

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

Van-Giang Trinh^a, Belaid Benhamou^a, Sylvain Soliman^{b,*}

^a*LIS, Aix-Marseille University, Marseille, France*

^b*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 (GRN) use thresholds or equivalently logical functions to represent the different regulations [1, 2, 3, 4].

Boolean net modeling 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 [5], even for modelling post-transcriptional mechanisms. This is even more true for very large models where such data is frequently missing and led to a constant increase in size of logical models *à la* Thomas [6]. 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 [7] 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. 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-implicants computation by relying on the *most-permissive* semantics of Boolean networks. This method has been implemented in the tool `mpbn`¹ demonstrated in [9] for handling medium-sized models from the literature and very large synthetic models (up to 100,000 nodes). However, this method is only applicable for *locally-monotonic* Boolean networks, whereas the prime-implicants based method [7]

¹<https://github.com/bnediction/mpbn>

30 is applicable for *general* Boolean networks (i.e., including both locally-monotonic
 31 and non-locally-monotonic ones). In addition, the `bioLQM` platform also pro-
 32 vides another method using Binary Decision Diagrams (BDDs) in [http://](http://colomoto.org/biolqm/doc/tools-trapspaces.html)
 33 colomoto.org/biolqm/doc/tools-trapspaces.html. This method avoids
 34 the prime-implicants computation as it characterizes the set of generic trap
 35 spaces of a Boolean network by a BDD, then filters this set to get the set
 36 of all minimal trap spaces. By this approach, it requires the computation
 37 of all solutions, whereas the ASP-based methods [7, 9] can start enumerat-
 38 ing them as they are found. Moreover, the main issue with the BDD-based
 39 method is that the number of generic trap spaces of a Boolean network may
 40 be extremely larger than the number of minimal trap spaces of this Boolean
 41 network. This issue limits the efficiency of the BDD-based method. The
 42 study [10] highlights the need for non-locally-monotonic Boolean networks
 43 in both biological and theoretical aspects. Hence, it is still necessary to
 44 develop efficient methods for computing minimal trap spaces of large-scale
 45 general Boolean networks.

46 Petri nets were introduced in the 60s as simple formalism for describing
 47 and analyzing information-processing systems that are characterized as be-
 48 ing concurrent, asynchronous, non-deterministic and possibly distributed [11,
 49 12]. The use of Petri nets for representing biochemical reaction systems, by
 50 mapping molecular species to places and reactions to transitions, hinted at
 51 already in [11, 12] was used more thoroughly quite late in [13], together with
 52 some Petri net concepts and tools for the analysis of metabolic networks.
 53 Siphons are such a concept, but they have not been used a lot for the study
 54 of biochemical systems [14, 15] even if the practical cost of computing their
 55 minimal/maximal elements appear much more manageable than the theoret-
 56 ical complexity would indicate [16, 17].

57 In this article we explore and prove for the first time a connection be-
 58 tween trap spaces of a general Boolean network and siphons of its Petri net
 59 encoding. Not only having important theoretical applications in studying
 60 properties of trap spaces in Boolean networks, the connection has impor-
 61 tant practical applications in the trap space computation. Specifically, based
 62 on the connection, we propose an alternative approach to compute minimal
 63 trap spaces, and hence complex attractors, of a general Boolean network. It
 64 replaces the need for prime-implicants by a completely different technique,
 65 namely the enumeration of maximal siphons in the Petri net encoding of the
 66 original model. We then demonstrate its efficiency and compare it to the
 67 state-of-the-art methods for computing minimal trap spaces in Boolean net-

68 works on many real-world models from various sources in the literature and
69 randomly generated models.

70 Herein we revise and extend our previous work in [18] as follows. First,
71 more formal definitions are given and the existing proofs are made more de-
72 tailed. In particular, an updated proof provides another way to prove the
73 independence of trap spaces of a Boolean network on its update scheme,
74 which was originally proved in [7]. Second, we showcase a theoretical ap-
75 plication of the connection between trap spaces in Boolean networks and
76 conflict-free siphons in Petri nets. Third, beyond the proposed ASP method
77 implementing the alternative approach [18], we propose several other possi-
78 ble methods for computing minimal trap spaces using Maximum Satisfiability
79 (MaxSAT), Constraint Programming (CP), and Integer Linear Programming
80 (ILP). Fourth, we discuss in detail how to compute several special types of
81 trap spaces in a Boolean network. Fifth, regarding the implementation, we
82 have developed a new converter that directly reads a `.bnet` file and builds
83 the Petri net encoding, instead of using the PNML conversion of `bioLQM` [18].
84 Finally, we conduct a more comprehensive benchmark on more real-world
85 models from various sources and randomly generated models to evaluate all
86 the proposed methods (the benchmark conducted in [18] considers only real-
87 world models).

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

97 2. Preliminaries

98 We shall briefly recall here some preliminaries on Boolean networks re-
99 lated to trap spaces and Petri nets. Remove this statement because there
100 is not sure if the encoded Boolean network preserves the trap spaces of the
101 original multi-level logical model.

102 2.1. Boolean networks

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

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

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

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

$$v_i(t+1) = f_i(v(t))$$

119 where $v(t)$ is the state of \mathcal{N} at time t and $v_i(t+1)$ is the state of node v_i at
120 time $t+1$. Note that for simplicity, we write $f_i(v(t))$ even $IN(v_i) \subset V$ (i.e.,
121 $IN(v_i)$ does not contain some nodes of V). An update scheme of a Boolean
122 network specifies the way that the nodes update their states through time
123 evolution [4]. Following the update scheme, the Boolean network transits
124 from a state to another state (possibly identical). This transition is called
125 the *state transition* and denoted by $\rightarrow \subseteq \mathcal{S}_{\mathcal{N}} \times \mathcal{S}_{\mathcal{N}}$. Then the dynamics of \mathcal{N}
126 is captured by the directed graph $(\mathcal{S}_{\mathcal{N}}, \rightarrow)$ called the State Transition Graph
127 (STG). There are two main types of update schemes [4]: synchronous, where
128 all the nodes are update simultaneously, and fully asynchronous, where only
129 one node is nondeterministically selected to be updated.

130 2.2. Traps spaces

131 We recall here some definitions from [7] for the introduction of *trap spaces*.
132 Minimal trap spaces prove to be a very good approximation of the attractors
133 of a Boolean network under asynchronous update schemes and have become
134 the *de facto* standard way to analyze models of a few tens of *genes* [19, 20].

135 An non-empty set $T \subseteq \mathcal{S}_{\mathcal{N}}$ is a trap set with respect to \rightarrow if for every
 136 $x \in T$ and $y \in S$ with $x \rightarrow y$ it holds that $y \in T$ [7]. An attractor of \mathcal{N}
 137 with respect to \rightarrow can be defined as an inclusion-wise minimal trap set of
 138 $(\mathcal{S}_{\mathcal{N}}, \rightarrow)$. An attractor can be also seen as a terminal strongly connected
 139 component of $(\mathcal{S}_{\mathcal{N}}, \rightarrow)$ [21]. An attractor of size 1 is called a fixed point,
 140 otherwise a cyclic attractor [7].

141 A subspace m of a Boolean network $\mathcal{N} = (V, F)$ is a mapping $m: V \mapsto$
 142 $\mathbb{B} \cup \{\star\}$. $m(v_i) \in \mathbb{B}$ means that the value of v_i is fixed in m and v_i is called
 143 a fixed variable. $m(v_i) \in \star$ means that the value of v_i is free in m and v_i is
 144 called a free variable. We denote D_m the set of all fixed variables of m . A
 145 subspace m is equivalent to a set of states:

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

146 For example, $m = \star \star 1$ (for simplicity, we write subspaces likes states)
 147 means that $D_m = \{v_3\}$, $m(v_3) = 1$, and it is equivalent to the set of states
 148 $\{001, 011, 101, 111\}$. We denote $\mathcal{S}_{\mathcal{N}}^* = (\mathbb{B} \cup \{\star\})^n$ the set of all possible
 149 subspaces of \mathcal{N} . Note that $|\mathcal{S}_{\mathcal{N}}^*| = 3^n$ and $\mathcal{S}_{\mathcal{N}} \subset \mathcal{S}_{\mathcal{N}}^*$ [7].

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

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

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

165 2.3. Petri net encoding of Boolean networks

166 **Definition 2.2.** A Petri net is a weighted bipartite directed graph (P, T, W) ,
 167 where P is a non-empty finite set of vertices called places, T is a non-empty



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

168 *finite set of vertices called transitions, $P \cap T = \emptyset$, and $W : (P \times T) \cup (T \times P) \mapsto$*
 169 *\mathbb{N} is a weight function attached to the arcs.*

170 A *marking* for a Petri net is a mapping $m : P \mapsto \mathbb{N}$ that assigns a number
 171 of tokens to each place. A place p is marked by a marking m if and only if
 172 $m(p) > 0$. Marking m can be seen as a subset of P that contains all marked
 173 places by m . We shall write $\text{pred}(x)$ (resp. $\text{succ}(x)$) to represent the set of
 174 vertices that have a (non-zero weighted) arc leading to (resp. coming from) x .
 175 In this work, we consider a class of Petri nets called 1-safe Petri nets where
 176 every place has at most 1 token and all arcs are of weight 1. In this case,
 177 weights are implicitly omitted in the arcs of a Petri net. Then, a transition
 178 $t \in T$ is *enabled* at a marking m if and only if $\text{pred}(t) \subseteq m$. A marking m
 179 is called a *deadlock* if there are no enabled transitions at m . The firing of
 180 t leads to a new marking m' specified by $m' = (m \setminus \text{pred}(t)) \cup \text{succ}(t)$. Note
 181 that when multiple transitions are enabled, we need to embed one firing
 182 scheme (similar to the update scheme of a Boolean network) to the Petri
 183 net. The classical firing scheme is that only one of the enabled transition is
 184 non-deterministically chosen to fire [12].

185 The link between Boolean networks *à la* Thomas and Petri nets was
 186 originally established in [22] in order to make available formal methods like
 187 model-checking for the analysis of such systems. The basic encoding into 1-
 188 safe (i.e., never more than one token in each place) nets only holds for purely
 189 Boolean networks but was later extended to multivalued logical models in
 190 two ways, either in [23] with non 1-safe Petri nets or more recently in [21]
 191 with 1-safe nets but many more places.

192 Since our study is focused on Boolean networks, we briefly recall the origi-
 193 nal encoding here. Its basis is that every node (*gene*) v of the original model
 194 $\mathcal{N} = (V, F)$ is represented by two separate places (p_v and \bar{p}_v), corresponding

195 to its two states, active, and inactive, respectively. Each conjunct of the
196 logical function that activates the *gene* will lead to a transition t , consuming
197 the inactive place (i.e., a directional arc from \bar{p}_v to t), producing the active
198 place (i.e., a directional arc from t to p_v), and with all other literals both
199 consumed and produced (i.e., a bidirectional arc). And conversely for the
200 inactivation. Let s be a state of the Boolean network and m_s be its corre-
201 sponding marking in the encoded Petri net. It holds that $\forall v \in V, s(v) = 0$ if
202 and only if $m_s(\bar{p}_v) = 1$ and $s(v) = 1$ if and only if $m_s(p_v) = 1$. Note also that
203 at any marking m of the Petri net encoding a Boolean network, it always
204 holds that $m(p_v) + m(\bar{p}_v) = 1$.

205 The main property of this encoding is that it is completely faithful with
206 respect to the update scheme of the original Boolean network. For each node
207 v of \mathcal{N} , only transitions corresponding to v can change the current marking
208 of p_v or \bar{p}_v . In addition, at any marking at most one of such transitions is en-
209 abled because $m(p_v) + m(\bar{p}_v) = 1$ holds. Hence, for any update scheme in \mathcal{N} ,
210 we have a corresponding firing scheme in \mathcal{P} , which preserves the equivalence
211 between the dynamics of \mathcal{N} and \mathcal{P} [24].

212 For illustration, let us reconsider the Boolean network shown in Exam-
213 ple 2.1. Figure 1(b) shows the Petri net encoding of this Boolean network.
214 Place p_{x_1} (resp. \bar{p}_{x_1}) in \mathcal{P} represents the activation (resp. the inactivation) of
215 node x_1 in \mathcal{N} . Marking $\{p_{x_1}, \bar{p}_{x_2}\}$ in \mathcal{P} represents state 10 in \mathcal{N} . Transitions
216 $t_{x_1}^1$ and $t_{x_1}^2$ represent the update of node x_1 . Of course, in any marking $t_{x_1}^1$
217 and $t_{x_1}^2$ cannot be both enabled. Then, the fully asynchronous update scheme
218 in \mathcal{N} corresponds to the classical firing scheme in \mathcal{P} where only one of the
219 enabled transitions for a given marking will be fired [12].

220 Note that given a Boolean network in the standard SBML-**Qual** format [25],
221 i.e., the package of SBML v3 [26] for such models, one can easily obtain its
222 Petri net encoding in the Petri Net Markup Language (PNML)² standard
223 using the **bioLQM**³ library. This piece of software extracted from **GINsim** [27]
224 and part of the **CoLoMoTo**⁴ [28] software suite allows for easy conversion
225 between standard formats. It also accepts many other common formats for
226 Boolean networks, notably the **.bnet** files of the BoolNet [29, 19] tools. The
227 conversion is executed as follows:

²<https://www.pnml.org/>

³<http://www.colomoto.org/biolqm/>

⁴<http://colomoto.org/>

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

229 Note that transforming a Boolean network defined by its functions into its
 230 Petri net encoding roughly relies on obtaining conditions for the activation
 231 and inactivation of the states. In [22] this took the form of the whole truth
 232 table of the Boolean functions, but as shown in Appendix 1 of [21] comput-
 233 ing Disjunctive Normal Forms (DNF) of each Boolean function is enough.
 234 Though this might appear quite computationally intensive it is important to
 235 remark first that contrary to the prime-implicants case, there is no need to
 236 find *minimal* DNFs. One way to look at this is to consider that this amounts
 237 to a similar approach as that used in [8] but with the encoding of both activa-
 238 tion and inhibition functions as DNFs in order to take into account possible
 239 non-local-monotonicity. This does not change the worst-case-complexity (ob-
 240 taining a single DNF being exponential) but might matter a lot in practice.
 241 As such, we will explore how this transformation, here using BDDs in `bioLQM`
 242 and directly in our tool using the `pyeda`⁵ library, and the one based on the
 243 most-permissive semantics compare in the Section 6 on evaluation.

244 2.4. Siphons

245 Siphons are a static and classical property of Petri nets [11]. Note how-
 246 ever that the use of siphons for the analysis of biological models, though it is
 247 not new, has been mostly relevant to the ODE-based continuous semantics
 248 of Chemical Reaction Networks [30, 31, 32]. We recall here the basic defini-
 249 tion establishing that to produce something in a siphon you must consume
 250 something from the siphon. This corresponds to the idea that a siphon is a
 251 set of places that once unmarked remains unmarked.

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

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

254 *Note that \emptyset is trivially a siphon.*

255 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
 256 conventionally $\text{pred}(S) = \text{succ}(S) = \emptyset$. We have an important property on
 257 siphons [33] as follows.

258 **Proposition 2.1.** *Let S be a siphon of a Petri net (P, T, W) . Then $\text{pred}(S) \subseteq$*
 259 *$\text{succ}(S)$.*

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

260 3. Minimal trap spaces as maximal conflict-free siphons

261 First, we add a definition related to any set of places of a Petri net
262 encoding a Boolean network, and notably a siphon of such a net.

263 **Definition 3.1.** *A set of places of Petri net \mathcal{P} encoding Boolean network*
264 *\mathcal{N} is conflict-free if it does not contain any two places corresponding to the*
265 *active and inactive states of the same node of \mathcal{N} . Then, a conflict-free siphon*
266 *S is said to be maximal if and only if there is no other conflict-free siphon*
267 *S' such that $S \subset S'$.*

268 Intuitively, a siphon is a set of places that once unmarked remains so.
269 If it is conflict-free then its dual corresponds to a partial-state of the model
270 such that whatever update, the fixed values remain so (since the unmarked
271 places remain unmarked). This is precisely the definition of a trap space and
272 maximality of the siphon is equivalent to as many fixed values as possible,
273 hence minimality of the trap space. For example, the Boolean network given
274 in Example 2.1 has two trap spaces, $m_1 = 11$ and $m_2 = \star\star$. The Petri net
275 encoding of this Boolean network has five generic siphons, $S_1 = \emptyset$, $S_2 =$
276 $\{p_{x_1}, \bar{p}_{x_1}\}$, $S_3 = \{p_{x_2}, \bar{p}_{x_2}\}$, $S_4 = \{\bar{p}_{x_1}, \bar{p}_{x_2}\}$, 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 **Definition 3.2.** *Let m be a subspace of Boolean network $\mathcal{N} = (V, F)$. A*
282 *mirror of m is a set of places S in the Petri net encoding \mathcal{P} of \mathcal{N} such that:*

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

283 and

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

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

287 *Proof.* First, we show that if m is a trap space of \mathcal{N} , then S is a conflict-free
288 siphon of \mathcal{P} (*). If $D_m = \emptyset$, then $S = \emptyset$ is trivially a conflict-free siphon of
289 \mathcal{P} . Thus, we consider the case that $D_m \neq \emptyset$ (resp. $S \neq \emptyset$). Assume that S is

290 not a siphon of \mathcal{P} . Then, there is a transition $t \in T$ such that $S \cap \text{succ}(t) \neq \emptyset$
 291 but $S \cap \text{pred}(t) = \emptyset$. This implies that there is a place $p \in S$ such that
 292 $p \in \text{succ}(t)$ but $p \notin \text{pred}(t)$. Let v be the corresponding node in \mathcal{N} of p . By
 293 the characteristics of the encoding [22], there is a directional arc from t to p
 294 and a directional arc from the complementary place of p to t . Without loss
 295 of generality, we assume that $p = p_v$, then there is a directional arc from t
 296 to p_v and a directional arc from \bar{p}_v to t . We follow the following procedure
 297 to find a state $s \in \mathcal{S}_{\mathcal{N}}[m]$ such that $m_s(p') = 1, \forall p' \in \text{pred}(t)$ where m_s is
 298 the corresponding marking in \mathcal{P} of s . For every place $p' \in \text{pred}(t)$, let p'' be
 299 the complementary place of p' and v' be the corresponding node in \mathcal{N} of p'
 300 and p'' . If $p'' \notin S$, then $v' \notin D_m$ and we can always set a Boolean value to
 301 $s(v')$ such that $s \in \mathcal{S}_{\mathcal{N}}[m]$ and $m_s(p') = 1$. If $p'' \in S$, then $v' \in D_m$ and we
 302 set $s(v') = m(v')$. In this case, if $p' = p_v$ then $s(v') = m(v') = 1$ leading to
 303 $m_s(p') = 1$, if $p' = \bar{p}_v$ then $s(v') = m(v') = 0$ leading to $m_s(p') = 1$. For
 304 the remaining nodes of \mathcal{N} , we can always set Boolean values to these nodes
 305 to preserve that $s \in \mathcal{S}_{\mathcal{N}}[m]$. We also have $m_s(p_v) = 0$ by the characteristics
 306 of the encoding [22]. Now, t is enabled at marking m_s . Its firing leads to
 307 a new marking m'_s such that $m'_s(p_v) = 1$ and $m'_s(\bar{p}_v) = 0$. Let s' be the
 308 corresponding state in \mathcal{N} of m'_s . We have $s'(v) = 1$ because $m'_s(p_v) = 1$ and
 309 $m(v) = 0$ because $p_v \in S$. This implies that $s' \notin \mathcal{S}_{\mathcal{N}}[m]$. For any firing
 310 scheme of \mathcal{P} , the firing of t always happens. Since a firing scheme of \mathcal{P} is
 311 equivalent to an update scheme of \mathcal{N} , s can escape from the trap space m
 312 for any update scheme of \mathcal{N} , which contradicts to the property of a trap
 313 space. Hence, S is a siphon of \mathcal{P} . By the definition of a mirror, S is also a
 314 conflict-free one.

315 Second, we show that if S is a conflict-free siphon of \mathcal{P} , then m is a trap
 316 space of \mathcal{N} (**). By the definition of a mirror, m is a subspace of \mathcal{N} . Let
 317 s be an arbitrary state in $\mathcal{S}_{\mathcal{N}}[m]$ and m_s be its corresponding marking in
 318 \mathcal{P} . Assume that there is a place $p \in S$ such that $m_s(p) = 1$. Let v be the
 319 corresponding node in \mathcal{N} of p . Since $p \in S$, $v \in D_m$ and $m(v) = s(v)$. If
 320 $p = p_v$, then $m_s(p_v) = 1$ leading to $m(v) = s(v) = 1$ by the characteristics of
 321 the encoding [22]. By the definition of a mirror, $m(v) = 0$ because $p_v \in S$,
 322 which is a contradiction. It is symmetric for the case that $p = \bar{p}_v$. Hence,
 323 $m_s(p) = 0, \forall p \in S$. In any marking m'_s reachable from m_s regardless of the
 324 firing scheme of \mathcal{P} , we have $m'_s(p) = 0, \forall p \in S$ by the dynamical property on
 325 markings of a siphon [33]. Let s' be the corresponding state in \mathcal{N} of m'_s . For
 326 every node $v \in D_m$, we have all two cases as follows. Case 1: $p_v \in S$, then
 327 $m'_s(p_v) = 0$, thus $s'(v) = 0 = m(v)$. Case 2: $\bar{p}_v \in S$, then $m'_s(\bar{p}_v) = 0$, thus

328 $s'(v) = 1 = m(v)$. Hence, $s'(v) = m(v)$ for every $v \in D_m$. Then, $s' \in \mathcal{S}_\mathcal{N}[m]$.
 329 By the definition of a trap space and the arbitrariness of s , m is a trap space
 330 of \mathcal{N} .

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

332 From the proof of Theorem 3.1, we can see that this theorem still holds
 333 for any update scheme of the Boolean network. Since the Petri net encoding
 334 of a Boolean network is independent of its update scheme and siphons are
 335 a static property of a Petri net, we can imply that trap spaces of a Boolean
 336 network are independent of its update scheme. Note that the original proof
 337 for this property of trap spaces (see Theorem 1 of [7]) only considers the two
 338 popular update schemes (i.e., synchronous and fully asynchronous). This
 339 exhibits the very first theoretical application of the connection between trap
 340 spaces of Boolean networks and siphons of Petri nets.

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

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

352 Second, we show that if S is a maximal conflict-free siphon of \mathcal{P} , then
 353 m is a minimal trap space of \mathcal{N} (**). Since S is a conflict-free siphon of \mathcal{P} ,
 354 m is a trap space of \mathcal{N} by Theorem 3.1. Assume that m is not minimal.
 355 Then, there is another trap space m' such that $m' < m$. By the definition of
 356 the partial order $<$ on subspaces, $\mathcal{S}_\mathcal{N}[m'] \subset \mathcal{S}_\mathcal{N}[m]$. Let S' be the mirror of
 357 m' . S' is a conflict-free siphon by Theorem 3.1. Following the definition of
 358 a mirror, $S \subset S'$, which contradicts to the maximality of S . Hence, m is a
 359 minimal trap space of \mathcal{N} .

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

361 We here showcase a theoretical application of the connection between trap
 362 spaces in Boolean networks and conflict-free siphons in Petri nets. We use it

363 to prove a property of minimal trap spaces, which has surprisingly not been
 364 formally proved. Specifically, all minimal trap spaces of a Boolean network
 365 are mutually disjoint. This property is important because we can use it to
 366 approximate the set of attractors of the Boolean network [7].

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

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

374 Consider two any distinct minimal trap spaces m_1 and m_2 . Assume that
 375 $\mathcal{S}_{\mathcal{N}}[m_1] \cap \mathcal{S}_{\mathcal{N}}[m_2] \neq \emptyset$. Let S_1 and S_2 be the mirrors of m_1 and m_2 , re-
 376 spectively. By Theorem 3.2, S_1 and S_2 are maximal conflict-free siphons
 377 of \mathcal{P} . We have that $S = S_1 \cup S_2$ is also a siphon because of Proposi-
 378 tion 2.1. For every node $v \in V$, assume that $p_v \in S$ and $\bar{p}_v \in S$ hold.
 379 Since S_1 and S_2 are conflict-free, there are all two cases. Case 1: $p_v \in S_1$
 380 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
 381 $m_1(v) \neq m_2(v), m_1(v) \neq \star, m_2(v) \neq \star$, then $\mathcal{S}_{\mathcal{N}}[m_1] \cap \mathcal{S}_{\mathcal{N}}[m_2] = \emptyset$. This is a
 382 contradiction. Hence, for every node $v \in V$, $p_v \in S$ and $\bar{p}_v \in S$ cannot hold
 383 together. Therefore, S is conflict-free. Now, we have that S is a conflict-free
 384 siphon but $S_1 \subset S$ or $S_2 \subset S$ holds because $S_1 \neq S_2$. This contradicts to the
 385 maximality of S_1 and S_2 . Hence, $\mathcal{S}_{\mathcal{N}}[m_1] \cap \mathcal{S}_{\mathcal{N}}[m_2] = \emptyset$ holds.

386 □

387 One naturally computational application of Theorem 3.1 is that we can ef-
 388 ficiently decide whether a subspace m is a trap space. In **PyBoolNet** [19], this
 389 is checked by using the percolation on the prime-implicants of the Boolean
 390 functions. As we have mentioned at the beginning of this article, the compu-
 391 tation of prime-implicants is a demanding task for complex Boolean networks,
 392 even is sometimes intractable. Hence, the checking method in [19] shows its
 393 limitations. Instead, we can first compute the mirror S_m of m in the Petri
 394 net encoding. Then, by Proposition 2.1 and Theorem 3.1, we can check if
 395 $\text{pred}(S_m) \subseteq \text{succ}(S_m)$. Note that the Petri net construction is less com-
 396 putationally demanding than the prime-implicant computation because it
 397 only requires computing generic (not prime) implicants of the Boolean func-

398 tions [21]. In addition, the time complexity of the above checking method is
 399 quadratic in the number of transitions of the Petri net in worst cases.

400 Furthermore, by Theorem 3.2, we can reduce the problem of computing
 401 all minimal trap spaces of a Boolean network to the problem of computing
 402 all maximal conflict-free siphons of its Petri net encoding. Note that in the
 403 case of special types of trap spaces (e.g., fixed points), this can be put in
 404 regard to special types of siphons in Petri nets. See Subsection 4.5 for more
 405 discussions about many special types of trap spaces. It might actually be
 406 possible to generalize our result to any 1-safe place-complementary Petri net
 407 to define a notion of trap spaces that might be useful for the analysis of Petri
 408 nets, but this is out of the scope of this present article.

409 It is noted that there are no existing methods specifically designed for
 410 computing maximal conflict-free siphons (even maximal siphons) of a Petri
 411 net. The reason might be that researchers mainly focus on minimal generic
 412 siphons [33] in the field of Petri nets. Hence, we here propose several methods
 413 for computing maximal conflict-free siphons of a Petri net. The details of
 414 the proposed methods shall be given in the next section.

415 4. Computation methods

416 4.1. Characterization

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

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

423 The system of all the rules of the above form with respect to all pairs (p, t)
 424 where $p \in P, t \in T, t \in \text{pred}(p)$ fully characterizes all generic siphons of a
 425 Petri net and has been used with SAT solvers in [16, 17]. To make S to be
 426 a conflict-free siphon, we need to add to the system the rule

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

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

429 4.2. Constraint satisfaction problem

430 The following Boolean Constraint Satisfaction Problem (CSP) directly
431 derives from the above characterization:

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

- 434 • $R = P$, i.e., a variable is introduced for each place of \mathcal{P} ,
- 435 • $D(p) = \mathbb{B}$ for all $p \in R$, i.e., the variables are Boolean,
- 436 • $C = \{\neg p_v \vee \neg \bar{p}_v = 1 \mid \forall v \in V\} \wedge \{(p = 1 \rightarrow \bigvee_{p' \in \text{pred}(t)} p' = 1) \mid p \in$
437 $P, t \in \text{pred}(p)\}$.

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

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

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

440 *Proof.* By the former part $\neg p_v \vee \neg \bar{p}_v = 1$ of C , the conflict-freeness is imposed
441 because for any satisfiable valuation r , $r(p_v) = r(\bar{p}_v) = 1$ is impossible for all
442 $v \in V$. As shown in [17], the latter part of C can characterize the set of all
443 generic siphons of \mathcal{P} . Hence, we can conclude the proof.

444 □

445 In [17], the set of all siphons of a given Petri net is characterized by a sim-
446 ilar Boolean CSP except the conflict-freeness constraint. From the encoded
447 CSP, the set of all *minimal* siphons of the Petri net can be enumerated in the
448 set inclusion order. For enumerating siphons in the set inclusion order, the
449 proposed method by [17] uses the technique that labels directly the Boolean
450 variables with increasing value selection (i.e., to test first the absence, then
451 the presence of a place in the candidate solution). The method has two
452 implementations, one uses an iterated SAT procedure and the other uses
453 Constraint Programming (CP) with backtracking.

454 One natural question is that how to use the CSP-based method for enu-
455 merating all the maximal conflict-free siphons of a Petri net encoding a
456 Boolean network? Of course, the set of all conflict-free siphons of the Petri
457 net can easily be characterized by the CSP model presented in [17] along with
458 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 Chuffed⁶ [34] via its MiniZinc [35] 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 correction subset” blocking strategy, which will ensure set-inclusion maximality of the solutions. This is what is implemented in **Trappist** using the RC2 MaxSAT solver [36] available through the **python-sat** package⁷.

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.

⁶<https://github.com/chuffed/chuffed>

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

489 $\mathbf{n-v}$) to denote place p_v (resp. \bar{p}_v), $\forall v \in V$. The set of all atoms in \mathcal{L} is given
 490 as $\mathcal{A} = \bigcup_{v \in V} \{\mathbf{p-v}, \mathbf{n-v}\}$. For each pair (p, t) where $p \in P, t \in T, t \in \text{pred}(p)$,
 491 we translate the rule (1) into the ASP rule

$$\mathbf{a_1}; \dots ; \mathbf{a_k} :- \mathbf{a}.$$

492 where $\mathbf{a} \in \mathcal{A}$ is the atom representing place p and $\{\mathbf{a_1}, \dots, \mathbf{a_k}\} \subseteq \mathcal{A}$ is the
 493 set of atoms representing places in $\text{pred}(t)$. The rule (2) is translated into
 494 the ASP rule

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

495 for each $v \in V$. This ASP rule guarantees that two places representing
 496 the same node in \mathcal{N} never belong to the same siphon of \mathcal{P} , representing
 497 the conflict-freeness. Naturally, a Herbrand model (see, e.g., [37]) of \mathcal{L} is
 498 equivalent to a conflict-free siphon of \mathcal{P} . To guarantee that a Herbrand
 499 model is also a stable model (an answer set), we need to add to \mathcal{L} the two
 500 choice rules

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

501 for each $v \in V$. Note that the number of atoms of \mathcal{L} is only $2n$, whereas
 502 the ASP encoding shown in [7] has as many atoms as the number of prime-
 503 implicants of the Boolean network and that number might be exponential in
 504 n . In [8], there is an ASP characterization of trap spaces that does not rely
 505 on minimal DNFs either and thus seems very similar to our ASP encoding.
 506 Remarkably it only requires the DNF for the *activation* part, using the in-
 507 formation that it will only be used for locally-monotonic Boolean networks.
 508 We would therefore expect that, when available, it will have comparable per-
 509 formance on the ASP part (the ASP program would be approximately twice
 510 smaller, though redundancy is not always bad in that field), but can also
 511 avoid combinatorial explosion of the Petri net encoding for some formula
 512 where the activation DNF is simple but the inhibition is not. Since **mpbn** is
 513 included in our benchmark this will be evaluated in our experiments.

514 Now, a solution (simply an answer set) $A \subseteq \mathcal{A}$ of \mathcal{L} is equivalent to a
 515 conflict-free siphon S of \mathcal{P} , thus a trap space m of \mathcal{N} . The conversion from A
 516 to m is straightforward. If $\mathbf{p-v} \in A$ then $v \in D_m$ and $m(v) = 0$. Conversely,
 517 if $\mathbf{n-v} \in A$ then $v \in D_m$ and $m(v) = 1$. Otherwise, $v \notin D_m$. Comput-
 518 ing multiple answer sets is built into ASP solvers and the solving collection
 519 **POTASSCO** [37] also features the option to find set-inclusion maximal answer
 520 sets with respect to the set of atoms. Naturally, a set-inclusion maximal

521 answer set of \mathcal{L} is equivalent to a maximal conflict-free siphon of \mathcal{P} , thus a
 522 minimal trap space of \mathcal{N} . By using this built-in option, we can compute all
 523 the set-inclusion maximal answer sets of \mathcal{L} (resp. all the minimal trap spaces
 524 of \mathcal{N}) in one execution.

525 4.4. Integer linear programming-based method

526 We first show how an Integer Linear Programming (ILP) \mathcal{I} can define
 527 a set of all conflict-free siphons of the encoded Petri net \mathcal{P} . We introduce
 528 *binary* variable $\mathbf{p-v}$ (resp. $\mathbf{n-v}$) to denote place p_v (resp. \bar{p}_v), $\forall v \in V$. The
 529 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
 530 $p \in P, t \in T, t \in \text{pred}(p)$, we translate the rule (1) into the ILP inequality

$$\mathbf{a} \leq \mathbf{a_1} + \dots + \mathbf{a_k}$$

531 where \mathbf{a} is the binary variable representing place p and $\{\mathbf{a_1}, \dots, \mathbf{a_k}\}$ is the
 532 set of binary variable representing places in $\text{pred}(t)$. The rule (2) is translated
 533 into the ILP inequality

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

534 for each $v \in V$. This inequality forbids both $\mathbf{p-v}$ and $\mathbf{n-p}$ receive the value
 535 1, thus representing the conflict-freeness. Since we only consider feasible
 536 solutions, the objective function is set to $\max \mathbf{p-v}$ for some $v \in V$. Naturally,
 537 a solution I of \mathcal{I} is equivalent to a conflict-free siphon S of \mathcal{P} . The conversion
 538 is that

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

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

540 We can see the similarity between \mathcal{I} and the encoded ASP shown in the
 541 previous subsection. However, due to the nature of solutions of an ILP, it is
 542 hard to compute all the set-inclusion maximal solutions of \mathcal{I} in one execution
 543 of an ILP solver. Hence, we propose an iterative approach as follows.

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

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

546 Now, \mathcal{I} can be solved using a general purpose ILP solver. If it admits any so-
 547 lution I^* , the corresponding conflict-free siphon (say S^*) is maximal. Hence,
 548 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

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

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

Since we used the MiniZinc framework to interface with the CP solver, it was simple to make the slight modifications described above and use that same interface to call the Coin-OR CBC solver⁸ [38].

4.5. Computation of special types of trap spaces

In the field of systems biology, biologists may want to compute more special types of trap spaces beyond minimal trap spaces [19]. We shall show that our proposed methods can be easily adjusted to compute popular types of trap spaces. We illustrate the adjustments via the ASP-based method (see Subsection 4.3), but these adjustments are completely applicable for other approaches such as MaxSAT, CP, and ILP.

First, the work by [39] uses the concept of stable motifs to build the succession diagram of a Boolean network, a summary of the decisions in the network dynamics that lead to successively more restrictive nested stable motifs. The succession diagram is useful for control and decision making on this Boolean network. In particular, the proposed control methods are independent to the update scheme. It has been shown that a stable motif of a Boolean network is equivalent to a maximal trap space of this Boolean network [39]. Hence, it is necessary to develop an efficient method for computing

⁸<https://github.com/coin-or/Cbc>

581 maximal trap spaces of a Boolean network. We shall show how to adjust the
 582 ASP-method presented in Subsection 4.3 to compute maximal trap spaces.

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

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

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

598 Second, we consider fixed points in Boolean networks. Let s be a fixed
 599 point of a Boolean network \mathcal{N} . We have a subspace m corresponding to s
 600 as follows: $\forall v \in V, m(v) = s(v)$, i.e., all nodes are fixed in m . Clearly, s is
 601 a trap set of \mathcal{N} regardless of the update scheme. Hence, m is a trap space
 602 of \mathcal{N} . In addition, since $|S_{\mathcal{N}}[m]| = 1$, m is also a minimal trap space. To
 603 compute all fixed points of \mathcal{N} , we can add more constraints to the encoded
 604 ASP characterizing all conflict-free siphons (equivalently trap spaces). For
 605 every $v \in V$, we add to the encoded ASP the rule

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

606 that ensures that for every conflict-free siphon S , it contains either **p-v** or **n-v**
 607 for every $v \in V$. Equivalently, the trap space corresponding to S is always
 608 a fixed point. Now, the set of answer sets of the encoded ASP is equivalent
 609 to the set of fixed points of \mathcal{N} . In particular, when solving the encoded ASP
 610 using an ASP solver, we do not need to use the built-in option for computing
 611 set-inclusion maximal answer sets. Note that we can also build another ASP
 612 characterizing all fixed points of \mathcal{N} based on the equivalence between a fixed
 613 point of \mathcal{N} and a deadlock of its Petri net encoding [21]. This approach may
 614 give a more compact ASP.

615 Third, we consider the trap spaces intersecting a given subspace m^* of
 616 a Boolean network. A trap space m intersects m^* if and only if $S_{\mathcal{N}}[m] \cap$
 617 $S_{\mathcal{N}}[m^*] \neq \emptyset$. It follows that for every v , if $m^*(v) = 0$ then $m(v) = 0$ or
 618 $m(v) = \star$, if $m^*(v) = 1$ then $m(v) = 1$ or $m(v) = \star$. For the former case, we
 619 add to \mathcal{L} the ASP rule

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

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

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

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

624 Finally, we consider the trap spaces that are inside a given subspace m^*
 625 of a Boolean network. We first adjust \mathcal{L} to characterize all such trap spaces.
 626 A trap space m is inside m^* if and only if $m(v) = m^*(v)$ for every $v \in D_{m^*}$.
 627 If $m^*(v) = 0$, we add to \mathcal{L} the ASP rule

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

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

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

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

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

633 where $\{v_{i1}, \dots, v_{ik}\}$ is the set of free nodes of m^* . This rule ensures that
 634 $m \neq m^*$. In the case that $m^* = \varepsilon$, we have all maximal trap spaces of the
 635 original Boolean network.

636 5. Motivating example

637 For a few years now we have been collaborating with biologists who build
 638 very large detailed and annotated maps and now wish to analyze the dy-
 639 namics of the corresponding models. One of the main maps studied this way
 640 represents knowledge about the Rheumatoid Arthritis [40], and was the main

641 motivation for the development of a tool to automatically transform it into
642 an executable Boolean network [6]. In the supplementary material of the pa-
643 per, an excerpt of the map, focused around the apoptosis (cell death) module
644 is transformed into a model of *reasonable* size, namely 180 Boolean variables
645 (model `F5_RA_apoptosis_executable_module.sbml` of supplementary ma-
646 terial S3, and model “RA-apoptosis” of Section 6). The study of such model,
647 though, is a big hurdle. Indeed, as stated in the article about another model
648 of the same size: “*The size of the CaSQ-inferred MAPK model (181 nodes)*
649 *made the calculation of stable states a non-realistic endeavour.*”

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

657 However, using a state-of-the-art tool like `PyBoolNet` [7] on that model
658 actually fails at the phase of prime-implicant generation. `mpbn` [9] can return
659 the first 1000 solution within 1.43s, but indeed, it limits the modeling range
660 of the modelers as it does not permit using non-locally-monotonic Boolean
661 functions. This is also true for the Alzheimer model also mentioned in that
662 same article and originally from [41] (F4 file in the original supplementary
663 material, and “Alzheimer” in Table 3), where `PyBoolNet` also fails at the
664 prime-implicant computation and `mpbn` does not give any answer because
665 this model is actually non-locally-monotonic. The current practice usually
666 revolves then around fixing some source nodes to plausible values and re-
667 ducing the model accordingly. While this approach makes sense, it relies
668 on potentially arbitrary decisions, and *hides away* critical modelling choices
669 that were actually not part of the original Boolean network or even of the
670 starting map.

671 Using the ASP-based method presented in Section 4.3, it is possible to
672 obtain the first 1000 minimal trap spaces (including ones that contain more
673 than one state) within 0.19s, which is much quicker than `mpbn`. Unfortu-
674 nately since this was not available at the time, the analysis of the model
675 remained very high-level and qualitative, instead of being able to use the
676 rich information of computed minimal trap spaces.

677 6. Evaluation

678 To evaluate the performance of the newly proposed methods (imple-
 679 mented as a Python package named **Trappist**) and the state-of-the-art meth-
 680 ods (**bioLQM**⁹, **PyBoolNet** [7, 19], and **mpbn** [9]), we compared them on both
 681 **PyBoolNet**’s own model repository and many real-world models from various
 682 sources in the literature. It is worth noting that **mpbn** [9] only handles locally-
 683 monotonic models, whereas the other methods can handle general models.
 684 To obtain a more comprehensive comparison, we also used random models
 685 generated by a third-party software (i.e., **BoolNet R** package [29]). As ex-
 686 plained in Section 5, in our benchmarks, we only searched for the first 1000
 687 minimal trap spaces for each model. It is worth noting that unlike existing
 688 analysis shown in the literature, we did not fix specific values for source nodes
 689 in all the considered models.

690 To solve the ASP problems, we used the same ASP solver **Clingo** [37] and
 691 the same configuration as that used in **PyBoolNet** [7, 19] and **mpbn** [9]. Specif-
 692 ically, we used the configuration `-heuristic=Domain -enum-mod=domRec`
 693 `-dom-mod=3` (subset maximality, equivalent to the deprecated `--dom-pref=32`
 694 `--heuristic=domain --dom-mod=7` used by **PyBoolNet**). We ran all the
 695 benchmarks on a machine whose environment is CPU: Intel® Core™ i9-
 696 11950H 2.60GHz \times 16, 16 GB DDR4 RAM, Ubuntu 20.04.5 LTS. Finally,
 697 we set a time limit of three minutes for each model.

698 All the models and a Jupyter notebook realizing the benchmarks can be
 699 found at <https://github.com/soli/trap-spaces-as-siphons>. These can
 700 be run on a Docker image in the cloud by clicking the “Binder” button.

701 6.1. *PyBoolNet* repository

702 Table 1 shows the experimental results on the models from the official
 703 **PyBoolNet** repository¹⁰. Column n denotes the number of nodes of each
 704 model. Column $|M|$ denotes the number of minimal trap spaces and for each
 705 method is given the computation time in seconds, asking only for the first
 706 1000 trap spaces. In the case of **bioLQM**, “N/A” means that the number
 707 of all minimal trap spaces of the model is larger than 1000 and we did not
 708 recorded the running time of **bioLQM** because it always requires to compute
 709 all minimal trap spaces. A number in bold indicates a ratio greater than

⁹<http://colomoto.org/biolqm/doc/tools-trap-space.html>

¹⁰<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	0.13	0.01	0.00	0.00	-	-	0.01
2 calzone_cellfate	28	27	0.12	0.02	0.01	0.01	-	-	0.01
3 dahlhaus_neuroplastoma	23	32	0.11	0.03	0.01	0.01	-	-	0.01
4 davidich_yeast	10	12	0.11	0.02	0.01	0.01	-	-	0.01
5 dinwoodie_life	15	7	0.11	0.01	0.00	0.01	-	-	0.01
6 dinwoodie_stomatal	13	1	0.10	0.01	0.00	0.00	-	-	0.01
7 faure_cellcycle	10	2	0.11	0.02	0.01	0.01	-	-	0.01
8 grieco_mapk	53	18	0.19	0.03	0.02	0.03	-	-	0.02
9 irons_yeast	18	1	0.12	0.03	0.01	0.01	-	-	0.02
10 jaoude_thdiff	103	> 1000	N/A	0.85	0.45	0.56	-	-	0.09
11 klamt_tcr	40	8	0.11	0.01	0.01	0.01	-	-	0.02
12 krumsiek_myeloid	11	6	0.10	0.01	0.00	0.00	-	-	0.01
13 multivalued	13	4	0.10	0.01	0.00	0.00	-	-	0.01
14 n12c5	11	5	0.11	17.83	0.01	0.01	-	-	0.01
15 n3s1c1a	2	2	0.10	0.01	0.00	0.00	-	-	0.01
16 n3s1c1b	2	2	0.09	0.02	0.00	0.00	-	-	0.01
17 n5s3	4	3	0.10	0.02	NM	0.00	-	-	0.01
18 n6s1c2	5	3	0.10	0.02	0.00	0.00	-	-	0.01
19 n7s3	6	3	0.11	0.02	0.00	0.00	-	-	0.01
20 raf	3	2	0.10	0.01	0.00	0.00	-	-	0.01
21 randomnet_n15k3	15	3	0.10	0.02	NM	0.01	-	-	0.01
22 randomnet_n7k3	7	10	0.10	0.01	NM	0.00	-	-	0.01
23 remy_tumorigenesis	34	25	0.15	0.94	0.02	0.02	-	-	0.02
24 saadatpour_guardcell	13	1	0.10	0.06	0.00	0.00	-	-	0.02
25 selvaggio_emt	56	> 1000	N/A	0.48	0.28	0.28	-	-	0.09
26 tournier_apoptosis	12	3	0.10	0.01	0.00	0.00	-	-	0.01
27 xiao_wnt5a	7	4	0.10	0.01	0.00	0.00	-	-	0.01
28 zhang_tlg1	60	156	0.60	0.09	0.09	0.07	-	-	0.04
29 zhang_tlg1_v2	60	258	0.64	0.04	0.08	0.11	-	-	0.04

three compared to the best result. “NM” indicates a non-locally-monotonic model. There are four variants of **Trappist**: SAT (i.e., the MaxSAT-based method shown in Subsection 4.2), CP (i.e., the CP-based method shown in Subsection 4.2), ILP (i.e., the ILP-based method shown in Subsection 4.4), and ASP (i.e., the ASP-based method shown in Subsection 4.3).

As shown in Table 1, for most of the models of the **PyBoolNet** reposi-

tory, the results are comparable with all minimal trap spaces found very fast. For 5 of the 29 models, `mpbn` did not give any answer because it recognized these models as not locally-monotonic. Note that on some very small models, `Trappist` is sometimes slower than `PyBoolNet` and/or `mpbn`, but still significantly under one second. On the contrary, on every model that was a bit challenging for `PyBoolNet` or `mpbn`, the new method is far more efficient with speedups between one and two orders of magnitude.

6.2. *BBM repository*

Currently, a research group has made a great effort for building a collection (called **BBM**) 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. It is released and maintained at <https://github.com/sybila/biodivine-boolean-models>. We here tested all the compared methods on this model repository.

Table 2: Results on the real-world models from the BBM repository.

Method	# failures	avg-lqm (s)	avg-mono (s)	avg-all (s)
<code>bioLQM</code>	9 (134)	12.87	N/A	N/A
<code>PyBoolNet</code>	12	8.87	11.00	13.59
<code>mpbn</code>	2 (187)	N/A	2.31	N/A
<code>Trappist-MaxSAT</code>	1	0.03	1.09	1.01
<code>Trappist-CP</code>	-	-	-	-
<code>Trappist-ILP</code>	-	-	-	-
<code>Trappist-ASP</code>	1	0.05	1.02	0.93

Table 2 shows the experimental results on the 211 real-world models from the **BBM** repository. Column 2 expresses the numbers of failures (i.e., did not finish the computation within a time limit of three minutes) of each method. For the case of `bioLQM`, we only considered the models that have at most 1000 minimal trap spaces. The number of such models is 134 (per all 211 models) and is denoted inside the parentheses. For the case of `mpbn`, we only considered the models that are locally-monotonic. The number of such models is 187 (per all 211 models) and is denoted inside the parentheses. Columns 3-5 express the average running time (in seconds) of each method for the models having at most 1000 minimal trap spaces, the locally-monotonic

740 models, and all the models, respectively. Note that when computing the
 741 average running time, if the running time exceeds 180s, it is considered as
 742 180s. From the results shown in Table 2, we reported several observations as
 743 follows.

744 6.3. Selected models

745 We used a set of real-world Boolean networks lying in various scales col-
 746 lected from numerous bibliographic sources. Most of these models are quite
 747 big (in size), complex (i.e., having high average in-degree, which is related to
 748 the number of prime-implicants) and have never been fully analyzed. Note
 749 that these models are not included in the `PyBoolNet` and `BBM` repositories. We
 750 then applied `bioLQM`, `PyBoolNet`, `mpbn`, and the four variants of `Trappist` to
 751 computing minimal trap spaces of these real-world models. Table 3 shows the
 752 obtained experimental results. “DNF” means that the method did not finish
 753 the computation (stopping at the first 1000 minimal trap spaces) within the
 754 timeout of two minutes. A number in bold indicates a ratio greater than or
 755 equal to 10 compared to the best result. The remaining notations are similar
 756 to those in Table 1. Hereafter, we analyze in detail the results with respect
 757 to minimal trap space computation.

758 The first observation is that for 26 of the 33 models (more than 78%),
 759 `mpbn` did not give any answer because it recognized that these models as
 760 not locally-monotonic. For 6 of the 33 models where `mpbn` returned the
 761 answers, `mpbn` and `Trappist` are comparable in computation time, though
 762 surprisingly `mpbn` appears a bit slower on average. Note however that `mpbn`
 763 was the only tool to provide a solution for the SN-5 model, thus confirming
 764 that if the activation function is in the right form, not having to compute the
 765 inactivation function’s disjunctive normal form can render a difficult problem
 766 tractable. However, since `mbpn` can handle only locally-monotonic models
 767 and `Trappist` can handle general models, it is difficult to further compare
 768 between them. Hence, we focus on only comparisons between `PyBoolNet`
 769 and `Trappist` in the following observations.

770 The second observation is that the proposed method vastly outperforms
 771 `PyBoolNet` in computational time, on each and every model, and sometimes
 772 with orders of magnitude of difference (e.g., for most models in the 100–1000
 773 nodes size range). Note that for all the cases where `PyBoolNet` did not man-
 774 age to finish before the timeout, as marked by “DNF” in Table 3, the timeout
 775 occurred during the computation of the prime-implicants. Hence, not even
 776 a single minimal trap space was output by that method. The computational

Table 3: Timing comparisons (in seconds) between bioLQM (LQM), PyBoolNet (PBN), mpbn and the four variants of Trappist on selected models from the literature.

model	n	$ M $	LQM	PBN	mpbn	Trappist			
						SAT	CP	ILP	ASP
1 metastatic [42]	10	4	0.10	0.04	NM	0.01	-	-	0.02
2 Arabidopsis_thaliana [42]	15	8	0.10	0.06	NM	0.01	-	-	0.02
3 p53_high_dna [42]	16	1	0.38	1.76	NM	0.08	-	-	0.14
4 p53_low_dna [42]	16	1	0.41	1.76	NM	0.07	-	-	0.14
5 FT-GRN [43]	23	32	DNF	DNF	NM	0.03	-	-	0.19
6 DNA_damage [42]	26	16	0.24	0.33	NM	0.02	-	-	0.05
7 Rho-GTPases [42]	33	2	0.17	0.57	40.39	0.07	-	-	0.11
8 Pluripotency [44]	36	440	DNF	DNF	NM	0.16	-	-	0.28
9 Pluripotent [42]	36	276	0.37	0.43	NM	0.07	-	-	0.06
10 Pancreatic_Cancer [42]	43	> 1000	N/A	0.11	0.36	0.17	-	-	0.06
11 Drosophila [45]	52	128	0.33	0.05	0.07	0.06	-	-	0.05
12 Cacace_TdevModel [46]	61	28	1.29	5.67	NM	0.06	-	-	0.08
13 hedgehog [42]	65	> 1000	N/A	DNF	0.50	0.34	-	-	0.33
14 EMT [39]	69	268	39.22	1.01	0.20	0.12	-	-	0.05
15 Bcell [47]	73	72	0.23	0.04	0.08	0.06	-	-	0.05
16 mast_cell [6]	73	> 1000	N/A	0.09	0.55	0.37	-	-	0.15
17 Corral_ThIL17diff [48]	92	> 1000	N/A	107.57	0.76	0.56	-	-	0.16
18 Adhesion_CIP [49]	121	78	56.81	4.25	0.23	0.17	-	-	0.19
19 EMT_Mech [50]	136	82	DNF	14.01	0.27	0.20	-	-	0.25
20 macrophage [42]	136	> 1000	N/A	0.54	1.09	0.84	-	-	0.27
21 angiogenesis [42]	141	> 1000	N/A	0.16	1.07	1.06	-	-	0.16
22 angiofull [51]	142	> 1000	N/A	0.17	1.06	0.88	-	-	0.23
23 EMT_Mech_TGFbeta [50]	150	492	DNF	11.28	0.78	0.69	-	-	0.35
24 RA_apoptosis [6]	180	> 1000	N/A	DNF	1.43	1.55	-	-	0.19
25 MAPK [6]	181	> 1000	N/A	13.58	1.76	1.51	-	-	0.27
26 Snf1-pathway [52]	202	> 1000	N/A	1.13	1.47	1.43	-	-	0.31
27 T-cell-co-receptor [42]	206	> 1000	N/A	DNF	1.52	2.26	-	-	0.35
28 TcellCheckPoint [53]	218	> 1000	N/A	4.99	NM	1.96	-	-	0.28
29 Mycobacterium [42]	317	> 1000	N/A	0.42	2.36	4.91	-	-	0.44
30 Leishmania [42]	342	> 1000	N/A	DNF	2.56	5.62	-	-	0.46
31 Cholocystokinin [6]	383	> 1000	N/A	0.36	2.99	4.81	-	-	0.37
32 Alzheimer [6]	762	> 1000	N/A	DNF	NM	18.21	-	-	0.79

777 advantage is therefore immediately a practical advantage since on the one
778 hand the state-of-the-art method did not allow any analysis whatsoever of
779 the models, and on the other hand the proposed method could provide, very
780 often under one second, the first thousand minimal trap spaces. For mod-
781 ellers having a critical look at a model and in a *model, invalidate, refine* loop

782 this means a huge difference in the models that are amenable to study.

783 Note that even with a very restricted time-limit of two minutes, it was
784 possible with the proposed technique to find *all* minimal trap spaces of small
785 models (roughly under 130 nodes, i.e., considered as quite big up to now).
786 Though it might seem impractical to handle tens of thousands of such pos-
787 sible complex attractors in a manual way, i.e., to compare them to specific
788 experimental conditions and corresponding data, we hope that an automatic
789 analysis of such attractors might become possible with systematic verification
790 methods, not unlike that described in [53]. Since the ASP code is declarative
791 by nature, it is also possible to add to it supplementary constraints coming
792 from the modeler in case one is looking for specific attractors. Finally, sam-
793 pling from the ASP-generated solutions as is done in [54] would allow for a
794 different type of exploration.

795 The third observation is that for all the models where **PyBoolNet** finished
796 before the timeout, once **PyBoolNet** went through the prime-implicant phase,
797 its ASP solving phase quickly returned the first 1000 minimal trap spaces, all
798 under one second. For these models, the ASP solving phase of the proposed
799 method also took very short time, all under one second. Hence, with the
800 experimental results shown in this paper, the practical differences between
801 our ASP encoding and that of **PyBoolNet** are not distinctly exposed. The
802 fact that our new ASP encoding is guaranteed to be linear in the number
803 of nodes of the original model does not seem to be crucial here, however a
804 much deeper analysis of those cases remains to be done.

805 Note that though enumerating the extremal siphons of a Petri net is ex-
806 ponential (see [17] for instance) this is apparently not the bottleneck of the
807 proposed method, showing once again that networks obtained from biochem-
808 ical models do have a specific structure.

809 6.4. Randomly generated models

810 We randomly generated a set of N-K models [1] with network size n in the
811 set $\{100, 150, 200, 250, 300, 350, 400\}$ and $K = 3$ (i.e., each node has exactly
812 three input nodes). We chose N-K models because they are a useful tool for
813 studying the dynamics of Boolean networks [1, 7]. For each network size, 50
814 instances were generated using the **generateRandomNKNetwork** function. In
815 total, we have 350 random models. We then applied the compared methods
816 to these models and recorded the numbers of failures (i.e., failed to obtain
817 the result within a time limit of three minutes) as well as the average running
818 time (inside the parentheses) in each method for each network size n . It is

819 worth noting that N-K models usually have small numbers of minimal trap
820 spaces [7]. Hence, we searched for all solutions in each model, which makes
821 the comparison to **bioLQM** more comprehensive. In addition, each node has
822 only three input nodes, i.e., the number of prime-implicants of the associated
823 Boolean function is small. Hence, **PyBoolNet** always passed the phase of
824 computing prime-implicants in every model even within 1s, which enables us
825 to compare the ASP encoding of **PyBoolNet** and that of **Trappist**.

Table 4: Results on N-K models.

n	LQM	mpbn	PBN	Trappist			
				SAT	CP	ILP	ASP
100	50 (> 180)	50 (N/A)	0 (0.07)	0 (0.05)	- ()	- ()	0 (0.09)
150	50 (> 180)	50 (N/A)	0 (0.14)	0 (0.10)	- ()	- ()	0 (0.14)
200	50 (> 180)	50 (N/A)	0 (0.43)	0 (0.25)	- ()	- ()	0 (0.24)
250	50 (> 180)	50 (N/A)	0 (1.92)	0 (1.04)	- ()	- ()	0 (0.56)
300	50 (> 180)	50 (N/A)	0 (9.68)	0 (4.46)	- ()	- ()	0 (1.83)
350	50 (> 180)	50 (N/A)	1 (46.54)	0 (20.09)	- ()	- ()	0 (6.10)
400	50 (> 180)	50 (N/A)	29 (144.09)	12 (90.36)	- ()	- ()	1 (33.01)

826 Table 4 shows the experimental results on N-K models. Column n de-
827 notes the network size. Columns LQM and PBN show the results of **bioLQM** and
828 **PyBoolNet**, respectively. For each method, the number outside the parenthe-
829 ses indicates the number of failures, whereas the number inside the paren-
830 theses indicates the average running time (in seconds). Note that when
831 computing the average running time, if the running time exceeds 180s, it
832 is considered as 180s. From these results, we obtained several observations
833 consistent with those obtained for real-world models.

834 TODO: ...

835 First, **mpbn** did not be able to handle any model because all the models
836 are non-locally-monotonic. Recall that a Boolean network is non-locally-
837 monotonic if only one of its Boolean functions is non-locally-monotonic.
838 Hence, it is apparent that all the randomly generated models are non-locally-
839 monotonic. This observation confirms the limit on the applicable model class
840 of **mpbn**.

841 Second, surprisingly **bioLQM** could not handle any model. One of the
842 reason may be that the BDD characterizing all trap spaces is too large, and

843 its computation is slow. It is apparent because the network size is large
844 (≥ 100) and the Boolean functions are not simple.

845 Third, **PyBoolNet** could handle every model of network size less than
846 or equal to 300. It only failed in one model of network size 350 but 29
847 models of network size 400. The average running time vastly increases as the
848 network size increases. As compared to the four methods of our approach,
849 the performance of **PyBoolNet** is comparable for the 100-node and 200-node
850 models. However, from $n = 250$, the performance difference is exhibited
851 more clearly.

852 Finally, ...

853 7. Conclusion

854 In this article we have explored and proved for the first time the equiva-
855 lence between (minimal) trap spaces of a general Boolean network and (max-
856 imal) conflict-free siphons of its Petri net encoding. We have shown several
857 important applications of this finding to studying properties of trap spaces
858 in Boolean networks. As an important practical application of the equiva-
859 lence, we have proposed a new approach for the computation of minimal trap
860 spaces in Boolean networks, based on the enumeration of maximal conflict-
861 free siphons of Petri nets. We have also proposed the four possible methods
862 using MaxSAT, CP, ILP, and ASP for implementing the new approach. The
863 proposed methods have been evaluated on many real-world models from the
864 literature as well as randomly generated models. The experimental results
865 show that the new approach vastly outperforms all the state-of-the-art meth-
866 ods in terms of general Boolean networks and is comparable to the **mpbn**
867 method even better in average in terms of locally-monotonic Boolean net-
868 works. We believe that this opens up the way to a much better analysis
869 of large Boolean networks, which is needed with the advent of automatic
870 model-generation pipelines [55].

871 Although the experimental results show the superiority of our approach
872 to **mpbn** in general, we however note that there is a model in the **BBM** repos-
873 itory (with identifier 122) where all the four proposed methods for the new
874 approach did not manage to finish the Petri net conversion before the time-
875 out, whereas **mpbn** can still handle this model. The model is not very large
876 but its Boolean functions are rather complicated. This points to the fact that
877 our current choice of using a BDD-based translation to obtain that Petri net
878 encoding, though it provides a small/efficient ASP might be too costly to

879 handle the complex models. In such a case, a more *naive* encoding might
 880 provide a much larger ASP program, with many redundant rules, but eas-
 881 ier/faster to obtain. The evaluation of the feasibility of such strategy, and
 882 of its impact on smaller instances, remains to be done. Recognizing that
 883 a model is locally-monotonic and applying in that specific case dedicated
 884 strategies as those of `mpbn` might also be a partial solution.

885 It is worth noting that there may be possibly other methods for comput-
 886 ing minimal/maximal conflict-free siphons in Petri nets, like the methods for
 887 generic siphon computation in the field of Petri nets (see [33] for a survey
 888 about these methods). Although these approaches do not directly support
 889 the minimal/maximal conflict-free siphon computation now, we plan to in-
 890 vestigate them in the future. They could replace our proposed methods if
 891 they give significantly better performance. However, the current methods
 892 appear to already perform very well even on the biggest models we have
 893 considered.

894 Finally, we think that the links between Petri nets and Boolean networks
 895 that we stumbled upon in this method might have deeper roots. Exploring
 896 those connections might lead both to interesting topics of research for Petri
 897 nets, like a notion of trap-spaces, and for Boolean networks. We also believe
 898 that the connection between trap spaces of Boolean networks and siphons
 899 of Petri nets can be a very useful tool for exploring and proving more new
 900 properties of trap spaces in Boolean networks, as we have used it to success-
 901 fully prove the separation of minimal trap spaces. Diving into this direction
 902 is one of our future work.

903 References

- 904 [1] L. Glass, S. A. Kauffman, The logical analysis of continuous, non-linear
 905 biochemical control networks, J. Theor. Biol. 39 (1973) 103–129.
- 906 [2] R. Thomas, Boolean formalisation of genetic control circuits, J. Theor.
 907 Biol. 42 (1973) 565–583.
- 908 [3] R. Thomas, R. d’Ari, Biological feedback, CRC press, 1990.
- 909 [4] R. Thomas, Regulatory networks seen as asynchronous automata: a
 910 logical description, J. Theor. Biol. 153 (1991) 1–23.

- 911 [5] R.-S. Wang, A. Saadatpour, R. Albert, Boolean modeling in systems
912 biology: an overview of methodology and applications, *Phys. Biol.* 9
913 (2012) 055001.
- 914 [6] S. S. Aghamiri, V. Singh, A. Naldi, T. Helikar, S. Soliman, A. Niarakis,
915 J. Xu, Automated inference of Boolean models from molecular interac-
916 tion maps using CaSQ, *Bioinform.* 36 (2020) 4473–4482.
- 917 [7] H. Klarner, A. Bockmayr, H. Siebert, Computing maximal and minimal
918 trap spaces of Boolean networks, *Nat. Comput.* 14 (2015) 535–544.
- 919 [8] S. Chevalier, C. Froidevaux, L. Paulevé, A. Y. Zinovyev, Synthesis of
920 Boolean networks from biological dynamical constraints using answer-
921 set programming, in: *International Conference on Tools with Artificial*
922 *Intelligence*, IEEE, 2019, pp. 34–41.
- 923 [9] L. Paulevé, J. Kolčák, T. Chatain, S. Haar, Reconciling qualitative,
924 abstract, and scalable modeling of biological networks, *Nat. Commun.*
925 11 (2020) 1–7.
- 926 [10] M. Noual, D. Regnault, S. Sené, About non-monotony in Boolean au-
927 tomata networks, *Theor. Comput. Sci.* 504 (2013) 12–25.
- 928 [11] J. L. Peterson, *Petri net theory and the modeling of systems*, Prentice
929 Hall PTR, 1981.
- 930 [12] T. Murata, Petri nets: Properties, analysis and applications, *Proc.*
931 *IEEE* 77 (1989) 541–580.
- 932 [13] V. N. Reddy, M. L. Mavrovouniotis, M. N. Liebman, Petri net rep-
933 resentations in metabolic pathways, in: *International Conference on*
934 *Intelligent Systems for Molecular Biology*, AAAI, 1993, pp. 328–336.
- 935 [14] I. Zevedei-Oancea, S. Schuster, Topological analysis of metabolic net-
936 works based on Petri net theory, *Silico Biol.* 3 (2003) 323–345.
- 937 [15] M. A. Blätke, M. Heiner, W. Marwan, Biomodel engineering with Petri
938 nets, in: *Algebraic and Discrete Mathematical Methods for Modern*
939 *Biology*, Elsevier, 2015, pp. 141–192.

- 940 [16] O. Oanea, H. Wimmel, K. Wolf, New algorithms for deciding the siphon-
941 trap property, in: International Conference on Applications and Theory
942 of Petri Nets, Springer, 2010, pp. 267–286.
- 943 [17] F. Nabli, T. Martinez, F. Fages, S. Soliman, On enumerating mini-
944 mal siphons in Petri nets using CLP and SAT solvers: theoretical and
945 practical complexity, *Constraints An Int. J.* 21 (2016) 251–276.
- 946 [18] V. Trinh, B. Benhamou, K. Hiraishi, S. Soliman, Minimal trap spaces of
947 logical models are maximal siphons of their Petri net encoding, in: In-
948 ternational Conference on Computational Methods in Systems Biology,
949 Springer, 2022, pp. 158–176.
- 950 [19] H. Klarner, A. Streck, H. Siebert, PyBoolNet: a python package for the
951 generation, analysis and visualization of Boolean networks, *Bioinform.*
952 33 (2017) 770–772.
- 953 [20] L. C. Fontanals, E. Tonello, H. Siebert, Control strategy identification
954 via trap spaces in Boolean networks, in: International Conference on
955 Computational Methods in Systems Biology, Springer, 2020, pp. 159–
956 175.
- 957 [21] T. Chatain, S. Haar, L. Jezequel, L. Paulevé, S. Schwoon, Characteriza-
958 tion of reachable attractors using Petri net unfoldings, in: International
959 Conference on Computational Methods in Systems Biology, Springer,
960 2014, pp. 129–142.
- 961 [22] C. Chaouiya, E. Remy, P. Ruet, D. Thieffry, Qualitative modelling of
962 genetic networks: From logical regulatory graphs to standard Petri nets,
963 in: International Conference on Applications and Theory of Petri Nets,
964 Springer, 2004, pp. 137–156.
- 965 [23] C. Chaouiya, A. Naldi, E. Remy, D. Thieffry, Petri net representation of
966 multi-valued logical regulatory graphs, *Nat. Comput.* 10 (2011) 727–750.
- 967 [24] T. Chatain, S. Haar, J. Kolcák, L. Paulevé, A. Thakkar, Concurrency
968 in Boolean networks, *Nat. Comput.* 19 (2020) 91–109.
- 969 [25] C. Chaouiya, D. Béranguier, S. M. Keating, A. Naldi, et al., SBML
970 qualitative models: a model representation format and infrastructure to

- 971 foster interactions between qualitative modelling formalisms and tools,
972 BMC Syst. Biol. 7 (2013) 1–15.
- 973 [26] S. M. Keating, D. Waltemath, M. König, F. Zhang, et al., SBML Level
974 3: an extensible format for the exchange and reuse of biological models,
975 Mol. Syst. Biol. 16 (2020) e9110.
- 976 [27] C. Chaouiya, A. Naldi, D. Thieffry, Logical modelling of gene regulatory
977 networks with GINSim, in: Bacterial Molecular Networks, Springer,
978 2012, pp. 463–479.
- 979 [28] A. Naldi, P. T. Monteiro, C. Müssel, C. for Logical Models, Tools,
980 H. A. Kestler, D. Thieffry, I. Xenarios, J. Saez-Rodriguez, T. Helikar,
981 C. Chaouiya, Cooperative development of logical modelling standards
982 and tools with CoLoMoTo, Bioinform. 31 (2015) 1154–1159.
- 983 [29] C. Müssel, M. Hopfensitz, H. A. Kestler, BoolNet - an R package for
984 generation, reconstruction and analysis of Boolean networks, Bioinform.
985 26 (2010) 1378–1380.
- 986 [30] D. Angeli, P. D. Leenheer, E. Sontag, A Petri net approach to persistence
987 analysis in chemical reaction networks, in: Biology and Control Theory:
988 Current Challenges, Springer, 2007, pp. 181–216.
- 989 [31] D. Angeli, P. D. Leenheer, E. D. Sontag, Persistence results for chemical
990 reaction networks with time-dependent kinetics and no global conserva-
991 tion laws, SIAM J. Appl. Math. 71 (2011) 128–146.
- 992 [32] E. Degrand, F. Fages, S. Soliman, Graphical conditions for rate inde-
993 pendence in chemical reaction networks, in: International Conference on
994 Computational Methods in Systems Biology, Springer, 2020, pp. 61–78.
- 995 [33] G. Liu, K. Barkaoui, A survey of siphons in Petri nets, Inf. Sci. 363
996 (2016) 198–220.
- 997 [34] E. Demirović, G. Chu, P. J. Stuckey, Solution-based phase saving for
998 CP: A value-selection heuristic to simulate local search behavior in com-
999 plete solvers, in: International Conference on Principles and Practice of
1000 Constraint Programming, Springer, 2018, pp. 99–108.

- 1001 [35] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, G. Tack,
1002 MiniZinc: Towards a standard CP modelling language, in: Interna-
1003 tional Conference on Principles and Practice of Constraint Program-
1004 ming, Springer, 2007, pp. 529–543.
- 1005 [36] A. Ignatiev, A. Morgado, J. Marques-Silva, RC2: an efficient MaxSAT
1006 solver, *J. Satisf. Boolean Model. Comput.* 11 (2019) 53–64.
- 1007 [37] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub,
1008 M. Schneider, Potassco: The Potsdam answer set solving collection,
1009 *AI Commun.* 24 (2011) 107–124.
- 1010 [38] J. Forrest, T. Ralphs, H. G. Santos, S. Vigerske, J. Forrest, L. Hafer,
1011 B. Kristjansson, jpfasano, EdwinStraver, M. Lubin, rlougee, jp-
1012 goncall, Jan-Willem, h-i gassmann, S. Brito, Cristina, M. Saltz-
1013 man, tosttost, B. Pitrus, F. MATSUSHIMA, to st, coin-or/Cbc: Re-
1014 lease releases/2.10.8, 2022. URL: [https://doi.org/10.5281/zenodo.](https://doi.org/10.5281/zenodo.6522795)
1015 6522795.
- 1016 [39] J. C. Rozum, J. G. T. Zañudo, X. Gan, D. Deritei, R. Albert, Parity
1017 and time reversal elucidate both decision-making in empirical models
1018 and attractor scaling in critical Boolean networks, *Sci. Adv.* 7 (2021)
1019 eabf8124.
- 1020 [40] V. Singh, M. Ostaszewski, G. D. Kalliolias, G. Chiocchia, R. Olaso,
1021 E. Petit-Teixeira, T. Helikar, A. Niarakis, Computational systems bi-
1022 ology approach for the study of rheumatoid arthritis: from a molecular
1023 map to a dynamical model, *Genom. Comput. Biol.* 4 (2018) 100050.
- 1024 [41] S. Ogishima, S. Mizuno, M. Kikuchi, A. Miyashita, R. Kuwano,
1025 H. Tanaka, J. Nakaya, AlzPathway, an updated map of curated sig-
1026 naling pathways: towards deciphering Alzheimer’s disease pathogenesis,
1027 in: *Systems Biology of Alzheimer’s Disease*, Springer, 2016, pp. 423–432.
- 1028 [42] C. Kadelka, T.-M. Butrie, E. Hilton, J. Kinseth, H. Serdarevic, A meta-
1029 analysis of Boolean network models reveals design principles of gene
1030 regulatory networks, *arXiv preprint arXiv:2009.01216* (2020).
- 1031 [43] E. C. Chávez-Hernández, S. Quiroz, B. García-Ponce, E. R. Álvarez-
1032 Buylla, The flowering transition pathways converge into a complex gene

1033 regulatory network that underlies the phase changes of the shoot apical
1034 meristem in *Arabidopsis thaliana*, *Front. Plant Sci.* 13 (2022) 852047.

1035 [44] A. Yachie-Kinoshita, K. Onishi, J. Ostblom, M. A. Langley, E. Posfai,
1036 J. Rossant, P. W. Zandstra, Modeling signaling-dependent pluripotency
1037 with Boolean logic to predict cell fate transitions, *Mol. Syst. Biol.* 14
1038 (2018) e7952.

1039 [45] M. R. Vega, Analyzing toys models of *Arabidopsis* and *Drosophila* us-
1040 ing Z3 SMT-LIB, in: *Independent Component Analyses, Compressive
1041 Sampling, Wavelets, Neural Net, Biosystems, and Nanoengineering XII*,
1042 volume 9118, SPIE, 2014, pp. 240–254.

1043 [46] E. Cacace, S. Collombet, D. Thieffry, Logical modeling of cell fate
1044 specification—Application to T cell commitment, in: *Current Topics in
1045 Developmental Biology*, Elsevier, 2020, pp. 205–238.

1046 [47] P. Dutta, L. Ma, Y. Ali, P. M. Sloot, J. Zheng, Boolean network model-
1047 ing of B-cell apoptosis and insulin resistance in type 2 diabetes mellitus,
1048 *BMC Syst. Biol.* 13 (2019) 1–12.

1049 [48] K. F. Corral-Jara, C. Chauvin, W. Abou-Jaoudé, M. Grandclaudeon,
1050 A. Naldi, V. Soumelis, D. Thieffry, Interplay between SMAD2 and
1051 STAT5A is a critical determinant of IL-17A/IL-17F differential expres-
1052 sion, *Mol. Biomed.* 2 (2021) 1–16.

1053 [49] E. Guberman, H. Sherief, E. R. Regan, Boolean model of anchorage
1054 dependence and contact inhibition points to coordinated inhibition but
1055 semi-independent induction of proliferation and migration, *Comput.
1056 Struct. Biotechnol. J.* 18 (2020) 2145–2165.

1057 [50] E. Sullivan, M. Harris, A. Bhatnagar, E. Guberman, I. Zonfa, E. R. Re-
1058 gan, Boolean modeling of mechanosensitive Epithelial to Mesenchymal
1059 Transition and its reversal, *bioRxiv* (2022).

1060 [51] N. Weinstein, L. Mendoza, I. Gitler, J. Klapp, A network model to
1061 explore the effect of the micro-environment on endothelial cell behavior
1062 during angiogenesis, *Front. Physiol.* 8 (2017) 960.

1063 [52] T. Lubitz, N. Welkenhuysen, S. Shashkova, L. Bendrioua, S. Hohmann,
1064 E. Klipp, M. Krantz, Network reconstruction and validation of the

- 1065 Snf1/AMPK pathway in baker's yeast based on a comprehensive litera-
1066 ture review, *npj Syst. Biol. Appl.* 1 (2015) 1–10.
- 1067 [53] C. Hernandez, M. Thomas-Chollier, A. Naldi, D. Thieffry, Computa-
1068 tional verification of large logical models—Application to the prediction
1069 of T cell response to checkpoint inhibitors, *Front. Physiol.* 11 (2020)
1070 558606.
- 1071 [54] S. Chevalier, V. Noël, L. Calzone, A. Y. Zinovyev, L. Paulevé, Synthesis
1072 and simulation of ensembles of Boolean networks for cell fate decision,
1073 in: *International Conference on Computational Methods in Systems Bi-*
1074 *ology*, Springer, 2020, pp. 193–209.
- 1075 [55] M. Ostaszewski, A. Niarakis, A. Mazein, I. Kuperstein, R. Phair,
1076 A. Orta-Resendiz, V. Singh, S. S. Aghamiri, M. L. Acencio, E. Glaab,
1077 et al., COVID19 Disease Map, a computational knowledge repository of
1078 virus–host interaction mechanisms, *Mol. Syst. Biol.* 17 (2021) e10387.