason might be that researchers mainly focus on minimal generic  
 435 siphons [35] in the field of Petri nets. While adapting those methods to ob-  
 436 tain minimal conflict-free siphons would sometimes be possible, the switch  
 437 from minimality to maximality is quite a leap. Hence, we here propose sev-  
 438 eral methods for computing maximal conflict-free siphons of a Petri net. The  
 439 details of the proposed methods shall be given in the next section.

## 440 4. Computation methods

### 441 4.1. Characterization

First, we show the characterization of all conflict-free siphons of the en-  
 coded Petri net  $\mathcal{P} = (P, T, W)$ . Suppose that  $S$  is a generic siphon of  $\mathcal{P}$ .  
 If a place  $p$  should belong to  $S$ , then by Proposition 2.1 all the transitions  
 in  $\text{pred}(p)$  must belong to  $\text{succ}(S)$ . A transition  $t$  belongs to  $\text{succ}(S)$  if and  
 only if there is at least one place  $p'$  in  $S$  such that  $p' \in \text{pred}(t)$ . Hence, for  
 each transition  $t \in \text{pred}(p)$ , we can state that

$$p \in S \Rightarrow \bigvee_{p' \in \text{pred}(t)} p' \in S. \quad (1)$$

The system of all the rules of the above form with respect to all pairs  $(p, t)$   
 where  $p \in P, t \in T, t \in \text{pred}(p)$  fully characterizes all generic siphons of a

Petri net and has been used with SAT solvers in [16, 17]. To make  $S$  to be a conflict-free siphon, we need to add to the system the rule

$$p_v \in S \Rightarrow \bar{p}_v \notin S \wedge \bar{p}_v \in S \Rightarrow p_v \notin S \quad (2)$$

for each node  $v \in V$ . By definition, the final system fully characterizes all conflict-free siphons of the encoded Petri net.

#### 4.2. Constraint satisfaction problem

A Constraint Satisfaction Problem (CSP) is defined by a triple giving its variables, their domains, and the constraints on those variables. The following Boolean CSP directly derives from the above characterization:

**Definition 4.1.** Given a Petri net  $\mathcal{P} = (P, T, W)$  encoding a Boolean network  $\mathcal{N} = (V, F)$ . The CSP  $\mathcal{C}(\mathcal{P})$  is the triple  $(R, D, C)$  where

- $R = P$ , i.e., a variable is introduced for each place of  $\mathcal{P}$ ,
- $D(p) = \mathbb{B}$  for all  $p \in R$ , i.e., the variables are Boolean,
- $C = \{\neg p_v \vee \neg \bar{p}_v = 1 \mid \forall v \in V\} \wedge \{(p = 1 \rightarrow \bigvee_{p' \in \text{pred}(t)} p' = 1) \mid p \in P, t \in \text{pred}(p)\}$ .

**Proposition 4.1.**  $\mathcal{C}(\mathcal{P})$  is satisfied by a valuation  $r$  if and only if

$$\{p \in P \mid r(p) = 1\}$$

is a conflict-free siphon of  $\mathcal{P}$ .

*Proof.* By the former part  $\neg p_v \vee \neg \bar{p}_v = 1$  of  $C$ , the conflict-freeness is imposed because for any satisfiable valuation  $r$ ,  $r(p_v) = r(\bar{p}_v) = 1$  is impossible for all  $v \in V$ . As shown in [17], the latter part of  $C$  can characterize the set of all generic siphons of  $\mathcal{P}$ . Hence, we can conclude the proof.  $\square$

In [17], the set of all siphons of a given Petri net is characterized by a similar Boolean CSP except the conflict-freeness constraint. From the encoded CSP, the set of all *minimal* siphons of the Petri net can be enumerated in the set inclusion order. For enumerating siphons in the set inclusion order, the method proposed in [17] uses the technique that labels directly the Boolean variables with increasing value selection (i.e., to test first the absence, then



the presence of a place in the candidate solution). The method has two implementations, one uses an iterated SAT procedure and the other uses Constraint Programming (CP) with backtracking.

One natural question is that how to use the CSP-based method for enumerating all the maximal conflict-free siphons of a Petri net encoding a Boolean network? Of course, the set of all conflict-free siphons of the Petri net can easily be characterized by the CSP model presented in [17] along with the additional constraint  $\neg p_v \vee \neg \bar{p}_v = 1$ , for each  $v \in V$ , which represents the conflict-freeness. However, the main concern is to enumerate all the *maximal* ones, which is not trivial to adapt from the CSP-based method. By Proposition 4.1, the set of all maximal conflict-free siphons of  $\mathcal{P}$  can be enumerated in the (maximality) set inclusion order, by restarting the search each time a conflict-free siphon  $S$  is found, with the following additional constraint for disallowing any subset of that conflict-free siphon:  $\bigvee_{p \notin S} p = 1$ . For enumerating conflict-free siphons in the set inclusion order, we can use the same technique as used in [17] but with the opposite setting, i.e., labeling directly the Boolean variables with decreasing value selection. The correctness of this technique comes from the fact that once  $S$  is found, it is the conflict-free siphon of maximum cardinality among all the remaining feasible conflict-free siphons. Similar to [17], the newly CSP-based method can also be implemented with SAT and CP solvers.

This method was implemented using the state-of-the-art CP solver Chuffed<sup>6</sup> [38] via its MiniZinc [39] interface. Because it is a high-level interface, the backtrack-and-replay method of [17] was not used but rather the alternative implementation with two global constraints for lexicographic ordering (ensuring enumeration of solutions) and iterated non-subset of each already found solution (for maximality).

For the SAT-based method, however a more direct method is to use a MaxSAT solver. We construct a MaxSAT problem with the following hard clauses:

$$(\neg p_v \vee \neg \bar{p}_v), \forall v \in V$$

and

$$(\neg p \vee \bigvee_{p' \in \text{pred}(t)} p'), \forall p \in P, \forall t \in \text{pred}(p).$$

We set a soft clause for each variable of the CSP and then use a “minimal cor-

---

<sup>6</sup><https://github.com/chuffed/chuffed>

rejection subset” blocking strategy, which will ensure set-inclusion maximality of the solutions. We implement this approach by using the RC2 MaxSAT solver [40] available through the `python-sat` package<sup>7</sup>.

#### 4.3. Answer set programming-based method

Another possible method is to translate the characterization shown in Subsection 4.1 into the ASP  $\mathcal{L}$  as follows. We introduce atom `p-v` (resp. `n-v`) to denote place  $p_v$  (resp.  $\bar{p}_v$ ),  $\forall v \in V$ . The set of all atoms in  $\mathcal{L}$  is given as  $\mathcal{A} = \bigcup_{v \in V} \{\text{p-v}, \text{n-v}\}$ . For each pair  $(p, t)$  where  $p \in P, t \in T, t \in \text{pred}(p)$ , we translate the rule (1) into the ASP rule

$$\text{a\_1}; \dots ; \text{a\_k} :- \text{a}.$$

where  $\text{a} \in \mathcal{A}$  is the atom representing place  $p$  and  $\{\text{a\_1}, \dots, \text{a\_k}\} \subseteq \mathcal{A}$  is the set of atoms representing places in  $\text{pred}(t)$ . The rule (2) is translated into the ASP rule

$$:- \text{p-v}, \text{n-v}.$$

for each  $v \in V$ . This ASP rule guarantees that two places representing the same node in  $\mathcal{N}$  never belong to the same siphon of  $\mathcal{P}$ , representing the conflict-freeness. Naturally, a Herbrand model (see, e.g., [41]) of  $\mathcal{L}$  is equivalent to a conflict-free siphon of  $\mathcal{P}$ . To guarantee that a Herbrand model is also a stable model (an answer set), we need to add to  $\mathcal{L}$  the two choice rules

$$\{\text{p-v}\}. \{\text{n-v}\}.$$

for each  $v \in V$ . Note that the number of atoms of  $\mathcal{L}$  is only  $2n$ , whereas the ASP encoding shown in [7] has as many atoms as the number of **prime implicants** of the Boolean network and that number might be exponential in  $n$ . In [8], there is an ASP characterization of trap spaces that does not rely on minimal DNFs either and thus seems very similar to our ASP encoding. Remarkably it only requires the DNF for the *activation* part, using the information that it will only be used for locally-monotonic Boolean networks. We would therefore expect that, when available, it will have comparable performance on the ASP part (the ASP program would be approximately twice smaller, though redundancy is not always bad in that field), but can also avoid combinatorial explosion of the Petri net encoding for some formula

---

<sup>7</sup><https://pysathq.github.io/docs/html/api/examples/rc2.html>

509 where the activation DNF is simple but the inhibition is not. Since `mpbn` is  
 510 included in our benchmark this will be evaluated in our experiments.

511 Now, a solution (simply an answer set)  $A \subseteq \mathcal{A}$  of  $\mathcal{L}$  is equivalent to a  
 512 conflict-free siphon  $S$  of  $\mathcal{P}$ , thus a trap space  $m$  of  $\mathcal{N}$ . The conversion from  $A$   
 513 to  $m$  is straightforward. If  $\mathbf{p-v} \in A$  then  $v \in D_m$  and  $m(v) = 0$ . Conversely,  
 514 if  $\mathbf{n-v} \in A$  then  $v \in D_m$  and  $m(v) = 1$ . Otherwise,  $v \notin D_m$ . Comput-  
 515 ing multiple answer sets is built into ASP solvers and the solving collection  
 516 `POTASSCO` [41] also features the option to find set-inclusion maximal answer  
 517 sets with respect to the set of atoms. Naturally, a set-inclusion maximal  
 518 answer set of  $\mathcal{L}$  is equivalent to a maximal conflict-free siphon of  $\mathcal{P}$ , thus a  
 519 minimal trap space of  $\mathcal{N}$ . By using this built-in option, we can compute all  
 520 the set-inclusion maximal answer sets of  $\mathcal{L}$  (resp. all the minimal trap spaces  
 521 of  $\mathcal{N}$ ) in one execution.

#### 522 4.4. Integer linear programming-based method

We first show how an Integer Linear Programming (ILP)  $\mathcal{I}$  can define  
 a set of all conflict-free siphons of the encoded Petri net  $\mathcal{P}$ . We introduce  
*binary* variable  $\mathbf{p-v}$  (resp.  $\mathbf{n-v}$ ) to denote place  $p_v$  (resp.  $\bar{p}_v$ ),  $\forall v \in V$ . The  
 set of all binary variables in  $\mathcal{I}$  is  $\bigcup_{v \in V} \{\mathbf{p-v}, \mathbf{n-v}\}$ . For each pair  $(p, t)$  where  
 $p \in P, t \in T, t \in \text{pred}(p)$ , we translate the rule (1) into the ILP inequality

$$\mathbf{a} \leq \mathbf{a\_1} + \dots + \mathbf{a\_k}$$

where  $\mathbf{a}$  is the binary variable representing place  $p$  and  $\{\mathbf{a\_1}, \dots, \mathbf{a\_k}\}$  is  
 the set of binary variables representing places in  $\text{pred}(t)$ . The rule (2) is  
 translated into the ILP inequality

$$\mathbf{p-v} + \mathbf{n-v} \leq 1$$

for each  $v \in V$ . This inequality forbids both  $\mathbf{p-v}$  and  $\mathbf{n-p}$  receive the value  
 1, thus representing the conflict-freeness. Since we only consider feasible  
 solutions, the objective function is set to  $\max \mathbf{p-v}$  for some  $v \in V$ . Naturally,  
 a solution  $I$  of  $\mathcal{I}$  is equivalent to a conflict-free siphon  $S$  of  $\mathcal{P}$ . The conversion  
 is that

$$S = \{p \in P \mid I(\mathbf{a-p}) = 1\}$$

523 where  $\mathbf{a-p}$  is the binary variable presenting place  $p$ .

524 We can see the similarity between  $\mathcal{I}$  and the encoded ASP shown in the  
 525 previous subsection. However, due to the nature of solutions of an ILP, it is

526 hard to compute all the set-inclusion maximal solutions of  $\mathcal{I}$  in one execution  
 527 of an ILP solver. Hence, we propose an iterative approach as follows.

The conflict-free siphon of maximum cardinality is of course maximal. Therefore, we impose the following objective function:

$$\max \sum_{v \in V} (p-v + n-v).$$

Now,  $\mathcal{I}$  can be solved using a general purpose ILP solver. If it admits any solution  $I^*$ , the corresponding conflict-free siphon (say  $S^*$ ) is maximal. Hence, it makes sense that it does not need to find any other conflict-free siphon of the net that is strictly contained in  $S^*$ . To do this, we add to  $\mathcal{I}$  a new inequality

$$1 \leq \sum_{p \in P \setminus S^*} a-p$$

528 where  $a-p$  is the binary variable presenting place  $p$ . Now, we solve  $\mathcal{I}$  again to  
 529 find a new solution. If a new solution  $I'$  exists, then let  $S'$  be its corresponding  
 530 conflict-free siphon. Indeed, abide by the newly added inequality, we have  
 531  $S' \cap (P \setminus S^*) \neq \emptyset$  because there is some  $a-p$  with  $p \in P \setminus S^*$  such that  
 532  $I'(a-p) = 1$ . This implies that it is impossible that  $S' = S^*$  or  $S' \subset S^*$ .  
 533 By the objective function, it means that  $S'$  is the conflict-free siphon of  
 534 maximum cardinality among the conflict-free siphons that are not contained  
 535 in  $S^*$ . Hence,  $S'$  is also a maximal conflict-free siphon. Again, we add to  $\mathcal{I}$   
 536 a new inequality with respect to the newly found siphon. The above process  
 537 is iterated until  $\mathcal{I}$  becomes unfeasible, this means that there is no further  
 538 maximal conflict-free siphon. Thus, all the maximal conflict-free siphons of  
 539 the Petri net have been found.

540 Since we used the MiniZinc framework to interface with the CP solver, it  
 541 was simple to make the slight modifications described above and to use that  
 542 same interface to call the Coin-OR CBC solver<sup>8</sup> [42].

#### 543 4.5. Computation of special types of trap spaces

544 In the field of systems biology, biologists may want to compute more  
 545 special types of trap spaces beyond minimal trap spaces [21], which also play  
 546 crucial roles in analysis and control of Boolean networks [22, 19]. We shall  
 547 show that our proposed methods can be easily adjusted to compute such

---

<sup>8</sup><https://github.com/coin-or/Cbc>

548 popular types of trap spaces. We illustrate the adjustments via the ASP-  
 549 based method (see Subsection 4.3) because ASP is declarative by nature,  
 550 but these adjustments are completely applicable for other approaches such  
 551 as MaxSAT, CP, and ILP.

552 First, the work presented in [19] uses the concept of *stable motifs* to build  
 553 the succession diagram of a Boolean network, a summary of the decisions in  
 554 the network dynamics that lead to successively more restrictive nested stable  
 555 motifs. The succession diagram is useful for control and decision making  
 556 on this Boolean network. In particular, the proposed control methods are  
 557 independent to the update scheme. Note that, in [19], the succession dia-  
 558 gram is also used to identify all attractors of a Boolean network under the  
 559 fully asynchronous update scheme. It has been shown that a stable motif  
 560 of a Boolean network is equivalent to a *maximal trap space* of this Boolean  
 561 network [19]. Indeed, the computation of stable motifs is a bottleneck of  
 562 the methods proposed in [19]. Hence, it is necessary to develop an efficient  
 563 method for computing maximal trap spaces of a Boolean network. We shall  
 564 show how to adjust the ASP-method presented in Subsection 4.3 to compute  
 565 maximal trap spaces.

We first provide the definition of maximal trap spaces. Let  $\varepsilon$  be the special  
 trap space of  $\mathcal{N}$  where all the nodes are free. Of course,  $\varepsilon$  corresponds to the  
 special conflict-free siphon  $\emptyset$ . A trap space  $m$  is called maximal if  $m \neq \varepsilon$  and  
 there is no other trap space  $m'$  such that  $m' \neq \varepsilon$  and  $m < m'$ . Analogously,  
 a conflict-free siphon  $S$  is called minimal if  $S \neq \emptyset$  and there is no other  
 trap space  $S'$  such that  $S' \neq \emptyset$  and  $S' \subset S$ . By using the reasoning similar  
 to the proof of Theorem 3.2, we can easily conclude that a maximal trap  
 space of  $\mathcal{N}$  is equivalent to a minimal conflict-free siphon of its encoded  
 Petri net  $\mathcal{P}$ . Let  $\mathcal{L}$  be the ASP characterizing all conflict-free siphons of  $\mathcal{P}$   
 (see Subsection 4.3). Naturally, we need to exclude  $\emptyset$  from the solution space  
 of  $\mathcal{L}$  (equivalently exclude  $\varepsilon$  from the set of trap spaces). To do this, we add  
 to  $\mathcal{L}$  the ASP rule

$$\text{p-v}_1; \text{n-v}_1; \dots; \text{p-v}_n; \text{n-v}_n.$$

566 that ensures that every answer set of  $\mathcal{L}$  cannot be empty. Then a set-inclusion  
 567 minimal answer set of  $\mathcal{L}$  is equivalent to a minimal conflict-free siphon of  $\mathcal{P}$ ,  
 568 thus a maximal trap space of  $\mathcal{N}$ .

Second, we consider *fixed points* in Boolean networks. To date, the anal-  
 ysis of the fixed points of a Boolean network remains a very useful tool in

understanding the behavior of complex biological models 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 [43]. Furthermore, the fixed point computation is also the crucial starting point for several state-of-the-art methods for computing complex attractors of Boolean networks [37]. Let  $s$  be a fixed point of a Boolean network  $\mathcal{N}$ . We have a subspace  $m$  corresponding to  $s$  as follows:  $\forall v \in V, m(v) = s(v)$ , i.e., all nodes are fixed in  $m$ . Clearly,  $s$  is a trap set of  $\mathcal{N}$  regardless of the update scheme. Hence,  $m$  is a trap space of  $\mathcal{N}$ . In addition, since  $|S_{\mathcal{N}}[m]| = 1$ ,  $m$  is also a minimal trap space. To compute all fixed points of  $\mathcal{N}$ , we can add more constraints to the encoded ASP characterizing all conflict-free siphons (equivalently trap spaces). For every  $v \in V$ , we add to the encoded ASP the rule

$$\text{p-v; n-v.}$$

569 that ensures that for every conflict-free siphon  $S$ , it contains either  $\text{p-v}$  or  $\text{n-v}$   
570 for every  $v \in V$ . Equivalently, the trap space corresponding to  $S$  is always  
571 a fixed point. Now, the set of answer sets of the encoded ASP is equivalent  
572 to the set of fixed points of  $\mathcal{N}$ . In particular, when solving the encoded ASP  
573 using an ASP solver, we do not need to use the built-in option for computing  
574 set-inclusion maximal answer sets. Note that we can also build another ASP  
575 characterizing all fixed points of  $\mathcal{N}$  based on the equivalence between a fixed  
576 point of  $\mathcal{N}$  and a deadlock of its Petri net encoding [23]. This approach may  
577 give a more compact ASP.

Third, we consider the trap spaces *intersecting* a given subspace  $m^*$  of a Boolean network. **Such trap spaces (along with minimal trap spaces) are used in the phenotype control method [22]. This method uses the prime implicant-based method [7, 21] to compute trap spaces, which has been shown inefficient. Hence, having a more efficient method for computing such trap spaces can push the barrier previously existing in this control method.** A trap space  $m$  intersects  $m^*$  if and only if  $S_{\mathcal{N}}[m] \cap S_{\mathcal{N}}[m^*] \neq \emptyset$ . It follows that for every  $v$ , if  $m^*(v) = 0$  then  $m(v) = 0$  or  $m(v) = \star$ , if  $m^*(v) = 1$  then  $m(v) = 1$  or  $m(v) = \star$ . For the former case, we add to  $\mathcal{L}$  the ASP rule

$$\text{: - n-v.}$$

that ensures that  $m(v)$  cannot be 1. For the latter case, we add to  $\mathcal{L}$  the ASP rule

$$\text{: - p-v.}$$

578 that ensures that  $m(v)$  cannot be 0. Now  $\mathcal{L}$  characterizes all trap spaces that  
 579 intersect  $m^*$ .

Finally, we consider the trap spaces that are *inside* a given subspace  $m^*$  of a Boolean network. Such trap spaces are used in the iterative procedure of building the succession diagram of a Boolean network [19], which is hierarchical. We first adjust  $\mathcal{L}$  to characterize all such trap spaces. A trap space  $m$  is inside  $m^*$  if and only if  $m(v) = m^*(v)$  for every  $v \in D_{m^*}$ . If  $m^*(v) = 0$ , we add to  $\mathcal{L}$  the ASP rule

$$\text{p-v.}$$

that ensures that  $m(v) = 0$ . If  $m^*(v) = 1$ , we add to  $\mathcal{L}$  the ASP rule

$$\text{n-v.}$$

that ensures that  $m(v) = 1$ . It is noted that if we want to compute maximal trap spaces inside  $m^*$ , we need to exclude the conflict-free siphon corresponding  $m^*$  from the solution space. Specifically, we need to add to  $\mathcal{L}$  the ASP rule

$$\text{p-v}_{i1}; \text{n-v}_{i1}; \dots; \text{p-v}_{ik}; \text{n-v}_{ik}.$$

580 where  $\{v_{i1}, \dots, v_{ik}\}$  is the set of free nodes of  $m^*$ . This rule ensures that  
 581  $m \neq m^*$ . In the case that  $m^* = \varepsilon$ , we have all maximal trap spaces of the  
 582 original Boolean network.

## 583 5. Motivating example

584 For a few years now we have been collaborating with biologists who build  
 585 very large detailed and annotated maps and now wish to analyze the dy-  
 586 namics of the corresponding models. One of the main maps studied this way  
 587 represents knowledge about the Rheumatoid Arthritis [44], and was the main  
 588 motivation for the development of a tool to automatically transform it into  
 589 an executable Boolean network [6]. In the supplementary material of the pa-  
 590 per, an excerpt of the map, focused around the apoptosis (cell death) module  
 591 is transformed into a model of *reasonable* size, namely 180 Boolean variables  
 592 (model `F5_RA_apoptosis_executable_module.sbml` of supplementary ma-  
 593 terial S3, and model “RA\_apoptosis” of Subsection 6.3). The study of such  
 594 model, though, is a big hurdle. Indeed, as stated in the article about another  
 595 model of the same size: “*The size of the CaSQ-inferred MAPK model (181*  
 596 *nodes) made the calculation of stable states a non-realistic endeavour.*”

597 In practice, even if there is a huge number of attractors in such a model,  
598 obtaining a sample of those can reveal very useful to invalidate the model and  
599 lead to further refinement. In particular, it provides a feature-rich alternative  
600 to random simulations for this type of very non-deterministic model. Being  
601 able to detect that there are inconsistencies with published experimental data  
602 in some of the first 1000 attractors, for instance, can lead to a much quicker  
603 Systems Biology loop: model, invalidate, refine.

604 However, using a state-of-the-art tool like `PyBoolNet` [7] on that model  
605 **unfortunately** fails at the phase of prime-implicant generation. `mpbn` [9] can  
606 return the first 1000 solutions within 1.43s, but indeed, it limits the model-  
607 ing range of the modelers as it does not permit using non-locally-monotonic  
608 Boolean functions. This is also true for the Alzheimer model also mentioned  
609 in that same article and originally from [45] (F4 file in the original supple-  
610 mentary material, and “Alzheimer” in Table 2), where `PyBoolNet` also fails  
611 at the prime-implicant computation and `mpbn` does not give any answer be-  
612 cause this model is actually non-locally-monotonic. The current practice  
613 usually revolves then around fixing some source nodes to plausible values  
614 and reducing the model accordingly. While this approach makes sense, it  
615 relies on potentially arbitrary decisions, and *hides away* critical modelling  
616 choices that were **clearly** not part of the original Boolean network or even of  
617 the starting map.

618 For the “RA\_apoptosis” model, using the ASP-based method presented  
619 in Subsection 4.3, it is **now** possible to obtain the first 1000 minimal trap  
620 spaces (including ones that contain more than one state) within 0.19s, which  
621 is much quicker than `mpbn`. The needed time for the “Alzheimer” model is  
622 0.79s.

## 623 6. Evaluation

624 To evaluate the performance of the newly proposed methods (imple-  
625 mented as a Python package named `Trappist` and available on the Python  
626 package index<sup>9</sup>) and the state-of-the-art methods (`bioLQM`<sup>10</sup>, `PyBoolNet` [7,  
627 21], and `mpbn` [9]), we compared them on both `PyBoolNet`’s own model repos-  
628 itory and many real-world models from various sources in the literature. To  
629 our knowledge, these models are a highly representative sample of Boolean

---

<sup>9</sup><https://pypi.org/project/trappist/>

<sup>10</sup><http://colomoto.org/biolqm/doc/tools-trap-space.html>



models currently available. It is worth noting that `mpbn` [9] only handles locally-monotonic models, whereas the other methods can handle general models. To obtain a more comprehensive comparison, we also used random models generated by a third-party software `BoolNet` R package [31]. As explained in Section 5, in our benchmarks, we only searched for the first 1000 minimal trap spaces for each model. It is worth noting that unlike existing analysis shown in the literature, we did not fix specific values for source nodes in all the considered models.

To solve the ASP problems, we used the same ASP solver `Clingo` [41] and the same configuration as that used in `PyBoolNet` [7, 21] and `mpbn` [9]. Specifically, 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  $\times$  16, 16 GB DDR4 RAM, Ubuntu 20.04.5 LTS. Finally, we set a time limit of three minutes for each model.

All the models and some Jupyter notebooks realizing the benchmarks (and named `TCS-Benchmark-<...>.ipynb`) can be found at <https://github.com/soli/trap-spaces-as-siphons/>. These can be run on a Docker image in the cloud by clicking the “Binder” button.

### 6.1. *PyBoolNet* repository

Table 1 shows the experimental results on the models from the official `PyBoolNet` repository<sup>11</sup>. Column  $n$  denotes the number of nodes of each model. Column  $|M|$  denotes the number of minimal trap spaces and for each method is given the computation time in seconds, asking only for the first 1000 minimal trap spaces. “NF” means that the method did not finish the computation within the time limit of three minutes. In the case of `bioLQM`, “N/A” means that the number of all minimal trap spaces of the model is larger than 1000 and we did not record the running time of `bioLQM` because it always requires to compute all minimal trap spaces. A number in bold indicates a ratio greater than three compared to the best result. “NM” indicates a non-locally-monotonic model. There are four variants of `Trappist: SAT` (i.e., `Trappist-MaxSAT`, the MaxSAT-based method shown in Subsection 4.2), `CP` (i.e., `Trappist-CP`, the CP-based method shown in

<sup>11</sup><https://github.com/hklarner/pyboolnet/tree/master/pyboolnet/repository>

Table 1: Timing comparisons (in seconds) between bioLQM (LQM), PyBoolNet (PBN), mpbn and the four variants of Trappist on the PyBoolNet repository.

model	$n$	$ M $	LQM	PBN	mpbn	Trappist			
						SAT	CP	ILP	ASP
1 arellano_rootstem	9	4	<b>0.13</b>	0.01	0.00	0.00	<b>0.97</b>	<b>0.96</b>	0.01
2 calzone_cellfate	28	27	<b>0.12</b>	0.02	0.01	0.01	<b>5.59</b>	<b>6.03</b>	0.01
3 dahlhaus_neuroplastoma	23	32	<b>0.11</b>	0.03	0.01	0.01	<b>6.56</b>	<b>6.99</b>	0.01
4 davidich_yeast	10	12	<b>0.11</b>	0.02	0.01	0.01	<b>2.56</b>	<b>2.21</b>	0.01
5 dinwoodie_life	15	7	<b>0.11</b>	0.01	0.00	0.01	<b>1.68</b>	<b>1.39</b>	0.01
6 dinwoodie_stomatal	13	1	<b>0.10</b>	0.01	0.00	0.00	<b>0.39</b>	<b>0.29</b>	0.01
7 faure_cellcycle	10	2	<b>0.11</b>	0.02	0.01	0.01	<b>0.58</b>	<b>0.46</b>	0.01
8 grieco_mapk	53	18	<b>0.19</b>	0.03	0.02	0.03	<b>3.93</b>	<b>10.46</b>	0.02
9 irons_yeast	18	1	<b>0.12</b>	0.03	0.01	0.01	<b>0.37</b>	<b>0.39</b>	0.02
10 jaoude_thdiff	103	1000 <sup>+</sup>	N/A	<b>0.85</b>	<b>0.45</b>	<b>0.56</b>	<b>NF</b>	<b>NF</b>	0.09
11 klamt_tcr	40	8	<b>0.11</b>	0.01	0.01	0.01	<b>1.98</b>	<b>1.22</b>	0.02
12 krumsiek_myeloid	11	6	<b>0.10</b>	0.01	0.00	0.00	<b>1.48</b>	<b>1.26</b>	0.01
13 multivalued	13	4	<b>0.10</b>	0.01	0.00	0.00	<b>0.93</b>	<b>0.86</b>	0.01
14 n12c5	11	5	<b>0.11</b>	<b>17.83</b>	0.01	0.01	<b>1.21</b>	<b>1.10</b>	0.01
15 n3s1c1a	2	2	<b>0.10</b>	0.01	0.00	0.00	<b>0.63</b>	<b>0.49</b>	0.01
16 n3s1c1b	2	2	<b>0.09</b>	0.02	0.00	0.00	<b>0.56</b>	<b>0.49</b>	0.01
17 n5s3	4	3	<b>0.10</b>	0.02	NM	0.00	<b>0.74</b>	<b>0.69</b>	0.01
18 n6s1c2	5	3	<b>0.10</b>	0.02	0.00	0.00	<b>0.91</b>	<b>0.59</b>	0.01
19 n7s3	6	3	<b>0.11</b>	0.02	0.00	0.00	<b>0.79</b>	<b>0.68</b>	0.01
20 raf	3	2	<b>0.10</b>	0.01	0.00	0.00	<b>0.55</b>	<b>0.39</b>	0.01
21 randomnet_n15k3	15	3	<b>0.10</b>	0.02	NM	0.01	<b>0.77</b>	<b>0.67</b>	0.01
22 randomnet_n7k3	7	10	<b>0.10</b>	0.01	NM	0.00	<b>2.07</b>	<b>1.46</b>	0.01
23 remy_tumorigenesis	34	25	<b>0.15</b>	<b>0.94</b>	0.02	0.02	<b>5.98</b>	<b>7.98</b>	0.02
24 saadatpour_guardcell	13	1	<b>0.10</b>	0.06	0.00	0.00	<b>0.53</b>	<b>0.45</b>	0.02
25 selvaggio_emt	56	1000 <sup>+</sup>	N/A	<b>0.48</b>	<b>0.28</b>	<b>0.28</b>	<b>NF</b>	<b>NF</b>	0.09
26 tournier_apoptosis	12	3	<b>0.10</b>	0.01	0.00	0.00	<b>0.74</b>	<b>0.75</b>	0.01
27 xiao_wnt5a	7	4	<b>0.10</b>	0.01	0.00	0.00	<b>1.00</b>	<b>0.89</b>	0.01
28 zhang_tlgl	60	156	<b>0.60</b>	0.09	0.09	0.07	<b>37.26</b>	<b>NF</b>	0.04
29 zhang_tlgl_v2	60	258	<b>0.64</b>	0.04	0.08	0.11	<b>69.95</b>	<b>NF</b>	0.04

Subsection 4.2), ILP (i.e., Trappist-ILP, the ILP-based method shown in Subsection 4.4), and ASP (i.e., Trappist-ASP, the ASP-based method shown in Subsection 4.3).

We first analyze the results of the four variants of Trappist. We can see that Trappist-MaxSAT and Trappist-ASP are comparable in most models, but Trappist-ASP is much faster for the jaoude\_thdiff and selvaggio\_emt

models where the number of minimal trap spaces is greater than 1000. The latter can be explained by the fact that **Trappist-MaxSAT** follows an iterative approach, i.e., it restarts the search with a new constraint each time a solution is found (see Subsection 4.2). This iterative approach may be less efficient than the way ASP solvers use to enumerate multiple solutions (answer sets), which is an advantage of ASP solvers [41]. Hence, when the number of solutions increases, the inferiority of **Trappist-MaxSAT** compared to **Trappist-ASP** will be exhibited more clearly. The two remaining variants, **Trappist-CP** and **Trappist-ILP**, are much less efficient than **Trappist-MaxSAT** and **Trappist-ASP** in every model, even are more than three orders of magnitude slower in some models. The first reason for their bad performance is that they are also iterative methods like **Trappist-MaxSAT**, thus they are not efficient for "enumeration" problems. Upon closer inspection, for the Boolean CSP characterizing conflict-free siphons, CP seems to be something that is a "less-efficient-SAT", handling mostly Boolean constraints and making little use of the global constraints only added for the iterative part. For ILP, it may be even worse, since the problem is purely Boolean (no real or integer numbers whatsoever). This is confirmed by the observation that for some quite large models (e.g., the grieco\_mapk, zhang\_tlg1, and zhang\_tlg1.v2 models), **Trappist-ILP** is much slower than **Trappist-CP**. Note that the inferiority of ILP compared to ASP with respect to the trap space enumeration has been reported in [7]. Hereafter, we shall compare the best variant of **Trappist** (i.e., **Trappist-ASP**) with other methods.

As shown in Table 1, for most of the models of the **PyBoolNet** repository, the results are comparable with all minimal trap spaces found very fast. However upon closer inspection, we can see some notable differences. First, **Trappist-ASP** is far more efficient than **bioLQM** in every model with speedups between  $5\times$  and  $16\times$ . Second, for small models, **PyBoolNet** and **mpbn** are comparable to **Trappist-ASP**. However, on every model that was a bit challenging for **PyBoolNet** or **mpbn**, **Trappist-ASP** is far more efficient with speedups between  $3\times$  and  $5\times$  for the case of **mpbn**, and between  $5\times$  and  $1783\times$  for the case of **PyBoolNet**. In particular, the second best variant of **Trappist** (i.e., **Trappist-MaxSAT**) is even far more efficient than **bioLQM** and **PyBoolNet**, and is comparable to **mpbn** on every model. It is worth noting that for 3 of the 29 models, **mpbn** did not give any answer because these models are **non**-locally-monotonic but all the other methods did, which confirms the limit of **mpbn** on the applicable class of models.

## 6.2. *BBM repository*

The research group behind the **BBM** repository has recently undertaken considerable effort for building a collection of real-world Boolean models from various sources used in systems biology. It aims to be a comprehensive collection suitable for benchmarking and testing new tools and methods. **BBM** consists of 211 models (24 out of them are non-locally-monotonic), peaking at 321 nodes, 1100 regulations among the nodes, and 133 source nodes, respectively. It is released and maintained at <https://github.com/sybila/biodivine-boolean-models>. We here tested all the compared methods on this model repository.

Figure 2 (upper panel) shows cumulative numbers of the **BBM** models that have less than 1000 minimal trap spaces solved by the compared methods with respect to enumerating the first 1000 minimal trap spaces. The number of such models is 134 (per all 211 models), and 15 of them are non-locally-monotonic. This model set allows us to fairly consider **bioLQM** for comparison, since **bioLQM** always requires to compute all minimal trap spaces. We can first see that **Trappist-ASP** and **Trappist-MaxSAT** are still the two best methods as they can handle every model within 1s and always can handle more models than all the remaining methods on every time limit. Second, **Trappist-CP** is better than **Trappist-ILP**, which is consistent with their comparison shown in the previous subsection. Third, one notable remark is that for the time limit of 100s or 180s, **Trappist-CP** can handle more models than all **bioLQM**, **PyBoolNet**, and **mpbn**. This remark shows that even without focusing on the optimization of our implementation, our alternative approach is still better than the state-of-the-art methods on a certain set of real-world models. This is supported by the fact that our alternative approach avoids the need for computing prime implicants (as opposed to **PyBoolNet**) and can handle non-locally-monotonic Boolean networks (as opposed to **mpbn**).

Figure 2 (lower panel) shows cumulative numbers of the **BBM** models solved by the compared methods (except **bioLQM**, **Trappist-CP**, and **Trappist-ILP**) with respect to enumerating the first 1000 minimal trap spaces. We omit the results of **Trappist-CP** and **Trappist-ILP** because they can handle no model with more than 1000 minimal trap spaces. Again, we can see that **Trappist-ASP** and **Trappist-MaxSAT** are the two best methods as they can handle every but one model within 5s. They also always handle many more models than both **PyBoolNet** and **mpbn** on every time limit. Note that with the time limit of 0.5s, **Trappist-ASP** can handle 14 more models than **Trappist-MaxSAT**, which is opposed to the case of models with less than

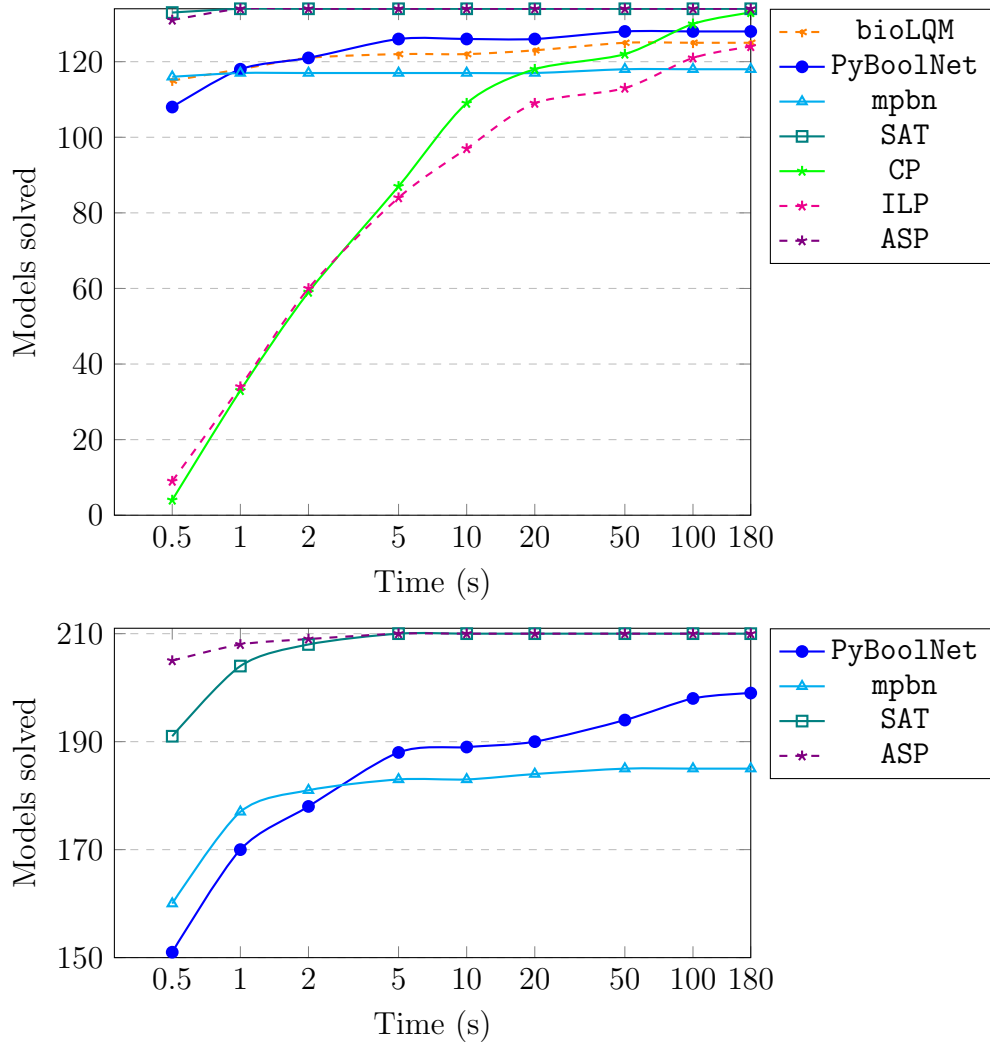


Figure 2: Cumulative numbers of the BBM models that have less than 1000 minimal trap spaces (upper panel) and BBM models solved by the compared methods with respect to enumerating the first 1000 minimal trap spaces (lower panel).

1000 minimal trap spaces (see Figure 2 (upper panel)). This observation confirms the disadvantage of Trappist-MaxSAT compared to Trappist-ASP for the case of many minimal trap spaces.

### 6.3. Selected models

We used a set of real-world Boolean networks lying in various scales collected from numerous bibliographic sources in the literature. Most of these models are quite big (in size), complex (i.e., having high average in-degree, which is related to the number of **prime implicants**), and have never been fully analyzed. Note that these models are not included in the **PyBoolNet** and **BBM** repositories. We then applied **bioLQM**, **PyBoolNet**, **mpbn**, and the four variants of **Trappist** to computing minimal trap spaces of these real-world models. Table 2 shows the obtained experimental results. A number in bold indicates a ratio greater than or equal to 10 compared to the best result. The remaining notations are similar to those in Table 1. Hereafter, we analyze in detail the results with respect to minimal trap space computation.

First, we obtained some observations on the four variants of **Trappist** consistent with the observations obtained in the previous subsections. More specifically, **Trappist-ASP** is still the best variant with a running time below one second for every model, and followed by **Trappist-MaxSAT**. In particular, the difference in running time between **Trappist-ASP** and **Trappist-MaxSAT** is bigger for larger models or models with more than 1000 minimal trap spaces. **Trappist-CP** and **Trappist-ILP** still have a much worse performance, with **Trappist-CP** better than **Trappist-ILP**. They still can handle no model with more than 1000 minimal trap spaces. However, **Trappist-CP** or **Trappist-ILP** can handle the **FT-GRN** and **Pluripotency** models, whereas all **bioLQM**, **PyBoolNet**, and **mpbn** cannot.

Second, **Trappist-ASP** (even **Trappist-MaxSAT**) is far more efficient than both **bioLQM** and **PyBoolNet** on every model where the comparison is possible. For most models, the speedups of **Trappist-ASP** compared to **bioLQM** and **PyBoolNet** are between one and three orders of magnitude. This again confirms the superiority of **Trappist-ASP** compared to the other methods that can handle general Boolean networks.

Third, for 11 of the 32 models (more than 34%), **mpbn** did not give any answer because these models are non-locally-monotonic. For 21 of the 32 models where **mpbn** returned the answers, **mpbn** and **Trappist-ASP** are roughly comparable in computation time, but **mpbn** appears quite slower on average. In particular, for the **Rho-GTPases** model, **mpbn** is  $577\times$  slower than **Trappist-ASP**. This observation along with the comparisons between **mpbn** and **Trappist-ASP** in the previous subsections are quite surprising because the **ASP** encoding of **mpbn** only requires the **DNF** for the activation part of a Boolean function, whereas that of **Trappist-ASP** requires both the activation

Table 2: Timing comparisons (in seconds) between bioLQM (LQM), PyBoolNet (PBN), mpbn and the four variants of Trappist on selected models from the literature. The models are sorted by size with a horizontal rule inserted to split at 100 and 200 nodes, as in [18]

model	$n$	$ M $	LQM	PBN	mpbn	Trappist			
						SAT	CP	ILP	ASP
1 metastatic [46]	10	4	<b>0.10</b>	0.04	NM	0.01	<b>1.15</b>	<b>0.89</b>	0.02
2 Arabidopsis_thaliana [46]	15	8	<b>0.10</b>	0.06	NM	0.01	<b>2.06</b>	<b>1.83</b>	0.02
3 p53_high_dna [46]	16	1	0.38	<b>1.76</b>	NM	0.08	0.53	0.43	0.14
4 p53_low_dna [46]	16	1	0.41	<b>1.76</b>	NM	0.07	0.58	0.48	0.14
5 FT-GRN [47]	23	32	<b>NF</b>	<b>NF</b>	NM	0.03	<b>8.41</b>	<b>12.38</b>	0.19
6 DNA_damage [46]	26	16	<b>0.24</b>	<b>0.33</b>	NM	0.02	<b>3.91</b>	<b>5.33</b>	0.05
7 Rho-GTPases [46]	33	2	0.17	0.57	<b>40.39</b>	0.07	<b>0.74</b>	0.56	0.11
8 Pluripotency [48]	36	440	<b>NF</b>	<b>NF</b>	NM	0.16	<b>138.92</b>	<b>NF</b>	0.28
9 Pluripotent [46]	36	276	0.37	0.43	NM	0.07	<b>72.40</b>	<b>NF</b>	0.06
10 Pancreatic.Cancer [46]	43	1000+	N/A	0.11	0.36	0.17	<b>NF</b>	<b>NF</b>	0.06
11 Drosophila [49]	52	128	0.33	0.05	0.07	0.06	<b>32.66</b>	<b>126.22</b>	0.05
12 Cacace_TdevModel [50]	61	28	<b>1.29</b>	<b>5.67</b>	NM	0.06	<b>7.51</b>	<b>23.15</b>	0.08
13 hedgehog [46]	65	1000+	N/A	<b>NF</b>	0.50	0.34	<b>NF</b>	<b>NF</b>	0.33
14 EMT [19]	69	268	<b>39.22</b>	<b>1.01</b>	0.20	0.12	<b>75.81</b>	<b>NF</b>	0.05
15 Bcell [51]	73	72	0.23	0.04	0.08	0.06	<b>18.95</b>	<b>81.85</b>	0.05
16 mast_cell [6]	73	1000+	N/A	0.09	0.55	0.37	<b>NF</b>	<b>NF</b>	0.15
17 Corral_ThIL17diff [43]	92	1000+	N/A	<b>107.57</b>	0.76	0.56	<b>NF</b>	<b>NF</b>	0.16
18 Adhesion_CIP [52]	121	78	<b>56.81</b>	<b>4.25</b>	0.23	0.17	<b>25.20</b>	<b>NF</b>	0.19
19 EMT_Mech [53]	136	82	<b>NF</b>	<b>14.01</b>	0.27	0.20	<b>27.55</b>	<b>NF</b>	0.25
20 macrophage [46]	136	1000+	N/A	0.54	1.09	0.84	<b>NF</b>	<b>NF</b>	0.27
21 angiogenesis [46]	141	1000+	N/A	0.16	1.07	1.06	<b>NF</b>	<b>NF</b>	0.16
22 angiofull [54]	142	1000+	N/A	0.17	1.06	0.88	<b>NF</b>	<b>NF</b>	0.23
23 EMT_Mech_TGFBeta [53]	150	492	<b>NF</b>	<b>11.28</b>	0.78	0.69	<b>NF</b>	<b>NF</b>	0.35
24 RA_apoptosis [6]	180	1000+	N/A	<b>NF</b>	1.43	1.55	<b>NF</b>	<b>NF</b>	0.19
25 MAPK [6]	181	1000+	N/A	<b>13.58</b>	1.76	1.51	<b>NF</b>	<b>NF</b>	0.27
26 Snf1-pathway [55]	202	1000+	N/A	1.13	1.47	1.43	<b>NF</b>	<b>NF</b>	0.31
27 T-cell-co-receptor [46]	206	1000+	N/A	<b>NF</b>	1.52	2.26	<b>NF</b>	<b>NF</b>	0.35
28 TcellCheckPoint [56]	218	1000+	N/A	<b>4.99</b>	NM	1.96	<b>NF</b>	<b>NF</b>	0.28
29 Mycobacterium [46]	317	1000+	N/A	0.42	2.36	<b>4.91</b>	<b>NF</b>	<b>NF</b>	0.44
30 Leishmania [46]	342	1000+	N/A	<b>NF</b>	2.56	<b>5.62</b>	<b>NF</b>	<b>NF</b>	0.46
31 Cholocystokinin [6]	383	1000+	N/A	0.36	2.99	<b>4.81</b>	<b>NF</b>	<b>NF</b>	0.37
32 Alzheimer [6]	762	1000+	N/A	<b>NF</b>	NM	<b>18.21</b>	<b>NF</b>	<b>NF</b>	0.79

and inhibition parts (see Subsection 4.3). However, the reason may lie on the differences in the ASP encoding characteristics of the two methods and the fact that mpbn needs to spend time checking the local-monotonicity of each Boolean function in a Boolean network. We expect that mpbn may outperform Trappist for a certain set of models, but not for the set of real-world



791 models considered in this article.

792 Fourth, regarding the comparison of the ASP-based methods (i.e.,  
793 `PyBoolNet`, `mpbn`, and `Trappist-ASP`), we note that for all the models where  
794 `PyBoolNet` did not finish before the time limit, the timeout occurred during  
795 the computation of the **prime implicants**. Hence, not even a single minimal  
796 trap space was output by that method. For all the remaining models, once  
797 `PyBoolNet` went through the prime-implicant phase, its ASP solving phase  
798 quickly returned the first 1000 minimal trap spaces, all under one second.  
799 Hence, with the experimental results shown in this subsection as well as the  
800 two previous subsections, the practical differences between the ASP encod-  
801 ing of `Trappist-ASP` and that of `PyBoolNet` are not distinctly exposed. The  
802 fact that our new ASP encoding is guaranteed to be linear in the number of  
803 nodes of the original model (see Subsection 4.3) does not seem to be crucial  
804 here, however a much deeper analysis of those cases shall be shown in the  
805 next subsection.

#### 806 6.4. Randomly generated models

807 We randomly generated a set of N-K models [1] with network size  $n$  in the  
808 set  $\{100, 150, 200, 250, 300, 350, 400\}$  and in-degree  $K = 3$  (i.e., each node  
809 has exactly three input nodes). We chose N-K models because they are a  
810 useful tool for studying the dynamics of Boolean networks [1, 7, 19]. For each  
811 network size, 50 instances were generated using the `generateRandomNKNetwork`  
812 function. In total, we have 350 random models. We then applied the com-  
813 pared methods to these models and recorded the running time of each method  
814 for each model. It is worth noting that N-K models usually have small num-  
815 bers of minimal trap spaces [7]. Hence, we searched for all solutions in each  
816 model, which makes the comparison to `bioLQM` more comprehensive. In addi-  
817 tion, each node has only three input nodes, **leading to a small number of prime**  
818 **implicants of the associated Boolean function**. Hence, `PyBoolNet` always  
819 passed the phase of computing **prime implicants** in every model even within  
820 one second, which enables us to compare the ASP encoding of `PyBoolNet`  
821 and that of `Trappist-ASP`.

822 Figure 3 shows cumulative numbers of random models solved by the com-  
823 pared methods with respect to enumerating all the minimal trap spaces. The  
824 number of succeeded models within three minutes for each method is: `bioLQM`  
825 (0), `PyBoolNet` (320), `mpbn` (0), `Trappist-maxSAT` (338), `Trappist-CP` (226),  
826 `Trappist-ILP` (39), `Trappist-ASP` (349). We can see that `Trappist-ASP` is  
827 the only method that can handle every model, but one. Note that none of



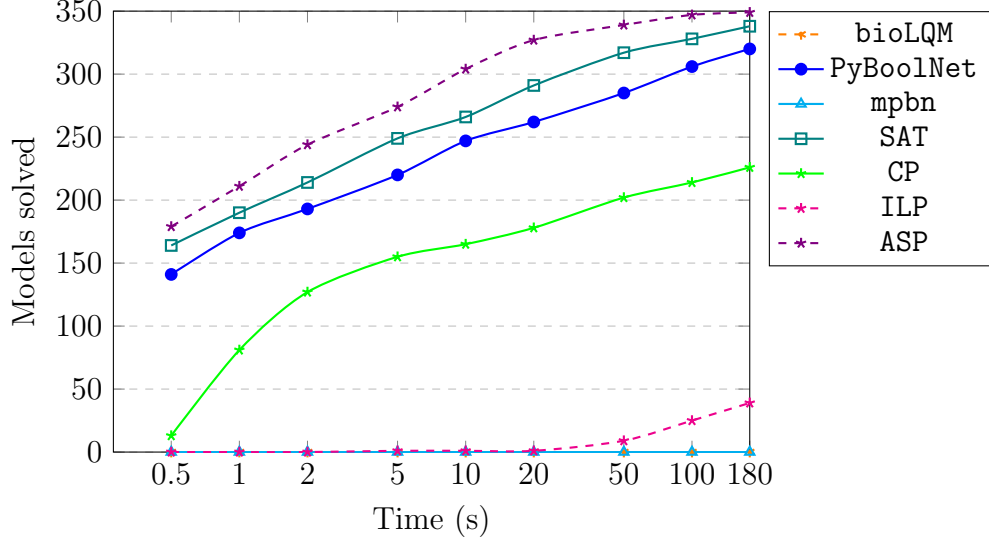


Figure 3: Cumulative numbers of random models solved by the compared methods with respect to enumerating all the minimal trap spaces.

the other methods can handle that only model failed by **Trappist-ASP**. We also obtained some observations consistent with those obtained for real-world models. More specifically, **Trappist-MaxSAT** is still the second best method and **Trappist-CP** is better than **Trappist-ILP**. Upon closer inspection, we obtained several notable observations as follows.

First, **mpbn** was not able to handle any model because all the models are non-locally-monotonic. Recall that a Boolean network is non-locally-monotonic if only one of its Boolean functions is non-locally-monotonic. Hence, it is apparent that all **these types** of randomly generated models are non-locally-monotonic because of the number of nodes is large ( $n \geq 100$ ). This observation confirms a limit on the applicable model class of **mpbn**.

Second, surprisingly **bioLQM** cannot handle any model. One of the reason may be that the BDD characterizing all **generic** trap spaces is too large, and its computation is slow. In addition, having too many generic trap spaces before the filtering process may be also a reason. It is apparent because the network size is large ( $n \geq 100$ ) and the Boolean functions are not simple.

Third, for every time limit, **Trappist-ASP** can always handle many more models than **PyBoolNet**, ranging from 29 to 65 more models. Since the time for the phase of computing **prime implicants** of **PyBoolNet** is negligible

in every model, most of the running time of `PyBoolNet` was spent for its ASP solving phase. Hence, we can easily see that the ASP encoding of `Trappist-ASP` is much better than that of `PyBoolNet`. This observation is consistent with the theoretical comparison in the ASP encoding between `Trappist-ASP` and `PyBoolNet` mentioned in Subsection 4.3.

## 6.5. Experimental summary

We have tested our alternative approach on many Boolean network models of various sizes and types (e.g., real-world models, randomly generated models) on existing and newly created benchmarks. This indicates the high coverage and comprehensiveness of the experiments.

Among the four variants of the alternative approach, `Trappist-ASP` is the best method as it vastly outperforms all the other variants. The second best one is `Trappist-MaxSAT`. The two remaining variants (i.e., `Trappist-CP` and `Trappist-ILP`) give bad performance for most models. However, for certain cases, they are still better than all state-of-the-art methods (i.e., `bioLQM`, `PyBoolNet`, and `mpbn`). This is evidence for the advantages of an alternative approach compared to what preexisted.

Regarding general Boolean networks, `Trappist-ASP` (even `Trappist-MaxSAT`) is far more efficient than both `bioLQM` and `PyBoolNet`. The speedups of `Trappist-ASP` or `Trappist-MaxSAT` are large, even between one and three orders of magnitude for most models. In addition, the experimental results also confirm that the ASP encoding of `Trappist-ASP` is much more efficient than that of `PyBoolNet`.

Regarding locally-monotonic Boolean networks, the performance of `mpbn` is roughly comparable to that of `Trappist-ASP` or `Trappist-MaxSAT`. However, `mpbn` is quite slower than `Trappist-ASP` on average. This shows the practical advantage of `Trappist-ASP` compared to `mpbn`, though its ASP encoding may be more complex than that of `mpbn` in theory.

## 7. Conclusion

In this article we have explored and proved for the first time the equivalence between (minimal) trap spaces of a general Boolean network and (maximal) conflict-free siphons of its Petri net encoding. We have shown several useful applications of this finding to studying properties of trap spaces in Boolean networks. As an important practical application of the equivalence, we have proposed a new approach for the computation of minimal trap

spaces in Boolean networks, based on the enumeration of maximal conflict-free siphons of Petri nets. We have also proposed four possible methods using MaxSAT, CP, ILP, and ASP for implementing the new approach. In particular, we have shown how to adjust our approach to compute several specific types of trap spaces (e.g., maximal trap spaces, fixed points), which besides minimal trap spaces also play crucial roles in the analysis and control of Boolean networks. The proposed methods for the minimal trap space computation have been evaluated on many real-world models from the literature as well as randomly generated models. The experimental results show that the new approach vastly outperforms all the state-of-the-art methods in terms of general Boolean networks and is comparable to the `mpbn` method even much better on average in terms of locally-monotonic Boolean networks. We believe that this opens up the way to a much better analysis of large Boolean networks, which is needed with the advent of automatic model-generation pipelines [57].

Although the experimental results show the superiority of our approach to `mpbn` in general, we however note that there is a model in the `BBM` repository (with identifier 122) where all the four proposed methods for the new approach did not manage to finish the Petri net conversion before the timeout, whereas `mpbn` can still handle this model. The model is not very large but its Boolean functions are rather complicated. This points to the fact that our current choice of using a BDD-based translation to obtain that Petri net encoding, though it provides a small/efficient ASP might be too costly to handle the complex models. In such a case, a more *naive* encoding might provide a much larger ASP program, with many redundant rules, but easier/faster to obtain. The evaluation of the feasibility of such strategy, and of its impact on smaller instances, remains to be done. Recognizing that a model is locally-monotonic and applying in that specific case dedicated strategies as those of `mpbn` might also be a partial solution.

Another direction to speed up our approach in the side of Boolean networks is to apply reduction techniques to the original Boolean network. Many reduction techniques on Boolean networks [58, 59] have been proposed and some of them fully preserve attractors of a Boolean network under the fully asynchronous update scheme. In particular, a reduction technique on elimination of negatively auto-regulated nodes with respect to asynchronous attractors has recently been proposed [59]. However, there are two major issues needed to be considered. First, the question of whether these reduction techniques fully preserve minimal trap spaces of a Boolean network is still open.

920 Second, although these reduction techniques can reduce the number of nodes,  
921 they can also increase the complexity of Boolean update functions [58], which  
922 is also an important factor for the efficiency of computation methods. It raises  
923 the question of whether they really simplify the computational burden of trap  
924 space computation. We will deeply investigate the two issues. Furthermore,  
925 we believe that the connection between trap spaces and siphons can be a  
926 very useful tool for addressing the first issue.

927 It is worth noting that there may be possibly other methods for comput-  
928 ing minimal/maximal conflict-free siphons in Petri nets, like the methods for  
929 generic siphon computation in the field of Petri nets (see [35] for a survey  
930 about these methods). Although these approaches do not directly support  
931 the minimal/maximal conflict-free siphon computation now, we plan to in-  
932 vestigate them in the future. They could replace our proposed methods if  
933 they give significantly better performance. Making use of the specific struc-  
934 ture (1-safe, place-complementary) might also reveal new techniques to be  
935 considered. However, the current methods appear to already perform very  
936 well even on the biggest models we have considered.

937 Finally, we think that the links between Petri nets and Boolean networks  
938 that we stumbled upon in this article might have deeper roots. Exploring  
939 those connections might lead both to interesting topics of research for Petri  
940 nets, like a notion of trap-spaces, and for Boolean networks. We also believe  
941 that the connection between trap spaces of Boolean networks and siphons  
942 of Petri nets can be a very useful tool for exploring and proving more new  
943 properties of trap spaces in Boolean networks, as we have used it to success-  
944 fully prove the independence of trap spaces to the update scheme and the  
945 separation of minimal trap spaces. Diving into this direction is promising  
946 and one of our future work.

## 947 References

- 948 [1] L. Glass, S. A. Kauffman, The logical analysis of continuous, non-linear  
949 biochemical control networks, J. Theor. Biol. 39 (1973) 103–129.
- 950 [2] R. Thomas, Boolean formalisation of genetic control circuits, J. Theor.  
951 Biol. 42 (1973) 565–583.
- 952 [3] R. Thomas, R. d’Ari, Biological feedback, CRC press, 1990.

- 953 [4] R. Thomas, Regulatory networks seen as asynchronous automata: a  
954 logical description, *J. Theor. Biol.* 153 (1991) 1–23.
- 955 [5] R.-S. Wang, A. Saadatpour, R. Albert, Boolean modeling in systems  
956 biology: an overview of methodology and applications, *Phys. Biol.* 9  
957 (2012) 055001.
- 958 [6] S. S. Aghamiri, V. Singh, A. Naldi, T. Helikar, S. Soliman, A. Niarakis,  
959 J. Xu, Automated inference of Boolean models from molecular interac-  
960 tion maps using CaSQ, *Bioinform.* 36 (2020) 4473–4482.
- 961 [7] H. Klarner, A. Bockmayr, H. Siebert, Computing maximal and minimal  
962 trap spaces of Boolean networks, *Nat. Comput.* 14 (2015) 535–544.
- 963 [8] S. Chevalier, C. Froidevaux, L. Paulevé, A. Y. Zinovyev, Synthesis of  
964 Boolean networks from biological dynamical constraints using answer-  
965 set programming, in: *International Conference on Tools with Artificial*  
966 *Intelligence*, IEEE, 2019, pp. 34–41.
- 967 [9] L. Paulevé, J. Kolčák, T. Chatain, S. Haar, Reconciling qualitative,  
968 abstract, and scalable modeling of biological networks, *Nat. Commun.*  
969 11 (2020) 1–7.
- 970 [10] M. Noual, D. Regnault, S. Sené, About non-monotony in Boolean au-  
971 tomata networks, *Theor. Comput. Sci.* 504 (2013) 12–25.
- 972 [11] J. L. Peterson, *Petri net theory and the modeling of systems*, Prentice  
973 Hall PTR, 1981.
- 974 [12] T. Murata, Petri nets: Properties, analysis and applications, *Proc.*  
975 *IEEE* 77 (1989) 541–580.
- 976 [13] V. N. Reddy, M. L. Mavrovouniotis, M. N. Liebman, Petri net rep-  
977 resentations in metabolic pathways, in: *International Conference on*  
978 *Intelligent Systems for Molecular Biology*, AAAI, 1993, pp. 328–336.
- 979 [14] I. Zevedei-Oancea, S. Schuster, Topological analysis of metabolic net-  
980 works based on Petri net theory, *Silico Biol.* 3 (2003) 323–345.
- 981 [15] M. A. Blätke, M. Heiner, W. Marwan, Biomodel engineering with Petri  
982 nets, in: *Algebraic and Discrete Mathematical Methods for Modern*  
983 *Biology*, Elsevier, 2015, pp. 141–192.

- 984 [16] O. Oanea, H. Wimmel, K. Wolf, New algorithms for deciding the siphon-  
985 trap property, in: International Conference on Applications and Theory  
986 of Petri Nets, Springer, 2010, pp. 267–286.
- 987 [17] F. Nabli, T. Martinez, F. Fages, S. Soliman, On enumerating mini-  
988 mal siphons in Petri nets using CLP and SAT solvers: theoretical and  
989 practical complexity, *Constraints An Int. J.* 21 (2016) 251–276.
- 990 [18] V. Trinh, B. Benhamou, K. Hiraishi, S. Soliman, Minimal trap spaces of  
991 logical models are maximal siphons of their Petri net encoding, in: In-  
992 ternational Conference on Computational Methods in Systems Biology,  
993 Springer, 2022, pp. 158–176.
- 994 [19] J. C. Rozum, J. G. T. Zañudo, X. Gan, D. Deritei, R. Albert, Parity  
995 and time reversal elucidate both decision-making in empirical models  
996 and attractor scaling in critical Boolean networks, *Sci. Adv.* 7 (2021)  
997 eabf8124.
- 998 [20] C. Gershenson, Introduction to random Boolean networks, in: Proceed-  
999 ings of the Ninth Int. Conf. on the Simulation and Synthesis of Living  
1000 Systems (ALife IX), MIT Press, 2004, p. 160–173.
- 1001 [21] H. Klarner, A. Streck, H. Siebert, PyBoolNet: a python package for the  
1002 generation, analysis and visualization of Boolean networks, *Bioinform.*  
1003 33 (2017) 770–772.
- 1004 [22] L. C. Fontanals, E. Tonello, H. Siebert, Control strategy identification  
1005 via trap spaces in Boolean networks, in: International Conference on  
1006 Computational Methods in Systems Biology, Springer, 2020, pp. 159–  
1007 175.
- 1008 [23] T. Chatain, S. Haar, L. Jezequel, L. Paulevé, S. Schwoon, Characteriza-  
1009 tion of reachable attractors using Petri net unfoldings, in: International  
1010 Conference on Computational Methods in Systems Biology, Springer,  
1011 2014, pp. 129–142.
- 1012 [24] C. Chaouiya, E. Remy, P. Ruet, D. Thieffry, Qualitative modelling of  
1013 genetic networks: From logical regulatory graphs to standard Petri nets,  
1014 in: International Conference on Applications and Theory of Petri Nets,  
1015 Springer, 2004, pp. 137–156.

- 1016 [25] C. Chaouiya, A. Naldi, E. Remy, D. Thieffry, Petri net representation of  
1017 multi-valued logical regulatory graphs, *Nat. Comput.* 10 (2011) 727–750.
- 1018 [26] T. Chatain, S. Haar, J. Kolcák, L. Paulevé, A. Thakkar, Concurrency  
1019 in Boolean networks, *Nat. Comput.* 19 (2020) 91–109.
- 1020 [27] C. Chaouiya, D. Bérenguier, S. M. Keating, A. Naldi, et al., SBML  
1021 qualitative models: a model representation format and infrastructure to  
1022 foster interactions between qualitative modelling formalisms and tools,  
1023 *BMC Syst. Biol.* 7 (2013) 1–15.
- 1024 [28] S. M. Keating, D. Waltemath, M. König, F. Zhang, et al., SBML Level  
1025 3: an extensible format for the exchange and reuse of biological models,  
1026 *Mol. Syst. Biol.* 16 (2020) e9110.
- 1027 [29] C. Chaouiya, A. Naldi, D. Thieffry, Logical modelling of gene regulatory  
1028 networks with GINsim, in: *Bacterial Molecular Networks*, Springer,  
1029 2012, pp. 463–479.
- 1030 [30] A. Naldi, P. T. Monteiro, C. Müssel, C. for Logical Models, Tools,  
1031 H. A. Kestler, D. Thieffry, I. Xenarios, J. Saez-Rodriguez, T. Helikar,  
1032 C. Chaouiya, Cooperative development of logical modelling standards  
1033 and tools with CoLoMoTo, *Bioinform.* 31 (2015) 1154–1159.
- 1034 [31] C. Müssel, M. Hopfensitz, H. A. Kestler, BoolNet - an R package for  
1035 generation, reconstruction and analysis of Boolean networks, *Bioinform.*  
1036 26 (2010) 1378–1380.
- 1037 [32] D. Angeli, P. D. Leenheer, E. Sontag, A Petri net approach to persistence  
1038 analysis in chemical reaction networks, in: *Biology and Control Theory:  
1039 Current Challenges*, Springer, 2007, pp. 181–216.
- 1040 [33] D. Angeli, P. D. Leenheer, E. D. Sontag, Persistence results for chemical  
1041 reaction networks with time-dependent kinetics and no global conserva-  
1042 tion laws, *SIAM J. Appl. Math.* 71 (2011) 128–146.
- 1043 [34] E. Degrand, F. Fages, S. Soliman, Graphical conditions for rate inde-  
1044 pendence in chemical reaction networks, in: *International Conference on  
1045 Computational Methods in Systems Biology*, Springer, 2020, pp. 61–78.

- 1046 [35] G. Liu, K. Barkaoui, A survey of siphons in Petri nets, *Inf. Sci.* 363  
1047 (2016) 198–220.
- 1048 [36] H. Klarner, H. Siebert, Approximating attractors of Boolean networks  
1049 by iterative CTL model checking, *Front. Bioeng. Biotechnol.* 3 (2015)  
1050 130.
- 1051 [37] V. Trinh, K. Hiraishi, B. Benhamou, Computing attractors of large-scale  
1052 asynchronous Boolean networks using minimal trap spaces, in: *ACM*  
1053 *International Conference on Bioinformatics, Computational Biology and*  
1054 *Health Informatics*, ACM, 2022, pp. 13:1–13:10.
- 1055 [38] E. Demirović, G. Chu, P. J. Stuckey, Solution-based phase saving for  
1056 CP: A value-selection heuristic to simulate local search behavior in com-  
1057 plete solvers, in: *International Conference on Principles and Practice of*  
1058 *Constraint Programming*, Springer, 2018, pp. 99–108.
- 1059 [39] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, G. Tack,  
1060 MiniZinc: Towards a standard CP modelling language, in: *International*  
1061 *Conference on Principles and Practice of Constraint Program-*  
1062 *ming*, Springer, 2007, pp. 529–543.
- 1063 [40] A. Ignatiev, A. Morgado, J. Marques-Silva, RC2: an efficient MaxSAT  
1064 solver, *J. Satisf. Boolean Model. Comput.* 11 (2019) 53–64.
- 1065 [41] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub,  
1066 M. Schneider, Potassco: The Potsdam answer set solving collection,  
1067 *AI Commun.* 24 (2011) 107–124.
- 1068 [42] J. Forrest, T. Ralphs, H. G. Santos, S. Vigerske, J. Forrest, L. Hafer,  
1069 B. Kristjansson, jpfasano, EdwinStraver, M. Lubin, rlougee, jp-  
1070 goncall, Jan-Willem, h-i gassmann, S. Brito, Cristina, M. Saltz-  
1071 man, tosttost, B. Pitrus, F. MATSUSHIMA, to st, coin-or/Cbc: Re-  
1072 lease releases/2.10.8, 2022. URL: [https://doi.org/10.5281/zenodo.](https://doi.org/10.5281/zenodo.6522795)  
1073 [6522795](https://doi.org/10.5281/zenodo.6522795).
- 1074 [43] K. F. Corral-Jara, C. Chauvin, W. Abou-Jaoudé, M. Grandclaudon,  
1075 A. Naldi, V. Soumelis, D. Thieffry, Interplay between SMAD2 and  
1076 STAT5A is a critical determinant of IL-17A/IL-17F differential expres-  
1077 sion, *Mol. Biomed.* 2 (2021) 1–16.



- 1078 [44] V. Singh, M. Ostaszewski, G. D. Kalliolias, G. Chiocchia, R. Olasso,  
1079 E. Petit-Teixeira, T. Helikar, A. Niarakis, Computational systems bi-  
1080 ology approach for the study of rheumatoid arthritis: from a molecular  
1081 map to a dynamical model, *Genom. Comput. Biol.* 4 (2018) 100050.
- 1082 [45] S. Ogishima, S. Mizuno, M. Kikuchi, A. Miyashita, R. Kuwano,  
1083 H. Tanaka, J. Nakaya, AlzPathway, an updated map of curated sig-  
1084 naling pathways: towards deciphering Alzheimer’s disease pathogenesis,  
1085 in: *Systems Biology of Alzheimer’s Disease*, Springer, 2016, pp. 423–432.
- 1086 [46] C. Kadelka, T.-M. Butrie, E. Hilton, J. Kinseth, H. Serdarevic, A meta-  
1087 analysis of Boolean network models reveals design principles of gene  
1088 regulatory networks, *arXiv preprint arXiv:2009.01216* (2020).
- 1089 [47] E. C. Chávez-Hernández, S. Quiroz, B. García-Ponce, E. R. Álvarez-  
1090 Buyla, The flowering transition pathways converge into a complex gene  
1091 regulatory network that underlies the phase changes of the shoot apical  
1092 meristem in *Arabidopsis thaliana*, *Front. Plant Sci.* 13 (2022) 852047.
- 1093 [48] A. Yachie-Kinoshita, K. Onishi, J. Ostblom, M. A. Langley, E. Posfai,  
1094 J. Rossant, P. W. Zandstra, Modeling signaling-dependent pluripotency  
1095 with Boolean logic to predict cell fate transitions, *Mol. Syst. Biol.* 14  
1096 (2018) e7952.
- 1097 [49] M. R. Vega, Analyzing toys models of *Arabidopsis* and *Drosophila* us-  
1098 ing Z3 SMT-LIB, in: *Independent Component Analyses, Compressive*  
1099 *Sampling, Wavelets, Neural Net, Biosystems, and Nanoengineering XII*,  
1100 volume 9118, SPIE, 2014, pp. 240–254.
- 1101 [50] E. Cacace, S. Collombet, D. Thieffry, Logical modeling of cell fate  
1102 specification—Application to T cell commitment, in: *Current Topics in*  
1103 *Developmental Biology*, Elsevier, 2020, pp. 205–238.
- 1104 [51] P. Dutta, L. Ma, Y. Ali, P. M. Slood, J. Zheng, Boolean network model-  
1105 ing of B-cell apoptosis and insulin resistance in type 2 diabetes mellitus,  
1106 *BMC Syst. Biol.* 13 (2019) 1–12.
- 1107 [52] E. Guberman, H. Sherief, E. R. Regan, Boolean model of anchorage  
1108 dependence and contact inhibition points to coordinated inhibition but  
1109 semi-independent induction of proliferation and migration, *Comput.*  
1110 *Struct. Biotechnol. J.* 18 (2020) 2145–2165.

- 1111 [53] E. Sullivan, M. Harris, A. Bhatnagar, E. Guberman, I. Zonfa, E. R. Re-  
1112 gan, Boolean modeling of mechanosensitive Epithelial to Mesenchymal  
1113 Transition and its reversal, *bioRxiv* (2022).
- 1114 [54] N. Weinstein, L. Mendoza, I. Gitler, J. Klapp, A network model to  
1115 explore the effect of the micro-environment on endothelial cell behavior  
1116 during angiogenesis, *Front. Physiol.* 8 (2017) 960.
- 1117 [55] T. Lubitz, N. Welkenhuysen, S. Shashkova, L. Bendrioua, S. Hohmann,  
1118 E. Klipp, M. Krantz, Network reconstruction and validation of the  
1119 Snf1/AMPK pathway in baker’s yeast based on a comprehensive litera-  
1120 ture review, *npj Syst. Biol. Appl.* 1 (2015) 1–10.
- 1121 [56] C. Hernandez, M. Thomas-Chollier, A. Naldi, D. Thieffry, Computa-  
1122 tional verification of large logical models—Application to the prediction  
1123 of T cell response to checkpoint inhibitors, *Front. Physiol.* 11 (2020)  
1124 558606.
- 1125 [57] M. Ostaszewski, A. Niarakis, A. Mazein, I. Kuperstein, R. Phair,  
1126 A. Orta-Resendiz, V. Singh, S. S. Aghamiri, M. L. Acencio, E. Glaab,  
1127 et al., COVID19 Disease Map, a computational knowledge repository of  
1128 virus–host interaction mechanisms, *Mol. Syst. Biol.* 17 (2021) e10387.
- 1129 [58] A. Naldi, E. Remy, D. Thieffry, C. Chaouiya, Dynamically consistent  
1130 reduction of logical regulatory graphs, *Theor. Comput. Sci.* 412 (2011)  
1131 2207–2218.
- 1132 [59] R. Schwieger, E. Tonello, Reduction for asynchronous Boolean net-  
1133 works: elimination of negatively autoregulated components, 2023.  
1134 [arXiv:2302.03108](https://arxiv.org/abs/2302.03108).