Towards Flexible Criteria in the Revision of Boolean Networks

Rafael Patronilo, Matthias Knorr and João Leite

NOVA LINCS, NOVA University Lisbon, 2829-516 Caparica, Portugal

Abstract

Biological regulatory networks, and Boolean networks (BNs) in particular, are widely used to investigate the dynamics of complex cellular processes and chemical reactions. BNs model the interactions between compounds through regulatory functions that determine a compound's state based on the activating or inhibiting effects of others. This enables the analysis of system behavior under incomplete, imprecise, or noisy information through in silico experiments, supporting hypothesis testing and the prediction of experimental outcomes—e.g., in healthcare research. In this context, new experimental data may reveal inconsistencies in the current model, necessitating its revision to account for these observations. This revision process has traditionally been manual, laborious, and error-prone due to the vast space of possible modifications. To address this, ARBoLoM was recently introduced as a tool for the automated revision of BNs. It leverages Answer Set Programming (ASP) to efficiently verify consistency and, when needed, compute repairs based on time series of experimental results. The revision is guided by an ordered set of criteria for determining minimal modifications, where the first criterion is fixed using iterative deepening, and the remaining ones are handled via optimization in ASP. However, fixing the first criterion limits the flexibility of the tool, and it remains an open question whether this first fixed criterion reflects practical or biological relevance. In this paper, we address this limitation by extending ARBoLoM to allow for a fully flexible order of the criteria. This extension also enables us to concisely characterize the behavior of prior approaches to BN revision and confirm that our tool remains orders of magnitude faster while still producing correct revision results.

Keywords

Biological Regulatory Networks, Revision, Answer Set Programming

1. Introduction

Over the past twenty years, systems biology has grown into a dynamic and influential discipline. It integrates concepts from engineering, mathematics, physics, and computer science to construct models that capture the complexity of biological phenomena. By adopting a comprehensive, system-level approach, researchers aim to uncover insights that were previously out of reach [1]. A key aspect of this area lies in the precise modeling of biological systems found in nature, with the overarching aim of deepening our understanding of cellular processes. Such insights may pave the way for novel scientific findings and conceptual frameworks regarding living systems.

In this context, Biological Regulatory Networks (BRNs) [2] are widely used to simulate the interactions between compounds in a computer environment or *in silico*. Among these, continuous models work directly with the concentration of each compound while logical models [3, 4] represent each compound as a Boolean variable, by measuring said concentration against a threshold. Though less concise, this is often advantageous in the absence of experimental data granular enough to support continuous models.

A prominent type of such logical models is Boolean logical models, also known as Boolean Networks (BNs) [5]. The interactions between compounds are represented as a directed graph, with an edge indicating whether a compound is a *regulator* of another compound, that is, whether its presence or absence influences the other compound. Each of these edges is labeled as a positive or negative interaction.[1] To allow more specific interactions to be described, each compound is usually accompanied

23rd International Workshop on Nonmonotonic Reasoning, November 11-13, 2025, Melbourne, Australia

r.patronilo@campus.fct.unl.pt (R. Patronilo); mkn@fct.unl.pt (M. Knorr); jleite@fct.unl.pt (J. Leite)
0009-0007-9456-8818 (R. Patronilo); 0000-0003-1826-1498 (M. Knorr); 0000-0001-6786-7360 (J. Leite)

[1]Sometimes, a compound can activate and inhibit another under different circumstances, but this is not common in practice, and we do not consider this case here.

by a regulatory function, a logical formula determining the compound's truth value according to the truth values of the regulators affecting it. Such BNs have been successfully applied in as models of, e.g., gene regulation networks [5].

It can happen that new experimental data is inconsistent with an existing BN, indicating one or more regulatory functions require correcting. This revision process has typically been done manually, but it can prove laborious and error-prone for large BNs. Unlike the automatic creation of BRNs [6, 7, 8, 9, 10], revision has only recently started to draw wider attention. Among the attempts at automating this revision process, most approaches are either limited on the type of repairs [11, 12, 13] or the complexity of the interactions described by the BN [14, 15]. ModRev [16] exceeds these limitations and is additionally capable of handling incomplete experimental data. However, it can take a long time to repair each BN, with the repair process often exceeding the predefined timeout of one hour and consequentially failing. ARBoLoM [17] addresses this problem, combining a deepening search approach with Answer Set Programming, in clingo [18], to find and optimize repairs. The tool proves orders of magnitude faster than ModRev, succeeding at finding a repair for a regulatory function in a number of real-life BNs often instantaneously and much before the previous timeout of one hour.

There is, however, a notable shortcoming that affects both tools, albeit to different extents. For finding optimal repairs, i.e., repairs that perform as few changes as possible to restore consistency, they employ an ordered set of criteria to determine the best solution, which is not fully flexible, making it difficult (or even impossible) for the user to apply either tool with some specific orders of the criteria. Notably, it is not possible to assign the same order to the criteria on both tools, making their direct comparison more challenging.

ModRev distinguishes three types of repairs, in the following order of preference (which is fixed):

1. Minimize changes to function regulators
2. Minimize changes to regulator signs
3. Minimize the changes in the formula of the regulatory function

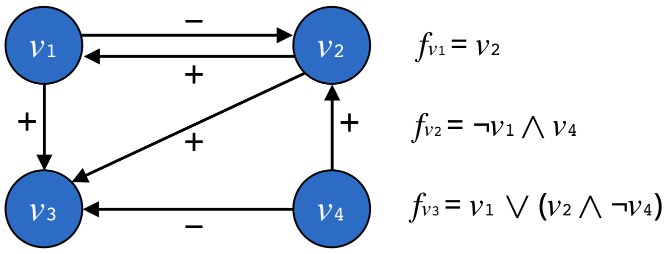
ARBoLoM distinguishes between changes in the formula due to changes in the number of function terms and changes in each term. ARBoLoM therefore distinguishes four types of criteria, in the following default order of preference (where the order of all but the first one can be changed):

1. Minimize the difference in the number of terms of the regulatory function
2. Minimize changes to function regulators
3. Minimize changes to regulator signs
4. Minimize changes in the format of each term.

This lack of complete flexibility in specifying the order of the criteria is, arguably, a shortcoming, as, to the best of our knowledge, no specific order has been identified as the preferred one to be used. It would therefore be clearly preferable to allow the user to adjust the order to their needs without limitation.

In this paper, we amend this problem and extend ARBoLoM in such a way that the user gains the desired flexibility on the choice of repair criteria, particularly on the order of preference for each type of repair. We achieve that by removing the hard-coded fixation of the first criterion in the iterative deepening search, introducing a parametrized solution which allows the user to indicate the order of preference on the criteria. We also perform a detailed evaluation analysing whether (a) performance is affected as such, (b) whether the order of criteria has an impact on the performance, and (c) how ARBoLoM's performance compares to that of ModRev when we adjust the optimization criteria to exactly match the order used in ModRev.

The remainder of this paper is structured as follows. We recall relevant material in Section 2 to make the paper self-contained. Then, in Section 3 we introduce the novel version of ARBoLoM that allows the flexible choice of the order of criteria for revision of BNs. We assess how well this new version performs in Section 4, before we conclude in Section 5.



v_1
v_2
v_3
v_4
$-$
$+$
$+$
$+$
$+$
$-$
$f_{v_1} = v_2$
$f_{v_2} = \neg v_1 \wedge v_4$
$f_{v_3} = v_1 \vee (v_2 \wedge \neg v_4)$

Figure 1: Example of a Boolean logical model.

2. Background

We briefly recall relevant notions and notation on Boolean Networks, Answer Set Programming, and ARBoLoM as a tool of revision in Boolean Networks, following the presentation in [17].

2.1. Boolean Networks

A Boolean Regulatory Network, or Boolean Network (BN), represents a set of biological compounds (including proteins or genes) and their interactions with each other modelling complex biological processes. Such BN makes use of a *regulatory graph*, which is a directed graph $G = (\mathrm{V}, E)$, where $V = \{v_1, ..., v_n\}$ is the set of vertices (nodes) representing the regulatory compounds, and $E = \{(u, v, s) : u, v \in V, s \in \{+, -\}\}$ is the set of signed edges representing the *interactions* between compounds such that E does not contain $(u, v, +)$ and $(u, v, -)$ for any $u, v \in V$. Edges with $s = +$ are called *positive interactions* (or *activation*), i.e., u activates v, while edges with $s = -$ are called *negative interactions* (or *inhibition*), i.e., u inhibits v.Nodes with no incoming edges in G are called *input nodes.* Such nodes represent external stimuli, and their values do not change over time.

Example 1. *Fig. 1 shows regulatory graph* $G = (V, E)$ *with* $V = \{v_1, v_2, v_3, v_4\}$ *and* $E = \{(v_1, v_2, -), (v_1, v_3, +), (v_2, v_1, +), (v_2, v_3, +), (v_4, v_2, +), (v_4, v_3, -)\}$.

Regulatory graphs alone do not specify how different compounds that affect the same node interact with each other for that node's activation. To that end, a regulatory function specifies, for each compound, what combination of regulators produces an effect on a given compound. Then, a *Boolean logical model* M of a regulatory network is defined as a tuple (V, F) where $V = \{v_1, v_2, ..., v_n\}$ is the set of variables representing the regulatory compounds of the network such that v_i is assigned to a value in $\{0, 1\}$, and $F = \{f_{v_1}, f_{v_2}, ..., f_{v_n}\}$ is the set of regulatory (Boolean) functions such that f_{v_i} defines the value of v_i and where $f_{v_i} = v_i$ if v_i is an input node. Regulatory functions of input nodes may be omitted (cf. v_4 in Fig. 1), but an explicit representation ($f_{v_4} = v_4$) is commonly used when involving implementational aspects [19].

Example 2. *Fig. 1 presents a Boolean logical model with* G *from Ex. 1 and regulatory functions for* G *on the right.*

Commonly, Blake canonical form (BCF), a special case of the disjunctive normal form, is used in the literature [12, 16, 17] to represent the regulatory functions of each biological compound by the disjunction of all its prime implicants [20]. A conjunction of literals is an implicant of a Boolean function if, whenever the conjunction takes the value 1, so does the function, and an implicant is *prime* if no subset of its literals forms itself an implicant [21]. BCF is unique up to reordering (of disjuncts), and since no compound can occur both positively and negatively in a specific regulatory function by definition of regulatory graphs, verification of prime implicants in such a function reduces to verifying that no disjunct is contained in the other. E.g., f_{v_3} (Fig. 1) is in BCF and both disjuncts are prime implicants.

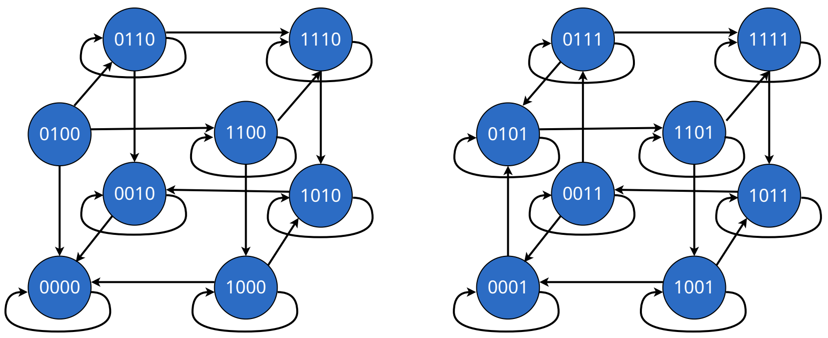


Figure 2: STG of the model in Ex. 2 for asynchronous updating scheme, with v_4 inactive on the left, and active on the right (adapted from [25]).

The dynamics of BNs is captured in their state resulting from the interactions between compounds over time. Formally, the *network state* of a BN with n compounds is a vector $S = [s_1, s_2, ...s_n]$ where s_i is the value of the variable representing the i-th compound of the network. Clearly, for Boolean logical models, the number of different states in a network is given by 2^n. E.g., if nodes 1 and 3 in Ex. 2 are active, and the other two are not, then this state is represented by 1010.

The evolution of states is captured in state transition graphs [22]. A State Transition Graph (STG) is a directed graph $G_{STG} = (S, T)$ where S is the set of vertices representing all the different states of the network, and T is the set of edges representing the viable transitions between states.

Updates to the values of the nodes in a BN are applied commonly in two different ways: synchronous and asynchronous [23, 24]. For synchronous update, at each time step, all compounds are updated simultaneously. Each network state has exactly one successor. For asynchronous update, at each time step, only one regulatory function may be applied. For a network of *n* compounds, each state can have at most n possible state transitions (including a transition to itself - cf. Fig. 2).

Boolean networks may enter so-called *attractors* during updates, i.e., sets of network states that form a cycle. These cycles are linked to many important cellular processes [26]. Among them, *stable states* are attractors that contain only a single state. An example for a stable state is 0000 on the left of Fig. 2.

2.2. Answer Set Programming

Answer Set Programming (ASP) aims at solving difficult combinatorial (primarily NP-hard) search problems [27]. It builds on the combination of a rich declarative knowledge representation language with efficient solvers [28]. Here, we briefly recall essential notions and notations for the sake of readability of the following material.

We start with the syntax of basic normal logic programs P that are a finite set of rules r of the form:

$$a_0 \leftarrow a_1, ..., a_m, \ not \ a_{m+1}, ..., \ not \ a_n. \tag{1}$$

where each a_i ($0 \leq i \leq n$, $0 \leq m \leq n$) is an *atom.* An atom is of the form $p(t_1, ..., t_k)$, with *p* a predicate symbol of arity *k*, and $t_1, ..., t_k$ terms, built from variables and constants of the (implicit) language of the program of *r*[2]. The atom to the left of the arrow ($\leftarrow$) is said to be the head of the rule, whereas the (negated) atoms to the right of the arrow, corresponding to a conjunction, are the body of the rule. We call literals both an atom a as well as its negation, $not \ a$ and we call a_0 the head of *r* ($head(r) = a_0$), and the set of all literals that occur in its body (also known as body literals) the body of *r* ($body(r) = \{a_1, ..., a_m, \ not \ a_{m+1}, ..., \ not \ a_n\}$). Rules with $m = n = 0$ are called *facts* and we admit that a_0 is $\bot$ to represent *constraints*, and we commonly omit $\leftarrow$ in the former case and $\bot$ in the latter. Facts allow us to express what necessarily must be true, while constraints provide us with the possibility to express conditions that we do not want to hold. The ground instantiation of a program, $ground(P)$ is obtained by replacing all (first-order) variables with constants from the given language

[2]Actually, function symbols are also permitted, but care must be taken as unbounded recursion is not admitted in ASP.

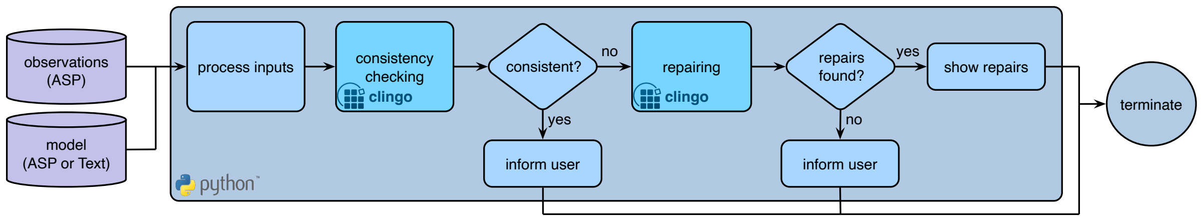


Figure 3: ARBoLoM – overview of the revision process of BNs

of the program in all possible ways. A set of ground atoms S is called an answer set of $ground(P)$ if S is the subset-minimal model of the reduct $ground(P)^S$ obtained as $\{a_0 \leftarrow a_1, ..., a_m \mid r$ of the form (1) in $ground(P)$ and $\{a_{m+1}, \ldots, a_n\} \cap S = \emptyset\}$.

ASP also admits a number of commonly used language constructs, most of which can be expressed by rules of form (1), but whose direct usage is much more convenient, and we indicate them briefly next.

First, choices permit that a number of atoms may be part of an answer set. E.g., for $\{activator(1); inhibitor(1)\}$ both $activator(1)$ and $inhibitor(1)$ can be part of an answer set. As a result, we obtain four different answer sets in this case, corresponding to the power set over the set of possible elements. Then, constraints can be used to filter out certain models. We can also turn such a choice into a cardinality constraint, providing lower and/or upper bounds on the number of atoms to appear in each answer set. E.g., $1\{activator(1); inhibitor(1)\}1$ admits exactly two answer sets as in each one exactly one of the two atoms can appear. Often, conditionals are employed, that allow us to determine ranges. E.g., a choice $\{node\_regulator(N, C) : compound(C)\}$ would admit answer sets with atoms over $node\_regulator/2$ as long as its second argument is a compound.

These constructs are also used in aggregates. E.g., $\#count\{C : node\_regulator(N1, C)\}$ counts all different values C that appear as second argument of the instances of $node\_regulator/2$. Optimization statements allow us not to just search for one (or more) solutions of a problem, but also assign weights to solutions which can be used to order solutions and optimize in accordance with these. E.g., $\#minimize\{1@2, C : sign\_changed(C)\}$ effectively sums the weights of all atoms over $sign\_changed/1$, where in this case, 1 is the weight of the atom and 2 the assigned priority (to be able to combine several optimization criteria with priorities – the higher priority is considered first). Based on that, the solver will aim to find the answer set with the lowest overall assigned weight.

Finally, constants can be declared using $\#const$ with a name and a default value. The latter can be overwritten when running that program. Comparison operators are naturally allowed, and the infix notation $X..Y$ for numeric values or variables representing numeric values is shorthand for all the values between and including X and Y.

2.3. ARBoLoM

ARBoLoM [17] has been developed for revision of BNs given a set of experimental results. The main idea is to check for consistency of the current model comparing its dynamic behaviour, i.e., what states are observed on what inputs, with the experimental observations. If the model's behaviour replicates these, then the model is consistent, otherwise it is inconsistent. In the latter case, repair is applied adjusting the functions whose results do not match the observations.

As shown in Fig. 3, ARBoLoM is written in Python to handle the processing of data and integration of the different components and employs ASP encodings for consistency checking and repair using the state-of-the-art ASP tool clingo [18]. ARBoLoM expects a set of (possibly incomplete) experimental observations and the specification of a BN. The former is assumed to be in a specific format, but any different representation of states of activations of the relevant compounds can be converted to it easily. The latter is expected to be either already encoded in ASP or in one of the standard formats, which is converted automatically by the tool into the required ASP format.

For checking consistency (and repair), three different modes of interaction between compounds are considered, namely stable states (without any time component), and synchronous and asynchronous updates, and three slightly varying ASP encodings exist. For the details, we refer to Aleixo et al. [17].[3]

The functions that are inconsistent need to be repaired by modifying one by one each inconsistent regulatory function. This requires again as input the representations of the network and the observations. The considered modifications are adding and removing regulators, changing regulators from activator to inhibitor and vice-versa, and changing the function's format, i.e., changing the number of terms as well as their composition. This results in a large variety of possible repaired functions. ARBoLoM aims at finding repairs that are as close as possible to the original functions in the spirit of minimal change, using the following order of priority:

1. Minimize the changes to the number of function terms;
2. Minimize the changes to the function's regulators;
3. Minimize the changes to the signs of regulators;
4. Minimize the changes to the format of each term.

While the order of criteria 2.–4. is easily adjustable (as we will see), criterion 1. is not, because ARBoLoM employs iterative deepening search on the number of changes to the number of function terms first. The central idea is to start searching for solutions with the same number of function terms as the function to be repaired, and if no solution is found, simultaneously, we search for solutions with one more and one less function term in parallel, determining the solution in accordance with the other criteria if both solutions exist, picking the single one if one of the cases fails, and repeat the process otherwise incrementing the distance w.r.t. number of terms present. The iteration is naturally limited by the lower bound of 0 function terms and by an upper bound determined based on the number of positive observations. This is possible because, in the limit, each positive observation is only covered by a single term in the function, so if we cannot find a function with at most as many terms as existing distinct observations (for the function to be repaired), then no such candidate can exist. This is further improved by not counting identical transitions of such observations.

The details for the respective ASP encodings can also be found in [17], but since, unlike the encodings for consistency checking, we revise these, we will discuss them with more detail next. [4]

3. Flexible Revision of Boolean Networks

Out of the four possible different repair operations distinguished in ARBoLoM, for the final three, their order of priority can be easily changed. The minimization of these is handled in ASP with optimization statements, each defined with a priority value corresponding to the order given in the end of the previous section. Reordering these criteria amounts to changing the priority value in these statements. For the first criterion however, which minimizes the change in the number of terms of the regulatory function, the optimization is handled by the iterative deepening search guided within the python code and therefore cannot be reordered in this way. This is a limitation, since a biologist might prefer a different order of preference for these criteria, which for a particular use case is more realistic. For example, if we know that a compound has been revealed to be relevant for a process in whose model it currently does not occur, then it needs to be added to the model in the revision process, and this requires adjusting the priority of criteria. But such modification cannot be achieved without substantial modifications to the implementation of the tool.

Therefore, we tackle this problem here and extend ARBoLoM to allow more flexibility on the choice of repair criteria, particularly in the order of preference for each type of repair. The main idea is to turn the criterion used for iterative deepening into one handled by ASP optimization. This allows parameterized optimization of all criteria in accordance with the preference of the user. We should note that this might affect overall performance, and we will analise this subsequently together with other important aspects in Sec. 4.

[3]The complete encodings can be found at https://github.com/fpaleixo/arbolom/tree/main/encodings/consistency.

[4]Still, the original complete encodings can be found at https://github.com/fpaleixo/arbolom/tree/main/encodings/repairs.

Listing 1: Simplified ASP encoding for repair with stable state observations.

```
 1 #const compound = c.
 2 #const max_node_number = n.
 3 fixed_regulator(C) :- fixed(C,compound), compound(C).
 4 fixed_regulator(C) :- fixed(C,compound,_), compound(C).
 5 fixed_activator(C) :- fixed(C,compound,0), compound(C).
 6 fixed_inhibitor(C) :- fixed(C,compound,1), compound(C).
 7 activator(C) :- fixed_activator(C).
 8 inhibitor(C) :- fixed_inhibitor(C).
 9 1 {activator(C); inhibitor(C)} 1 :- compound(C), not fixed_activator(C), not fixed_inhibitor(C).
10 available_node_ID(1..TERM_NO) :- function(compound, TERM_NO).
11 available_node_ID(1..max_node_number):- function(compound, TERM_NO), max_node_number > TERM_NO.
12 {node_regulator(N,C) : compound(C)} :- available_node_ID(N).
13 node_ID(N) :- node_regulator(N,_).
14 :- fixed_regulator(C), not node_regulator(_,C).
15 :- N1_VARNO = #count{ C : node_regulator(N1,C) },
16    COMMON_VARNO = #count{C : node_regulator(N1,C), node_regulator(N2,C)},
17    node_ID(N1), node_ID(N2), N1 != N2, COMMON_VARNO = N1_VARNO.
18 experiment_negative_node(E,N) :- node_regulator(N,C), curated_observation(E,C,0), activator(C).
19 experiment_negative_node(E,N) :- node_regulator(N,C), curated_observation(E,C,1), inhibitor(C).
20 experiment_positive_node(E,N) :- not experiment_negative_node(E,N),
21    node_ID(N), curated_observation(E,_,_).
22 original_regulator(C) :- regulates(C,compound,_).
23 present_regulator(C) :- node_regulator(N,C).
24 sign_changed(C) :- regulates(C,compound,0), inhibitor(C).
25 sign_changed(C) :- regulates(C,compound,1), activator(C).
26 :- experiment_positive_node(E,N), curated_observation(E,compound,0).
27 :- not experiment_positive_node(E,_), curated_observation(E,compound,1).
```

The resulting ASP encoding for repair in the case of stable states is shown in Listing 1. The encodings for synchronous and asynchronous updates are similar in the same vein as in Aleixo et al. [17], covering also the necessary update aspects due to the dynamics, which are not essential for our modifications. [5] In the following, we briefly describe the encoding and the main modifications, starting by briefly recalling the input format [17].

The regulatory graph $G = (V, E)$ is encoded as a set of facts over $compound/1$ to represent all elements in V and over $regulates/3$ s.t. $regulates(u, v, s)$ represents that compound u regulates v with the sign s (where $s = 0$ encodes an activator, and $s = 1$ an inhibitor). For regulatory functions, we use predicates $function/2$ and $term/3$ to represent the number of their terms and the constituting terms themselves, respectively. E.g., $\mathsf{function}(\mathsf{v1}, 1)$ represents that v_1's regulatory function has one term, while $\mathsf{term}(\mathsf{v1}, 1, \mathsf{v2})$ represents that term 1 of v_1's regulatory function contains v_2. This notation is a bit less compact, but facilitates specific modifications.

The observations are encoded by means of predicates, $experiment/1$ and $observation/3$. The former is used to represent the id of an experiment, while $\mathsf{observation}(\mathsf{e}, \mathsf{c}, \mathsf{s})$ represents the observed state s (0 for inactive, 1 for active) of compound c of experiment e. Note that, for updates where the network does change its state over time, an additional argument for discrete time t is required.

The repair encoding itself first introduces two constants as parameters (Lines 1–2), one for the compound whose function is to be repaired, the other for the upper bound of the number of terms (called nodes here). Then, we incorporate additional knowledge on biologically mandatory known regulators (possibly including the fixed signs, i.e., whether they are known to be activating or inhibiting) (Lines 3–8), and for those not fixed whether they are activating or inhibiting (Line 9). Then, we determine the number of available nodes (i.e., conjunctions of regulators) picking the larger number among the current number of terms and the given parameter (Lines 10–11). For each such node, we generate possible regulators (Line 12), and turn a node effective if it has at least one regulator associated to it (Line 13). Then, we ensure that the fixed regulators indeed appear in some node (Line 14), and that the function be in BCF (Line 15–17) verifying that the node is not fully contained in another node, by assuring its number of compounds differs from the number of compounds in common with another node.

[5]All new encodings can be found at https://github.com/rafael-patronilo/arbolom/tree/terms-criterion-asp/encodings/repairs.

Listing 2: New ASP minimization statements with parameterized priority.

```
 1 % number of terms
 2 #minimize{1@P, N : node_ID(N), function(compound, TERM_NO), N > TERM_NO}.
 3 #minimize{1@P, N : available_node_ID(N), not node_ID(N), function(compound, TERM_NO), N <= TERM_NO}.
 4 % regulators
 5 #minimize{1@P, C : not original_regulator(C), present_regulator(C)}.
 6 #minimize{1@P, C : original_regulator(C), not present_regulator(C)}.
 7 % signs
 8 #minimize{1@P, C : sign_changed(C)}.
 9 % regulators in each term
10 #minimize{1@P, ID, R : term(compound, ID, R), node_ID(ID), not node_regulator(ID, R)}.
11 #minimize{1@P, ID, R : node_regulator(ID, R), term(compound, ID, _), not term(compound, ID, R)}.
```

We identify nodes as negative if one of its activators is inactive (Line 18) or one of its inhibitors active (Line 19), and as positive if it is not negative (Line 20–21). Likewise, we determine which regulators appear in the original function and in the modified one (Lines 22–23) and which compounds exhibit changes in sign between the original and the modified function (Lines 24–25). Finally, we discard all solutions for which an experimental observation of the considered compound differs from the value computed by the modified function (Lines 26–27).

Let us discuss the major changes applied to this encoding. First and foremost, it differs conceptionally. Namely, it no longer includes the minimization statements directly, since these are formatted and added by python code, allowing their priority to be set at runtime. We explain how this works in a moment.

This principal modification causes a significant change in this encoding relative to the original one of ARBoLoM, required for handling the minimization of the change in the number of terms. We have removed the restriction which fixated the number of nodes of the functions to consider. This restriction was required by the deepening search to restrict the search space to the number of nodes being currently considered. As this minimization of the change in the number of nodes has been moved to the ASP program, this is neither required nor desirable as we wish for the search space to contain all possible functions regardless of the number of nodes. The constant max_node_number (formerly node_number) now represents the upper bound for the number of the terms rather than an exact restriction. This upper bound was already calculated in ARBoLoM to restrict the range of the deepening search as described in Section 2.3. We still calculate this value in the same way, by counting unique positive observations of the compound being repaired. We note that equivalent modifications were applied to the encodings for synchronous and asynchronous updates.

Regarding the minimization of changes, the tool admits now as additional input the parameters for the priority of criteria. This can be used within the Python code to prepare a number of optimization statements to be added to the ASP encoding. Listing 2 shows an abstract presentation of the minimization statements added by the python code, each with the placeholder P which is substituted with the correct priority for each statement according to the user's selection. Note that this is indeed an abstracted representation, and each of the lines can be associated with a different value, permitting in principle a very flexible attribution of priorities. Our focus though is to assign these in accordance with the characteristics of modifications considered in the previous sections and also used as separating comments in the listing.

In more detail, we have added two new minimization statements which count and minimize changes in the number of terms for both addition (Line 2) and removal (Line 3). This accounts for the minimization of changes previously handled by the iterative deepening process. Lines 5 and 6 account for changes in the number of regulators and allows their minimization, while Line 8 captures minimization of sign changes of the involved regulators. Finally, Lines 10 and 11 handle minimization in changes of terms, by accounting for regulators that have been added to a node or removed from it. The resulting minimized numbers of modifications are also extracted from the results and can be used for analysis and comparison.

In addition, we also modified the minimize statements in Lines 5, 6, 10 and 11 in the sense of internalising former auxiliary predicates inside of these minimize statements. While this change has

Abbreviation	$\#Comp.$	$\#Inter.$	$\#StableS$	$Avg.\#\ Reg.$	$Max.\#\ Reg.$
MCC	10	35	1	3.50	6
FY	10	27	12	2.70	5
TCR	40	57	7	1.42	5
SP	19	57	7	3.00	8
Th	23	35	3	1.52	5

Table 1
Boolean models data with their numbers of compounds (#Comp), compound interactions (#Inter), stable states (#StableS), average number of regulators (Avg.#Reg.), and maximum number of regulators (Max.#Reg.).

no effect on the result of the optimization itself, this kind of encoding commonly improves processing times in search with ASP and may thus impact positively overall performance of the repair program.

4. Evaluation

We now assess in a detailed manner how these modifications affect the overall performance, studying in particular to what extent changing the order of preference itself impacts on performance and how ARBoLoM with a preference order of criteria aligned with that of ModRev compares with ModRev.

To that end, we have taken advantage of the benchmarks employed before for ARBoLoM [17] which were generated based on the same methodology used in [29]. Essentially real BNs are considered and corrupted, and sets of observations, generated from the original BN, are used to repair its corrupted version. The BNs are corrupted by applying four operations, with varying probability X:

- Altering each regulatory function with a probability X (0.1 to 1), changing the number of terms and/or the format of the terms of the function.
- Altering the sign of each edge (regulator) in the function with a probability X (0.05 to 0.75).
- Removing a regulator from the function with probability X (0.01 to 0.15).
- Adding a new regulator to the function with probability X (0.01 to 0.15)

Based on that, 24 different corruption configurations were defined, varying both the available corruption operations and respective probabilities. Then, five benchmarks were created using real biological models: **MCC** [23], **FY** [30], **TCR** [31], **SP** [32], and **Th** [33] – some information on these is summarized in Table 1. For each configuration 100 corrupted models were generated, resulting in a total of 2400 models for each benchmark. Aside the corrupted models, each benchmark includes four sets of observations generated using the original model, ranging from a set with 1 experiment with 3 timepoints to a set with 5 experiments with 20 timepoints.

In addition, to test the tool on a larger network, a benchmark using a model that combines all the five real models was created – its corresponding data can be inferred from Table 1 as these 5 models are distinct for all but one compound shared between two BNs. Even though this network is thus a bit disconnected, this still provides a challenge, as the corruptions are generated arbitrarily, and possible fixes need to inspect the entire search space. Given the size of this new network, in this benchmark only 10 corrupted models for each of the 24 configurations were generated and only one set of observations with 5 experiments and 20 timepoints.

Out of the six benchmarks the original ARBoLoM was tested on, so far we have evaluated the new version of the tool on two of them: the **TCR** benchmark, which is the largest (by number of compounds) among the 5 original networks and consequentially the model whose benchmark takes longer to repair on average; and the benchmark combining all five models, which we will refer to as **All5**. We ran our tests on a Linux machine with an AMD Ryzen 7 9700X CPU and 32GB of RAM.

We tested five orders of priority which we consider significant, analyzing both the time the revision process takes and the cost of each criterion in the optimal repairs, that is the number of changes of each type applied in the repaired function. Table 2 shows the mean, min and max costs of each criterion considering each of the five orders of priority. Note that the order **A** is equivalent to the order used

Table 2
Costs of the optimal repairs in **TCR** for 4 different orders of criteria.

		stable state			synchronous			asynchronous		
ID	Criteria Priority	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.
A	1. term number	0.00	0	0	0.00	0	0	0.00	0	0
	2. regulators	0.05	0	1	0.13	0	2	0.04	0	2
	3. signs	0.94	0	3	0.84	0	3	0.89	0	4
	4. term format	0.27	0	7	0.38	0	7	0.36	0	5
B	1. regulators	0.02	0	1	0.05	0	2	0.01	0	1
	2. signs	0.85	0	3	0.82	0	3	0.78	0	4
	3. term format	0.26	0	3	0.24	0	3	0.25	0	5
	4. term number	0.17	0	3	0.27	0	5	0.38	0	4
C	1. signs	0.05	0	1	0.01	0	1	0.00	0	0
	2. term format	0.25	0	4	0.58	0	4	0.73	0	5
	3. term number	0.89	0	3	0.79	0	4	0.86	0	3
	4. regulators	1.29	0	5	1.49	0	5	1.52	0	5
D	1. term format	0.03	0	1	0.03	0	1	0.02	0	1
	2. term number	0.11	0	3	0.18	0	3	0.08	0	3
	3. regulators	0.13	0	3	0.20	0	3	0.05	0	2
	4. signs	0.98	0	3	0.91	0	3	1.07	0	4
E	1. regulators	0.02	0	1	0.05	0	2	0.01	0	1
	2. signs	0.85	0	3	0.82	0	3	0.78	0	4
	3. term number and format	0.42	0	4	0.47	0	5	0.53	0	5

in the original ARBoLoM. Orders **B**, **C** and **D** are obtained by moving the criterion with the highest priority in the previous order to the lowest priority, therefore cycling the highest priority between all criteria. Order **E** is different. It is equivalent to the order used in ModRev [16], that is, it first prioritizes changes to the regulators, second changes to the signs of the regulators and lastly changes to the format of the regulatory function, be it term number or format of each term. This last order is therefore similar to order **B** with the difference that the same priority is assigned to reducing the change in term number and the change in the format of each term.

We can see that the criterion with the highest priority always has the lowest average cost, indicating that the prioritization works as intended. This is however not always the case for the other criteria. For example, in the order **A**, the average cost of the change in signs is higher than the cost in the change to term format, despite signs having a higher priority. This can be explained by the fact that the space of possible repairs has already been limited by the first and second criteria, and therefore it may not be possible to pick a set of repairs with fewer changes in signs than in the term format.

Table 3 shows the times for the whole revision process, consistency checking and repairing for each of the five orders of priority listed in Table 2. Note that we include the consistency checking time merely for context when analyzing the total revision time. Consistency checking should not depend on the order of the criteria as this is a parameter of the repair process, and the small differences in the respective times between different orders are therefore most likely related to circumstantial factors of the testing environment. Variations in the repair time however can be related to the choice of criteria, but even these do not show much variation. Looking at the maximum times, we see that the tool seems to take under 5 seconds for any model, regardless of the interaction mode or order of priority in the repair criteria, and, on average for all cases, revision is performed in less than 0.5 seconds. This shows that for **TCR**, and therefore likely for the other four smaller real models as well, this new version of ARBoLoM is as efficient as the original [17], with the advantage of flexibility in the order of the criteria.

We were also interested in understanding how our new solution performs on a larger model. Table 4 shows the times for the benchmark **All5** in asynchronous mode, using each of the five orders of priority we have established. We note that times overall increase, but results are still obtained quickly with average repair times varying between 1 and 4 seconds and maximum values between 7 and 33 seconds. Notably, among the 5 orders, the one corresponding to that previously employed by ARBoLoM (order

Table 3
Times for benchmark **TCR** under each of the 5 orders of priority for the criteria.

		stable state			synchronous			asynchronous		
Order	Time	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.
A	Revision (s)	0.06	< 0.01	4.11	0.25	< 0.01	4.36	0.11	< 0.01	2.41
	Consistency (s)	< 0.01	< 0.01	< 0.01	0.03	< 0.01	0.24	0.02	< 0.01	0.28
	Repair (s)	0.07	< 0.01	4.10	0.27	< 0.01	4.25	0.17	< 0.01	2.34
B	Revision (s)	0.04	< 0.01	0.24	0.20	< 0.01	4.68	0.12	< 0.01	2.27
	Consistency (s)	< 0.01	< 0.01	< 0.01	0.02	< 0.01	0.20	0.02	< 0.01	0.32
	Repair (s)	0.05	< 0.01	0.24	0.21	< 0.01	4.57	0.17	< 0.01	2.20
C	Revision (s)	0.04	< 0.01	0.27	0.22	< 0.01	3.66	0.12	< 0.01	1.81
	Consistency (s)	< 0.01	< 0.01	< 0.01	0.03	< 0.01	0.23	0.02	< 0.01	0.34
	Repair (s)	0.05	< 0.01	0.26	0.23	< 0.01	3.54	0.17	< 0.01	1.75
D	Revision (s)	0.04	< 0.01	0.24	0.21	< 0.01	3.62	0.11	< 0.01	2.39
	Consistency (s)	< 0.01	< 0.01	< 0.01	0.03	< 0.01	0.19	0.02	< 0.01	0.27
	Repair (s)	0.05	< 0.01	0.23	0.23	< 0.01	3.50	0.17	< 0.01	2.32
E	Revision (s)	0.06	< 0.01	0.45	0.37	0.01	3.95	0.20	< 0.01	2.68
	Consistency (s)	< 0.01	< 0.01	< 0.01	0.05	< 0.01	0.30	0.04	< 0.01	0.42
	Repair (s)	0.07	< 0.01	0.44	0.39	0.01	3.81	0.28	0.01	2.61

Table 4
Times for benchmark **All5** under each of the 5 orders of priority for the criteria, in asynchronous mode compared to the version with deepening search.

		Flexible Criteria			Deepening Search		
Order	Times	Avg.	Min.	Max.	Avg.	Min.	Max.
A	Revision (s)	3.10	0.10	33.17	1.39	0.08	8.08
	Consistency (s)	0.14	0.09	0.53	0.12	0.07	0.34
	Repair (s)	3.38	0.10	32.98	1.46	0.09	7.91
B	Revision (s)	1.45	0.09	6.90	-	-	-
	Consistency (s)	0.15	0.09	0.53	-	-	-
	Repair (s)	1.48	0.10	6.74	-	-	-
C	Revision (s)	1.62	0.10	6.57	-	-	-
	Consistency (s)	0.15	0.09	0.52	-	-	-
	Repair (s)	1.68	0.10	6.43	-	-	-
D	Revision (s)	1.30	0.10	7.07	-	-	-
	Consistency (s)	0.12	0.09	0.38	-	-	-
	Repair (s)	1.35	0.10	6.89	-	-	-
E	Revision (s)	2.63	0.19	10.67	-	-	-
	Consistency (s)	0.23	0.18	0.65	-	-	-
	Repair (s)	2.75	0.18	10.46	-	-	-

A) turns out to be the slowest, followed by the one used by ModRev (order E). We deem that, for **A**, this results from the fact that there is in principle a large number of nodes to be added, and the optimization process takes more time due to that. For **E**, we conjecture that this is influenced by the two conjoined criteria, as mixing them might make it more complicated to distinguish between different solutions.

Still, this shows that the solution employed by ARBoLoM is clearly superior to that of ModRev, even when using the same criteria, as revision always succeeds and with, on average, interactive response times. ModRev, on the other hand, only succeeds in 79% of the cases and takes on average around 105 seconds on those instances it successfully revises, even if on a slightly slower system [17].

We also include on the right of Table 4, for order **A**, the times for the former version of ARBoLoM using deepening search. On average, deepening search proves to be over two times as fast and on the worst case it proves four times and over 20 seconds faster. Despite the new version being notably slower, in exchange we obtain the flexibility, and even for this large BN, we are able to provide repairs well below a minute in asynchronous mode for all models, regardless of the order of priority for the criteria.

We cannot provide the same reassurance for synchronous mode. Our experiments so far show that

Table 5
Comparison of the new ASP criteria against the former ASP criteria in the **All5** benchmark in synchronous mode. Note that both versions in this table used the deepening search strategy

	New Criteria			Old Criteria		
Time	Avg.	Min.	Max.	Avg.	Min.	Max.
Revision (s)	115.76	22.47	425.31	115.70	24.10	408.62
Consistency (s)	0.18	0.11	0.32	0.18	0.11	0.33
Repair (s)	115.58	22.36	425.17	115.52	23.99	408.28

synchronous mode poses a significant challenge, with the repair of multiple compounds timing out. All these compounds have a high max number of nodes (terms) relative to the compounds which get repaired within reasonable time. Recall that this upper bound for the number of terms in the repaired function is calculated by counting the number of unique positive observations of the compound. This is a limitation of the new version over the former version, which uses deepening search to progressively try different numbers of terms and therefore is not susceptible to problems related to a large upper limit. We are currently studying possible strategies to mitigate this issue. One easy solution would be to use the deepening search strategy whenever it is possible, so we can at least provide the same performance in those situations, while retaining the flexibility of criteria otherwise.

Finally, as described in Section 3, we modified existing ASP minimization criteria, internalising auxiliary predicates into the minimize statements, with the goal of improving performance. To understand if and how much our changes to the existing ASP minimization criteria affect the performance, we transposed these changes to the former version of ARBoLoM (with the deepening search) and test on the benchmark **All5** in synchronous mode, which, due to the overall expected longer revision times, should allow us to give a better idea of the impact on performance. Table 5 shows the times for both versions, both obtained in the same testing environment. Surprisingly, there is no significant difference between both versions, and we will have to investigate this matter further.

5. Conclusions

We have introduced a new version of ARBoLoM, a tool for the revision of Boolean Networks given a set of experimental observations. The main improvement lies in the new capacity for the user to be able to freely choose the preference order of application of optimization criteria, employed to ensure that revision is done with minimal changes. Our assessment shows that this is indeed viable in the sense that revision of a real BN, for several variants of preference orders, can be done very fast. This includes the specific order employed by another tool, ModRev [16], validating the previous assessment [17], now for identical optimization criteria, that ARBoLoM's solution based on ASP is indeed superior by a considerable margin.

We have however also seen, that for larger models, dropping the iterative deepening search in general has a negative impact on performance, in particular if the search space in terms of possible nodes to be added increases. We plan to incorporate the previous solution of ARBoLoM for this particular choice of criteria to take advantage of the improved performance. We also will do more exhaustive experiments with a wider set of real BNs, aiming to confirm our positive results here, and to tackle open issues including the ongoing study of the impact of different forms of encoding the optimizations.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. This work is supported by project BIO-REVISE (2023.13327.PEX) and by UID/04516/NOVA Laboratory for Computer Science and Informatics (NOVA LINCS) with the financial support of FCT.IP.

Declaration on Generative AI

The author(s) have not employed any generative AI tools.

References

[1] A. Siegel, O. Radulescu, M. L. Borgne, P. Veber, J. Ouy, S. Lagarrigue, Qualitative analysis of the relation between DNA microarray data and behavioral models of regulation networks, Biosystems 84 (2006) 153–174. doi:10.1016/j.biosystems.2005.10.006.

[2] H. Jong, Modeling and simulation of genetic regulatory systems: A literature review, Journal of computational biology : a journal of computational molecular cell biology 9 (2002) 67–103. doi:10.1089/10665270252833208.

[3] L. Glass, S. A. Kauffman, The logical analysis of continuous, non-linear biochemical control networks, Journal of Theoretical Biology 39 (1973) 103–129. doi:https://doi.org/10.1016/0022-5193(73)90208-7.

[4] R. Thomas, Boolean formalization of genetic control circuits, Journal of Theoretical Biology 42 (1973) 563–585. doi:https://doi.org/10.1016/0022-5193(73)90247-6.

[5] L. Salinas, L. Gómez, J. Aracena, Existence and non existence of limit cycles in boolean networks, in: Automata and Complexity, volume 42, Springer, 2022, pp. 233–252. doi:10.1007/978-3-030-92551-2_15.

[6] M. Ostrowski, L. Paulevé, T. Schaub, A. Siegel, C. Guziolowski, Boolean network identification from perturbation time series data combining dynamics abstraction and logic programming, Biosystems 149 (2016) 139–153. doi:https://doi.org/10.1016/j.biosystems.2016.07.009, selected papers from the Computational Methods in Systems Biology 2015 conference.

[7] C. Réda, B. Wilczyński, Automated inference of gene regulatory networks using explicit regulatory modules, Journal of Theoretical Biology 486 (2020) 110091. doi:10.1016/j.jtbi.2019.110091.

[8] A. Mitsos, I. N. Melas, P. Siminelakis, A. D. Chairakaki, J. Saez-Rodriguez, L. G. Alexopoulos, Identifying drug effects via pathway alterations using an integer linear programming optimization formulation on phosphoproteomic data, PLoS Computational Biology 5 (2009) e1000591. doi:10.1371/journal.pcbi.1000591.

[9] T. Schaub, A. Siegel, S. Videla, Reasoning on the Response of Logical Signaling Networks with ASP, 2014, pp. 49–92. doi:10.1002/9781119005223.ch2.

[10] S. Videla, C. Guziolowski, F. Eduati, S. Thiele, N. Grabe, J. Saez-Rodriguez, A. Siegel, Revisiting the training of logic models of protein signaling networks with ASP, in: Computational Methods in Systems Biology, Springer Berlin Heidelberg, 2012, pp. 342–361. doi:10.1007/978-3-642-33636-2_20.

[11] E. Merhej, S. Schockaert, M. D. Cock, Repairing inconsistent answer set programs using rules of thumb: A gene regulatory networks case study, International Journal of Approximate Reasoning 83 (2017) 243–264. doi:10.1016/j.ijar.2017.01.012.

[12] A. Lemos, I. Lynce, P. Monteiro, Repairing boolean logical models from time-series data using answer set programming, Algorithms for Molecular Biology 14 (2019) 9. doi:10.1186/s13015-019-0145-8.

[13] X. Chai, Reachability Analysis and Revision of Dynamics of Biological Regulatory Networks, Ph.D. thesis, École centrale de Nantes, 2019. URL: https://theses.hal.science/tel-02145438/.

[14] M. Gebser, C. Guziolowski, M. Ivanchev, T. Schaub, A. Siegel, S. Thiele, P. Veber, Repair and prediction (under inconsistency) in large biological networks with answer set programming, Principles of Knowledge Representation and Reasoning: Proceedings of the 12th International Conference, KR 2010 (2010). URL: https://cdn.aaai.org/ocs/1334/1334-7439-1-PB.pdf.

[15] N. Mobilia, A. Rocca, S. Chorlton, E. Fanchon, L. Trilling, Logical modeling and analysis of regulatory genetic networks in a non monotonic framework, in: Bioinformatics and Biomedical Engineer-

ing, Springer International Publishing, 2015, pp. 599–612. doi:10.1007/978-3-319-16483-0_58.

[16] F. Gouveia, I. Lynce, P. T. Monteiro, Revision of boolean models of regulatory networks using stable state observations, J. Comput. Biol. 27 (2020) 144–155. doi:10.1089/cmb.2019.0289.

[17] F. Aleixo, M. Knorr, J. Leite, Revising Boolean Logical Models of Biological Regulatory Networks, in: Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, 2023, pp. 12–22. doi:10.24963/kr.2023/2.

[18] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot ASP solving with clingo, Theory and Practice of Logic Programming 19 (2019) 27–82. doi:10.1017/S1471068418000054.

[19] H. Klarner, A. Bockmayr, H. Siebert, Computing symbolic steady states of boolean networks, in: J. Wąs, G. C. Sirakoulis, S. Bandini (Eds.), Cellular Automata, Springer International Publishing, Cham, 2014, pp. 561–570. doi:10.1007/978-3-319-11520-7_59.

[20] A. Blake, Canonical expressions in Boolean algebra, Ph.D. thesis, The University of Chicago, 1937. doi:10.2307/2267634.

[21] Y. Crama, P. L. Hammer, Boolean functions: Theory, algorithms, and applications, Cambridge University Press, 2011. doi:10.1017/CBO9780511852008.

[22] A. Naldi, E. Remy, D. Thieffry, C. Chaouiya, Dynamically consistent reduction of logical regulatory graphs, Theoretical Computer Science 412 (2011) 2207–2218. doi:10.1016/j.tcs.2010.10.021.

[23] A. Faure, A. Naldi, C. Chaouiya, D. Thieffry, Dynamical analysis of a generic boolean model for the control of the mammalian cell cycle, Bioinformatics 22 (2006) e124–e131. doi:10.1093/bioinformatics/btl210.

[24] A. Garg, A. D. Cara, I. Xenarios, L. Mendoza, G. D. Micheli, Synchronous versus asynchronous modeling of gene regulatory networks, Bioinformatics 24 (2008) 1917–1925. doi:10.1093/bioinformatics/btn336.

[25] F. Gouveia, Model Revision of Boolean Logical Models of Biological Regulatory Networks, Ph.D. thesis, Instituto Superior Técnico, Universidade de Lisboa, 2021. URL: https://scholar.tecnico.ulisboa.pt/records/Dc_NRZICXqamFOgaJK7aATltOOGTAgr625vU.

[26] M. Hopfensitz, C. Müssel, M. Maucher, H. A. Kestler, Attractors in boolean networks: a tutorial, Computational Statistics 28 (2012) 19–36. doi:10.1007/s00180-012-0324-2.

[27] V. Lifschitz, What is answer set programming?, in: Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3, AAAI'08, AAAI Press, 2008, p. 1594–1597. URL: https://www.cs.utexas.edu/~vl/papers/wiasp.pdf.

[28] M. Gebser, B. Kaufmann, T. Schaub, Conflict-driven answer set solving: From theory to practice, Artificial Intelligence 187-188 (2012) 52–89. doi:10.1016/j.artint.2012.04.001.

[29] F. Gouveia, I. Lynce, P. T. Monteiro, Modrev - model revision tool for boolean logical models of biological regulatory networks, in: A. Abate, T. Petrov, V. Wolf (Eds.), Computational Methods in Systems Biology - 18th International Conference, CMSB 2020, Konstanz, Germany, September 23-25, 2020, Proceedings, volume 12314 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 339–348. doi:10.1007/978-3-030-60327-4_18.

[30] M. Davidich, S. Bornholdt, Boolean network model predicts cell cycle sequence of fission yeast, PloS one 3 (2008) e1672. doi:10.1371/journal.pone.0001672.

[31] S. Klamt, J. Saez-Rodriguez, J. A. Lindquist, L. Simeoni, E. D. Gilles, A methodology for the structural and functional analysis of signaling and regulatory networks, BMC Bioinformatics 7 (2006). doi:10.1186/1471-2105-7-56.

[32] L. Sanchez, C. Chaouiya, D. Thieffry, Segmenting the fly embryo: logical analysis of the role of the segment polarity cross-regulatory module, The International Journal of Developmental Biology 52 (2008) 1059–1075. doi:10.1387/ijdb.072439ls.

[33] L. Mendoza, I. Xenarios, A method for the generation of standardized qualitative dynamical systems of regulatory networks, Theoretical Biology and Medical Modelling 3 (2006). doi:10.1186/1742-4682-3-13.