Scalable Counting of Minimal Trap Spaces and Fixed Points in Boolean Networks

Mohimenul Kabir[1] ✉ 📷
School of Computing, National University of Singapore, Singapore

Van-Giang Trinh ✉ 📷
Inria Saclay, EP Lifeware, Palaiseau, France

Samuel Pastva ✉ 📷
Faculty of Informatics, Masaryk University, Brno, Czechia

Kuldeep S Meel ✉ 📷
School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA

―――― **Abstract** ――――――――――――――――――――――――――――――――――

Boolean Networks (BNs) serve as a fundamental modeling framework for capturing complex dynamical systems across various domains, including systems biology, computational logic, and artificial intelligence. A crucial property of BNs is the presence of trap spaces – subspaces of the state space that, once entered, cannot be exited. Minimal trap spaces, in particular, play a significant role in analyzing the long-term behavior of BNs, making their efficient enumeration and counting essential. The fixed points in BNs are a special case of minimal trap spaces. In this work, we formulate several meaningful counting problems related to minimal trap spaces and fixed points in BNs. These problems provide valuable insights both within BN theory (e.g., in probabilistic reasoning and dynamical analysis) and in broader application areas, including systems biology, abstract argumentation, and logic programming. To address these computational challenges, we propose novel methods based on *approximate answer set counting*, leveraging techniques from answer set programming. Our approach efficiently approximates the number of minimal trap spaces and the number of fixed points without requiring exhaustive enumeration, making it particularly well-suited for large-scale BNs. Our experimental evaluation on an extensive and diverse set of benchmark instances shows that our methods significantly improve the feasibility of counting minimal trap spaces and fixed points, paving the way for new applications in BN analysis and beyond.

---

[1] Corresponding author

1 Introduction

Boolean Networks (BNs) serve as a fundamental modeling framework for representing complex dynamical systems across domains such as systems biology, computational logic, and artificial intelligence [19, 31, 56, 58, 73]. Their ability to capture intricate interactions and complex system behaviors makes them a valuable tool for studying diverse phenomena such as gene regulatory networks, logical circuits, and reasoning processes [5, 56]. A crucial aspect of BN dynamics is the existence of trap spaces – subspaces of the state space that, once entered, cannot be exited [44]. These structures are instrumental in understanding the long-term behavior of BNs, as they often correspond to stable system configurations or attractors [44].

Among trap spaces, minimal trap spaces (fixed points are a special case of minimal trap spaces in which all variables are *fixed* [44]) are of particular interest due to their role in characterizing the dynamical landscape of a BN [44, 75]. The identification and counting of minimal trap spaces provide valuable insights into the system stability, attractor structure, and probabilistic behavior [44, 32]. However, enumerating these structures poses a significant challenge due to their computational complexity, especially for large-scale BNs encountered in practical applications [68]. Scalable methods for counting minimal trap spaces are therefore essential for advancing the BN analysis.

Unfortunately, minimal trap spaces have received little attention in the context of counting problems for Boolean networks. In contrast, the problem of counting fixed points has been studied in several works [67, 66, 32, 12]. These works show that fixed point counting is generally #P-complete and examine its complexity under structural restrictions [67, 66] or specific classes of Boolean update functions [32]. Some studies also consider scenarios in which the update logic is only partially known [12]. On the practical side, while no dedicated implementation exists for fixed point counting, the task can be reduced to propositional model counting [60] via a CNF encoding of fixed points [20]. To the best of our knowledge, no prior work – either theoretical or practical – has addressed the problem of counting minimal trap spaces in Boolean networks.

Our paper addresses the research gap by formulating six meaningful problems related to counting minimal trap spaces and fixed points in BNs. These counting problems capture core tasks such as counting all minimal trap spaces or fixed points, counting those that satisfy a given *property* (e.g., a *phenotype*), and counting solutions under *perturbations*. These problems provide valuable insights not only within BN theory (e.g., probabilistic reasoning and dynamical analysis) but also in broader application areas such as abstract argumentation and logic programming (discussed in Section 4). We subsequently propose novel and efficient methods to solve the counting problems by exploiting the expressive power of Answer Set Programming (ASP) [50]. ASP is a declarative problem solving paradigm and has widely been applied in the field of systems biology [57, 76], in particular in the analysis and control of BNs [1, 42, 44, 55, 68, 71] (see Section 3). As in existing ASP counting literature [39], our ASP-based reduction leverages the approximate answer set counter ApproxASP [38], which employs a *hashing-based* technique for approximate answer set counting. Finally, we conduct an extensive experimental evaluation on a diverse benchmark dataset. Our analysis shows that ApproxASP efficiently estimates the number of minimal trap spaces and fixed points, and ApproxMC [77] efficiently estimates fixed point counts. Both approaches avoid exhaustive enumeration, significantly improving the feasibility of counting compared to enumeration and BDD-based methods used in other tools.

The remainder of the paper is structured as follows. In Section 2, we review the necessary background on propositional logic, BNs, answer set programming, and model counting. Section 3 surveys related work. Section 4 defines the six counting problems we consider

and discusses their applications. Section 5 introduces our ASP-based reduction methods for solving these problems. Section 6 presents experimental results, demonstrating the effectiveness of our methods. Finally, Section 7 concludes the paper and outlines potential directions for future research.

2 Preliminaries

In this section, we present background and some notations from propositional logic, Boolean networks, and answer set programming.

2.1 Propositional Formulas

In this work, we employ $\mathbb{B} = \{0, 1\}$ as the Boolean domain and $\mathbb{B}_\star = \{0, 1, \star\}$ as the three-valued domain. Under propositional semantics, each *propositional variable v* takes its value from $\mathbb{B}$. A propositional formula is defined recursively: the basic formulas include the Boolean constants 0 and 1, any variable v, and its negation $\neg v$. Moreover, if a formula is prefixed by $\neg$ (not) or if two formulas are connected by a logical connective – $\wedge$ (conjunction), $\vee$ (disjunction), and $\leftrightarrow$ (bi-implication) – the resulting expression is also a formula. A formula is in Conjunctive Normal Form (CNF) if it is expressed as a conjunction of disjunctions of literals (variables or their negations), and it is in Negation Normal Form (NNF) if all negations are applied directly to propositional variables. Any propositional formula can be converted into a semantically equivalent CNF or NNF by recursively applying specific rewriting rules for its logical connectives [10, 51].

2.2 Boolean Networks

A Boolean Network (BN) f is defined as a finite set of Boolean functions over a finite set of Boolean variables, denoted by $\mathsf{Var}(f)$. Each variable $v \in \mathsf{Var}(f)$ is associated with a Boolean function $f_v : \mathbb{B}^{|\mathsf{Var}(f)|} \to \mathbb{B}$. A function f_v is termed *constant* if it is always either 0 or 1 regardless of the values of its arguments. A variable v is considered a *source variable* if f_v is the identity function on v, i.e., $f_v = v$. A state s of f is a Boolean vector $s \in \mathbb{B}^{|\mathsf{Var}(f)|}$ that can be viewed as a mapping: $s : \mathsf{Var}(f) \to \mathbb{B}$; we denote the value of variable v in state s by s_v. For convenience, a state is often represented as a string of values (e.g., "0110" instead of $(0, 1, 1, 0)$).

At each discrete time step t, each variable v can update its state according to its Boolean function f_v; that is, v's state at time $t + 1$ is given by $s'_v = f_v(s)$. An *update scheme* specifies how these state updates occur over time [58]. The two primary schemes are synchronous, in which all variables update simultaneously, and fully asynchronous, where a single variable is chosen non-deterministically to update. Under arbitrary update scheme, the BN transitions from one state to another – a process known as a *state transition*. The overall dynamics of the BN are captured by the *State Transition Graph* (STG), a directed graph whose nodes represent states and edges represent transitions. We denote the STG under the synchronous update scheme as $\mathsf{sstg}(f)$ and that under the fully asynchronous scheme as $\mathsf{astg}(f)$.

A non-empty set A of states is a *trap set* if there is no transition from a state in A to a state outside A in the State Transition Graph (STG) of f (i.e., there is no pair $x \in A$ and $y \notin A$ such that (x, y) is an arc in the STG) [44]. A trap set that is minimal with respect to set inclusion is termed an *attractor*. In particular, an attractor containing a single state is called a *fixed point*, while one with two or more states is referred to as a *cyclic* attractor. A *sub-space m* of a BN f is a mapping $m : \mathsf{Var}(f) \to \mathbb{B}_\star$. A variable $v \in \mathsf{Var}(f)$ is said to be

fixed (resp. *free*) in m if $m(v) \neq \star$ (resp. $m(v) = \star$). For convenience, a sub-space is often represented as a string of values (e.g., $0\star$ instead of $\{v_1 = 0, v_2 = \star\}$). The sub-space m represents a set of states, denoted by $\mathcal{S}[m]$, defined as

$$\mathcal{S}[m] = \{s \in \mathbb{B}^{|\mathsf{Var}(f)|} \mid s_v = m(v), \forall v \in \mathsf{Var}(f), m(v) \neq \star\}$$

For example, if $m = \star 11$, then $\mathcal{S}[m] = \{011, 111\}$. If a sub-space is also a trap set, it is a *trap space*. Unlike trap sets and attractors, trap spaces are independent of the update scheme employed [44]. Notably, a fixed point of f is a special trap space in which all variables are fixed. A trap space m is *minimal* if there is no trap space m' such that $\mathcal{S}[m'] \subset \mathcal{S}[m]$. Since an attractor is a subset-minimal trap set, a minimal trap space contains at least one attractor of the BN, regardless of the update scheme employed [44].

▶ **Example 1.** Let us consider BN f with $\mathsf{Var}(f) = \{a, b\}$, $f_a = a \wedge \neg b$, and $f_b = a$. The synchronous STG of f is shown in Figure 1a. The set $\{00, 01, 11\}$ is a trap set but not a trap space. It is easy to check that f has three trap spaces: $m_1 = 00$, $m_2 = 0\star$, and $m_3 = \star\star$. Among these, m_1 is a minimal trap space (also a fixed point) of f. In this case, m_1 is also the only synchronous attractor of f.

(a) **(b)**
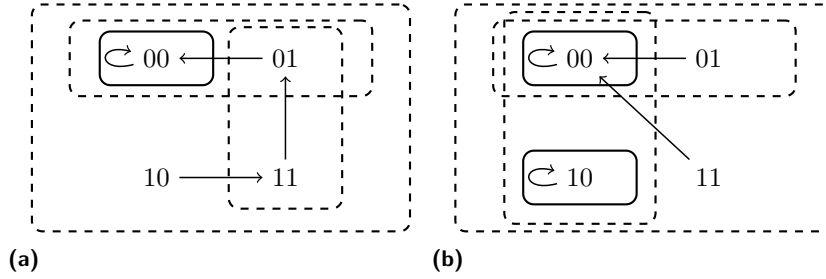
Figure 1 (a) Synchronous STG $\mathsf{sstg}(f)$ of BN f from Example 1. (b) Synchronous STG $\mathsf{sstg}(f)$ where variable b is subject to a *knockout* (i.e., its value is forced to 0). Trap spaces (resp. minimal trap spaces) are enclosed by dashed (resp. solid) rectangular frames.

2.3 Answer Set Programming

An Answer Set Program (ASP) P expresses logical constraints between a set of propositional variables. In answer set programming, such variables are also called *atoms*, and the set of atoms appearing in P is denoted by $\mathsf{at}(P)$. For notational convenience, we will use the terms "variable" and "atom" interchangeably throughout the paper.

An ASP is a finite set of *rules* of the form

$$a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_m, \sim c_1, \ldots, \sim c_n$$

where $k, m, n \geq 0$ and $\sim$ denotes *default negation* [17]. Given a rule r of the above form, we define $\mathsf{h}(r) = \{a_1, \ldots, a_n\}$, $\mathsf{b}^+(r) = \{b_1, \ldots, b_m\}$, and $\mathsf{b}^-(r) = \{c_1, \ldots, c_n\}$ as the *head*, *positive body*, and *negative body* of r, respectively. The rule r is a *fact* if $\mathsf{b}^+(r) = \mathsf{b}^-(r) = \emptyset$ and an (integrity) *constraint* if $\mathsf{h}(r) = \emptyset$. We often use the notations $\top$ and $\bot$ to denote empty body ($\mathsf{b}^+(r) \cup \mathsf{b}^-(r) = \emptyset$) and empty head ($\mathsf{h}(r) = \emptyset$), respectively. The rule r is called *normal* if $|\mathsf{h}(r)| \leq 1$, and the ASP program is normal (or a normal logic program) if all its rules are normal. A program is called *disjunctive* if there is a rule $r \in P$ such that $|\mathsf{h}(r)| > 1$ [7].

Answer Set Semantics

A subset M of atoms (called an *interpretation*) *satisfies* a rule r if $(\mathsf{h}(r) \cup \mathsf{b}^-(r)) \cap M \neq \emptyset$ or $\mathsf{b}^+(r) \setminus M \neq \emptyset$. The interpretation M is a *supported model* of P if it satisfies all rules of P, denoted as $M \models P$. The *Gelfond-Lifschitz (GL) reduct* of the program P with respect to the interpretation M is defined as follows: $P^M := \{\mathsf{h}(r) \leftarrow \mathsf{b}^+(r) | r \in P, M \cap \mathsf{b}^-(r) = \emptyset\}$ [30]. The interpretation M is an *answer set* (or a *stable model*) of P if $\nexists M' \subsetneq M$ such that $M' \models P^M$. We use the notation $\mathsf{AS}(P)$ to denote the set of answer sets of P.

Answer Set Counting

Given an ASP P, the *answer set counting* problem (denoted as #ASP) seeks to compute the number of answer sets of P, written as $|\mathsf{AS}(P)|$. The projected answer set counting problem (denoted as #PASP) extends #ASP by counting the number of distinct answer sets of P with respect to a given set of *projection atoms* $I \subseteq \mathsf{at}(P)$. Two answer sets are considered equivalent if they differ only on atoms in $\mathsf{at}(P) \setminus I$ [25]. We denote the set of projected answer sets as $\mathsf{AS}(P, I)$, so that #PASP computes $|\{M \cap I \mid M \in \mathsf{AS}(P)\}|$. As a special case, when $I = \mathsf{at}(P)$, #PASP reduces to #ASP. We use the notation $\#\mathsf{PASP}(P, I)$ to denote the *projected answer set count* of program P w.r.t. projection atoms I.

In the probably approximately correct (PAC) framework for answer set counting, given a program P and a projection set I, the goal is to estimate a count cnt satisfying

$$\Pr\left[\frac{|\mathsf{AS}(P, I)|}{(1 + \varepsilon)} \leq \mathsf{cnt} \leq (1 + \varepsilon) \times |\mathsf{AS}(P, I)|\right] \geq 1 - \delta$$

where $0 < \varepsilon < 1$ is the *tolerance* and $0 < \delta < 1$ is the *confidence* [38, 14].

3 Related Work

ASP-Based Computation of Fixed Points and Minimal Trap Spaces

Several ASP encodings have been proposed to characterize fixed points and minimal trap spaces in BNs. In many cases, an ASP encoding for fixed points is derived from its minimal trap space counterpart by adding *integrity constraints* to capture the specific properties of fixed points [69, 72]. The first ASP encoding that requires the computation of *prime implicants* for each Boolean function was proposed and implemented in [44]. A major bottleneck of this encoding is that computing even a single prime implicant is NP-hard and the total number of prime implicants can be exponential in the number of function inputs [15]. Subsequent encodings [55, 69, 72] that were proposed to overcome this bottleneck still suffer from scalability and efficiency issues, particularly for very large and complex models, primarily because they require the *disjunctive normal forms* of all the Boolean functions of the original BN. For fixed points, the ASP encoding by Trinh et al. [71] (called `fASP`) uses a *negation normal form* for each Boolean function, which is much more efficient to obtain. This encoding was later generalized for minimal trap spaces [68] (called `tsconj`); however, when dealing with *unsafe formulas* that might yield incorrect solutions, it still requires a disjunctive normal form to ensure the correctness.

BN Encoding of Normal Logic Programs

The theoretical work by Inoue [33] was among the first to establish a connection between ASP and BNs. It defines a BN encoding for finite ground normal logic programs, which relies on the notion of the Clark's completion [17], and points out that the two-valued models of

the Clark's completion of a finite ground normal logic program one-to-one correspond to the fixed points of the encoded BN. The subsequent work [34] points out that the strict supported classes of a finite ground normal logic program one-to-one correspond to the synchronous attractors of the encoded BN. Very recently, Trinh et al. [73] related the regular models in a finite ground normal logic program and the minimal trap spaces in the respective BN, and further applied these theoretical results to explore graphical conditions for the existence, uniqueness, and number of regular models.

SAT Characterization of Fixed Points

The set of fixed points of a BN f can be characterized as the set of *satisfying assignments* of the propositional formula $\bigwedge_{v \in \mathsf{Var}(f)} (v \leftrightarrow f_v)$ [20]. Hence, we can apply #SAT tools [64, 13, 60] to counting the number of fixed points of a BN. To the best of our knowledge, to date, there is no SAT characterization for the set of minimal trap spaces of a BN.

Answer Set Counting

For general ASP programs, #ASP is $\# \cdot \mathrm{coNP}$-complete [27], while #ASP is $\# \cdot \mathrm{P}$-complete for normal ASP programs, which follows from standard reductions [35]. The projected answer set counting #PASP is simply Σ_2^p-complete if the set of projection atoms is empty; otherwise, the complexity is $\# \cdot \Sigma_2^p$-complete [25]. Fichte et al. [25, 27] exploited the *tree decomposition*-based technique for counting answer sets. Kabir et al. [38] introduced the hashing-based approximate counting technique for answer set counting.

4   Problem Formulation

In this section, we introduce several counting problems related to minimal trap spaces in Boolean networks (BNs), covering both minimal trap spaces and fixed points. We begin with a straightforward counting variant, then propose a specialized variant that requires solutions to satisfy a specific *property*, and finally present a more complex variant focused on phenotype measurement in BNs under perturbations. To highlight the biological utility of these theoretical problems, a brief case study is also given in Appendix D.

4.1   Counting Minimal Trap Spaces and Fixed Points

We introduce two fundamental counting problems for Boolean networks (BNs): one that counts the number of minimal trap spaces (Definition 2) and another one counts the number of fixed points (Definition 3). These problems address the basic question: How many minimal trap spaces or fixed points does a given BN have?

▶ **Definition 2** (C-MTS-1). *Given a BN f, compute the number of minimal trap spaces of f.*

▶ **Definition 3** (C-FIX-1). *Given a BN f, compute the number of fixed points of f.*

In BN research, both counting problems – C-MTS-1 and C-FIX-1 – are valuable when full enumeration is infeasible, as in gene regulatory network models with many source variables [2, 68, 71]. They are also useful when divergent solutions are sought [16]. Notably, the fixed point counting problem (C-FIX-1) can enhance methods for enumerating asynchronous attractors by enabling the selection of a smaller candidate set, thereby potentially speeding up the filtering process [74]. Finally, both C-MTS-1 and C-FIX-1 can lay the groundwork for probabilistic reasoning in BNs [61].

Recent research has connected Boolean networks (BNs) with two broad fields: abstract argumentation and logic programming. Although we omit the preliminaries on Abstract Argumentation Frameworks (AFs) [18], Abstract Dialectical Frameworks (ADFs) [49], and Normal Logic Programs (NLPs) [73], interested readers can consult the cited papers for details. Specifically, Trinh et al. [70] and Dimopoulos et al. [19] demonstrate that the *preferred* (resp. *stable*) extensions of an AF one-to-one correspond to the minimal trap spaces (resp. fixed points) of the respective BN. Similarly, Heyninck et al. [31] and Azpeitia et al. [5] demonstrate that ADFs and BNs are identical – in this context, the *preferred* (resp. *two-valued*) interpretations of an ADF match the minimal trap spaces (resp. fixed points) of the respective BN. In the realm of logic programming, the subset-minimal *supported partial* models (resp. *supported* models) of a *finite ground* NLP correspond one-to-one with the minimal trap spaces (resp. fixed points) of the respective BN [33, 73]. Moreover, if the finite ground NLP is *tight* [22, 47], then its regular models (resp. stable models) one-to-one correspond to the minimal trap spaces (resp. fixed points) of the respective BN. These types of extensions, interpretations, and models are central to the study of AFs, ADFs, and finite ground NLPs [26, 36, 49].

Recent work on AFs has focused on counting stable and preferred extensions, yielding both complexity results [26] and dynamic programming methods [18] (note that the dynamic programming methods do not support preferred extensions). In contrast, counting problems for ADFs and finite ground NLPs remain largely unexplored, aside from a few studies on answer sets [4, 38]. Thanks to the connections between BNs and these formalisms, these results from C-MTS-1 and C-FIX-1 can be extended to ADFs – where they correspond to preferred and two-valued interpretations – as well as to general finite ground NLPs (for supported partial and supported models) and tight finite ground NLPs (for regular and stable models).

4.2 Counting with Satisfying Properties

We examine a specialized variant of problems C-MTS-1 and C-FIX-1, focusing on counting minimal trap spaces (Definition 4) and fixed points (Definition 5) that satisfy a specified property. This formulation addresses the natural question: How many minimal trap spaces (or fixed points) in a BN exhibit a given property or assumption?

In systems biology, BNs are used to model biological phenotypes, which reflect an organism's functional characteristics [48]. Several definitions of phenotype in BNs have been proposed [9, 45, 46]; in this work, we adopt one of the most widely used notions. Given a BN f, we define *trait* as a statement of the form $(v \leftrightarrow e)$, where $v \in \mathsf{Var}(f)$ and $e \in \mathbb{B}_\star$. Note that $v \leftrightarrow \star$ is evaluated true if and ony if $v = \star$. A phenotype β is then defined as the conjunction of a set of traits. A sub-space m satisfies a phenotype β (denoted by $m \models \beta$) if, upon replacing each variable $v \in \beta$ with its value $m(v)$, the resulting formula evaluates to true under propositional semantics. Unlike minimal trap spaces, fixed points require that all variables take Boolean values ($e \in \mathbb{B}$), and hence phenotypes involving "$\star$" are not meaningful in this context.

In our problem formulation, the property of interest is a desirable phenotype. In systems biology, a minimal trap space satisfying a phenotype suggests the phenotype's potential emergence in vivo. Furthermore, this counting variant is applicable beyond systems biology. For instance, in abstract argumentation frameworks (resp. normal logic programs), a phenotype may represent the presence of particular set of arguments (resp. atoms) in an extension (resp. a model) [26, 36], a concept closely linked to *credulous reasoning*.

▶ **Definition 4** (C-MTS-2). *Given a BN f and a phenotype β, compute the number of minimal trap spaces of f that satisfy β.*

▶ **Definition 5** (C-FIX-2). *Given a BN f and a phenotype β, compute the number of fixed points of f that satisfy β.*

When the specified phenotype is a *tautology*, C-MTS-2 (resp. C-FIX-2) reduces directly to C-MTS-1 (resp. C-FIX-1). Thus, all implications discussed for C-MTS-1 and C-FIX-1 also apply to C-MTS-2 and C-FIX-2, respectively. In the following, we examine these applications through the lens of abstract argumentation frameworks.

First, C-MTS-2 and C-FIX-2 facilitate more nuanced reasoning between *skeptical* and *credulous* approaches by integrating quantitative and probabilistic methods [23]. For instance, by running C-MTS-2 (or C-FIX-2) twice – once with a specific set of arguments and once with a tautology – we can gain insights into preferred (or stable) extensions [26]. Second, this approach shifts reasoning from simple decision-making (i.e., determining whether a set of arguments is present in an extension) to probabilistic reasoning (i.e., assessing the likelihood of a set of arguments appears in an extension). Moreover, this framework supports the development of advanced probabilistic semantics for abstract argumentation frameworks [41]. To our knowledge, these aspects have not yet been explored in the context of BNs, and our work paves the way for integrating such reasoning into BN analysis.

▶ **Example 6.** Consider again the BN f shown in Example 1. It has a unique minimal trap space $m_1 = 00$, which is also its only fixed point of f. Hence, the answer to C-MTS-1 (resp. C-FIX-1) is 1. Considering the phenotype $\beta = (b \leftrightarrow \star)$, the answer to C-MTS-2 is 0.

4.3   Counting Under Perturbations and Measuring Robustness

We consider a main contribution of this paper to be the identification of new, relevant counting problems (other than C-MTS-2 and C-FIX-2, which were previously known): C-MTS-3 and C-FIX-3. To formalize these problems, we introduce the concept of a *perturbation*.

Consider a BN f. A *perturbation* σ [63] on a set $\mathcal{X} \subseteq \mathsf{Var}(f)$ of *perturbable* variables is defined as a mapping from $\mathcal{X} \to \mathbb{B}_\star$. In practice, $\mathcal{X}$ can be any subset of variables whose perturbation is biologically meaningful. Since each variable in $\mathcal{X}$ can assume one of three values under perturbation, there are $3^{|\mathcal{X}|}$ possible perturbations. For each $v \in \mathcal{X}$, setting $\sigma(v) = 0$ forces $f_v = 0$ (*knockout perturbation*), setting $\sigma(v) = 1$ forces $f_v = 1$ (*over-expression perturbation*), and setting $\sigma(v) = \star$ leaves f_v unchanged. Consequently, the perturbed BN, denoted f^σ, is defined by $\mathsf{Var}(f^\sigma) = \mathsf{Var}(f)$ and, for every $v \in \mathsf{Var}(f^\sigma)$,

$$f_v^\sigma = \begin{cases} \sigma(v) & \text{if } v \in \mathcal{X} \text{ and } \sigma(v) \neq \star \\ f_v & \text{otherwise} \end{cases}$$

Note that, the value $\star$ is used to distinguish imperturbable variables and perturbable variables that are unchanged under a certain perturbation.

In biological systems, a perturbation refers to any disturbance that disrupts the normal functioning of a BN. Such disturbances may arise from *genetic mutations* [62], external factors such as *medications* [11], or other influences [52]. These perturbations can substantially alter the phenotypes exhibited by a BN, making it crucial to quantify their effects on network behavior. In systems biology, this impact is commonly assessed in terms of *robustness* [43], which motivates our definitions of problems C-MTS-3 and C-FIX-3.

▶ **Definition 7** (C-MTS-3). *Given a BN f, a set of perturbable variables $\mathcal{X}$, and a target phenotype β, determine the number of perturbations σ on $\mathcal{X}$ such that the perturbed BN f^σ exhibits at least one minimal trap space that satisfies β.*

▶ **Definition 8** (C-FIX-3). *Given a BN f, a set of perturbable variables $\mathcal{X}$, and a target phenotype β, determine the number of perturbations σ on $\mathcal{X}$ such that the perturbed BN f^σ exhibits at least one fixed point that satisfies β.*

Leveraging the results of C-MTS-3 (or C-FIX-3 if focusing solely on fixed points), we define the robustness of a phenotype as the fraction of perturbations that preserve the phenotype relative to the total number of perturbations ($3^{|\mathcal{X}|}$). In other words, robustness measures the probability that a phenotype remains active following a random, admissible perturbation is applied to the network. This measure of phenotype robustness can inform the selection of specific perturbations and guide targeted treatment strategies [9, 65]. A small case study on an interferon model with 121 variables presents these concepts more practically in Appendix D.

▶ **Example 9.** Consider BN f given in Example 1. Consider $\mathcal{X} = \{b\}$ denote the set of perturbable variables and define the set of desirable phenotype as $\beta = (a \leftrightarrow 0 \wedge b \leftrightarrow 0)$. There are three possible perturbations: $\sigma_1 = \{b = 0\}$, $\sigma_2 = \{b = 1\}$, and $\sigma_3 = \{b = \star\}$. The perturbation σ_1 represents the knockout perturbation of variable b and $\mathsf{sstg}(f^{\sigma_1})$ is given in Figure 1b. It is straightforward to observe that f^{σ_1} has two minimal trap spaces 00 and 10. In constrast, the BN f^{σ_2} has one minimal trap space 01, which does not satisfy the given phenotype. The BN f^{σ_3} equals f and has one minimal trap space 00. Therefore, for BN f, phenotype β and perturbable variables $\mathcal{X}$, the answer to C-MTS-3 is 2 and the perturbation robustness of phenotype β in BN f w.r.t. $\mathcal{X}$ is $2/3$.

On the impact of precision. Finally, it should be noted that while these problems are defined *exactly*, in practice their results mainly serve as a means of comparison. For example, the results of C-MTS-2 could be used to compare the abundance of two biological phenotypes. Then the exact count may not be important, as long as the two phenotypes can be compared reliably. As such, these problems are particularly suitable for approximate counting. This also impacts the choice of method parameters (ϵ and δ), as in practice, even low precision (as used in our benchmarks) can be sufficient to distinguish between significantly different phenotypes. For closely matched results, the precision can be then increased as needed.

5 Computational Methods

Given a BN f, our approach is to construct an ASP program P such that the answer sets of P one-to-one correspond to the minimal trap spaces (or fixed points) of f. This reduction allows us to leverage existing answer set counters to efficiently count the answer sets of P. For C-MTS-1 and C-FIX-1, we simply use the ASP encodings of `tsconj` and `fASP`, respectively. For C-MTS-2 and C-FIX-2, we complement the encoding of the given phenotype. For C-MTS-3 and C-FIX-3, we propose a new perturbation encoding, which we consider to be a main contribution of this paper. Now, we begin by briefly reviewing the `tsconj` and `fASP` encodings.

5.1 `tsconj` and `fASP` Encodings

We first present the common components of the two encodings, followed by their differences. The ASP encodings represent sub-spaces using atoms $\mathsf{p}(v)$ and $\mathsf{n}(v)$, indicating whether variable v is fixed to 1, 0, or left free. The goal is to translate BN trap space and fixed point properties into ASP rules whose answer sets correspond to these configurations.

Given a BN f, the encodings compute an ASP program P as follows: for each variable $v \in \mathsf{Var}(f)$, two atoms $\mathsf{p}(v)$ and $\mathsf{n}(v)$ are introduced to indicate positive and negative assignments of the variable v, respectively. Additionally, for every $v \in \mathsf{Var}(f)$, one rule of the form: $\mathsf{p}(v) \vee \mathsf{n}(v) \leftarrow \top$ is added to ensure that each answer set corresponds to a sub-space of f. The translation from an answer set M to a sub-space m is defined as follows: for each $v \in \mathsf{Var}(f)$, we have (i) $m(v) = 1$ if and only if $\mathsf{p}(v) \in M \wedge \mathsf{n}(v) \notin M$, (ii) $m(v) = 0$ if and only if $\mathsf{p}(v) \notin M \wedge \mathsf{n}(v) \in M$, and (iii) $m(v) = \star$ if and only if $\mathsf{p}(v) \in M \wedge \mathsf{n}(v) \in M$. Recall that a trap space of f can be characterized by $\bigwedge_{v \in \mathsf{Var}(f)}(v \leftarrow f_v) \wedge (\neg v \leftarrow \neg f_v)$ [68]. For every $v \in \mathsf{Var}(f)$, two ASP rules – (i) $\gamma(v) \leftarrow \gamma(\mathsf{NNF}(f_v))$ and (ii) $\gamma(\neg v) \leftarrow \gamma(\mathsf{NNF}(\neg f_v))$ – are added to P to express the characterization, where $\mathsf{NNF}(\Phi)$ denotes a negation normal form of a Boolean formula Φ and γ is a procedure defined as follows:

$$\gamma(v) = \mathsf{p}(v), \gamma(\neg v) = \mathsf{n}(v), v \in \mathsf{Var}(f),$$

$$\gamma(\bigwedge_{1 \leq j \leq J} \alpha_j) = \gamma(\alpha_1) \wedge \ldots \wedge \gamma(\alpha_J), \gamma(\bigvee_{1 \leq j \leq J} \alpha_j) = \mathsf{aux}_k,$$

where aux_k is a new *auxiliary* atom, k is a global counter starting from 1 and shall be increased by 1 after each new auxiliary atom is created, and for each j, the rule $\mathsf{aux}_k \leftarrow \gamma(\alpha_j)$ is added to P.

For the correctness of the `tsconj` encoding, Trinh et al. [68] defined a syntactic safeness condition for a Boolean formula Φ: Φ is considered safe if it does not contain any conjunction of two subformulas Φ_1 and Φ_2 such that there exists a variable x appearing in Φ_1 with $\neg x$ appearing in Φ_2. When both f_v and $\neg f_v$ are safe for every $v \in \mathsf{Var}(f)$, then the set of answer sets of P one-to-one corresponds with the set of minimal trap spaces of f. When a Boolean formula Φ (f_v or $\neg f_v$) is unsafe, the Disjunctive Normal Form (DNF) of Φ is used instead, which is always safe by definition.

There is no notion of "safeness" in the `fASP` encoding. Rather an additional rule $\bot \leftarrow \mathsf{p}(v), \mathsf{n}(v)$ is added to P, for every $v \in \mathsf{Var}(f)$ such that the variable v cannot take the value of $\star$. The answer sets of P one-to-one correspond to the fixed points of f.

Following on, given a BN f, we use the notations $\mathsf{P\text{-}tsconj}(f)$ and $\mathsf{P\text{-}fASP}(f)$ to denote the encoded ASP programs of f, according to the `tsconj` and `fASP` encodings, respectively.

5.2   Methods for Problems C-MTS-1 and C-FIX-1

We make use of `tsconj` and `fASP` encodings for C-MTS-1 and C-FIX-1, respectively. The choice of encodings is due to following two reasons: first, these encodings rely on less expensive representations – specifically, negation normal forms (ref. Section 3). Second, they establish a one-to-one correspondence between the minimal trap spaces (or fixed points) of the original BN and the answer sets of the encoded ASP program, whereas other encodings yield a one-to-one correspondence with the subset-minimal (or subset-maximal) answer sets [3], which prevents the direct use of existing ASP counters.

5.3   Methods for Problems C-MTS-2 and C-FIX-2

We add the encoding of phenotype to the encodings of `tsconj` and `fASP` to solve counting problems C-MTS-2 and C-FIX-2, respectively. Given a BN f and a phenotype β, we compute an ASP program $\mathsf{ToASP}(\beta)$ to capture the phenotype β by invoking Algorithm 1. The algorithm exploits atoms introduced in the `tsconj` encoding to interpret the phenotype β. The main idea of Algorithm 1 is exploiting *faceted answer set navigation* to constrain the search space of answer sets [24]. We prove that the minimal trap spaces of f satisfying the phenotype β one-to-one correspond to the answer sets of $\mathsf{P\text{-}tsconj}(f) \cup \mathsf{ToASP}(\beta)$ (Theorem 10).

Algorithm 1 ToASP(β).

Input: Phenotype β
Output: ASP program Q
1: $Q \leftarrow \emptyset$
2: **for each** $(v \leftrightarrow e) \in \beta$ **do**
3:    **if** $e = 1$ **then**
4:       $Q.\mathsf{add}(\bot \leftarrow \sim\mathsf{p}(v), \qquad \bot \leftarrow \mathsf{n}(v))$
5:    **else if** $e = 0$ **then**
6:       $Q.\mathsf{add}(\bot \leftarrow \mathsf{p}(v), \qquad \bot \leftarrow \sim\mathsf{n}(v))$
7:    **else if** $e = \star$ **then**
8:       $Q.\mathsf{add}(\bot \leftarrow \sim\mathsf{p}(v), \qquad \bot \leftarrow \sim\mathsf{n}(v))$
9: **return** Q

▶ **Theorem 10.** *Given a BN f and a phenotype β, the minimal trap spaces of f satisfying β one-to-one correspond to the answer sets of* $\mathsf{P\text{-}tsconj}(f) \cup \mathsf{ToASP}(\beta)$.

For C-FIX-2, we can apply the `fASP` encoding and Algorithm 1 similarly. Note that in the counting problem C-FIX-2, the term e is either 0 or 1, for each $(v \leftrightarrow e) \in \beta$, since fixed points require all variables to be fixed. We formally prove the correctness of our proposed method for C-FIX-2 in Theorem 11.

▶ **Theorem 11.** *Given a BN f and a phenotype β, the fixed points of f satisfying β one-to-one correspond to the answer sets of* $\mathsf{P\text{-}fASP}(f) \cup \mathsf{ToASP}(\beta)$.

▶ **Example 12.** Consider the BN f of Example 1. The program $\mathsf{P\text{-}tsconj}(f)$ is as follows:

$$\mathsf{p}(a) \vee \mathsf{n}(a) \leftarrow \top \qquad \mathsf{p}(a) \leftarrow \mathsf{p}(a), \mathsf{n}(b) \qquad \mathsf{n}(a) \leftarrow \mathsf{aux}_1 \qquad \mathsf{aux}_1 \leftarrow \mathsf{n}(a) \qquad \mathsf{aux}_1 \leftarrow \mathsf{p}(b)$$
$$\mathsf{p}(b) \vee \mathsf{n}(b) \leftarrow \top \qquad \mathsf{p}(b) \leftarrow \mathsf{p}(a) \qquad \mathsf{n}(b) \leftarrow \mathsf{n}(a)$$

The program $\mathsf{P\text{-}tsconj}(f)$ has a unique answer set $\{\mathsf{n}(a), \mathsf{n}(b), \mathsf{aux}_k\}$ corresponding to the unique minimal trap space 00 of f. Consider the phenotype $\beta = (b \leftrightarrow \star)$, the BN f has no minimal trap space satisfying β (see Example 6), thus C-MTS-2 returns 0. Following the procedure outlined above, the ASP program $\mathsf{ToASP}(\beta)$ is as follows:

$$\bot \leftarrow \sim\mathsf{p}(b) \qquad \bot \leftarrow \sim\mathsf{n}(b)$$

Indeed, the program $\mathsf{P\text{-}tsconj}(f) \cup \mathsf{ToASP}(\beta)$ has no answer set.

5.4 Methods for Problems C-MTS-3 and C-FIX-3

Given a BN f, a phenotype β, and a set of perturbable variables $\mathcal{X}$, one possible approach to solving these problems is to introduce new atoms to represent possible perturbations over perturbable variables, then correspondingly to intervene in the encoded ASP program obtained by applying the encodings proposed for C-MTS-2 and C-FIX-2, and finally to apply projected counting restricted to these new atoms. Instead, we propose a more convenient approach that reduces C-MTS-3 (resp. C-FIX-3) to C-MTS-2 (resp. C-FIX-2) along with projected answer set counting.

▶ **Definition 13.** *Consider a BN f and a set of perturbable variables $\mathcal{X} \subseteq \mathsf{Var}(f)$, we construct a new BN g such that for every $v \in \mathsf{Var}(f)$, if $v \in \mathsf{Var}(f) \setminus \mathcal{X}$, then the variable $v \in \mathsf{Var}(g)$ and $g_v = f_v$, and if $v \in \mathcal{X}$, then three variables $v, v^k, v^o \in \mathsf{Var}(g)$ and*

$$g_v = \neg v^k \wedge (v^o \vee f_v),$$
$$g_{v^k} = v^k,$$
$$g_{v^o} = v^o \wedge \neg v^k.$$

Instead of modifying the ASP encoding to model perturbations, we construct a perturbed version of the original BN, thereby preserving the semantics of the original encodings. For each perturbable variable $v \in \mathcal{X}$, we introduce two new variables in g: v^k, encoding whether v is knocked out, and v^o, encoding whether it is over-expressed. By construction:

- If $v^k = 1$, then $v^o = 0$ and $v = 0$, modeling a knockout of v.
- If $v^k = 0$ and $v^o = 1$, then $v = 1$, modeling over-expression.
- If $v^k = 0$ and $v^o = 0$, then v follows its original update function f_v (v is unperturbed).
- The case $v^k = 1$ and $v^o = 1$ is infeasible due to the constraint $g_{v^o} = v^o \wedge \neg v^k$.

Since $\mathsf{Var}(f) \subseteq \mathsf{Var}(g)$, any phenotype β defined over f remains valid in g. The minimal trap spaces (resp. fixed points) of g correspond to those of f under all possible perturbations over $\mathcal{X}$. Thus, the minimal trap spaces (resp. fixed points) of g that satisfy the phenotype β represent the perturbed solutions of f that also satisfy β. Crucially, for each perturbation, multiple satisfying minimal trap spaces (or fixed points) are counted once, enabling us to count the number of satisfying perturbations. Hence, the counting problem C-MTS-3 (resp. C-FIX-3) reduces to the projected version of C-MTS-2 (resp. C-FIX-2) over the newly defined BN g.

We now discuss how we solve C-MTS-3 and C-FIX-3 by applying projected answer set counting. We focus on the case for minimal trap spaces, and the case of fixed points is trivially similar. Following Theorem 10, the set of answer sets of $\mathsf{P\text{-}tsconj}(g) \cup \mathsf{ToASP}(\beta)$ represents the set of minimal trap spaces of g satisfying the phenotype β. Let $\Omega = \bigcup_{v \in \Delta} \{\mathsf{p}(v), \mathsf{n}(v)\}$ denote the set of perturbation-related variables, where $\Delta = \bigcup_{v \in \mathcal{X}} \{v^k, v^o\}$ be the set of ASP atoms used in the encoding. It follows that the number of answer sets of $\mathsf{P\text{-}tsconj}(g) \cup \mathsf{ToASP}(\beta)$, projected onto the set Ω is equal to the number of perturbation settings (i.e., assignments to variables in $\mathcal{X}$) under which g admits a minimal trap space satisfying β.

▶ **Theorem 14.** *Given a BN f, a set of perturbable variables $\mathcal{X} \subseteq \mathsf{Var}(f)$, a phenotype β, and $\Omega = \bigcup_{v \in \Delta} \{\mathsf{p}(v), \mathsf{n}(v)\}$, where $\Delta = \bigcup_{v \in \mathcal{X}} \{v^k, v^o\}$, then C-MTS-3 can be computed as $\#\mathsf{PASP}(\mathsf{P\text{-}tsconj}(g) \cup \mathsf{ToASP}(\beta), \Omega)$, where g is the new BN according to Definition 13.*

▶ **Theorem 15.** *Given a BN f, a set of perturbable variables $\mathcal{X} \subseteq \mathsf{Var}(f)$, a phenotype β, and $\Omega = \bigcup_{v \in \Delta} \{\mathsf{p}(v), \mathsf{n}(v)\}$, where $\Delta = \bigcup_{v \in \mathcal{X}} \{v^k, v^o\}$, then C-FIX-3 can be computed as $\#\mathsf{PASP}(\mathsf{P\text{-}fASP}(g) \cup \mathsf{ToASP}(\beta), \Omega)$, where g is the new BN following Definition 13.*

▶ **Example 16.** Consider again Example 9. Following Definition 13, we obtain the BN g: $\mathsf{Var}(g) = \{a, b, b^k, b^o\}$, $g_a = a \wedge \neg b$, $g_{b^k} = b^k$, $g_{b^o} = b^o \wedge \neg b^k$, and $g_b = \neg b^k \wedge (b^o \vee a)$. (see Appendix B for the details of $\mathsf{P\text{-}tsconj}(g)$ and $\mathsf{ToASP}(\beta)$). Let $\Omega = \{\mathsf{p}(b^k), \mathsf{n}(b^k), \mathsf{p}(b^o), \mathsf{n}(b^o)\}$. Then C-MTS-3 can be computed as $\#\mathsf{PASP}(\mathsf{P\text{-}tsconj}(g) \cup \mathsf{ToASP}(\beta), \Omega)$. Indeed, $\#\mathsf{PASP}(\mathsf{P\text{-}tsconj}(g) \cup \mathsf{ToASP}(\beta), \Omega)$ returns 2, which is consistent with the result shown in Example 9.

6   Experimental Evaluation

This section presents the experimental evaluation of the presented methods. We use existing minimal trap space and fixed point computation tools as baselines – namely, AEON [8], k++ADF (ADF) [49], and clingo [29]. The ADF tool is applicable here due to the equivalence

between ADFs and BNs [5, 31], which supports only C-MTS-1 and C-FIX-1. k++ADF and Clingo count minimal trap spaces and fixed points via enumeration on the ADF and the encoded ASP respectively, and AEON via BDD-based encoding. For the fixed point problem variants, we further considered the propositional model counter [60] (GANAK) and the approximate model counter [77] (ApproxMC), using the translation to CNF (see Section 3). However, note that these techniques cannot be directly used for minimal trap space counting. For approximate answer set counting, we employed the hashing-based approximate answer set counter ApproxASP [38] with parameters $\varepsilon = 0.8$ and $\delta = 0.2$. Following prior work on counting [40], we provided ApproxASP with an independent support of a disjunctive ASP program exploiting Padoa theorem [53]. We also tested the tree decomposition-based answer set counter DynASP [27]; however, we did not include it in the final analysis because it has been significantly outperformed by the remaining baselines. Finally, note that we could not include #SAT-based ASP counters aspmc [21] and sharpASP [37] as baselines, since these counters are designed for normal logic programs, while `tsconj` and `fASP` encodings produce disjunctive ASP programs.

We compiled our benchmark set from prior studies on minimal trap spaces and fixed points in BNs [54, 68, 71]. The set comprises 645 BN instances – 245 real-world models and 400 randomly generated – with up to 5,000 variables. To evaluate counting problems C-MTS-2, C-FIX-2, C-MTS-3, and C-FIX-3, we pseudo-randomly fixed three variables to represent the target phenotype and selected up to 50 perturbable variables (yielding as many as 3^{50} possible perturbations). Phenotypes are typically not published in machine-readable format, thus we maintain biological interpretability by deriving the phenotype from a known trap space, linking it to an existing biological feature of the network. As for the chosen size, only a few variables are sufficient to identify phenotypes: e.g., in [28], only 10-200 entities are needed out of 10,000. Since our tests are often significantly smaller, we scaled down the phenotypes accordingly. Appendix C provides further details about the benchmark. The code and dataset of experiment evaluation is available at: `https://zenodo.org/records/15141045`

Environmental Settings. All experiments were conducted on a high-performance computing cluster, with each node consisting of Intel Xeon Gold 6248 CPUs. Each benchmark instance was allocated one core, with runtime and memory limits set to 5000 seconds and 8 GB respectively, for all the tools considered.

6.1 Experimental Results

C-MTS-1 and C-FIX-1. The results for C-MTS-1 and C-FIX-1 are shown in Table 1 and Table 2, respectively. Each table reports the number of instances solved (i.e., instances for which a count was successfully returned) by each tool and their corresponding PAR2 scores [6] (PAR2 score is a runtime metric that also penalizes benchmark timeouts). Here, approximate counting (ApproxASP and ApproxMC) clearly outperform all existing solutions. Even compared to exact model counting (GANAK), this approach achieves significantly better performance. Note that for C-FIX-1, ApproxASP and ApproxMC are roughly comparable, but (as discussed in detail later), ApproxASP is faster on simpler instances.

It is worth noting that unsafe formulas are quite rare in 245 real-world models, which is consistent with the observation in [68]. All 400 randomly generated models have no unsafe formulas because of the nature of the generation [68].

C-MTS-2 and C-FIX-2. The results for C-MTS-2 and C-FIX-2 are shown in Table 3 and Table 4, respectively, using the same metric (solved instances and PAR2 score) as Table 1 and Table 2. Here, the relative performance of individual tools mirrors that observed for

▮ **Table 1** The performance comparison of different counters on C-MTS-1 counting problem.

	AEON	ADF	clingo	ApproxASP
#Solved	179	200	211	**364**
PAR2	7255	6923	6742	**4448**

▮ **Table 2** The performance comparison of different counters on C-FIX-1 counting problem.

	AEON	ADF	clingo	GANAK	ApproxMC	ApproxASP
#Solved	247	217	227	317	**420**	413
PAR2	6172	6656	6493	5269	3801	**3760**

C-MTS-1 and C-FIX-1. However, all tools solved more instances overall, largely due to the inclusion of the phenotype property, which typically reduces the number of solutions. Since phenotype properties are generally much simpler than the update functions describing network dynamics, we expect them to simplify the counting problem – often substantially.

▮ **Table 3** The performance comparison of different counters on C-MTS-2 counting problem.

	AEON	clingo	ApproxASP
#Solved	231	308	**464**
PAR2	6428	5237	**2919**

▮ **Table 4** The performance comparison of different counters on C-FIX-2 counting problem.

	AEON	clingo	GANAK	ApproxMC	ApproxASP
#Solved	252	236	333	**438**	429
PAR2	6099	6360	5030	3527	**3499**

C-MTS-3 and C-FIX-3. The results for problems C-MTS-3 and C-FIX-3 are presented in Tables 5 and 6, respectively. This benchmark confirms the leading performance of ApproxASP in minimal trap space and fixed point counting as it was able to solve 644/645 and 645/645 problem instances for C-MTS-3 and C-FIX-3, respectively. Here, ApproxASP significantly outperforms even ApproxMC, and outperforms all other tools by a factor of $2\times$ or more. In contrast to C-MTS-2 and C-FIX-2, where all tools benefited from a reduced number of solutions, the presence of perturbations in this setting generally increases both the number of solutions and the underlying complexity of BNs. The key performance differentiation lies in the tools' ability to handle *projected counting*. For this problem, the *independent support* for XOR constraints is derived from the BN perturbable variables. Since this is only a subset of the network variables, the independent support size is relatively small, reducing the size of the XORs. This results in the superior performance of ApproxASP and ApproxMC. Note that in these countings, the number of perturbable variables is at most 50 and the count is upper-bounded by 3^{50}.

(a) C-MTS-1.   **(b)** C-MTS-2.   **(c)** C-MTS-3.

(d) C-FIX-1.   **(e)** C-FIX-2.   **(f)** C-FIX-3.
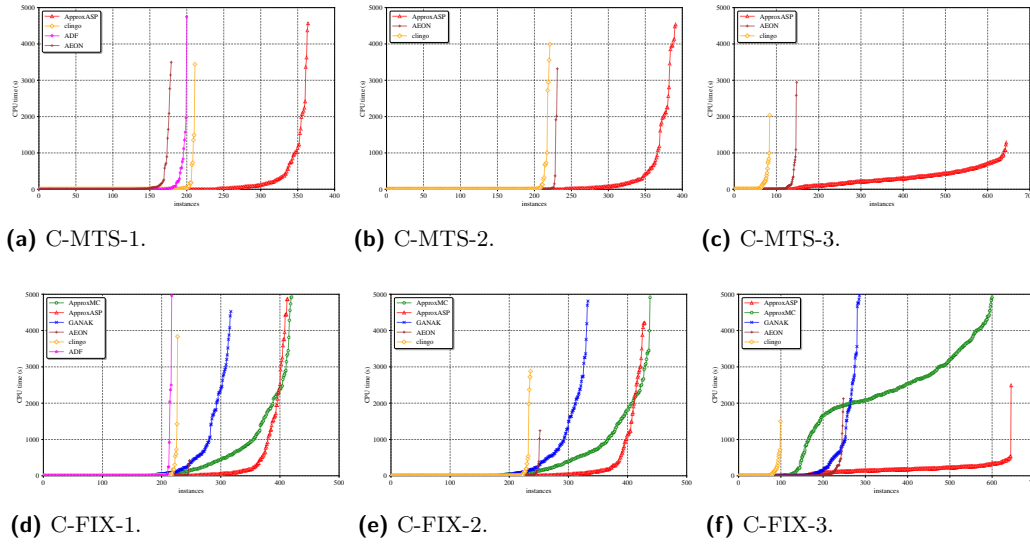
Figure 2 Performance comparison of different counters across all counting problems. The x axis shows the number of benchmarks completed before the corresponding CPU runtime on the y axis.

Table 5 The performance comparison of different counters on C-MTS-3 counting problem.

	AEON	clingo	ApproxASP
#Solved	148	84	**644**
PAR2	7725	8711	**283**

Table 6 The performance comparison of different counters on C-FIX-3 counting problem.

	AEON	clingo	GANAK	ApproxMC	ApproxASP
#Solved	248	99	286	600	**645**
PAR2	6176	8476	5757	2481	**150**

6.2 Detailed Runtime Analysis

The runtime performance of different tools is depicted in Figure 2. In these cactus plots, a point (x, y) indicates that a tool successfully completes x benchmark instances, with each instance taking at most y seconds. The plots highlight the superiority of the hashing-based counting techniques, ApproxASP and ApproxMC. Notably, even in cases where ApproxASP solves fewer instances than ApproxMC (e.g. C-FIX-1), it is typically faster on simpler problem instances, which is also reflected in its PAR2 score.

By examining the number of solutions successfully computed for different tasks, we observe that only ApproxASP, ApproxMC, and GANAK can reliably count instances having a large number of solutions (e.g. $\geq 10^{30}$). Here, BDD-based counters like AEON perform somewhat better on fixed point problems compared to tools using plain enumeration (ADF, clingo), but cannot compete in the (arguably more complex) trap space problems.

The performance of GANAK and ApproxMC is also severely affected by the time required to compute their input CNF formulas (ref. Section 3). Here, deriving the CNF problem representation is often considerably more time-consuming than computing the comparable

ASP encoding. Our results reveal that, on average, it took about 545 seconds to compute the CNF formula for each BN. Moreover, for 39 BNs, the corresponding CNF could not be computed within the 5000-second timeout. In contrast, the ASP encodings for all BNs – including the more challenging ones with perturbable variables – were generated within seconds. This demonstrates the superior flexibility of the ASP-based approach.

Note that the approximate counters ApproxMC and ApproxASP provide an (ε, δ)-guarantee. Thus for each solved instance, we computed the *observed tolerance*, which is defined as $\max(\mathsf{cnt}/|\mathsf{AS}(P)|, |\mathsf{AS}(P)|/\mathsf{cnt}) - 1$, where cnt is the count returned by ApproxASP or ApproxMC, and $\mathsf{AS}(P)$ denotes the answer set count of program P. On average, ApproxMC and ApproxASP exhibit observed tolerances of 0.032 and 0.007, respectively. The maximum observed tolerances were 0.39 for ApproxASP and 0.07 for ApproxMC, both of which are well below the theoretical bound of $\varepsilon = 0.8$.

We evaluated ApproxMC and ApproxASP with a tighter guarantee, setting $\varepsilon = 0.01$ and $\delta = 0.05$. In these setting, ApproxASP solved $244, 243, 114, 258, 259, 130$ for C-MTS-1, C-MTS-2, C-MTS-3, C-FIX-1, C-FIX-2, and C-FIX-3, respectively, while ApproxMC solved $263, 259, 120$ for C-FIX-1, C-FIX-2, and C-FIX-3, respectively. Consequently, the higher precision significantly reduces the number of solved instances.

We compared the size of instances solved by different counting techniques across various counting problems. We observed that, for minimal trap spaces, ApproxASP solved instances with up to 4000 variables, while `tsconj` and AEON solved instances with up to 321 variables. For fixed points, although there were large instances (up to 4000 variables), these did not contain fixed points, making a direct comparison unfeasible.

7 Conclusion

This paper addresses the problem of counting minimal trap spaces and fixed points in Boolean networks. These are critical concepts in understanding of long-term BN behavior and are relevant across a diverse set of application domains, including systems biology, abstract argumentation, and logic programming. Trap space counting is especially important in systems biology: due to the inherent robustness of biological phenomena, biologically motivated BNs admit a high degree of redundancy, resulting in a vast repertoire of closely related trap spaces (or fixed points) that cannot be explored solely through enumeration.

Here, we propose novel methods for determining trap space and fixed point counts using approximate answer set counting, thus entirely avoiding costly enumeration. We apply this methodology to three biologically motivated problems: (a) general counting; (b) counting occurrences of a known biological phenotype, and (c) projected counting of perturbations that ensure the emergence of a known biological phenotype. The last problem is particularly timely, as it allows us to determine *perturbation robustness* [9, 43], a vital measure that determines how stable a phenotype appears under external stimuli. Through extensive experiments on a diverse set of benchmarks, we show that approximate counting substantially improves the feasibility of counting in this domain, outperforming traditional enumeration-based and exact approaches whenever applicable.

Our work opens several promising directions for future research: First is to integrate reduction techniques – previously shown to be effective in BN analysis [56, 59, 65] – to further improve counting accuracy and scalability. Second direction explores hybrid strategies that combine exact and approximate counting, aiming to strike a balance between efficiency and precision. Finally, a deeper investigation into the computational complexity of the counting problems would help refine our understanding of their theoretical underpinnings.

References

1 Emna Ben Abdallah, Maxime Folschette, Olivier F. Roux, and Morgan Magnin. ASP-based method for the enumeration of attractors in non-deterministic synchronous and asynchronous multi-valued networks. *Algorithms Mol. Biol.*, 12(1):20:1–20:23, 2017. `doi:10.1186/S13015-017-0111-2`.

2 Sara Sadat Aghamiri, Vidisha Singh, Aurélien Naldi, Tomás Helikar, Sylvain Soliman, Anna Niarakis, and Jinbo Xu. Automated inference of Boolean models from molecular interaction maps using CaSQ. *Bioinform.*, 36(16):4473–4482, 2020. `doi:10.1093/BIOINFORMATICS/BTAA484`.

3 Mario Alviano, Carmine Dodaro, Salvatore Fiorentino, Alessandro Previti, and Francesco Ricca. ASP and subset minimality: Enumeration, cautious reasoning and MUSes. *Artif. Intell.*, 320:103931, 2023. `doi:10.1016/J.ARTINT.2023.103931`.

4 Rehan Abdul Aziz, Geoffrey Chu, Christian J. Muise, and Peter James Stuckey. Stable model counting and its application in probabilistic logic programming. In *AAAI*, pages 3468–3474. AAAI Press, 2015. `doi:10.1609/AAAI.V29I1.9691`.

5 Eugenio Azpeitia, Stan Muñoz Gutiérrez, David A. Rosenblueth, and Octavio Zapata. Bridging abstract dialectical argumentation and Boolean gene regulation. *CoRR*, abs/2407.06106, 2024. `doi:10.48550/arXiv.2407.06106`.

6 Tomas Balyo, Marijn JH Heule, and Matti Järvisalo. SAT competition 2017–solver and benchmark descriptions, 2017.

7 Rachel Ben-Eliyahu and Rina Dechter. Propositional semantics for disjunctive logic programs. *Ann. Math. Artif. Intell.*, 12(1-2):53–87, 1994. `doi:10.1007/BF01530761`.

8 Nikola Benes, Lubos Brim, Jakub Kadlecaj, Samuel Pastva, and David Safránek. AEON: attractor bifurcation analysis of parametrised Boolean networks. In *CAV*, pages 569–581. Springer, 2020. `doi:10.1007/978-3-030-53288-8_28`.

9 Nikola Benes, Lubos Brim, Samuel Pastva, David Safránek, and Eva Smijáková. Phenotype control of partially specified Boolean networks. In *CMSB*, pages 18–35. Springer, 2023. `doi:10.1007/978-3-031-42697-1_2`.

10 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

11 Peter Bloomingdale, Van Anh Nguyen, Jin Niu, and Donald E Mager. Boolean network modeling in systems pharmacology. *J. Pharmacokinet. Pharmacodyn.*, 45:159–180, 2018.

12 Florian Bridoux, Amélia Durbec, Kévin Perrot, and Adrien Richard. Complexity of fixed point counting problems in Boolean networks. *J. Comput. Syst. Sci.*, 126:138–164, 2022. `doi:10.1016/J.JCSS.2022.01.004`.

13 Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. A scalable approximate model counter. In Christian Schulte, editor, *CP*, pages 200–216. Springer, 2013. `doi:10.1007/978-3-642-40627-0_18`.

14 Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In *IJCAI*, pages 3569–3576. IJCAI/AAAI Press, 2016. URL: `http://www.ijcai.org/Abstract/16/503`.

15 Ashok K. Chandra and George Markowsky. On the number of prime implicants. *Discrete Math.*, 24(1):7–11, 1978. `doi:10.1016/0012-365X(78)90168-1`.

16 Stéphanie Chevalier, Vincent Noël, Laurence Calzone, Andrei Yu. Zinovyev, and Loïc Paulevé. Synthesis and simulation of ensembles of Boolean networks for cell fate decision. In *CMSB*, pages 193–209. Springer, 2020. `doi:10.1007/978-3-030-60327-4_11`.

17 Keith L. Clark. *Negation as Failure*, pages 293–322. Springer US, 1978. `doi:10.1007/978-1-4684-3384-5_11`.

18   Ridhwan Dewoprabowo, Johannes Klaus Fichte, Piotr Jerzy Gorczyca, and Markus Hecher. A practical account into counting Dung's extensions by dynamic programming. In *LPNMR*, pages 387–400. Springer, 2022. `doi:10.1007/978-3-031-15707-3_30`.

19   Yannis Dimopoulos, Wolfgang Dvořák, and Matthias König. Connecting abstract argumentation and Boolean networks. In *COMMA*, pages 85–96. IOS Press, 2024. `doi:10.3233/FAIA240312`.

20   Elena Dubrova and Maxim Teslenko. A SAT-based algorithm for finding attractors in synchronous Boolean networks. *IEEE ACM Trans. Comput. Biol. Bioinform.*, 8(5):1393–1399, 2011. `doi:10.1109/TCBB.2010.20`.

21   Thomas Eiter, Markus Hecher, and Rafael Kiesel. aspmc: New frontiers of algebraic answer set counting. *Artif. Intell.*, 330:104109, 2024. `doi:10.1016/J.ARTINT.2024.104109`.

22   François Fages. Consistency of Clark's completion and existence of stable models. *Methods Log. Comput. Sci.*, 1(1):51–60, 1994.

23   Johannes K. Fichte, Markus Hecher, and Mohamed A. Nadeem. Plausibility reasoning via projected answer set counting - a hybrid approach. In *IJCAI*, volume 22, pages 2620–2626, 2022. `doi:10.24963/IJCAI.2022/363`.

24   Johannes Klaus Fichte, Sarah Alice Gaggl, and Dominik Rusovac. Rushing and strolling among answer sets - navigation made easy. In *AAAI*, pages 5651–5659. AAAI Press, 2022. `doi:10.1609/AAAI.V36I5.20506`.

25   Johannes Klaus Fichte and Markus Hecher. Treewidth and counting projected answer sets. In *LPNMR*, pages 105–119. Springer, 2019. `doi:10.1007/978-3-030-20528-7_9`.

26   Johannes Klaus Fichte, Markus Hecher, and Arne Meier. Counting complexity for reasoning in abstract argumentation. *J. Artif. Intell. Res.*, 80, 2024. `doi:10.1613/JAIR.1.16210`.

27   Johannes Klaus Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. Answer set solving with bounded treewidth revisited. In *LPNMR*, pages 132–145. Springer, 2017. `doi:10.1007/978-3-319-61660-5_13`.

28   Stephan Fischer and Jesse Gillis. How many markers are needed to robustly determine a cell's type? *iScience*, 24(11):103292, November 2021.

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

30   Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP*, pages 1070–1080. MIT Press, 1988.

31   Jesse Heyninck, Matthias Knorr, and João Leite. Abstract dialectical frameworks are Boolean networks. In *LPNMR*, pages 98–111. Springer, 2024. `doi:10.1007/978-3-031-74209-5_8`.

32   Christopher M. Homan and Sven Kosub. Dichotomy results for fixed point counting in Boolean dynamical systems. *Theor. Comput. Sci.*, 573:16–25, 2015. `doi:10.1016/J.TCS.2015.01.040`.

33   Katsumi Inoue. Logic programming for Boolean networks. In *IJCAI*, pages 924–930. IJCAI/AAAI, 2011. `doi:10.5591/978-1-57735-516-8/IJCAI11-160`.

34   Katsumi Inoue and Chiaki Sakama. Oscillating behavior of logic programs. In *Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, pages 345–362. Springer, 2012. `doi:10.1007/978-3-642-30743-0_23`.

35   Tomi Janhunen. Some (in)translatability results for normal logic programs and propositional theories. *J. Appl. Non Class. Logics*, 16(1-2):35–86, 2006. `doi:10.3166/JANCL.16.35-86`.

36   Tomi Janhunen, Ilkka Niemelä, Dietmar Seipel, Patrik Simons, and Jia-Huai You. Unfolding partiality and disjunctions in stable model semantics. *ACM TOCL*, 7(1):1–37, 2006. `doi:10.1145/1119439.1119440`.

37   Mohimenul Kabir, Supratik Chakraborty, and Kuldeep S. Meel. Exact ASP counting with compact encodings. In *AAAI*, pages 10571–10580. AAAI Press, 2024. `doi:10.1609/AAAI.V38I9.28927`.

38 Mohimenul Kabir, Flavio O. Everardo, Ankit K. Shukla, Markus Hecher, Johannes Klaus Fichte, and Kuldeep S. Meel. ApproxASP - a scalable approximate answer set counter. In *AAAI*, pages 5755–5764. AAAI Press, 2022. `doi:10.1609/AAAI.V36I5.20518`.

39 Mohimenul Kabir and Kuldeep S Meel. A fast and accurate ASP counting based network reliability estimator. In *LPAR*, volume 94, pages 270–287, 2023. `doi:10.29007/KC6Q`.

40 Mohimenul Kabir and Kuldeep S Meel. On lower bounding minimal model count. *TPLP*, 24(4):586–605, July 2024. `doi:10.1017/S147106842400036X`.

41 Nikolai Käfer, Christel Baier, Martin Diller, Clemens Dubslaff, Sarah Alice Gaggl, and Holger Hermanns. Admissibility in probabilistic argumentation. *J. Artif. Intell. Res.*, 74, 2022. `doi:10.1613/JAIR.1.13530`.

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

43 Hiroaki Kitano. Towards a theory of biological robustness. *Mol. Syst. Biol.*, 3(1):137, 2007.

44 Hannes Klarner, Alexander Bockmayr, and Heike Siebert. Computing maximal and minimal trap spaces of Boolean networks. *Nat. Comput.*, 14(4):535–544, 2015. `doi:10.1007/S11047-015-9520-7`.

45 Hannes Klarner, Frederike Heinitz, Sarah Nee, and Heike Siebert. Basins of attraction, commitment sets, and phenotypes of Boolean networks. *IEEE ACM Trans. Comput. Biol. Bioinform.*, 17(4):1115–1124, 2020. `doi:10.1109/TCBB.2018.2879097`.

46 Hannes Klarner, Elisa Tonello, Laura Fontanals, Florence Janody, Claudine Chaouiya, and Heike Siebert. Detection of markers for discrete phenotypes. In *CSBio*, pages 64–68, 2021.

47 Joohyung Lee and Vladimir Lifschitz. Loop formulas for disjunctive logic programs. In *ICLP*, pages 451–465. Springer, 2003. `doi:10.1007/978-3-540-24599-5_31`.

48 Qin Li, Anders Wennborg, Erik Aurell, Erez Dekel, Jie-Zhi Zou, Yuting Xu, Sui Huang, and Ingemar Ernberg. Dynamics inside the cancer cell attractor reveal cell heterogeneity, limits of stability, and escape. *Proc. Natl. Acad. Sci. U.S.A.*, 113(10):2672–2677, February 2016.

49 Thomas Linsbichler, Marco Maratea, Andreas Niskanen, Johannes Peter Wallner, and Stefan Woltran. Advanced algorithms for abstract dialectical frameworks based on complexity analysis of subclasses and SAT solving. *Artif. Intell.*, 307:103697, 2022. `doi:10.1016/J.ARTINT.2022.103697`.

50 Victor W Marek and Miroslaw Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm*, pages 375–398. Springer, 1999. `doi:10.1007/978-3-642-60085-2_17`.

51 Gabriele Masina, Giuseppe Spallitta, and Roberto Sebastiani. On CNF conversion for disjoint SAT enumeration. In *SAT*, pages 15:1–15:16, 2023. `doi:10.4230/LIPICS.SAT.2023.15`.

52 Arnau Montagud, Jonas Béal, Luis Tobalina, Pauline Traynard, Vigneshwari Subramanian, Bence Szalai, Róbert Alföldi, László Puskás, Alfonso Valencia, Emmanuel Barillot, et al. Patient-specific Boolean models of signalling networks guide personalised treatments. *Elife*, 11:e72626, 2022.

53 Alessandro Padoa. Essai d'une théorie algébrique des nombres entiers, précédé d'une introduction logique à une théorie déductive quelconque. *Bibliothèque du Congrès International de Philosophie*, 3:309–365, 1901.

54 Samuel Pastva, David Šafránek, Nikola Beneš, Luboš Brim, and Thomas Henzinger. Repository of logically consistent real-world Boolean network models. *bioRxiv*, 2023. URL: `https://www.biorxiv.org/content/early/2023/06/12/2023.06.12.544361`.

55 Loïc Paulevé, Juraj Kolčák, Thomas Chatain, and Stefan Haar. Reconciling qualitative, abstract, and scalable modeling of biological networks. *Nat. Commun.*, 11(1):1–7, August 2020.

56 Jordan C. Rozum, Jorge Gómez Tejeda Zañudo, Xiao Gan, Dávid Deritei, and Réka Albert. Parity and time reversal elucidate both decision-making in empirical models and attractor scaling in critical Boolean networks. *Sci. Adv.*, 7(29):eabf8124, July 2021.

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

58   Julian D Schwab, Silke D Kühlwein, Nensi Ikonomi, Michael Kühl, and Hans A Kestler. Concepts in Boolean network modeling: What do they all mean? *Comput. Struct. Biotechnol. J.*, 18:571–582, 2020.

59   Robert Schwieger and Elisa Tonello. Reduction for asynchronous Boolean networks: elimination of negatively autoregulated components. *Discret. Math. Theor. Comput. Sci.*, 25(2), 2023. `doi:10.46298/DMTCS.10930`.

60   Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S. Meel. GANAK: A scalable probabilistic exact model counter. In *IJCAI*, pages 1169–1176. ijcai.org, 2019. `doi:10.24963/IJCAI.2019/163`.

61   Ilya Shmulevich, Edward R. Dougherty, and Wei Zhang. From Boolean to probabilistic Boolean networks as models of genetic regulatory networks. *Proc. IEEE*, 90(11):1778–1792, 2002. `doi:10.1109/JPROC.2002.804686`.

62   Ilya Shmulevich, Edward R Dougherty, and Wei Zhang. Gene perturbation and intervention in probabilistic Boolean networks. *Bioinf.*, 18(10):1319–1331, 2002.

63   Cui Su and Jun Pang. Sequential temporary and permanent control of Boolean networks. In *CMSB*, pages 234–251. Springer, 2020. `doi:10.1007/978-3-030-60327-4_13`.

64   Marc Thurley. sharpSAT - counting models with advanced component caching and implicit BCP. In *SAT*, pages 424–429. Springer, 2006. `doi:10.1007/11814948_38`.

65   Elisa Tonello and Loïc Paulevé. Phenotype control and elimination of variables in Boolean networks. *Peer Community Journal*, 4, August 2024.

66   Predrag T. Tosic. On the complexity of counting fixed points and gardens of eden in sequential dynamical systems on planar bipartite graphs. *Int. J. Found. Comput. Sci.*, 17(5):1179–1204, 2006. `doi:10.1142/S0129054106004339`.

67   Predrag T. Tosic and Gul A. Agha. On computational complexity of counting fixed points in symmetric Boolean graph automata. In *UC*, pages 191–205. Springer, 2005. `doi:10.1007/11560319_18`.

68   Van-Giang Trinh, Belaid Benhamou, Samuel Pastva, and Sylvain Soliman. Scalable enumeration of trap spaces in Boolean networks via answer set programming. In *AAAI*, pages 10714–10722. AAAI Press, 2024. `doi:10.1609/AAAI.V38I9.28943`.

69   Van-Giang Trinh, Belaid Benhamou, and Loïc Paulevé. mpbn: a simple tool for efficient edition and analysis of elementary properties of Boolean networks. *CoRR*, abs/2403.06255, 2024. `doi:10.48550/arXiv.2403.06255`.

70   Van-Giang Trinh, Belaid Benhamou, and Vincent Risch. Graphical analysis of abstract argumentation frameworks via Boolean networks. In *ICAART*, pages 745–756, 2025. `doi:10.5220/0013346400003890`.

71   Van-Giang Trinh, Belaid Benhamou, and Sylvain Soliman. Efficient enumeration of fixed points in complex Boolean networks using answer set programming. In *CP*, pages 35:1–35:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/LIPICS.CP.2023.35`.

72   Van-Giang Trinh, Belaid Benhamou, and Sylvain Soliman. Trap spaces of Boolean networks are conflict-free siphons of their Petri net encoding. *Theor. Comput. Sci.*, 971:114073, September 2023. `doi:10.1016/J.TCS.2023.114073`.

73   Van-Giang Trinh, Belaid Benhamou, Sylvain Soliman, and François Fages. Graphical conditions for the existence, unicity and number of regular models. In *ICLP*, pages 175–187, 2024. `doi:10.4204/EPTCS.416.16`.

74   Van-Giang Trinh, Kunihiko Hiraishi, and Belaid Benhamou. Computing attractors of large-scale asynchronous Boolean networks using minimal trap spaces. In *ACM-BCB*, pages 13:1–13:10. ACM, 2022. `doi:10.1145/3535508.3545520`.

75   Van-Giang Trinh, Kyu Hyong Park, Samuel Pastva, and Jordan C Rozum. Mapping the attractor landscape of Boolean networks with biobalm. *Bioinformatics*, 41(5):btaf280, 2025.

76   Santiago Videla, Carito Guziolowski, Federica Eduati, Sven Thiele, Martin Gebser, Jacques
        Nicolas, Julio Saez-Rodriguez, Torsten Schaub, and Anne Siegel. Learning Boolean logic
        models of signaling networks with ASP. *Theor. Comput. Sci.*, 599:79–101, 2015. `doi:`
        `10.1016/J.TCS.2014.06.022`.
77   Jiong Yang and Kuldeep S Meel. Rounding meets approximate model counting. In *CAV*,
        pages 132–162. Springer, 2023. `doi:10.1007/978-3-031-37703-7_7`.

A   Details of Proofs

▶ **Theorem 10.** *Given a BN f and a phenotype β, the minimal trap spaces of f satisfying β
one-to-one correspond to the answer sets of $\mathsf{P\text{-}tsconj}(f) \cup \mathsf{ToASP}(\beta)$.*

Proof. To prove the correctness of Theorem 10, we reuse the correctness proof of Theorem 2
of [68], which establishes that the answer sets of $\mathsf{P\text{-}tsconj}(f)$ one-to-one correspond to the
minimal trap spaces of f. Let us recall the translation between an answer set M of $\mathsf{P\text{-}tsconj}(f)$
and its respective minimal trap space m of f: for each variable $v \in \mathsf{Var}(f)$, $m(v) = 1$ if and
only if $\mathsf{p}(v) \in M \wedge \mathsf{n}(v) \notin M$, $m(v) = 0$ if and only if $\mathsf{p}(v) \notin M \wedge \mathsf{n}(v) \in M$, and $m(v) = \star$ if
and only if $\mathsf{p}(v) \in M \wedge \mathsf{n}(v) \in M$.

We employ the theories of faceted answer set navigation [24] to prove the correctness
of Theorem 10. According to faceted navigation, for a program P and an atom $a \in \mathsf{at}(P)$,
adding the integrity constraint $\{\bot \leftarrow a\}$ to program P restricts the search space of P, where
no answer set contains the atom a. Conversely, adding the integrity constraint $\{\bot \leftarrow \sim a\}$ to
program P ensures that every answer set in the modified program contains the atom a.

We prove the correctness of our encoding for each trait $(v \leftrightarrow e) \in \beta$. We show that Algo-
rithm 1 selects *facets* in such a way that the answer sets of $\mathsf{P\text{-}tsconj}(f) \cup \mathsf{ToASP}(\beta)$ one-to-one
correspond to the minimal trap spaces satisfying β of f. When $(v \leftrightarrow 1) \in \beta$ (line 4 in Algo-
rithm 1), two constraints are added to $\mathsf{ToASP}(\beta)$ to ensure that every answer set contains the
atom $\mathsf{p}(v)$ and none contains the atom $\mathsf{n}(v)$. These constraints effectively assign the value 1
to variable v. When $(v \leftrightarrow 0) \in \beta$ (line 6 in Algorithm 1), two added constraints ensure that
every answer set contains the atom $\mathsf{n}(v)$ and none contains $\mathsf{p}(v)$, thereby assigning the value
0 to variable v. When $(v \leftrightarrow \star) \in \beta$ (line 8 in Algorithm 1), two added constraints ensure
that every answer set contains both $\mathsf{p}(v)$ and $\mathsf{n}(v)$. These constraints effectively assign the
value $\star$ to variable v.
Combining all the cases of Algorithm 1, we can claim the correctness of our encoding.   ◀

▶ **Theorem 11.** *Given a BN f and a phenotype β, the fixed points of f satisfying β one-to-one
correspond to the answer sets of $\mathsf{P\text{-}fASP}(f) \cup \mathsf{ToASP}(\beta)$.*

Proof. The proof technique of Theorem 10 can be similarly extended for fixed point counting
with the program $\mathsf{P\text{-}fASP}(f)$.   ◀

To prove Theorems 14 and 15, we prepare the following preliminaries.

▶ **Definition 17.** *The total order $\leq_t$ on $\mathbb{B}_\star$ is defined by $0 <_t \star <_t 1$.*

▶ **Definition 18.** *The partial order $\leq_s$ on $\mathbb{B}_\star$ is defined by $0 <_s \star$, $1 <_s \star$, and it contains
no other relation.*

▶ **Definition 19.** *Consider a BN f, a sub-space m of f, and a Boolean expression e over
$\mathsf{Var}(f)$. The value of e under sub-space m w.r.t. the Kleene three-valued logic, denoted as
$m(e)$, is recursively defined as follows:*

$$m(e) = \begin{cases} e & \text{if } e \in \mathbb{B}_\star \\ m(a) & \text{if } e = a, a \in \mathsf{Var}(f) \\ \neg m(e_1) & \text{if } e = \neg e_1 \\ min_{\leq_t}(m(e_1), m(e_2)) & \text{if } e = e_1 \wedge e_2 \\ max_{\leq_t}(m(e_1), m(e_2)) & \text{if } e = e_1 \vee e_2 \end{cases}$$

where $\neg 1 = 0, \neg 0 = 1, \neg\star = \star$*, and* $min_{\leq_t}$ *(resp.* $max_{\leq_t}$*) is the function to get the minimum (resp. maximum) value of two values w.r.t. the order* $\leq_t$*.*

▶ **Theorem 20** (Theorem 1 of [44])**.** *Consider a BN* f *and a sub-space* m *of* f*. A sub-space* m *is a trap space of* f *iff* $m(f_v) \leq_s m(v)$ *for every* $v \in \mathsf{Var}(f)$*.*

▶ **Corollary 21.** *Consider a BN* f *and a sub-space* m *of* f*. A sub-space* m *is a minimal trap space of* f *iff* m *is a* $\leq_s$*-minimal trap space of* f*.*

▶ **Corollary 22.** *Consider a BN* f *and a sub-space* m *of* f*. A sub-space* m *is a fixed point of* f *iff* m *is a trap space of* f *and* $m(v) \neq \star$ *for every* $v \in \mathsf{Var}(f)$*.*

Now, we show the formal proof of Theorem 14.

▶ **Lemma 23.** *Consider a BN* f *and a set of perturbable variables* $\mathcal{X} \subseteq \mathsf{Var}(f)$*. Let* g *be the BN obtained from* f*, according to Definition 13. Let* Δ *be the set* $\bigcup_{v \in \mathcal{X}} \{v^k, v^o\}$*. If* m *is a minimal trap space of* g*, then* $m(v) \neq \star$ *for every* $v \in \Delta$*.*

Proof. Let m be a minimal trap space of g. Assume that there exists $v^k \in \Delta$ such that $m(v^k) = \star$ or $v^o \in \Delta$ such that $m(v^o) = \star$. We consider two cases as follows.

Case 1: $m(v^k) = \star$**.** Let m' be a sub-space of g such that $m'(u) = m(u)$, for every $u \in \mathsf{Var}(g) \setminus \{v^k\}$ and $m'(v^k) = 0$. We have $m'(g_{v^k}) = m'(v^k)$. The variable v^k only affects v^o and v. Regarding v^o, we have $m'(g_{v^o}) = m'(v^o \wedge \neg v^k) = min_{\leq_t}(m'(v^o), \neg m'(v^k)) = min_{\leq_t}(m'(v^o), 1) = m'(v^o)$. Regarding v, we have $m'(g_v) = m'(\neg v^k \wedge (v^o \vee f_v)) = min_{\leq_t}(\neg m'(v^k), max_{\leq_t}(m'(v^o), m'(f_v))) = min_{\leq_t}(1, max_{\leq_t}(m'(v^o), m'(f_v))) = max_{\leq_t}(m'(v^o), m'(f_v)) = max_{\leq_t}(m(v^o), m(f_v))$. Following Theorem 20, $m(g_v) \leq_s m(v)$. We have $m(g_v) = m(\neg v^k \wedge (v^o \vee f_v)) = min_{\leq_t}(\neg m(v^k), max_{\leq_t}(m(v^o), m(f_v))) = min_{\leq_t}(\star, max_{\leq_t}(m(v^o), m(f_v)))$. Since $m(v) <_s max_{\leq_t}(m(v^o), m(f_v))$ implies $m(v) <_s m(g_v)$ which is a contradiction, we derive that $max_{\leq_t}(m(v^o), m(f_v)) \leq_s m(v)$. This implies $m'(g_v) \leq_s m(v) = m'(v)$. For any $u \in \mathsf{Var}(g) \setminus \{v, v^k, v^o\}$, $m'(g_u) = m(g_u) \leq_s m(u) = m'(u)$. Hence, m' is trap space of g and $m' <_s m$, which is a contradiction.

Case 2: $m(v^o) = \star$**.** Let m' be a sub-space of g such that $m'(u) = m(u)$ for every $u \in \mathsf{Var}(g) \setminus \{v^o\}$ and $m'(v^o) = 0$. We have $m'(g_{v^o}) = m'(v^o \wedge \neg v^k) = min_{\leq_t}(m'(v^o), \neg m'(v^k)) = min_{\leq_t}(0, \neg m'(v^k)) = 0 = m'(v^o)$. The variable v^o only affects v. We have $m'(g_v) = m'(\neg v^k \wedge (v^o \vee f_v)) = min_{\leq_t}(\neg m'(v^k), max_{\leq_t}(m'(v^o), m'(f_v))) = min_{\leq_t}(\neg m'(v^k), max_{\leq_t}(0, m'(f_v))) = min_{\leq_t}(\neg m'(v^k), m'(f_v)) = min_{\leq_t}(\neg m(v^k), m(f_v))$. Following Theorem 20, $m(g_v) \leq_s m(v)$. We have $m(g_v) = m(\neg v^k \wedge (v^o \vee f_v)) = min_{\leq_t}(\neg m(v^k), max_{\leq_t}(m(v^o), m(f_v))) = min_{\leq_t}(\neg m(v^k), max_{\leq_t}(\star, m(f_v)))$. Suppose that $m'(v) = m(v) <_s min_{\leq_t}(\neg m(v^k), m(f_v))$. If $m(f_v) = 0$, then $m(v) <_s 0$ which contradicts to the definition of $\leq_s$. Hence $m(f_v) \neq 0$, leading to $max_{\leq_t}(\star, m(f_v)) = m(f_v)$. Then $m(v) <_s min_{\leq_t}(\neg m(v^k), m(f_v)) = m(g_v)$, which is a contradiction. Hence, $min_{\leq_t}(\neg m(v^k), m(f_v)) \leq_s m'(v)$. This implies that $m'(g_v) \leq_s m'(v)$. For any $u \in \mathsf{Var}(g) \setminus \{v, v^o\}$, $m'(g_u) = m(g_u) \leq_s m(u) = m'(u)$. Hence, m' is trap space of g and $m' <_s m$, which is a contradiction.

Combining Case 1 and Case 2, we can conclude that for $m(v) \neq \star$ for every $v \in \Delta$. ◀

▶ **Lemma 24.** *Consider a BN f and a set of perturbable variables $\mathcal{X} \subseteq \mathsf{Var}(f)$. Let g be the BN obtained from f, according to Definition 13 and Δ be the set $\bigcup_{v \in \mathcal{X}} \{v^k, v^o\}$. Let m be a trap space of g such that $m(v) \neq \star$ for every $v \in \Delta$. Let σ be the perturbation of f such that $\sigma(v) = 1$ if and only if $m(v^k) = 0$ and $m(v^o) = 1$, $\sigma(v) = 0$ if and only if $m(v^k) = 1$ and $m(v^o) = 0$, and $\sigma(v) = \star$ if and only if $m(v^k) = 0$ and $m(v^o) = 0$. Then m' is a trap space of f^σ where $m'(v) = m(v)$ for every $v \in \mathsf{Var}(f)$.*

Proof. Assume that there exists $v \in \mathcal{X}$ such that $m(v^k) = m(v^o) = 1$. Since m is a trap space of g, $m(g_{v^o}) \leq_s m(v^o)$. We have $m(g_{v^o}) = m(v^o \wedge \neg v^k) = 0$, whereas $m(v^o) = 1$, leading to $0 \leq_s 1$, which contradicts to the definition of $\leq_s$. Hence, the case of $m(v^k) = m(v^o) = 1$ is impossible for every $v \in \mathcal{X}$. Since $m(v) \neq \star$ for every $v \in \Delta$, σ is well specified.

Recall that $\mathsf{Var}(g) = \mathsf{Var}(f) \cup \Delta$ and $\mathsf{Var}(f) = \mathsf{Var}(f^\sigma)$. Consider $v \in \mathsf{Var}(f^\sigma)$. If $v \notin \mathcal{X}$, we have $m'(f_v^\sigma) = m'(f_v) = m'(g_v) = m(g_v) \leq_s m(v) = m'(v)$. If $v \in \mathcal{X}$, we have the following cases:

Case 1: $m(v^k) = 0$ **and** $m(v^o) = 0$. Then $\sigma(v) = \star$, thus $m'(f_v^\sigma) = m'(f_v)$. We have $m(g_v) = m(\neg v^k \wedge (v^o \vee f_v)) = m(f_v) \leq_s m(v) = m'(v)$. Since $m'(f_v) = m(f_v)$, it follows that $m'(f_v^\sigma) \leq_s m'(v)$.

Case 2: $m(v^k) = 1$ **and** $m(v^o) = 0$. Then $\sigma(v) = 0$, thus $m'(f_v^\sigma) = 0$. We have $m(g_v) = m(\neg v^k \wedge (v^o \vee f_v)) = 0 \leq_s m(v) = m'(v)$. Hence, $m'(f_v^\sigma) \leq_s m'(v)$.

Case 3: $m(v^k) = 0$ **and** $m(v^o) = 1$. Then $\sigma(v) = 1$, thus $m'(f_v^\sigma) = 1$. We have $m(g_v) = m(\neg v^k \wedge (v^o \vee f_v)) = 1 \leq_s m(v) = m'(v)$. Hence, $m'(f_v^\sigma) \leq_s m'(v)$.

Now we can conclude that $m'(f_v^\sigma) \leq_s m'(v)$ for every $v \in \mathsf{Var}(f^\sigma)$. Hence, m' is a trap space of f^σ. ◀

▶ **Theorem 14.** *Given a BN f, a set of perturbable variables $\mathcal{X} \subseteq \mathsf{Var}(f)$, a phenotype β, and $\Omega = \bigcup_{v \in \Delta} \{\mathsf{p}(v), \mathsf{n}(v)\}$, where $\Delta = \bigcup_{v \in \mathcal{X}} \{v^k, v^o\}$, then C-MTS-3 can be computed as $\#\mathsf{PASP}(\mathsf{P\text{-}tsconj}(g) \cup \mathsf{ToASP}(\beta), \Omega)$, where g is the new BN according to Definition 13.*

Proof. First, we consider a perturbation $\sigma \colon \mathcal{X} \to \mathbb{B}_\star$ of BN f. Let $m_\Delta \colon \Delta \to \mathbb{B}$ be a mapping such that for every $v \in \mathcal{X}$, $\sigma(v) = 1$ if and only if $m_\Delta(v^k) = 0$ and $m_\Delta(v^o) = 1$, $\sigma(v) = 0$ if and only if $m_\Delta(v^k) = 1$ and $m_\Delta(v^o) = 0$, and $\sigma(v) = \star$ if and only if $m_\Delta(v^k) = 0$ and $m_\Delta(v^o) = 0$. Recall that $\mathsf{Var}(g) = \mathsf{Var}(f) \cup \Delta$ and $\mathsf{Var}(f) = \mathsf{Var}(f^\sigma)$. Let m be a minimal trap space of f^σ and $m \models \beta$. Let m' be a sub-space of g such that $m'(v) = m(v)$ if $v \in \mathsf{Var}(f)$ and $m'(v) = m_\Delta(v)$ if $v \in \Delta$. We show that m' is a minimal trap space of g and $m' \models \beta$ (1).

Consider $v \in \mathcal{X}$, we have $m'(g_{v^k}) = m'(v^k)$. If $m'(v^k) = 0$, then $m'(v^o \wedge \neg v^k) = m'(v^o)$. If $m'(v^k) = 1$, then $m'(v^o) = 0$ due to the definition of m_Δ, leading to $m'(v^o \wedge \neg v^k) = 0 = m'(v^o)$. Hence, we can derive that $m'(g_{v^o}) = m'(v^o \wedge \neg v^k) = m'(v^o)$. Regarding $m'(g_v) = m'(\neg v^k \wedge (v^o \vee f_v))$, we have the following cases:

Case 1: $\sigma(v) = \star$. Then $m'(v^k) = 0$ and $m'(v^o) = 0$. We have $m'(g_v) = m'(f_v) = m(f_v) = m(f_v^\sigma) \leq_s m(v) = m'(v)$.

Case 2: $\sigma(v) = 1$. Then $m'(v^k) = 0$ and $m'(v^o) = 1$. We have $m'(g_v) = 1 = m(f_v^\sigma) \leq_s m(v) = m'(v)$.

Case 3: $\sigma(v) = 0$. Then $m'(v^k) = 1$ and $m'(v^o) = 0$. We have $m'(g_v) = 0 = m(f_v^\sigma) \leq_s m(v) = m'(v)$. Consider $v \in \mathsf{Var}(f) \setminus \mathcal{X}$. We have $m'(g_v) = m'(f_v) = m(f_v) = m(f_v^\sigma) \leq_s m(v) = m'(v)$.

Now, we can conclude that m' is a trap space of g. Assume that m' is not minimal. Then there is a trap space n of g such that $n <_s m'$. Since $m'(v) \neq \star$ for every $v \in \Delta$, $n(v) = m'(v)$ for every $v \in \Delta$, leading to $n(v) \neq \star$ for every $v \in \Delta$. Following the Lemma 24, n' is a trap

space of f^σ where $n'(v) = n(v)$ for every $v \in \mathsf{Var}(f)$. Since $n(v) = m'(v)$ for every $v \in \Delta$, we have $n' <_s m$, which contradicts to the $\leq_s$-minimality of m w.r.t. f^σ. Hence, m' is a minimal trap space of g. In addition, since β only contains the variables in $\mathsf{Var}(f)$, it is trivial that $m' \models \beta$.

Second, we consider a minimal trap space m of g such that $m \models \beta$. By Lemma 23, $m(v) \neq \star$ for every $v \in \Delta$. The case of $m(v^k) = m(v^o) = 1$ is impossible because if it holds, then $m(g_{v^o}) = m(v^o \wedge \neg v^k) = 0 \leq_s m(v^o) = 1$, which is a contradiction. Let σ be the perturbation of f such that $\sigma(v) = 1$ if and only if $m(v^k) = 0$ and $m(v^o) = 1$, $\sigma(v) = 0$ if and only if $m(v^k) = 1$ and $m(v^o) = 0$, and $\sigma(v) = \star$ if and only if $m(v^k) = 0$ and $m(v^o) = 0$. Let m' be a sub-space of f such that $m'(v) = m(v)$ for every $v \in \mathsf{Var}(f)$. We show that m' is a minimal trap space of f^σ and $m' \models \beta$ (2).

By Lemma 24, m' is a trap space of f^σ. Assume that m' is not minimal. Then there is a minimal trap space n of f^σ such that $n <_s m'$. Let n' be a sub-space of g such that $n'(v) = m(v)$ for every $v \in \Delta$ and $n'(v) = n(v)$ for every $v \in \mathsf{Var}(f)$. By following the same reasoning for (1), we have n' is a trap space of g. However, $n' <_s m$, which contradicts to the $\leq_s$-minimality of m w.r.t. g. Hence, m' is a minimal trap space of f^σ. In addition, since β only contains the variables in $\mathsf{Var}(f)$, it is trivial that $m' \models \beta$.

From (1) and (2), we can conclude that, given $f, \mathcal{X}$, and β, the result of the counting problem C-MTS-3 is equivalent to the number of minimal trap spaces of g that satisfy β where multiple minimal trap spaces with the same values on the variables in Δ are only counted once. By Theorem 10, the answer sets of $\mathsf{P\text{-}tsconj}(g) \cup \mathsf{ToASP}(\beta)$ one-to-one correspond to the minimal trap spaces of g satisfying β. The set Ω includes the atoms of $\mathsf{P\text{-}tsconj}(g) \cup \mathsf{ToASP}(\beta)$ corresponding to the variables in Δ of g. It follows that the number of answer sets of $\mathsf{P\text{-}tsconj}(g) \cup \mathsf{ToASP}(\beta)$ projected to Ω is equal to the number of minimal trap spaces of g satisfying β projected to Δ. This implies that C-MTS-3 can be computed as the projected answer set counting query $\#\mathsf{PASP}(\mathsf{P\text{-}tsconj}(g) \cup \mathsf{ToASP}(\beta), \Omega)$.   ◀

Finally, we show the formal proof of Theorem 15.

▶ **Theorem 15.** *Given a BN f, a set of perturbable variables $\mathcal{X} \subseteq \mathsf{Var}(f)$, a phenotype β, and $\Omega = \bigcup_{v \in \Delta} \{\mathsf{p}(v), \mathsf{n}(v)\}$, where $\Delta = \bigcup_{v \in \mathcal{X}} \{v^k, v^o\}$, then C-FIX-3 can be computed as $\#\mathsf{PASP}(\mathsf{P\text{-}fASP}(g) \cup \mathsf{ToASP}(\beta), \Omega)$, where g is the new BN following Definition 13.*

Proof. The proof technique of Theorem 14 can be similarly extended for C-FIX-3 with the program $\mathsf{P\text{-}fASP}(g)$.   ◀

B   Details of Example ASP Programs

The ASP program $\mathsf{P\text{-}tsconj}(f)$ considered in Example 16 is:

$\mathsf{p}(a) \vee \mathsf{n}(a) \leftarrow \top \qquad \mathsf{p}(a) \leftarrow \mathsf{p}(a), \mathsf{n}(b) \qquad \mathsf{n}(a) \leftarrow \mathsf{aux}_1 \qquad \mathsf{aux}_1 \leftarrow \mathsf{n}(a) \qquad \mathsf{aux}_1 \leftarrow \mathsf{p}(b)$

$\mathsf{p}(b) \vee \mathsf{n}(b) \leftarrow \top$

$\mathsf{p}(b) \leftarrow \mathsf{n}(b^k), \mathsf{aux}_2 \qquad \mathsf{aux}_2 \leftarrow \mathsf{p}(b^o) \qquad \mathsf{aux}_2 \leftarrow \mathsf{p}(a)$

$\mathsf{n}(b) \leftarrow \mathsf{aux}_3 \qquad \mathsf{aux}_3 \leftarrow \mathsf{p}(b^k) \qquad \mathsf{aux}_3 \leftarrow \mathsf{n}(b^o), \mathsf{n}(a)$

$\mathsf{p}(b^k) \vee \mathsf{n}(b^k) \leftarrow \top \qquad \mathsf{p}(b^k) \leftarrow \mathsf{p}(b^k) \qquad \mathsf{n}(b^k) \leftarrow \mathsf{n}(b^k)$

$\mathsf{p}(b^o) \vee \mathsf{n}(b^o) \leftarrow \top \qquad \mathsf{p}(b^o) \leftarrow \mathsf{p}(b^o), \mathsf{n}(b^k) \qquad \mathsf{n}(b^o) \leftarrow \mathsf{aux}_4 \quad \mathsf{aux}_4 \leftarrow \mathsf{n}(b^o) \quad \mathsf{aux}_4 \leftarrow \mathsf{p}(b^k)$

The ASP program $\mathsf{ToASP}(\beta)$ considered in Example 16 is:

$$\perp \leftarrow \mathsf{p}(a) \qquad \perp \leftarrow \sim\!\mathsf{n}(a)$$
$$\perp \leftarrow \mathsf{p}(b) \qquad \perp \leftarrow \sim\!\mathsf{n}(b)$$

C   Details of Benchmark Instances

Our benchmarks are based on the 245 real-world biological models from the BBM dataset [54] available at the time of writing, plus 400 random Boolean networks that were first tested as part of [68]. The real-world Boolean models range up to 1076 variables, out of which up to 223 are source variables. However, the median network size in this dataset is less than 100 variables. As such, we also consider larger, random Boolean networks ranging between 1,000 and 5,000 variables. All source variables across all networks are left unrestricted, meaning they can take the value 0 or 1, thereby maximizing the number of admissible trap spaces.

These 645 instances are used directly as inputs for benchmarking the C-FIX-1 and C-MTS-1 problems. To test C-FIX-2 and C-MTS-2, we augment each network with a pseudo-random phenotype specification. Here, the specification is chosen as follows: we first compute an arbitrary, fixed minimal trap space using `tsconj`. We then randomly select the values of three fixed variables – excluding all the source variables of the network. The conjunction of these values represents the tested phenotype. This process ensures that for each Boolean network, problem C-MTS-2 always has at least one valid minimal trap space solution (existence of a fixed point solution cannot be guaranteed regardless of the chosen phenotype).

Finally, to evaluate C-FIX-3 and C-MTS-3, we use the same phenotype but also pseudo-randomly select up to 50 perturbable variables, excluding both the source variables and those fixed by the phenotype. For networks with less than 50 such candidates, we simply select all viable variables as perturbable. We then use the transformation proposed in Definition 13 to construct a new variant of each benchmark network in which the selected variables can be perturbed. Each such perturbed network, together with the phenotype, represents the input for C-FIX-3 and C-MTS-3.

D   Phenotype robustness analysis, case study of Interferon 1 model

Model no. 118 in the BBM dataset [54] represents an interaction network related to the activation of the so-called *Interferon 1*, a biochemical species closely tied to immune response present in T-cells. The model was initially derived using [2] and then later tuned by domain experts to correctly capture relevant biological phenotypes. It consists of 121 variables, of which 55 are "inputs", meaning they are not regulated by other variables.

The model defines three phenotype variables, *ISG* (full name *ISG expression antiviral response phenotype*), *PCK* (full name *Proinflammatory cytokine expression inflammation phenotype*), and *IFN* (full name *Type 1 IFN response phenotype*). As these are separate output variables of the network, each trap space can exhibit any combination of active and inactive phenotype variables.

Since the model defines response of T-cells to immunological stimuli and environmental factors, it is important to understand how these mechanisms respond to potential permanent perturbations, either due to genetic mutations or therapeutic treatments. Here, we provide an overview of the model phenotypes through the lens of phenotype robustness.

■ **Table 7** Phenotype robustness analysis for the Interferon 1 model. First three columns describe the desired phenotype (− meaning the value is unconstrained). The C-MTS-3 column lists the number of perturbations for which said phenotype appears in the network. Finally, robustness r indicates what portion of the possible perturbations still enable the corresponding phenotype.

ISG	PCK	IFN	C-MTS-3	Robustness (r)
1	−	−	3486784401	1.000
−	1	−	2114072298	0.606
−	−	1	2313362673	0.663
0	0	0	478296900	0.137
1	0	0	621785970	0.178
0	1	0	478296900	0.137
0	0	1	813104730	0.233
1	1	0	782989740	0.224
1	0	1	1096362783	0.314
0	1	1	813104730	0.233
1	1	1	1409735826	0.404

For simplicity, we selected 20 variables of the model as potential perturbation targets. In reality, this choice would be further influenced by known genetic risk factors or drug targets. Consequently, this choice results in $3^{20} = 3486784401$ admissible perturbations in our Boolean system. We then consider two phenotype variants: First, where a single phenotype variable is expected to be 1 and the remaining phenotype variables are unconstrained (i.e. they can be 0, 1, or $\star$), yielding three combinations. And second, more specific one, where each phenotype variable is fixed to either 1 or 0, yielding eight combinations.

The results for C-MTS-3 and the subsequent robustness computation are given in Table 7. Here, we can notice several biologically interesting outcomes:

- Regardless of perturbation, the network always exhibits a trap space with the *ISG* phenotype. The remaining phenotypes are usually also present ($r = 0.606$ and $r = 0.663$), but are significantly less robust than *ISG*. This indicates that (assuming favorable environmental conditions), expression of ISG as a response to viral activity is robust and cannot be disrupted by a perturbation.
- Among the fully defined phenotypes, 010 and 000 are the least robust while 111 is the most robust. This shows the general tendency of T-cells to reliably and consistently respond to immunological stimuli, as the 111 phenotype indicates the maximal level of immunological activity.
- Even though *ISG* is the most robust phenotype when taken in isolation, phenotypes with active *IFN* generally achieve higher robustness when other phenotypes are required to be inactive.

Such outcomes serve several functions: First, they can be used to validate (or refute) assumptions about the biological tendencies of the studied system. Second, if observations do not match model predictions, better quantitative understanding can provide possible sources of further model refinement. Third, this knowledge can enable us to better (more reliably or efficiently) select a phenotype that should be targeted by a treatment among several related but distinct cellular phenotypes. Finally, note that the number of solutions in each case is significantly higher than what would be countable using standard enumeration, underscoring the importance of dedicated counting methods, and approximate counting in particular.