Reasoning over Biological Networks Using Maximum Satisfiability

João Guerra and Inês Lynce

INESC-ID/Instituto Superior Técnico,
Technical University of Lisbon, Portugal
{jguerra,ines}@sat.inesc-id.pt

Abstract. Systems biology is with no doubt one of the most compelling fields for a computer scientist. Modelling such systems is per se a major challenge, but the ultimate goal is to reason over those systems. We focus on modelling and reasoning over biological networks using Maximum Satisfiability (MaxSAT). Biological networks are represented by an influence graph whose vertices represent the genes of the network and edges represent interactions between genes. Given an influence graph and an experimental profile, the first step is to check for mutual consistency. In case of inconsistency, a repair is suggested. In addition, what is common to all solutions/optimal repairs is also provided. This information, named prediction, is of particular interest from a user's point of view. Answer Set Programming (ASP) has been successfully applied to biological networks in the recent past. In this work, we give empirical evidence that MaxSAT is by far more adequate for solving this problem. Moreover, we show how concepts well studied in the fields of MaxSAT and CP, such as backbones and unsatisfiable subformulas, can be naturally related to this practical problem.

1 Introduction

The field of systems biology has seen a tremendous boost due to the advances in molecular biology, which are responsible for the availability of large sets of comprehensive data. The existence of large-scale data sets is a key motivation for developing reliable algorithmic solutions to solve different problems in the field. Understanding those problems comprises reasoning about them.

We address the problem of reasoning over biological networks, in particular gene regulatory networks (GRNs), using influence graphs and the Sign Consistency Model (SCM) to represent GRNs. Reasoning is performed using Boolean Satisfiability (SAT) and Maximum Satisfiability (MaxSAT). These formalisms seem to be particularly well suited to this problem given the Boolean domains of the variables of the problem. SAT is used when reasoning can be formulated as a decision problem, whereas MaxSAT is used when reasoning can be formulated as an optimization problem.

This paper has three main contributions. First, we propose a SAT encoding for modelling biological networks and checking their consistency. This encoding has the advantage of making trivial the task of computing a prediction in case

M. Milano (Ed.): CP 2012, LNCS 7514, pp. 941–956, 2012.

of satisfiability, simply by using available tools for identifying backbones. Second, a MaxSAT encoding is used to repair inconsistent biological networks. Third, we propose an iterative solution, which invokes a MaxSAT solver, to compute predictions under inconsistency. Experimental results show that our contributions outperform existing solutions and as a result solve (many) practical instances which could not be solved by the existing alternatives. All the software implemented in this work is available online[1].

The paper is organized as follows. The next section introduces preliminaries, namely influence graphs, SCM, SAT, MaxSAT and related work. Section 3 describes how SAT and MaxSAT can be applied to reasoning over biological networks, including consistency checking, repairing and predicting. Section 4 is devoted to the experimental evaluation of the solutions proposed. Finally, the paper concludes and suggests future research directions.

2 Preliminaries

A gene regulatory network (GRN) is a kind of biological network in which we are concerned with the interactions between genes. To be precise, genes do not interact with each others directly, but rather through regulatory proteins (and other molecules). Each gene is influenced by the concentration levels of the proteins in its cellular context. Nonetheless, proteins are usually abstracted away and we speak of interactions between genes.

GRN models can be classified as either static or dynamic and qualitative or quantitative. Dynamic models describe the change of gene expression levels over time, whilst static models measure the variation of the gene expression levels between two steady states.

For many biological processes there is no detailed quantitative information available, e.g. accurate experimental data on chemical reactions kinetics is rarely available. This led to the creation of simpler models, the qualitative models. Qualitative models only consider, for example, the sign of the difference between the gene expression levels of two conditions. Despite being a simplification, these models are useful when there is a lack of information about the biological processes and still allow modelling the behaviour of a biological system correctly. Qualitative formalisms have also been successfully applied to other areas besides molecular biology (e.g. see [4]).

Our approach relies on a static qualitative model for GRNs and on the use of SAT and MaxSAT to reason over it. To describe the model, we borrow the notation introduced in [16,12].

For a survey of different models for GRNs refer to the relevant literature (e.g. see [9]).

2.1 Influence Graphs and Sign Consistency Model

Influence graphs are a common representation for a wide range of qualitative dynamical systems, notably biological systems [32]. These kind of graphs offer

[1] `http://sat.inesc-id.pt/~jguerra/rbnms/`

a logical representation of the interactions between the elements of a dynamic system.

An influence graph is a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \sigma)$, where $\mathcal{V}$ is a set of vertices representing the genes of a GRN, $\mathcal{E}$ is a set of edges representing the interactions between the genes of the GRN and $\sigma : \mathcal{E} \to \{+, -\}$ is a (partial) labelling of the edges. An edge from a vertex u to a vertex v, with $u, v \in \mathcal{V}$, is denoted as $u \to v$. Biologically, an interaction with label $+$ $(-)$ represents the activation (inhibition) of gene transcription or protein activation (inactivation).

To impose constraints between GRNs and experimental profiles we use the Sign Consistency Model (SCM) [29], which is based on influence graphs. This static qualitative model is particularly well suited for dealing with incomplete and noisy data [16,12]. In the SCM, experimental profiles only contain qualitative information about the observed variation of the gene expression levels.

Given an influence graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \sigma)$, an experimental profile $\mu : \mathcal{V} \to \{+, -\}$ is a (partial) labelling of the vertices of the graph. Additionally, each vertex of the graph is classified as either input or non-input. The labelling $\mu(v)$ of a non-input vertex $v \in \mathcal{V}$ is *consistent iff* there is at least one edge that explains its sign, i.e. one edge $u \to v \in \mathcal{E}$ such that $\mu(v) = \mu(u) \cdot \sigma(u \to v)$, where $\cdot$ corresponds to the multiplication of signs in the multiplication of signed numbers. For example, if $\mu(v) = +$ then either $\mu(u) = +$ and $\sigma(u \to v) = +$ or $\mu(u) = -$ and $\sigma(u \to v) = -$. Biologically, label $+$ $(-)$ means that there was an increase (decrease) in the gene expression levels. Note that the definition of consistency does not apply to input vertices.

An influence graph (model) $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \sigma)$ and an experimental profile (data) μ are *mutually consistent iff* there are total labellings $\sigma' : \mathcal{E} \to \{+, -\}$ and $\mu' : \mathcal{V} \to \{+, -\}$, which are total extensions of σ and μ, respectively, such that $\mu'(v)$ is consistent for every non-input vertex $v \in \mathcal{V}$. When considering an influence graph without an experimental profile, i.e. when μ is undefined for every vertex of the graph, we talk about *self-consistency* of the graph [18].

Example 1. Figure 1 illustrates an influence graph (left) and an experimental profile for that influence graph (right). The graph has three vertices, a, b and c and five edges, $a \to b$, $a \to c$, $b \to a$, $b \to c$ and $c \to b$. All vertices are non-input vertices. Lighter (green) edges ending with $\to$ have label $+$, whereas darker (red) edges ending with $\dashv$ have label $-$. Likewise, in the experimental profile, lighter (green) vertices have label $+$, darker (red) vertices have label $-$ and white vertices have no label.

In section 3 we will reason over this example.

2.2 Maximum Satisfiability

The Boolean Satisfiability (SAT) problem is the problem of deciding whether there is an assignment to the variables of a Boolean formula that satisfies it.

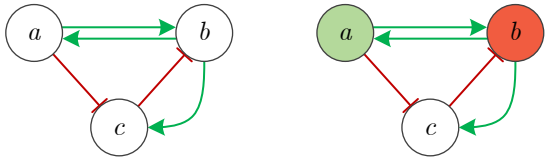

Fig. 1. An influence graph (left) along with an experimental profile (right)

Without lack of generality, one may assume that the formula is in conjunctive normal form (CNF). A CNF formula is a conjunction ($\wedge$) of clauses, where a clause is a disjunction ($\vee$) of literals and a literal is a Boolean variable (x) or its negation ($\neg x$). A CNF formula is satisfied *iff* all of its clauses are satisfied. A clause is satisfied if at least one of its literals is satisfied. A literal x ($\neg x$) is satisfied *iff* the corresponding Boolean variable is assigned the value *true* (*false*).

The Maximum Satisfiability (MaxSAT) problem is closely related to SAT. The goal in MaxSAT is to find an assignment that maximizes (minimizes) the number of satisfied (unsatisfied) clauses.

There are a few interesting variants of the MaxSAT problem. One of them, relevant to the application being of interest, is the Partial MaxSAT problem. In Partial MaxSAT, some clauses are classified as hard, whereas the remaining ones are classified as soft. The goal is to find an assignment that satisfies all hard clauses and maximizes the number of satisfied soft clauses. Hard clauses are usually represented within square brackets, whereas soft clauses are represented within parentheses.

Example 2. Consider the following Partial MaxSAT formula $\mathcal{F} = [x_1 \vee x_2 \vee x_3] \wedge [\neg x_1 \vee x_3] \wedge (\neg x_2) \vee (\neg x_3)$. The two optimal solutions are $\{x_1, \neg x_2, x_3\}$ and $\{\neg x_1, x_2, \neg x_3\}$. In any of these solutions, only one of the two soft clauses is satisfied.

2.3 Related Work

The same problem of modelling and reasoning over biological networks has been tackled in the past using Answer Set Programming (ASP) [16,12,15]. First, the authors analyse whether a biological network is consistent. If the network is consistent that means that there is a solution corresponding to total extensions of σ and μ. Moreover, a prediction corresponding to the intersection of all solutions is computed. If the network is inconsistent then minimal explanations for inconsistency are provided. As an alternative, an optimal repair is given. In addition, the user is also given a prediction, which now summarizes what is common to all optimal repairs. In ASP, the computation of predictions is achieved through cautious reasoning [14].

Reasoning over biological networks with ASP can find similarities with concepts well known in SAT and CP.

Minimal explanations for inconsistency are often called minimal unsatisfiable cores (MUCs) in CP [20] and minimal unsatisfiable subformulas (MUSes) in SAT [24]. Repairing with MaxSAT may be related with the identification of MUSes in SAT. A MaxSAT solution does not satisfy exactly one clause from each MUS of the corresponding SAT formula. The number of unsatisfied clauses may be less than the number of MUSes when some of the MUSes have a non-empty intersection.

The identification of assignments common to all solutions corresponds to the definition of backbone [28,23]. Backbones find applications not only in decision problems, but also in optimization problems [30]. In the context of MaxSAT, backbones have inspired the development of new search strategies [33,27] and the same occurred in other domains (e.g. see [17,21]). Recent work in SAT has focused on implementing efficient algorithms for identifying backbones in practical settings [26,34]. The solution developed in this paper follows one of the solutions proposed for post-silicon fault localization [34].

More sophisticated models exist for GRNs and other biological networks using CP solutions. An example is the framework developed by Corblin et al. [8,7,6]. The authors use a more complex formalism for modelling GRNs that allows multivalued variables and uses transition rules, amongst other particularities. For a survey of CP solutions to solve related biological problems refer to the relevant literature (e.g. see [31,3,1]).

3 Reasoning with Satisfiability

Our goal is to provide SAT and MaxSAT solutions for reasoning over biological networks. We begin by describing how to encode a GRN into SAT using the model introduced in the previous section. This encoding allows to validate an influence graph against an experimental profile. In case the graph and profile are mutually inconsistent, the identification of repairs to restore consistency is of interest. Building on the SAT encoding, we then introduce a MaxSAT encoding that allows to repair the graph and/or profile in order to restore consistency. In addition, we also provide information about what is common to all total labellings, in the case of consistency, or to all optimal repairs, in the case of inconsistency. This information is called prediction.

3.1 Checking Consistency

An influence graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \sigma)$ and respective experimental profile μ can be encoded into SAT as follows. For the sake of clarity, the constraints will not be presented in CNF. Translating such constrains to CNF should be a standard task though.

Let us first introduce three types of Boolean variables. For each vertex $v \in \mathcal{V}$, there is a Boolean variable inp_v such that inp_v is assigned the value *true* if v is an input vertex and *false* otherwise. For each vertex $v \in \mathcal{V}$, there is a Boolean variable $lvtx_v$ (label vertex) such that $lvtx_v$ is assigned the value *true/false* if

the corresponding label $\mu(v)$ is $+/-$. Likewise, for each edge $u \rightarrow v \in \mathcal{E}$, there is Boolean variable $ledg_{uv}$ (label edge) such that $ledg_{uv}$ is assigned the value *true/false* if the corresponding label $\sigma(u \rightarrow v)$ is $+/-$.

An additional type of (auxiliary) variables is needed to represent the value of $\mu \cdot \sigma$, which denotes the influence between vertices. For each edge $u \rightarrow v \in \mathcal{E}$, create a Boolean variable $infl_{uv}$ such that $infl_{uv}$ is assigned the value *true/false* if $\mu(u) \cdot \sigma(u \rightarrow v)$ is $+/-$.

Let us now introduce the constraints, starting with the ones corresponding to unit clauses. For each vertex $v \in \mathcal{V}$, introduce a unit clause (inp_v) if v is an input vertex. Otherwise, introduce a unit clause $(\neg inp_v)$. Given labellings μ/σ, introduce one unit clause for each vertex/edge that has a label. (Remember that μ and σ may be partial labellings and therefore not all vertices/edges may have a corresponding label.) For each vertex $v \in \mathcal{V}$ with $\mu(v) = +/-$ introduce a unit clause $(lvtx_v)/(\neg lvtx_v)$. For each edge $u \rightarrow v \in \mathcal{E}$ with $\sigma(u \rightarrow v) = +/-$ introduce a unit clause $(ledg_{uv})/(\neg ledg_{uv})$.

Example 3. Consider again the example in Figure 1. Encoding the influence graph into SAT produces the following unit clauses: $(\neg inp_a)$, $(\neg inp_b)$, $(\neg inp_c)$, $(ledg_{ab})$, $(\neg ledg_{ac})$, $(ledg_{ba})$, $(ledg_{bc})$, $(\neg ledg_{cb})$. Moreover, encoding the experimental profile into SAT produces the following unit clauses: $(lvtx_a)$, $(\neg lvtx_b)$.

In order to define the value of variables *infl* a few additional constraints are needed. The value of these variables is given by $\mu \cdot \sigma$. For each edge $u \rightarrow v$, the following constraints are added:

$$\begin{aligned} infl_{uv} &\longrightarrow (lvtx_u \wedge ledg_{uv}) \vee (\neg lvtx_u \wedge \neg ledg_{uv}) \\ \neg infl_{uv} &\longrightarrow (lvtx_u \wedge \neg ledg_{uv}) \vee (\neg lvtx_u \wedge ledg_{uv}) \end{aligned} \tag{1}$$

Finally, consistency must be ensured. An influence graph and an experimental profile are mutually consistent if total extensions for μ and σ can be found such that all non-input vertices are consistent. The consistency of a vertex v is ensured by making use of variables $infl_{uv}$ where u may be any vertex to which v is adjacent. For each vertex v, the following constraints are added:

$$\begin{aligned} inp_v \vee (lvtx_v &\longrightarrow \bigvee_u infl_{uv}) \\ inp_v \vee (\neg lvtx_v &\longrightarrow \bigvee_u \neg infl_{uv}) \end{aligned} \tag{2}$$

Example 4. Equations 1 and 2 applied to vertex c of Figure 1 produce the following constraints:

$$\begin{aligned} &infl_{ac} \longrightarrow (lvtx_a \wedge ledg_{ac}) \vee (\neg lvtx_a \wedge \neg ledg_{ac}), \\ &\neg infl_{ac} \longrightarrow (lvtx_a \wedge \neg ledg_{ac}) \vee (\neg lvtx_a \wedge ledg_{ac}), \\ &infl_{bc} \longrightarrow (lvtx_b \wedge ledg_{bc}) \vee (\neg lvtx_b \wedge \neg ledg_{bc}), \\ &\neg infl_{bc} \longrightarrow (lvtx_b \wedge \neg ledg_{bc}) \vee (\neg lvtx_b \wedge ledg_{bc}), \end{aligned}$$

$$inp_c \vee (lvtx_c \longrightarrow (infl_{ac} \vee infl_{bc})),$$
$$inp_c \vee (\neg lvtx_c \longrightarrow (\neg infl_{ac} \vee \neg infl_{bc})).$$

A SAT call to the complete CNF encoding of the influence graph reveals that the graph by itself is self-consistent. If we add the encoding of the experimental profile, i.e. $(lvtx_a) \wedge (\neg lvtx_b)$, another SAT call reveals that the influence graph and the experimental profile are mutually inconsistent. Observe that vertex a has only one incoming edge $b \rightarrow a$ with label $+$. Given that vertex b has label $-$, vertex a cannot have label $+$.

3.2 Repairing

When an influence graph is inconsistent, whether by itself or mutually with an experimental profile, one may consider repairing the graph and/or the profile in order to restore consistency. The motivation is that some of the observations may be unreliable. To this end, we allow three types of repair operations (and combinations thereof): `e`, flip edges labels; `i`, make non-input vertices become input vertices; and `v`, flip vertices labels. The goal is to identify *cardinality-minimal* repairs.

Restoring consistency is an optimization problem that can be encoded into Partial MaxSAT as follows. The SAT encoding described in the previous section is still valid when allowing repairs. However, the unit clauses referring to non-input vertices, vertices labels and edges labels can be unsatisfied if needed, depending on the type of repair. Hence, these clauses can be made *soft* in the Partial MaxSAT encoding. All other clauses, i.e. clauses encoding constraints (1) and (2), are hard clauses. For repairs of type `e`, unit clauses referring to edges labels are soft clauses. For repairs of type `i`, unit clauses referring to input vertices are soft clauses. For repairs of type `v`, unit clauses referring to vertices labels are soft clauses. All other unit clauses are hard clauses. The MaxSAT solution will satisfy all hard clauses and the largest number of soft clauses. Note that all soft clauses are unit clauses. The size of the repair corresponds to the number of unsatisfied clauses. The actual repairs can be trivially obtained from the unsatisfied clauses.

This encoding can be easily adapted to consider other repairs. The user could manually indicate which repairs would be reasonable to perform. For example, in some cases it can make sense to restrict repairs to a subset of vertices and respective edges.

Example 5. As discussed in Example 4, the influence graph and experimental profile illustrated in Figure 1 are mutually inconsistent. To identify repairs of type `e`, the following clauses are declared as soft clauses: $(ledg_{ab})$, $(\neg ledg_{ac})$, $(ledg_{ba})$, $(ledg_{bc})$, $(\neg ledg_{cb})$. The solution can be any one of the four cardinality-minimal repairs, all of them with size two: $\{\neg ledg_{ab}, \neg ledg_{ba}\}$, $\{\neg ledg_{ba}, ledg_{ac}\}$, $\{\neg ledg_{ba}, \neg ledg_{bc}\}$ and $\{\neg ledg_{ba}, ledg_{cb}\}$. Had been allowed only repairs of type `v`, the clauses to be made soft would be: $(lvtx_a)$, $(\neg lvtx_b)$. In this case, two different optimal repairs with size one could be obtained: $\{\neg lvtx_a\}$ and $\{lvtx_b\}$.

3.3 Predicting

Analysing all possible solutions to a given problem instance can become a puzzling task when the number of solutions is too large. In this context, it is certainly useful to know what is common to all solutions. Note that this concept applies both to decision problems and optimization problems as long as (optimal) solutions to a given problem instance can be found.

The intersection of all solutions to a given problem instance is called prediction. This concept can be applied either to consistent problem instances when checking consistency or to inconsistent problem instances when repairing. In SAT and MaxSAT, the assignments which are common to all (optimal) solutions correspond to the backbones of a given formula. However, when predicting, only a subset of the variables is relevant. Still, approaches used for identifying backbones in SAT and MaxSAT can be adapted to compute predictions.

Predicting under consistency can be applied to the SAT encoding described in Section 3.1 using a tool designed to identify backbone variables in SAT formulas. A prediction can be obtained after filtering irrelevant variables from the set of variables returned by the tool.

For the case of prediction under inconsistency, one has to consider the MaxSAT encoding described in Section 3.2. The set of repairs common to all optimal solutions can be obtained from the unit soft clauses that were not satisfied in all optimal solutions. (Observe that each unsatisfied unit soft clause corresponds to a repair.) Next we describe how a MaxSAT solver can be instrumented to efficiently compute predictions.

A naïve approach consists in enumerating all solutions and computing their intersection [26]. This approach requires n calls to a MaxSAT solver, being n the number of optimal solutions. (In practice, this number can be reduced taking into account that only a subset of the variables is relevant.) After each call, a blocking clause corresponding to the negation of the computed solution is added to the formula, in order to prevent the same solution from being found again in a future call. Moreover, after each iteration the actual intersection of the solutions computed so far is updated.

One key optimization to the naïve approach can make a significant difference. Instead of adding one blocking clause for each solution found, there is only one blocking clause, which corresponds to the prediction computed so far. This implies that not all solutions have to be computed. Only a solution that reduces the size of the current prediction can be computed at each iteration.

Algorithm 1 has been implemented to compute predictions under inconsistency using the optimization described above. Similarly to the naïve approach, this algorithm consists in retrieving different optimal solutions and reducing the prediction at each iteration. (Again, we can take advantage of the fact that only a subset of the variables is relevant.) But in practice less iterations are expected to be required.

Given a partial MaxSAT formula $\mathcal{F}$, the algorithm is initialized with an *optimum* value, obtained from calling MaxSAT($\mathcal{F}$) and the initial *prediction*, obtained with function Get-Repairs. Each call to the MaxSAT solver returns

Algorithm 1. Predicting under Inconsistency

Input : Partial MaxSAT Formula $\mathcal{F}$
Output : Predicted Repairs of $\mathcal{F}$, *prediction*

```
(out, opt, sol) ← MaxSAT(F)                          // compute initial solution
optimum ← opt
prediction ← Get-Repairs(sol)
while |prediction| ≠ 0 do
    (out, opt, sol) ← MaxSAT(F ∪ [¬prediction])      // block current prediction
    if out == UNSAT or opt > optimum then
        break
    prediction ← prediction ∩ Get-Repairs(sol)       // update prediction
return prediction
```

a 3-tuple (out, opt, sol), where *out* corresponds to the outcome (either SAT or UNSAT); *opt* corresponds to the optimal value, i.e. the number of *unsatisfied* clauses; and *sol* corresponds to the variable assignments which result in the optimum value.

The initial prediction corresponds to the set of repairs obtained from the first MaxSAT solution. At each iteration, the MaxSAT solver is called with the initial formula and the blocking clause corresponding to the current prediction, which is added as a hard clause. If the new solution is still optimal, then the corresponding repairs are intersected with the current prediction; otherwise the final prediction has been found. Note that either the current prediction is reduced at each step or the algorithm terminates. In this later case, the final prediction is returned.

Although the worst-case scenario of Algorithm 1 is as bad as the naïve approach, it performs generally well for this domain, as we will see later in Section 4.2.

Example 6. In Example 5 were listed four optimal repairs of type `e` for the influence graph and the experimental profile in Figure 1. The prediction corresponds to the intersection of all the solutions, i.e. the set $\{\neg ledg_{ba}\}$. A possible run of Algorithm 1 could be to first find repair $\{\neg ledg_{ab}, \neg ledg_{ba}\}$, thus next calling the MaxSAT solver with the original formula and the clause $[ledg_{ab} \vee ledg_{ba}]$. Suppose that afterwards repair $\{\neg ledg_{ba}, ledg_{ac}\}$ is found. So the next step is to call the MaxSAT solver with the original formula and the clause $[ledg_{ba}]$. This call returns UNSAT and so the algorithm terminates. Only two calls to the MaxSAT solver were needed, instead of the four calls of a naïve algorithm.

4 Experimental Evaluation

The experimental evaluation is driven by the goal of comparing the performance of ASP solvers with the performance of SAT and MaxSAT solvers. With this goal in mind, our evaluation is based on the same instances that were used to

evaluate the performance of ASP solvers in the past. These instances, as well as the experiments performed, are detailed in the literature [16,12]. In one of the experiments [16], ASP is used to solve randomly generated instances. The target in this case is to check the consistency of each one of the problem instances. In the other experiment [12], real instances are used and the target is to repair unsatisfiable problem instances. In addition, prediction is applied only to the instances for which a repair can be provided.

The computations were performed using the ASP solver *clasp* 2.0.6 together with grounder *gringo* 3.0.4 [13], an improved version of the MiniSat 2.2.0 SAT solver [10] available from *github*, the backbone identification tool for SAT `minibones` [26] and the MaxSAT solver MSUnCore 2.5 [25]. The experiments were run on several Intel Xeon 5160 machines (dual-cores with 3.00 GHz of clock speed, 4 MB of cache, 1333 MHz of FSB speed and 4 GB of RAM each), running 64-bit versions of Linux 2.6.33.3-85.fc13. All tools were configured to not take advantage of any sort of parallelism.

All times are shown in seconds and correspond to the average execution times taken by each tool for solving a set of problem instances. For each aborted instance, the timeout value of 600 seconds is added to the total sum. The time needed to translate each instance from raw data to ASP, SAT or MaxSAT format is not taken into account, as it is negligible and similar for any of these three formats.

4.1 Checking Consistency and Predicting under Consistency

The first experiment evaluates consistency checking and prediction under consistency for the randomly generated instances. These instances have between 500 and 4000 vertices. The degree of a vertex is on average 2.5, following what is assumed as standard in biological networks [22]. In total, there are 400 instances, 50 for each one of the eight different graph sizes (starting with 500 vertices and up to 4000, with an increment of 500 vertices).

Table 1 shows the average run times for each graph size. We distinguish between consistent and inconsistent instances, denoted as *sat* and *unsat*, respectively. The first columns relate to consistency checking. In these columns, ASP refers to running *gringo* together with *clasp* using the VSIDS heuristic (the most efficient heuristic according to [16]) and SAT refers to MiniSat.

The remaining columns relate to prediction under consistency. Prediction is only applied to satisfiable instances. As before, ASP refers to running *gringo* together with *clasp* using the VSIDS heuristic but now using the cautious reasoning mode (`--cautious`), which makes *clasp* compute the intersection between all answer sets. SAT now refers to running `minibones`, which is used to compute the backbones of SAT formulas.

The results are clear. Checking consistency is trivial for both ASP and SAT solvers. As expected, larger instances require more time but still the time required is not significant. The results obtained with the ASP solver are in conformity with the results available in the literature [16]. Prediction for satisfiable instances is also trivial. We believe that such information is very important from

Table 1. Times for consistency checking and prediction under consistency

		Consistency		Prediction	
		ASP	SAT	ASP	SAT
500	sat	0.11	0.01	0.15	0.05
	unsat	0.10	0.00		
1000	sat	0.26	0.02	0.42	0.18
	unsat	0.23	0.01		
1500	sat	0.42	0.03	0.79	0.39
	unsat	0.37	0.01		
2000	sat	0.58	0.03	1.26	0.69
	unsat	0.51	0.01		
2500	sat	0.75	0.04	1.88	1.08
	unsat	0.66	0.01		
3000	sat	0.91	0.06	2.79	1.57
	unsat	0.79	0.02		
3500	sat	1.08	0.07	3.97	2.14
	unsat	0.95	0.02		
4000	sat	1.24	0.05	5.37	2.82
	unsat	1.10	0.02		

a user's point of view. Whereas analysing dozens or hundreds of solutions is infeasible in practice, knowing which assignments *must* belong to any solution is certainly of interest.

Checking consistency and predicting under consistency is expected to scale well for larger instances using either approach. Nevertheless, at the light of the results obtained for real instances (see next section), extending this evaluation method to larger instances did not seem to be the best way to follow.

4.2 Repairing and Predicting under Inconsistency

This second experiment was conducted using as test set the transcriptional regulatory network of *Escherichia coli* K-12 [11] along with two experimental profiles: the Exponential-Stationary Growth Shift study [5] and the Heatshock experiment [2].

The goal of this experiment is to evaluate the feasibility of (optimally) repairing inconsistent problem instances, as well as of predicting under inconsistency, i.e. computing which assignments are common to all optimal repairs. We used the transcriptional regulatory network of *Escherichia coli* K-12, which contains 5140 interactions between 1915 genes. Each one of the two profiles, Exponential-Stationary Growth Shift and Heatshock, has slightly over 850 gene expression level variations, which correspond to vertex labellings. For a better assessment of the scalability of the approaches used, several data samples were generated by randomly selecting 5%, 10%, 20% and 50% of the whole data for each experimental profile. For each percentage were generated 50 inconsistent instances. The whole test set is made of 400 instances. (In previous work [12] were used percentages 3%, 6%, 9%, 12% and 15% instead, but the better performance of the most recent versions of *gringo* and *clasp* required the generation of more difficult instances.)

Table 2. Times for repair and prediction under inconsistency

Exponential-Stationary Growth Shift							
	e	i	v	ei	ev	iv	eiv
Repair (ASP)							
5%	0.67	0.35	0.27	0.67	0.59	0.41	0.69
10%	0.64	0.35	0.27	0.82	0.75	0.38	1.35
20%	0.94	0.36	0.28	18.28	7.05	0.67	77.98 (**3**)
50%	2.89	0.35	0.29	587.48 (**48**)	572.02 (**46**)	481.16 (**35**)	600.00 (**50**)
Prediction (ASP)							
5%	0.65	0.33	0.26	0.68	0.58	0.39	0.67
10%	0.61	0.32	0.26	0.75	0.67	0.36	0.96
20%	0.90	0.32	0.26	1.80	3.28	0.51	13.86
50%	1.92	0.32	0.27	41.71	320.30	302.43 (**4**)	–
Repair (MaxSAT)							
5%	0.22	0.22	0.21	0.17	0.17	0.17	0.17
10%	0.24	0.24	0.23	0.20	0.20	0.20	0.20
20%	0.37	0.35	0.34	0.29	0.29	0.29	0.29
50%	0.74	0.73	0.72	0.60	0.59	0.59	0.59
Prediction (MaxSAT)							
5%	0.41	0.41	0.39	0.44	0.45	0.45	0.45
10%	0.60	0.58	0.54	0.78	0.76	0.76	0.79
20%	1.41	1.18	0.99	2.11	1.95	1.99	1.97
50%	4.55	3.06	2.26	9.32	7.71	7.83	7.58

Heatshock							
	e	i	v	ei	ev	iv	eiv
Repair (ASP)							
5%	0.69	0.35	0.27	0.66	0.60	0.33	0.67
10%	0.69	0.35	0.27	0.99	0.83	0.34	1.50
20%	1.27	0.34	0.27	112.94 (**8**)	10.61	0.39	42.02
50%	279.43 (**21**)	0.35	0.28	572.03 (**47**)	504.76 (**37**)	202.10 (**12**)	600.00 (**50**)
Prediction (ASP)							
5%	0.67	0.33	0.26	0.66	0.59	0.30	0.66
10%	0.68	0.33	0.27	0.88	0.70	0.30	1.39
20%	0.91	0.31	0.27	17.86	1.07	0.33	9.88
50%	43.76	0.31	0.26	28.01	276.07 (**3**)	126.58 (**1**)	–
Repair (MaxSAT)							
5%	0.21	0.21	0.20	0.17	0.16	0.16	0.16
10%	0.25	0.24	0.24	0.21	0.19	0.19	0.19
20%	0.39	0.38	0.30	0.31	0.25	0.25	0.25
50%	1.02	0.88	0.61	0.86	0.50	0.51	0.50
Prediction (MaxSAT)							
5%	0.43	0.42	0.39	0.48	0.40	0.41	0.40
10%	0.76	0.63	0.56	1.06	0.71	0.71	0.68
20%	1.98	1.46	1.01	3.44	1.68	1.58	1.61
50%	15.14	4.96	2.43	31.71	7.88	6.79	6.58

Table 3. Times for repair and prediction under inconsistency (100% instances)

	Exponential-Stationary Growth Shift				Heatshock			
	Repair		**Prediction**		**Repair**		**Prediction**	
	ASP	MaxSAT	ASP	MaxSAT	ASP	MaxSAT	ASP	MaxSAT
e	4.97	1.46	3.56	13.49	600.00	2.88	–	99.97
i	0.33	1.41	0.30	5.97	0.30	2.77	0.27	16.15
v	0.31	1.43	0.29	4.38	0.28	1.22	0.28	7.73
e i	600.00	1.17	–	28.70	600.00	2.19	–	222.43
e v	600.00	1.14	–	22.01	600.00	0.94	–	26.50
i v	600.00	1.16	–	18.55	600.00	0.94	–	21.45
e i v	600.00	1.14	–	17.25	600.00	0.96	–	16.40

Table 2 shows the average run times for the ASP and MaxSAT approaches. Timeouts, shown within parentheses, represent that the imposed time limit of 600 seconds was exceeded before finding a solution. Observe that there were no timeouts for the MaxSAT runs. In the experiment, we allowed the following repair operations and combinations thereof (previously introduced in Section 3.2): `e`, flip edges signs; `i`, make non-input vertices become input vertices; and `v`, flip vertices signs. This results in 7 types of repairs, thus $400 \cdot 7 = 2800$ instances. Observe that the possibility of making a vertex become an input (`i` repair operation) makes that vertex trivially consistent.

The repairing phase determines which instances will be used in the prediction phase. It would make no sense to apply prediction to instances for which not even one optimal repair was provided within the time limit. Hence, prediction is applied only to the instances for which repairing was successful.

The results presented in Table 2 were obtained using *gringo* together with *clasp* using flag `--opt-heuristic=1` for better performance. To compute the predictions, *clasp* applies cautious reasoning to all optimal solutions (`--cautious`, `--opt-all=<opt-value>`, with `<opt-value>` being the optimal repair value). To repair an instance using MaxSAT we used MSUnCore. For prediction, we used Algorithm 1, described in Section 3.3. Note that the operation corresponding to the first line of the algorithm was already computed during the repair phase and therefore the computation is not repeated.

The results are again clear. ASP aborts 357 out of 2800 instances in the repair phase, plus 8 instances in the prediction phase, whereas MaxSAT solves all the instances in a few seconds. MaxSAT is far more adequate than ASP to repair inconsistent instances. Many instances could not be repaired within the time limit of 600 seconds with ASP. As a result, prediction could not be applied to these instances. This is true for the two experimental profiles. In contrast, the MaxSAT solver is able not only to repair all instances, but also to do it in a few seconds. Similar results are obtained for prediction. The number of calls to the MaxSAT solver range from 1 to 52 (on average from 1.08 to 28.30). There seems to be no clear relation between the number of times the MaxSAT solver is called and the prediction size.

Despite the clear trend in favour of the MaxSAT approach, the next step would be to evaluate more difficult problem instances. The most difficult instances would be the ones where the complete experimental profiles are used. This results in 7 instances for each profile, one for each combination of repair operations. Detailed results are given in Table 3. From these 14 instances, the ASP solver was able to repair 3 instances while the MaxSAT solver was able to repair *all* the instances, taking on average less than 3 seconds. Prediction was successfully applied to all the instances but one for which the ASP solver was able to provide a repair. MaxSAT was able to predict all the instances.

Even though the MaxSAT solver is able to provide a prediction taking on average 20 seconds, two outliers exist. Both outliers refer to the Heathstock profile, one to the `e` repair operation and the other to the `ei` combination. The first requires 99.97 seconds and the second requires 222.43 seconds. Compared to the remaining instances, these are the ones requiring more calls to the MaxSAT solver (35 and 94, respectively). However, these instances are not the ones with a larger prediction. Actually, the hardest instance has prediction size 0, which means that there are at least two disjoint optimal repairs. If two disjoint optimal repairs had been identified at the first two iterations, then no more iterations would have been needed. This fact suggests instrumenting the MaxSAT solver to find diverse solutions [19], which will be investigated in the future.

Evaluating the accuracy of prediction is out of the scope of this evaluation. Accuracy results have already been provided for the ASP approach [12]. The accuracy of prediction is quite high, being always above 90%. Given that both approaches are equivalent, it makes no sense to repeat such evaluation.

5 Conclusions and Future Work

We have studied how SAT and MaxSAT can be applied to reasoning over biological networks. The use of SAT and MaxSAT is certainly adequate, given that the domains of the variables of the actual problem are Boolean. SAT and MaxSAT encodings have been shown to be more competitive than other approaches used in the past, namely Answer Set Programming (ASP).

As future work we will consider other optimization criteria for repairs. For example, subset-minimal repairs, as already suggested by Gebser et al. [12]. Finding subset-minimal repairs comprises proving that removing any repair from the proposed solution no longer allows to achieve consistency. Additional types of repairs could be considered as well, such as adding edges to the influence graph [12]. As already mentioned, the proposed MaxSAT encoding can easily accommodate other types of repairs.

Another direction for the future is the evaluation of different algorithmic approaches to compute predictions under inconsistency. Ideas coming from existing algorithms for backbone identification in SAT [26] can be discussed as a starting point.

Acknowledgements. This work is partially supported by the Portuguese Foundation for Science and Technology (FCT) under the research project ASPEN (PTDC/EIA-CCO/110921/2009) and by INESC-ID multiannual funding from the PIDDAC program funds. We thank the reviewers for their insightful comments.

References

1. Workshops on Constraint Based Methods for Bioinformatics (WCB) (2005-2012)
2. Allen, T., Herrgård, M., Liu, M., Qiu, Y., Glasner, J., Blattner, F., Palsson, B.: Genome-scale analysis of the uses of the escherichia coli genome: model-driven analysis of heterogeneous data sets. Journal of Bacteriology 185(21), 6392–6399 (2003)
3. Barahona, P., Krippahl, L., Perriquet, O.: Bioinformatics: a challenge to constraint programming. In: Hybrid Optimization, vol. 45, pp. 463–487. Springer (2011)
4. Bobrow, D.: Qualitative reasoning about physical systems: an introduction. Artificial Intelligence 24(1-3), 1–5 (1984)
5. Bradley, M., Beach, M., de Koning, A., Pratt, T., Osuna, R.: Effects of fis on escherichia coli gene expression during different growth stages. Microbiology 153(9), 2922–2940 (2007)
6. Corblin, F., Bordeaux, L., Fanchon, E., Hamadi, Y., Trilling, L.: Connections and integration with SAT solvers: a survey and a case study in computational biology. In: Hybrid Optimization, vol. 45, pp. 425–461. Springer (2011)
7. Corblin, F., Fanchon, E., Trilling, L.: Applications of a formal approach to decipher discrete genetic networks. BMC Bioinformatics 11, 385 (2010)
8. Corblin, F., Tripodi, S., Fanchon, E., Ropers, D., Trilling, L.: A declarative constraint-based method for analyzing discrete genetic regulatory networks. Biosystems 98(2), 91–104 (2009)
9. de Jong, H.: Modeling and simulation of genetic regulatory systems: a literature review. Journal of Computational Biology 9(1), 67–103 (2002)
10. Eén, N., Sörensson, N.: An Extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
11. Gama-Castro, S., Jiménez-Jacinto, V., Peralta-Gil, M., Santos-Zavaleta, A., Peñaloza-Spinola, M., Contreras-Moreira, B., Segura-Salazar, J., Muñiz-Rascado, L., Martínez-Flores, I., Salgado, H., Bonavides-Martínez, C., Abreu-Goodger, C., Rodríguez-Penagos, C., Miranda-Ríos, J., Morett, E., Merino, E., Huerta, A., Treviño-Quintanilla, L., Collado-Vides, J.: RegulonDB (version 6.0): gene regulation model of escherichia coli k-12 beyond transcription, active (experimental) annotated promoters and textpresso navigation. Nucleic Acids Research 36(Database Issue), 120–124 (2008)
12. Gebser, M., Guziolowski, C., Ivanchev, M., Schaub, T., Siegel, A., Thiele, S., Veber, P.: Repair and prediction (under inconsistency) in large biological networks with answer set programming. In: International Conference on Principles of Knowledge Representation and Reasoning, pp. 497–507 (2010)
13. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: the Potsdam answer set solving collection. AI Communications 24(2), 107–124 (2011)
14. Gebser, M., Kaufmann, B., Schaub, T.: The Conflict-Driven Answer Set Solver *clasp*: Progress Report. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 509–514. Springer, Heidelberg (2009)

15. Gebser, M., König, A., Schaub, T., Thiele, S., Veber, P.: The BioASP library: ASP solutions for systems biology. In: IEEE International Conference on Tools with Artificial Intelligence, pp. 383–389 (2010)
16. Gebser, M., Schaub, T., Thiele, S., Veber, P.: Detecting inconsistencies in large biological networks with answer set programming. Theory and Practice of Logic Programing 11(2-3), 323–360 (2011)
17. Gregory, P., Fox, M., Long, D.: A New Empirical Study of Weak Backdoors. In: Stuckey, P.J. (ed.) CP 2008. LNCS, vol. 5202, pp. 618–623. Springer, Heidelberg (2008)
18. Guziolowski, C., Veber, P., Le Borgne, M., Radulescu, R., Siegel, A.: Checking consistency between expression data and large scale regulatory networks: a case study. Journal of Biological Physics and Chemistry 7(2), 37–43 (2007)
19. Hebrard, E., Hnich, B., O'Sullivan, B., Walsh, T.: Finding diverse and similar solutions in constraint programming. In: AAAI Conference on Artificial Intelligence, pp. 372–377 (2005)
20. Hemery, F., Lecoutre, C., Sais, L., Boussemart, F.: Extracting MUCs from constraint networks. In: European Conference on Artificial Intelligence, pp. 113–117 (2006)
21. Hsu, E.I., Muise, C.J., Christopher Beck, J., McIlraith, S.A.: Probabilistically Estimating Backbones and Variable Bias: Experimental Overview. In: Stuckey, P.J. (ed.) CP 2008. LNCS, vol. 5202, pp. 613–617. Springer, Heidelberg (2008)
22. Jeong, H., Tombor, B., Albert, R., Oltvai, Z., Barabási, A.-L.: The large-scale organization of metabolic networks. Nature 407(6804), 651–654 (2000)
23. Kilby, P., Slaney, J., Thiébaux, S., Walsh, T.: Backbones and backdoors in satisfiability. In: AAAI Conference on Artificial Intelligence, pp. 1368–1373 (2005)
24. Marques-Silva, J.: Minimal unsatisfiability: models, algorithms and applications. In: IEEE International Symposium on Multiple-Valued Logic, pp. 9–14 (2010)
25. Marques-Silva, J., Manquinho, V.: Towards More Effective Unsatisfiability-Based Maximum Satisfiability Algorithms. In: Kleine Büning, H., Zhao, X. (eds.) SAT 2008. LNCS, vol. 4996, pp. 225–230. Springer, Heidelberg (2008)
26. Marques-Silva, J., Mikoláš, J., Lynce, I.: On computing backbones of propositional theories. In: European Conference on Artificial Intelligence, pp. 15–20 (2010)
27. Menaï, M.: A two-phase backbone-based search heuristic for partial MAX-SAT – an initial investigation. In: Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, pp. 681–684 (2005)
28. Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., Troyansky, L.: Determining computational complexity from characteristic 'phase transitions'. Nature 400(6740), 133–137 (1999)
29. Siegel, A., Radulescu, O., Le Borgne, M., Veber, P., Ouy, J., Lagarrigue, S.: Qualitative analysis of the relation between DNA microarray data and behavioral models of regulation networks. Biosystems 84(2), 153–174 (2006)
30. Slaney, J., Walsh, T.: Backbones in optimization and approximation. In: International Joint Conference on Artificial Intelligence, pp. 254–259 (2001)
31. Soliman, S.: Constraint programming for the dynamical analysis of biochemical systems – a survey. Technical Report Deliverable 1.6, ANR CALAMAR, ANR-08-SYSC-003 (2011)
32. Soulé, C.: Mathematical approaches to differentiation and gene regulation. Comptes Rendus Biologies 329(1), 13–20 (2006)
33. Zhang, W., Rangan, A., Looks, M.: Backbone guided local search for maximum satisfiability. In: International Joint Conference on Artificial Intelligence, pp. 1179–1184 (2003)
34. Zhu, C., Weissenbacher, G., Sethi, D., Malik, S.: SAT-based techniques for determining backbones for post-silicon fault localisation. In: IEEE International High Level Design Validation and Test Workshop, pp. 84–91 (2011)