Detecting Inconsistencies in Large Biological Networks with Answer Set Programming*

Martin Gebser[1], Torsten Schaub[1], Sven Thiele[1], Björn Usadel[2], and Philippe Veber[1]

[1] University of Potsdam, Institute for Informatics, August-Bebel-Str. 89, D-14482 Potsdam
[2] Max Planck Institute of Molecular Plant Physiology, Am Mühlenberg 1, D-14476 Potsdam

Abstract. We introduce an approach to detecting inconsistencies in large biological networks by using Answer Set Programming. To this end, we build upon a recently proposed notion of consistency between biochemical/genetic reactions and high-throughput profiles of cell activity. We then present an approach based on Answer Set Programming to check the consistency of large-scale data sets. Moreover, we extend this methodology to provide explanations for inconsistencies in the data by determining minimal representations of conflicts. In practice, this can be used to identify unreliable data or to indicate missing reactions.

1 Introduction

Molecular biology has seen a technological revolution with the establishment of high-throughput methods in the last years. These methods allow for gathering multiple orders of magnitude more data than was procurable before. Furthermore, there is an increasing number of biological repositories on the web, such as KEGG, AraCyc, EcoCyc, RegulonDB, and others, incorporating thousands of biochemical reactions and genetic regulations. For combining huge amounts of experimental data with the knowledge gathered in these repositories, one needs appropriate and powerful knowledge representation tools that allow for modeling complex biological systems and their behavior.

In this paper, we deal with the analysis of high-throughput measurements in molecular biology, like microarray data or metabolic profiles [1]. Up to now, it is still a common practice to use expression profiles merely for detecting over- or under-expressed genes under specific conditions, leaving the task of making biological sense out of tens of gene identifiers to human experts. However, many efforts have also been made these years to make a better use of high-throughput data, in particular, by integrating them into large-scale models of transcriptional regulation or metabolic processes [2,3].

One possible approach consists in investigating the compatibility between the experimental measurements and the knowledge available in reaction databases. This can be done by using formal frameworks, for instance, those developed in [4] and [5]. A crucial feature of this methodology is its ability to cope with qualitative knowledge (for instance, reactions lacking kinetic details) and noisy data. In this work, we rely on the so-called *Sign Consistency Model* (SCM) due to Siegel et al. [4]. SCM imposes

* A preliminary version of this paper was presented at the Workshop on Constraint Based Methods for Bioinformatics (WCB'08).

constraints between experimental measurements and a graph representation of cellular interactions, called an *influence graph* [6].

Building on SCM, we develop declarative techniques based on *Answer Set Programming* (ASP) [7,8,9] to detect and explain inconsistencies in large data sets. This approach has several advantages. First, it allows us to formulate biological problems in a declarative way, thus easing the communication with biological experts. Second, although we do not detail it here, the rich modeling language facilitates integrating different knowledge representation and reasoning techniques, like abduction, planning, explanation, prediction, etc., in a uniform and transparent way. And finally, modern ASP solvers are based on advanced Boolean constraint solving technology and thus provide us with highly efficient inference engines. Apart from modeling the aforementioned biological problems in ASP, our major concern lies with the scalability of the approach. To this end, we do not only illustrate our application domain on an example but, moreover, design an artificial yet biologically meaningful benchmark suite indicating that an ASP-based approach scales well on the considered class of applications.

To begin with, we introduce SCM in Section 2. Section 3 briefly describes ASP, providing the syntax and semantics used in our application. In Section 4, we develop an ASP formulation of checking the consistency between experimental profiles and influence graphs. We further extend this approach in Section 5 to identifying minimal representations of conflicts if the experimental data is inconsistent with an influence graph. Section 6 is dedicated to an empirical evaluation of our approach along with an exemplary case study illustrating our application domain. Section 7 concludes this paper with a brief discussion and an outlook on future work.

2 Influence Graphs and Sign Consistency Constraints

Influence graphs [6] are a common representation for a wide range of dynamical systems. In the field of genetic networks, they have been investigated for various classes of systems, ranging from ordinary differential equations [10] to synchronous [11] and asynchronous [12] Boolean networks. Influence graphs have also been introduced in the field of qualitative reasoning [13] to describe physical systems where a detailed quantitative description is not available. This has also been the main motivation for using influence graphs for knowledge representation in the context of biological systems.

An *influence graph* is a directed graph whose vertices are the input and state variables of a system and whose edges express the effects of variables on each other. An edge $j \to i$ means that the variation of j in time influences the level of i. Every edge $j \to i$ of an influence graph is labeled with a sign, either **+** or **–**, denoted by $\sigma(j,i)$, where **+** (**–**) indicates that j tends to increase (decrease) i. An example influence graph is given in Figure 1; it represents a simplified model for the operon lactose in *E. coli*.

In SCM, *experimental profiles* are supposed to come from steady state shift experiments where, initially, the system is at steady state, then perturbed using control parameters, and eventually, it settles into another steady state. It is assumed that the data measures the differences between the initial and the final state. Thus, for genes, proteins, or metabolites, we know whether the concentration has increased or decreased, while quantitative values are unavailable, unessential, or unreliable. By $\mu(i)$, we denote

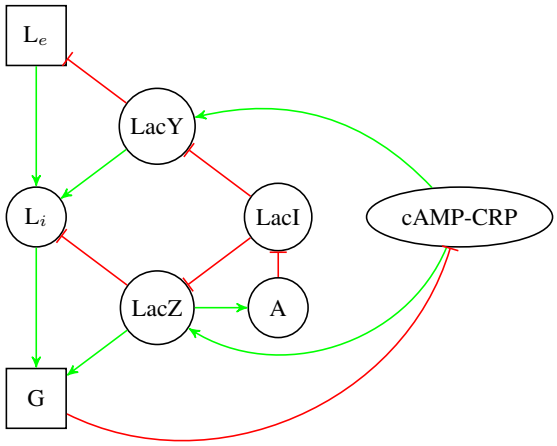

Fig. 1. Simplified model of operon lactose in *E. coli*, represented as an influence graph. The vertices represent either genes, metabolites, or proteins, while the edges indicate the regulations among them. Green edges with an arrow stand for positive regulations (activations), while red edges with a tee head stand for negative regulations (inhibitions). Vertices G and L_e are considered to be inputs of the system, that is, their signs are not constrained via their incoming edges.

the sign, again either **+** or **–**, of the variation of a species i between the initial and the final condition. One can easily enhance this setting by also considering null (or more precisely, non-significant) variations, by exploiting the concept of sign algebra [13].

Given an influence graph (as a representation of cellular interactions) and a labeling of its vertices with signs (as a representation of experimental profiles), we now describe the constraints that relate both. Informally, for every non-input vertex i, the observed variation $\mu(i)$ should be explained by the influence of at least one predecessor j of i in the influence graph. Thereby, the *influence* of j on i is given by the sign $\mu(j)\sigma(j,i) \in \{+,-\}$, where the multiplication of signs is derived from that of numbers. Sign consistency constraints can then be formalized as follows.

Definition 1 (Sign Consistency Constraints). *Let (V, E, σ) be an influence graph, where V is the set of vertices, E the set of edges, and $\sigma : E \to \{+,-\}$ a labeling of the edges. Furthermore, let $\mu : V \to \{+,-\}$ be a vertex labeling.*

Then, for every non-input vertex $i \in V$, the sign $\mu(i)$ of i is consistent, if there is some edge $j \to i$ in E such that $\mu(i) = \mu(j)\sigma(j,i)$.

The notion of (sign) consistency is extended to whole influence graphs in the natural way, requiring the sign of each non-input vertex to be consistent. In practice, influence graphs and experimental profiles are likely to be partial. Thus, we say that a partial labeling of the vertices is consistent with a partially labeled influence graph, if there is some consistent extension of vertex and edge labelings to all vertices and edges.

Table 1 shows four different vertex labelings for the influence graph given in Figure 1. Total labeling μ_1 is consistent with the influence graph: the variation of each

Species	L_e	L_i	G	LacY	LacZ	LacI	A	cAMP-CRP
μ_1	−	−	−	−	−	+	−	+
μ_2	+	+	−	+	−	+	−	−
μ_3	+	?	−	?	?	+	?	?
μ_4	?	?	?	−	+	?	?	+

Table 1. Some vertex labelings (reflecting measurements of two steady states) for the influence graph depicted in Figure 1; unobserved values are indicated by a question mark '**?**'.

vertex (except for input vertex L_e) can be explained by the effect of one of its regulators. For instance, in μ_1, LacY receives a positive influences from cAMP-CRP as well as a negative influence from LacI, the latter accounting for the decrease of LacY. The second labeling, μ_2, is not consistent: this time LacY receives only negative influences from cAMP-CRP and LacI, and its increase cannot be explained. Furthermore, partial vertex labeling μ_3 is consistent with the influence graph in Figure 1, as setting the signs of L_i, LacY, LacZ, A, and cAMP-CRP to **+**, **–**, **–**, **–**, and **+**, respectively, extends μ_3 to a consistent total labeling. In contrast, μ_4 cannot be extended consistently.

3 Answer Set Programming

This section provides a brief introduction to ASP (see [9] for details), a declarative paradigm for knowledge representation and reasoning, offering a rich modeling language [14,15] along with highly efficient inference engines based on Boolean constraint solving technology [16,17,18,19]. The basic idea of ASP is to encode a problem as a logic program such that its answer sets represent solutions to the original problem.

In view of our application, we take advantage of the elevated expressiveness of disjunctive programs, being able to capture problems at the second level of the polynomial hierarchy [20,21]. A *disjunctive logic program* over an alphabet $\mathcal{A}$ is a finite set of *rules* of the form

$$a_1; \ldots; a_l \leftarrow b_{l+1}, \ldots, b_m, not\ c_{m+1}, \ldots, not\ c_n\ , \tag{1}$$

where a_i, b_j, c_k are *atoms* for $0 < i \leq l < j \leq m < k \leq n$. A *literal* is an atom a or its (default) negation $not\ a$. A rule r as in (1) is called a *fact*, if $l = n = 1$, and an *integrity constraint*, if $l = 0$. Let $head(r) = \{a_1, \ldots, a_l\}$ be the *head* of r and $body(r) = \{b_{l+1}, \ldots, b_m, not\ c_{m+1}, \ldots, not\ c_n\}$ be the *body* of r. Given a set L of literals, let $L^+ = \{a \in \mathcal{A} \mid a \in L\}$ and $L^- = \{a \in \mathcal{A} \mid not\ a \in L\}$.

An interpretation is represented by the set of atoms that are true in it. A *model* of a program P is an interpretation in which all rules in P are true according to the standard definition of truth in propositional logic (while treating rules and default negation as implications and classical negation, respectively). Note that the (empty) head of an integrity constraint is false wrt every interpretation, while the empty body is true wrt every interpretation. Answer sets of P are particular models of P satisfying an additional stability criterion. Roughly, a set X of atoms is an answer set, if for every rule of form (1), X contains a minimum of atoms among $a_1, \ldots, a_l$ whenever $b_{l+1}, \ldots, b_m$ belong to X and no $c_{m+1}, \ldots, c_n$ belongs to X. However, note that the disjunction in

heads of rules, in general, is not exclusive. Formally, an *answer set* X of a program P is a $\subseteq$-minimal model of

$$\{head(r) \leftarrow body(r)^+ \mid r \in P, body(r)^- \cap X = \emptyset\} .$$

For example, program $\{a; b \leftarrow.\ c; d \leftarrow a, not\ b.\ \leftarrow b.\}$ has answer sets $\{a, c\}$ and $\{a, d\}$.

Although answer sets are usually defined on ground (i.e., variable-free) programs, the rich modeling language of ASP allows for non-ground problem encodings, where schematic rules stand for their ground instantiations. Grounders, like *gringo* [22] and *lparse* [15], are capable of combining a problem encoding and an instance (typically a set of ground facts) into an equivalent ground program, processed by some ASP solver. We follow this methodology and provide encodings for the problems considered below.

4 Checking Consistency

We now come to the first main question addressed in this paper, namely, how to check whether an experimental profile is consistent with a given influence graph. Note that, if the profile provides us with a sign for each vertex of the influence graph, the task can be accomplished simply by checking whether each non-input vertex receives at least one influence matching its variation. However, as soon as the experimental profile has missing values (which is very likely in practice), the problem becomes NP-hard [23]. In fact, a Boolean satisfiability problem over clauses $\{C_1, \ldots, C_m\}$ and variables $\{x_1, \ldots, x_n\}$ can be reduced as follows: introduce unlabeled input vertices $x_1, \ldots, x_n$, non-input vertices $C_1, \ldots, C_m$ labeled **+**, and edges $x_j \rightarrow C_i$ labeled **+** (**–**) if x_j occurs positively (negatively) in C_i. It is not hard to check that the labeling of $C_1, \ldots, C_m$ by **+** is consistent with the obtained influence graph iff $\{C_1, \ldots, C_m\}$ is satisfiable.

We next provide a logic program such that each of its answer sets matches a consistent extension of vertex and edge labelings. Our encodings as well as instances are available at [24]. For clarity, we here present them in a simplified manner and omit some convenient but unessential encoding optimizations. Our program is composed of three parts, described in the following subsections.

4.1 Problem Instance

An influence graph as well as an experimental profile are given by ground facts. For each species i, we introduce a fact *vertex*(i), and for each edge $j \rightarrow i$, a fact *edge*(j, i). If $s \in \{\textbf{+}, \textbf{–}\}$ is known to be the variation of a species i or the sign of an edge $j \rightarrow i$, it is expressed by a fact *observedV*(i, s) or *observedE*(j, i, s), respectively. Finally, a vertex i is declared to be input via a fact *input*(i).

For example, negative regulation LacI $\rightarrow$ LacY in the influence graph shown in Figure 1 and observation **+** for LacI (as with μ_3 in Table 1) give rise to the following facts:

$$\begin{array}{l} \textit{vertex}(\text{LacI}). \\ \textit{vertex}(\text{LacY}). \\ \textit{edge}(\text{LacI}, \text{LacY}). \\ \textit{observedV}(\text{LacI}, \textbf{+}). \\ \textit{observedE}(\text{LacI}, \text{LacY}, \textbf{–}). \end{array} \qquad (2)$$

Note that the absence of a fact of form $\mathit{observedV}(\mathrm{LacY}, s)$ means that the variation of LacY is unobserved (as with μ_3). In (2), we use LacI and LacY as names for constants associated with the species in Figure 1, but not as first-order variables. Similarly, for uniformity of notations, + and – are written in (2) for constants identifying signs.

4.2 Generating Solution Candidates

As mentioned above, our goal is to check whether an experimental profile is consistent with an influence graph. If so, it is witnessed by total labelings of the vertices and edges, which are generated via the following rules:

$$\begin{aligned} \mathit{labelV}(V, +); \mathit{labelV}(V, -) &\leftarrow \mathit{vertex}(V). \\ \mathit{labelE}(U, V, +); \mathit{labelE}(U, V, -) &\leftarrow \mathit{edge}(U, V). \end{aligned} \tag{3}$$

Moreover, the following rules ensure that known labels are respected by total labelings:

$$\begin{aligned} \mathit{labelV}(V, S) &\leftarrow \mathit{observedV}(V, S). \\ \mathit{labelE}(U, V, S) &\leftarrow \mathit{observedE}(U, V, S). \end{aligned} \tag{4}$$

Note that the stability criterion for answer sets demands that a known label derived via rules in (4) is also derived via rules in (3), thus, excluding the opposite label. In fact, the disjunctive rules used in this section could actually be replaced with non-disjunctive rules via "shifting" [25], given that our first encoding results in a so-called *head-cycle-free* (HCF) [26] ground program. However, the disjunctive rules in (3) will be reused in Section 5 where they cannot be compiled away. Also note that HCF programs, for which deciding answer set existence stays in NP, are recognized as such by disjunctive ASP solvers [19,27,28]. Hence, the purely syntactic use of disjunction is not harmful.

The following ground rules are obtained by combining the schematic rules in (3) and (4) with the facts in (2):

$$\begin{aligned} \mathit{labelV}(\mathrm{LacI}, +); \mathit{labelV}(\mathrm{LacI}, -) &\leftarrow \mathit{vertex}(\mathrm{LacI}). \\ \mathit{labelV}(\mathrm{LacY}, +); \mathit{labelV}(\mathrm{LacY}, -) &\leftarrow \mathit{vertex}(\mathrm{LacY}). \\ \mathit{labelE}(\mathrm{LacI}, \mathrm{LacY}, +); \mathit{labelE}(\mathrm{LacI}, \mathrm{LacY}, -) &\leftarrow \mathit{edge}(\mathrm{LacI}, \mathrm{LacY}). \\ \mathit{labelV}(\mathrm{LacI}, +) &\leftarrow \mathit{observedV}(\mathrm{LacI}, +). \\ \mathit{labelE}(\mathrm{LacI}, \mathrm{LacY}, -) &\leftarrow \mathit{observedE}(\mathrm{LacI}, \mathrm{LacY}, -). \end{aligned} \tag{5}$$

One can check that the program consisting of the facts in (2) and the rules in (5) admits two answer sets, the first one including $\mathit{labelV}(\mathrm{LacY}, +)$ and the second one including $\mathit{labelV}(\mathrm{LacY}, -)$. On the remaining atoms, both answer sets coincide by containing the atoms in (2) along with $\mathit{labelV}(\mathrm{LacI}, +)$ and $\mathit{labelE}(\mathrm{LacI}, \mathrm{LacY}, -)$.

4.3 Testing Solution Candidates

We now check whether generated total labelings satisfy the sign consistency constraints stated in Definition 1, requiring an influence of sign s for each non-input vertex i with variation s. We thus define $\mathit{receive}(i, s)$ to indicate that i receives an influence of sign s:

$$\begin{aligned} \mathit{receive}(V, +) &\leftarrow \mathit{labelE}(U, V, S), \mathit{labelV}(U, S). \\ \mathit{receive}(V, -) &\leftarrow \mathit{labelE}(U, V, S), \mathit{labelV}(U, T), S \neq T. \end{aligned} \tag{6}$$

Inconsistent labelings, where a non-input vertex does not receive any influence matching its variation, are then ruled out by integrity constraints of the following form:

$$\leftarrow \mathit{labelV}(V, S), \mathit{not}\ \mathit{receive}(V, S), \mathit{not}\ \mathit{input}(V). \tag{7}$$

Note that the schematic rules in (6) and (7) are given in the input language of grounder *gringo* [22], available at [29]. This allows us to omit an explicit listing of some (domain) predicates in the bodies of rules, which would be necessary when using *lparse* [15]. At [24], we provide encodings both for *gringo* and also more verbose ones for *lparse*.

Starting from the answer sets described in the previous subsection, the included atoms $\mathit{labelE}(\mathrm{LacI}, \mathrm{LacY}, \mathbf{-})$ and $\mathit{labelV}(\mathrm{LacI}, \mathbf{+})$ allow us to derive $\mathit{receive}(\mathrm{LacY}, \mathbf{-})$ via a ground instance of the second rule in (6), while $\mathit{receive}(\mathrm{LacY}, \mathbf{+})$ is underivable. After adding $\mathit{receive}(\mathrm{LacY}, \mathbf{-})$, the solution candidate containing $\mathit{labelV}(\mathrm{LacY}, \mathbf{-})$ satisfies the ground instances of the integrity constraint in (7) obtained by substituting LacY for V. Assuming LacI to be an input, as it can be declared via fact $\mathit{input}(\mathrm{LacI})$, we thus obtain an answer set containing $\mathit{labelV}(\mathrm{LacY}, \mathbf{-})$, expressing a decrease of LacY. In contrast, since $\mathit{receive}(\mathrm{LacY}, \mathbf{+})$ is underivable, the solution candidate containing $\mathit{labelV}(\mathrm{LacY}, \mathbf{+})$ violates the following ground instance of (7):

$$\leftarrow \mathit{labelV}(\mathrm{LacY}, \mathbf{+}), \mathit{not}\ \mathit{receive}(\mathrm{LacY}, \mathbf{+}), \mathit{not}\ \mathit{input}(\mathrm{LacY}).$$

That is, the solution candidate with $\mathit{labelV}(\mathrm{LacY}, \mathbf{+})$ does not pass the consistency test.

5 Identifying Minimal Inconsistent Cores

In view of the usually large amount of data, it is crucial to provide concise explanations, whenever an experimental profile is inconsistent with an influence graph (i.e., if the logic program given in the previous section has no answer set). To this end, we adopt a strategy that was successfully applied on real biological data [30]. The basic idea is to isolate minimal subgraphs of an influence graph such that the vertices and edges cannot be labeled consistently. This task is closely related to extracting Minimal Unsatisfiable Cores (MUCs) [31] in the context of Boolean satisfiability (SAT) [16]. In allusion, we call a minimal subgraph of an influence graph whose vertices and edges cannot be labeled consistently a *Minimal Inconsistent Core* (MIC). Note that identifying a MUC is D^{P}-complete [31,32], which is why we use disjunctive programs to encode MICs.

For illustration, consider the influence graph and the MIC shown in Figure 2. One can check that the observed simultaneous increase of **B** and **D** is not consistent with the influence graph, but the reason for this might not be apparent at first glance. However, once the depicted MIC is extracted, we immediately see that the increase of **B** implies an increase of **A**, so that the observed increase of **D** cannot be explained.

We next provide an encoding for identifying MICs, where a problem instance, that is, an influence graph along with an experimental profile, is represented by facts as specified in Section 4.1. The encoding then consists of three parts: the first generating MIC candidates, the second asserting inconsistency, and the third verifying minimality. The generating part comprises the rules in (3) and (4), and in addition, it includes:

$$\mathit{active}(V); \mathit{inactive}(V) \leftarrow \mathit{vertex}(V), \mathit{not}\ \mathit{input}(V). \tag{8}$$

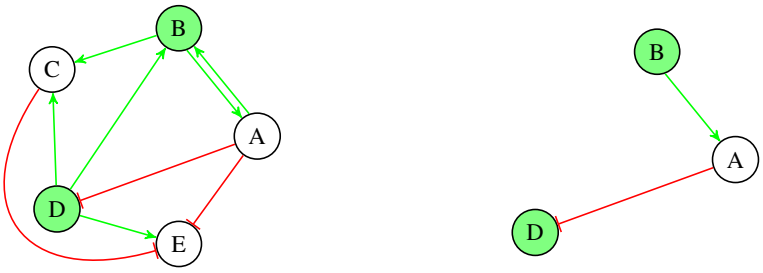

Fig. 2. A partially labeled influence graph and a contained MIC.

This additional rule permits guessing non-input vertices to be marked as active. The subgraph of the influence graph consisting of the active vertices, their regulators, and the connecting edges forms a MIC candidate, tested via the two encoding parts below.

5.1 Testing for Inconsistency

By adapting a methodology used in [21], the following subprogram makes sure that the active vertices belong to a subgraph that cannot be labeled consistently, while all possible labelings of the residual vertices and edges are (implicitly) taken into account:[3]

$$
\begin{aligned}
\mathit{opposite}(U,V) &\leftarrow \mathit{labelE}(U,V,\mathbf{-}), \mathit{labelV}(U,S), \mathit{labelV}(V,S).\\
\mathit{opposite}(U,V) &\leftarrow \mathit{labelE}(U,V,\mathbf{+}), \mathit{labelV}(U,S), \mathit{labelV}(V,T), S \neq T.\\
\mathit{bottom} &\leftarrow \mathit{active}(V), \mathit{opposite}(U,V) : \mathit{edge}(U,V).\\
&\leftarrow \mathit{not}\ \mathit{bottom}.\\
\mathit{labelV}(V,\mathbf{+}) &\leftarrow \mathit{bottom}, \mathit{vertex}(V).\\
\mathit{labelV}(V,\mathbf{-}) &\leftarrow \mathit{bottom}, \mathit{vertex}(V).\\
\mathit{labelE}(U,V,\mathbf{+}) &\leftarrow \mathit{bottom}, \mathit{edge}(U,V).\\
\mathit{labelE}(U,V,\mathbf{-}) &\leftarrow \mathit{bottom}, \mathit{edge}(U,V).
\end{aligned}
$$

In this (part of the) encoding, $\mathit{opposite}(U,V)$ indicates that the influence of regulator U on V is opposite to the variation of V. If all regulators of an active vertex V have such an opposite influence, the sign consistency constraint for V is violated, in which case atom *bottom* along with all labels for vertices and edges are derived. Note that the stability criterion for an answer set X imposes that *bottom* and all labels belong to X only if the active vertices cannot be labeled consistently. Finally, integrity constraint $\leftarrow \mathit{not}\ \mathit{bottom}$ necessitates the inclusion of *bottom* in any answer set, thus, stipulating an inevitable sign consistency constraint violation for some active vertex.

Reconsidering our example in Figure 2, the ground instances of (8) permit guessing $\mathit{active}(\mathbf{A})$ and $\mathit{active}(\mathbf{D})$. When labeling **A** with **+** (or assuming $\mathit{labelV}(\mathbf{A},\mathbf{+})$ to be true), we derive $\mathit{opposite}(\mathbf{A},\mathbf{D})$ and *bottom*, producing in turn all labels for vertices and

[3] In the language of *gringo* (and *lparse* [15]), the expression $\mathit{opposite}(U,V) : \mathit{edge}(U,V)$ used below refers to the conjunction of all ground atoms $\mathit{opposite}(j,i)$ for which $\mathit{edge}(j,i)$ holds.

edges. Furthermore, setting the sign of **A** to **–** (or *labelV*$(\mathbf{A}, \text{–})$ to true) makes us derive *opposite*$(\mathbf{B}, \mathbf{A})$, which again gives *bottom* and all labels for vertices and edges. We have thus verified that the sign consistency constraints for **A** and **D** cannot be jointly satisfied, given the observed increases of **B** and **D**. That is, active vertices **A** and **D** are sufficient to explain the inconsistency between the observations and the influence graph.

5.2 Testing for Minimality

It remains to be verified whether the sign consistency constraints for all active vertices are necessary to identify an inherent inconsistency. This test is based on the idea that, excluding any active vertex, the sign consistency constraints for the other active vertices should be satisfied by appropriate labelings. This can be implemented as follows:

$$
\begin{aligned}
\textit{labelV'}(W, V, \text{+}); \textit{labelV'}(W, V, \text{–}) &\leftarrow \textit{active}(W), \textit{vertex}(V).\\
\textit{labelE'}(W, U, V, \text{+}); \textit{labelE'}(W, U, V, \text{–}) &\leftarrow \textit{active}(W), \textit{edge}(U, V).\\
\textit{labelV'}(W, V, S) &\leftarrow \textit{active}(W), \textit{observedV}(V, S).\\
\textit{labelE'}(W, U, V, S) &\leftarrow \textit{active}(W), \textit{observedE}(U, V, S).\\
\textit{receive'}(W, V, \text{+}) &\leftarrow \textit{labelE'}(W, U, V, S), \textit{labelV'}(W, U, S).\\
\textit{receive'}(W, V, \text{–}) &\leftarrow \textit{labelE'}(W, U, V, S), \textit{labelV'}(W, U, T), S \neq T.\\
&\leftarrow \textit{labelV'}(W, V, S), \textit{active}(V), V \neq W, \textit{not receive'}(W, V, S).
\end{aligned}
$$

This subprogram is similar to the consistency check encoded via the rules in (3), (4), (6), and (7). However, sign consistency constraints are only checked for active vertices, and they must be satisfiable for all but an arbitrary active vertex W. As W ranges over all (non-input) vertices of an influence graph, each active vertex is taken into consideration.

For the influence graph in Figure 2, it is easy to see that the sign consistency constraint for **A** is satisfied by setting the sign of **A** to **+**, expressed by atom *labelV'*$(\mathbf{D}, \mathbf{A}, \text{+})$ in the ground rules obtained from the above encoding. In turn, the sign consistency constraint for **D** is satisfied by setting the sign of **A** to **–**. This is reflected by atom *labelV'*$(\mathbf{A}, \mathbf{A}, \text{–})$, allowing us to derive *receive'*$(\mathbf{A}, \mathbf{D}, \text{+})$, so that the ground instance of the above integrity constraint containing *labelV'*$(\mathbf{A}, \mathbf{D}, \text{+})$ is satisfied.

6 Empirical Evaluation and Application

For assessing the scalability of our approach, we start by conceiving a parameterizable set of artificial yet biologically meaningful benchmarks. After that, we present a typical application stemming from real biological data, illustrating the exertion in practice.

6.1 Checking Consistency

We first evaluate the efficiency of our approach on randomly generated instances, aiming at structures similar to those found in biological applications. Instances are composed of an influence graph, a complete labeling of its edges, and a partial labeling of its vertices. Our random generator takes three parameters: (i) the number α of vertices in the influence graph, (ii) the average degree β of the graph, and (iii) the proportion γ

α	*claspD Berkmin*	*claspD VMTF*	*claspD VSIDS*	*cmodels*	*dlv*	*gnt*
500	0.14	0.11	0.11	0.16	0.46	0.71
1000	0.41	0.25	0.25	0.35	1.92	3.34
1500	0.79	0.38	0.38	0.53	4.35	7.50
2000	1.33	0.51	0.51	0.71	8.15	13.23
2500	2.10	0.66	0.66	0.89	13.51	21.88
3000	3.03	0.80	0.79	1.07	20.37	31.77
3500	3.22	0.93	0.92	1.15	21.54	34.39
4000	4.35	1.06	1.06	1.36	30.06	46.14

Table 2. Run-times for consistency checking with *claspD*, *cmodels*, *dlv*, and *gnt*.

of observed variations for vertices. To generate an instance, we compute a random graph with α vertices (the value of α varying from 500 to 4000) under the model by Erdős-Rényi [33]. Each pair of vertices has equal probability to be connected via an edge, whose label is chosen independently with probability 0.5 for both signs. We fix the average degree β to 2.5, which is considered to be a typical value for biological networks [34]. Finally, $\lfloor \gamma\alpha \rfloor$ vertices are chosen with uniform probability and assigned a label with probability 0.5 for both signs. For each number α of vertices, we generated 50 instances using five different values for γ, viz., 0.01, 0.02, 0.033, 0.05, and 0.1. All instances can be found at [24].

We used *gringo* [22,29] (version 2.0.0) for combining the generated instances and the encoding given in Section 4 into equivalent ground logic programs. For deciding consistency by computing an answer set (if it exists), we ran disjunctive ASP solvers *claspD* [19] (version 1.1) with "Berkmin", "VMTF", and "VSIDS" heuristics, *cmodels* [17,27] (version 3.75) using *zchaff* [35], *dlv* [28] (build BEN/Oct 11), and *gnt* [36] (version 2.1). All runs were performed on a Linux machine equipped with an AMD Opteron 2 GHz processor and a memory limit set to 2GB RAM.

Table 2 shows the average run-times over 50 instances per number α of vertices in seconds, including grounding times of *gringo* and solving times. We checked that grounding times of *gringo* increase linearly with the number α of vertices, and they do not vary significantly over γ. For all solvers, run-times also increase linearly in α.[4] In fact, for fixed α values, we found two clusters of instances: consistent ones where total labelings were easy to compute and inconsistent ones where inconsistency was detected from preassigned labels only. This tells us that the influence graphs generated as described above are usually (too) easy to label consistently, and inconsistency only happens if it is explicitly introduced via fixed labels. However, such constellations are not unlikely in practice (cf. Section 6.3), and isolating MICs from them, as done in the next subsection, turned out to be hard for most solvers. Finally, greater values for γ led to an increased proportion of inconsistent instances, without making them much harder.

[4] Longer run-times of *claspD* with "Berkmin" in comparison to the other heuristics are due to a more expensive computation of heuristic values in the absence of conflict information. Furthermore, the time needed for performing "Lookahead" slows down *dlv* as well as *gnt*.

α	*gringo*	*claspD Berkmin*	*claspD VMTF*	*claspD VSIDS*
50	0.24	1.16 (0)	0.65 (0)	0.97 (0)
75	0.55	39.11 (1)	1.65 (0)	3.99 (0)
100	0.87	41.98 (1)	3.40 (0)	4.80 (0)
125	1.37	15.47 (0)	47.56 (1)	10.73 (0)
150	2.02	54.13 (0)	48.05 (0)	15.89 (0)
175	2.77	30.98 (0)	116.37 (2)	23.07 (0)
200	3.82	42.81 (0)	52.28 (1)	24.03 (0)
225	4.94	99.64 (1)	30.71 (0)	41.17 (0)
250	5.98	194.29 (3)	228.42 (5)	110.90 (1)
275	7.62	178.28 (2)	193.03 (4)	51.11 (0)
300	9.45	241.81 (2)	307.15 (7)	124.31 (0)

Table 3. Run-times for grounding with *gringo* and solving with *claspD*.

6.2 Identifying Minimal Inconsistent Cores

We now investigate the problem of finding a MIC within the same setting as in the previous subsection. Because of the elevated size of ground instantiations and problem difficulty, we varied the number α of vertices from 50 to 300, thus, using considerably smaller influence graphs than before. We again use *gringo* for grounding, now taking the encoding given in Section 5. As regards solving, we restrict our attention to *claspD* because all three of the other solvers showed drastic performance declines.

Table 3 shows average run-times over 50 instances per number α of vertices in seconds for grounding with *gringo* and solving with *claspD* using "Berkmin","VMTF", and "VSIDS" heuristics. Timeouts, indicated in parentheses, are taken as maximum time of 1800 seconds. We observe a quadratic increase in grounding times of *gringo*, which is in line with the fact that ground instantiations for our MIC encoding grow quadratically with the size of influence graphs. In fact, the schematic rules in Section 5.2 give rise to α copies of an influence graph. Considering solving times spent by *claspD* for finding one MIC (if it exists), we observe that they are relatively stable, in the sense that they are tightly correlated to grounding times. This regularity again confirms that, though it is random, the applied generation pattern tends to produce rather uniform influence graphs. Finally, we observed that unsatisfiable instances, i.e., consistent instances without any MIC, were easier to solve than the ones admitting answer sets. We conjecture that this is because consistent total labelings provide a disproof of inconsistency as encoded in Section 5.1.

As our experimental results demonstrate, computing a MIC is computationally harder than just checking consistency. This is not surprising because the related problem of identifying a MUC is D^{P}-complete [31,32]. With our declarative technique, we spot the quadratic space blow-up incurred by the MIC encoding in Section 5 as a bottleneck. It is an interesting open question whether more economical encodings can be found.

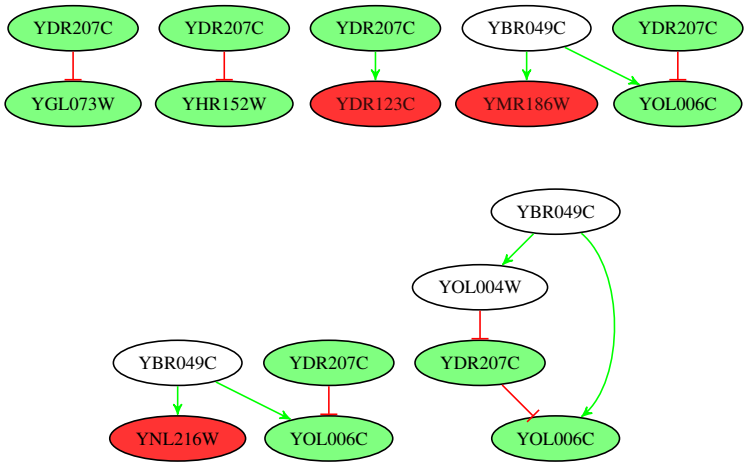

Fig. 3. Some exemplary MICs obtained by comparing the regulatory network in [37] with a genetic profile from [38].

6.3 Biological Case Study

In the following, we present the results of applying our approach to real-world data of genetic regulations in yeast. We tested the gene-regulatory network of yeast provided in [37] against genetic profile data of SNF2 knock-outs [38] from the Saccharomyces Genome Database. The regulatory network of yeast contains 909 genetic or biochemical regulations, all of which have been established experimentally, among 491 genes.

Comparing the yeast regulatory network with the genetic profile of SNF2, we found the data to be inconsistent with the network, which was easily detected using the approach from Section 4. Applying our diagnosis technique from Section 5, we obtained a total of 19 MICs. While computing the first MIC took only about 2.5 seconds using *gringo* and *claspD*, the computation of all MICs was considerably harder, taking 3 minutes and 38 seconds with *claspD* using "VMTF" embedded into a wrapper script that excludes already computed MICs via integrity constraints. In fact, the minimality encoding in Section 5.2 admits multiple answer sets corresponding to the same MIC because the variations of vertices not connected to the MIC can be chosen freely, thus producing copies of the same solution. Even though the encodings available at [24] already address this redundancy, they do not yet establish a one-to-one correspondence between MICs and answer sets since the determined consistent labelings of the subgraphs of a MIC are not necessarily unique. For achieving one-to-one correspondence, this redundancy must also be eliminated, which is a subject to future work.

Six of the computed MICs are exemplarily shown in Figure 3. While the first three of them are pretty obvious, we also identified more complex topologies. However, our example demonstrates that the MICs obtained in practice are still small enough to be understood easily. For finding suitable corrections to the inconsistencies, it is often even more helpful to display the connections between several overlapping MICs. Observe that all six MICs in Figure 3 are related to gene YDR207C, and in Figure 4, we show

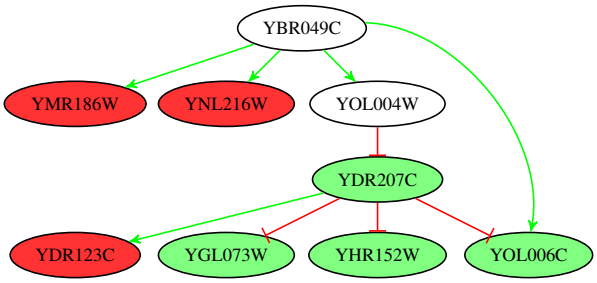

Fig. 4. Subgraph obtained by connecting the six MICs given in Figure 3.

the subgraph of the yeast regulatory network obtained by connecting them. In this representation, one can see that the observed increase of YDR207C is not compatible with the variation of any of its four targets, but the variation of YDR207C itself can be explained by its direct and indirect regulators. This suggests to first check the correctness of the observation that YDR207C has increased, and depending on the result, to consider additional regulations that might be missing in the yeast regulatory network. In fact, potential uses of our diagnosis technique applied to real-world data include identifying unreliable data and missing reactions in a systematic and more targeted way.

7 Discussion

We have provided an approach based on ASP to investigate the consistency between experimental profiles and influence graphs. In case of inconsistency, the concept of a MIC can be exploited for identifying concise explanations, pointing to unreliable data or missing reactions. The problem of finding MICs is closely related to the extraction of MUCs in the context of SAT. From a knowledge representation point of view, however, we argue for our ASP-based technique, as it allows for an elegant declarative way to describe problems in terms of a uniform encoding and specific instances.

By now, a variety of efficient ASP tools are available, both for grounding and for solving logic programs. Our empirical assessment of them (on random as well as real data) has in principle demonstrated the scalability of the approach. As elegance and flexibility in problem modeling are major advantages of ASP, our investigation might make it attractive also for related biological questions, beyond the ones addressed in this paper. For instance, natural extensions of the presented techniques allow for accomplishing prediction and repair. In the future, it will also be interesting to explore how far the performance of ASP tools can be tuned by varying and optimizing the given encodings, e.g., in order to compute all MICs more effectively. In turn, challenging applications like the one presented here might contribute to the further improvement of ASP tools, as they might be geared towards efficiency in such application domains.

Acknowledgments Philippe Veber was supported by a grant from DAAD. This work was partially funded by the GoFORSYS project (http://www.goforsys.org/; Grant 0313924).

The authors would like to thank Roland Kaminski for fruitful comments on our encoding and Carito Guziolowski for providing the data on yeast.

References

1. Joyce, A., Palsson, B.: The model organism as a system: Integrating 'omics' data sets. Nature Reviews Molecular Cell Biology **7**(3) (2006) 198–210
2. Klamt, S., Stelling, J.: Stoichiometric and constraint-based modelling. In: System Modeling in Cellular Biology: From Concepts to Nuts and Bolts. MIT Press (2006) 73–96
3. Friedman, N., Linial, M., Nachman, I., Pe'er, D.: Using Bayesian networks to analyze expression data. Journal of Computational Biology **7**(3-4) (2000) 601–620
4. 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) (2006) 153–174
5. Gutierrez-Rios, R., Rosenblueth, D., Loza, J., Huerta, A., Glasner, J., Blattner, F., Collado-Vides, J.: Regulatory network of Escherichia coli: Consistency between literature knowledge and microarray profiles. Genome Research **13**(11) (2003) 2435–2443
6. Soulé, C.: Graphic requirements for multistationarity. Complexus **1**(3) (2003) 123–133
7. Marek, V., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: The Logic Programming Paradigm: a 25-Year Perspective. Springer (1999) 375–398
8. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Annals of Mathematics and Artificial Intelligence **25**(3-4) (1999) 241–273
9. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
10. Soulé, C.: Mathematical approaches to differentiation and gene regulation. Comptes Rendus Biologies **329** (2006) 13–20
11. Remy, É., Ruet, P., Thieffry, D.: Graphic requirements for multistability and attractive cycles in a Boolean dynamical framework. Advances in Applied Mathematics (2008) To appear
12. Richard, A., Comet, J.: Necessary conditions for multistationarity in discrete dynamical systems. Discrete Applied Mathematics **155**(18) (2007) 2403–2413
13. Kuipers, B.: Qualitative reasoning: Modeling and simulation with incomplete knowledge. MIT Press (1994)
14. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2) (2002) 181–234
15. Syrjänen, T.: Lparse 1.0 user's manual. http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz
16. Mitchell, D.: A SAT solver primer. Bulletin of the European Association for Theoretical Computer Science **85** (2005) 112–133
17. Giunchiglia, E., Lierler, Y., Maratea, M.: Answer set programming based on propositional satisfiability. Journal of Automated Reasoning **36**(4) (2006) 345–377
18. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: clasp: A conflict-driven answer set solver. In: Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07). Springer (2007) 260–265
19. Drescher, C., Gebser, M., Grote, T., Kaufmann, B., König, A., Ostrowski, M., Schaub, T.: Conflict-driven disjunctive answer set solving. In: Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08). AAAI Press (2008) 422–432
20. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. Freeman and Co. (1979)

21. Eiter, T., Gottlob, G.: On the computational cost of disjunctive logic programming: Propositional case. Annals of Mathematics and Artificial Intelligence **15**(3-4) (1995) 289–323
22. Gebser, M., Schaub, T., Thiele, S.: GrinGo: A new grounder for answer set programming. In: Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07). Springer (2007) 266–271
23. Veber, P., Le Borgne, M., Siegel, A., Lagarrigue, S., Radulescu, O.: Complex qualitative models in biology: A new approach. Complexus **2**(3-4) (2004) 140–151
24. http://www.cs.uni-potsdam.de/wv/bioasp
25. Gelfond, M., Lifschitz, V., Przymusinska, H., Truszczyński, M.: Disjunctive defaults. In: Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91). Morgan Kaufmann (1991) 230–237
26. Ben-Eliyahu, R., Dechter, R.: Propositional semantics for disjunctive logic programs. Annals of Mathematics and Artificial Intelligence **12**(1-2) (1994) 53–87
27. Lierler, Y.: cmodels – SAT-based disjunctive answer set solver. In: Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05). Springer (2005) 447–451
28. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Transactions on Computational Logic **7**(3) (2006) 499–562
29. http://sourceforge.net/projects/gringo
30. Guziolowski, C., Veber, P., Le Borgne, M., Radulescu, O., Siegel, A.: Checking consistency between expression data and large scale regulatory networks: A case study. Journal of Biological Physics and Chemistry **7**(2) (2007) 37–43
31. Dershowitz, N., Hanna, Z., Nadel, A.: A scalable algorithm for minimal unsatisfiable core extraction. In: Proceedings of the Ninth International Conference on Theory and Applications of Satisfiability Testing (SAT'06). Springer (2006) 36–41
32. Papadimitriou, C., Yannakakis, M.: The complexity of facets (and some facets of complexity). In: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing (STOC'82). ACM Press (1982) 255–260
33. Erdős, P., Rényi, A.: On random graphs. Publicationes Mathematicae **6** (1959) 290–297
34. Jeong, H., Tombor, B., Albert, R., Oltvai, Z., Barabási, A.: The large-scale organization of metabolic networks. Nature **407** (2000) 651–654
35. http://www.princeton.edu/∼chaff/zchaff.html
36. Janhunen, T., Niemelä, I., Seipel, D., Simons, P., You, J.: Unfolding partiality and disjunctions in stable model semantics. ACM Transactions on Computational Logic **7**(1) (2006) 1–37
37. Guelzim, N., Bottani, S., Bourgine, P., Képès, F.: Topological and causal structure of the yeast transcriptional regulatory network. Nature Genetics **31** (2002) 60–63
38. Sudarsanam, P., Iyer, V., Brown, P., Winston, F.: Whole-genome expression analysis of snf/swi mutants of Saccharomyces cerevisiae. Proceedings of the National Academy of Sciences of the United States of America **97**(7) (2000) 3364–3369