Repair and Prediction (under Inconsistency) in Large Biological Networks with Answer Set Programming

Martin Gebser
University of Potsdam
D-14482 Potsdam

Carito Guziolowski
INRIA/Irisa
F-35042 Rennes

Mihail Ivanchev
University of Potsdam
D-14482 Potsdam

Torsten Schaub
University of Potsdam
D-14482 Potsdam

Anne Siegel
INRIA/Irisa
F-35042 Rennes

Sven Thiele
University of Potsdam
D-14482 Potsdam

Philippe Veber
Institut Cochin
F-75014 Paris

Abstract

We address the problem of repairing large-scale biological networks and corresponding yet often discrepant measurements in order to predict unobserved variations. To this end, we propose a range of different operations for altering experimental data and/or a biological network in order to re-establish their mutual consistency—an indispensable prerequisite for automated prediction. For accomplishing repair and prediction, we take advantage of the distinguished modeling and reasoning capacities of Answer Set Programming. We validate our framework by an empirical study on the widely investigated organism *Escherichia coli*.

Introduction

The availability of high-throughput methods in molecular biology has led to a tremendous increase of measurable data along with resulting knowledge repositories, gathered on the web (e.g. KEGG, MetaCyc, RegulonDB) usually in terms of biological networks. However, both measurements as well as biological networks are prone to considerable incompleteness (Ernst et al. 2008), heterogeneity (Gutiérrez-Ríos et al. 2003), and mutual inconsistency (Ferrazzi et al. 2007), making it non-trivial to draw biologically meaningful conclusions in an automated way.

We addressed the problem of detecting and explaining inconsistencies in large-scale biological networks and datasets in (Gebser et al. 2008) by introducing a declarative approach based on Answer Set Programming (ASP; (Baral 2003)). Our approach builds upon the Sign Consistency Model (SCM; (Siegel et al. 2006)), imposing constraints between experimental measurements and cellular interactions, expressed as influence graphs. In contrast to available probabilistic methods (cf. (Ferrazzi et al. 2007)), this model is particularly well-suited for dealing with qualitative knowledge (for instance, reactions lacking kinetic details) as well as incomplete and noisy data.

The natural question arising now is how to *repair* networks and data that have been found to be inconsistent, that is, how to modify network and/or data in order to re-establish their mutual consistency. A major challenge lies in the range of possible repair operations, since an inconsistency can be explained by missing interactions or inaccurate information in a network as well as by aberrant data. However, once consistency is re-established, network and data can be used for predicting unobserved variations.

To this end, we extend our basic approach and propose a framework for repairing large-scale biological networks and corresponding measurements in order to allow for predicting unobserved variations. We discuss the interest of different repair operations wrt several criteria: biological meaning, minimality measures, and computational cost. We implement our framework by taking advantage of the distinguished modeling and reasoning capacities of ASP. In particular, we detail how reasoning modes can be used for efficient prediction under minimal repairs, complying with the objective of minimal change. We then apply ASP to empirically evaluate the effect of different repair operations, both quantitatively and qualitatively, by considering the well-studied organism *Escherichia coli* along with published experimental data. The obtained results demonstrate the advantages of a highly declarative approach combined with high-performance inference engines. From the application perspective, the distinguishing novel features of our approach are as follows: (i) it offers a flexible concept of repair to overcome inconsistencies in biological networks, (ii) it is fully automated, (iii) it deals with large-scale systems in a global way, and finally (iv) it allows for predicting unobserved variations (even in the presence of inconsistency).

Background

Our approach relies on the qualitative framework of SCM for modeling biological networks and experimental measurements, and on ASP as declarative programming paradigm. These concepts are introduced in the following.

Sign Consistency Model

SCM is based on influence graphs, a common representation for a wide range of dynamical systems, lacking or abstracting from detailed quantitative descriptions, in biology (Soulé 2006) or physics (Kuipers 1994). While multivalued logical formalisms (Thomas 1979) may precisely describe the dynamics of small-scale systems, disjunctive causal rules on influence graphs were originally introduced in random dynamical frameworks to study global properties of large-scale networks, using a probabilistic approach (Kauffman 1993) mainly illustrated on the transcriptional network of yeast (Kauffman et al. 2003). Further probabilistic and heuristic methods (Gutiérrez-Ríos et al. 2003; Ideker 2004) exploit disjunctive causal rules to derive regulatory networks from high-throughput experimental data. Complementary to such data-driven methods, the main interest of SCM lies in its global approach to confront a network and data. To this end, SCM imposes a collection of constraints on experimental data together with information on regulations between network components (Siegel et al. 2006). Let us now detail the construction of such a model.

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. The vertices of an influence graph are biologically mapped to genes, proteins, or metabolites. An edge $j \rightarrow i$ means that the variation of j in time influences the level of i. The edges $j \rightarrow i$ of an influence graph are labeled with signs, either 1 or -1, denoted by $\sigma(j, i)$. Sign 1 (-1) indicates that regulator j tends to increase (decrease) i. Biologically, 1 stands for DNA transcription, protein activation, or molecule production, while -1 expresses inhibition of DNA transcription or protein inactivation. An example influence graph is given in Figure 1 (left), depicting positive (negative) influences in green (red).

In SCM, *experimental profiles* are supposed to come from steady state shift experiments. Initially, the system is in a 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, one observes whether the concentration has increased or decreased, while quantitative values are unavailable, unessential, or unreliable. By $\mu(i)$, we denote the sign, either 1 or -1, of the variation of a species i between the initial and the final state. In Figure 1 (middle and right), an observed increase 1 (decrease -1) is indicated by a green (red) colored vertex; variations of uncolored vertices are unknown.

Given an influence graph (as a representation of cellular interactions) and a labeling of its vertices with signs (as observations of an experimental profile), we now describe the constraints that relate both. For every non-input vertex i, an observed variation $\mu(i)$ should be explained by the *influence* of at least one regulator j on i, given by $\mu(j)*\sigma(j, i)$. If a vertex i receives at least one influence matching $\mu(i)$, we say that its sign is *consistent*. 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 total extension of vertex and edge labelings to all vertices and edges, respectively.

Answer Set Programming

We here only briefly introduce main concepts related to logic programs under answer set semantics (see (Baral 2003) for details). A *rule* is of the form

$$a \leftarrow a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n. \qquad (1)$$

Each a_i is an atom for $1 \leq i \leq n$, a is either an atom or $\bot$, and $\sim$ denotes (default) negation. A rule of form (1) such that $n = m = 0$ is called a *fact*. An *integrity constraint* is a rule of form (1) such that $a = \bot$; its semantic role is to deny interpretations where the right-hand side of $\leftarrow$ holds. In the following, we omit $\leftarrow$ when writing facts, and similarly $\bot$ when writing integrity constraints. A (logic) *program* Π is a set of rules; its ground instantiation, denoted by $\mathbf{\Pi}$, is obtained by applying all substitutions of first-order variables with terms from the (implicitly given) Herbrand universe of Π. A (Herbrand) interpretation X is a subset of the Herbrand base of Π; X is a model of Π if $\{a, a_{m+1}, \dots, a_n\} \cap X \neq \emptyset$ or $\{a_1, \dots, a_m\} \not\subseteq X$ for every rule of form (1) in $\mathbf{\Pi}$. Finally, X is an *answer set* of Π if X is a subset-minimal model of $\{a \leftarrow a_1, \dots, a_m. \mid a \leftarrow a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n. \in \mathbf{\Pi}, \{a_{m+1}, \dots, a_n\} \cap X = \emptyset\}$. Note that any answer set of Π is as well a model of Π. The idea of ASP is to encode a given problem by a program such that its answer sets correspond to solutions for the problem.

Approach

Our goal is to provide ASP solutions for reasoning over influence graphs and experimental profiles, in particular, if they are inconsistent with each other. To this end, we identify below several operations, called *repairs*, which can be applied to re-establish consistency. The framework we develop is configurable, so that biological experts may selectively investigate critical parts of biological networks and/or measurements. In what follows, we provide logic program representations of repair in the input language of ASP grounder *gringo* (Gebser et al. 2009). After describing the format of instances, repair operations, and our repair encoding, we consider *minimal* repairs. Finally, we explain the usage of minimal repairs for *prediction* (under inconsistency).

Problem Instance

An influence graph is represented by a fact $vtx(i)$, for each species i, a fact $edge(j, i)$, for each (known) regulation $j \rightarrow i$, and a fact $obs\_e(j, i, s)$ with $s \in \{1, -1\}$, for each (known) regulation type. Furthermore, an experimental profile is declared via a fact $exp(p)$; its observed variations and inputs are specified by facts $obs\_v(p, i, s)$ with $s \in \{1, -1\}$ and $inp(p, j)$, respectively. We assume that, for a given species i (or regulation $j \rightarrow i$) and an experimental profile p, an instance contains at most one of the facts $obs\_v(p, i, 1)$ and $obs\_v(p, i, -1)$ (or $obs\_e(j, i, 1)$ and $obs\_e(j, i, -1)$), but not both of them.

Example 1 *The facts describing the influence graph (Π_g) and experimental profiles (Π_{p_1} and Π_{p_2}) shown in Figure 1 are provided in Figure 2. Note that experimental profile p_1 (cf. middle in Figure 1) and p_2 (cf. right in Figure 1) are inconsistent with the given influence graph. Both necessitate labeling vertex b with -1 in order to explain the observed decrease of c. With p_1, such a decrease of b is unexplained; with p_2, it can be explained by labeling a with -1, which in turn leaves the observed decrease of e unexplained. However, there were no such inherent inconsistencies, e.g., if increases of c had been observed in p_1 and p_2.* $\diamond$

$$\Pi_g = \left\{\begin{array}{llllllll} vtx(a). & vtx(b). & vtx(c). & vtx(d). & vtx(e). & & & \\ edge(a,b). & obs\_e(a,b,1). & edge(b,a). & obs\_e(b,a,1). & edge(d,b). & obs\_e(d,b,1). \\ edge(a,d). & obs\_e(a,d,-1). & edge(b,c). & obs\_e(b,c,1). & edge(d,c). & obs\_e(d,c,1). \\ edge(a,e). & obs\_e(a,e,-1). & edge(c,e). & obs\_e(c,e,-1). & edge(d,e). & obs\_e(d,e,1). \end{array}\right\} \quad (2)$$

$$\Pi_{p_1} = \{exp(p_1). \quad inp(p_1,d). \quad obs\_v(p_1,d,1). \quad obs\_v(p_1,c,-1). \quad obs\_v(p_1,a,1).\} \quad (3)$$

$$\Pi_{p_2} = \{exp(p_2). \quad inp(p_2,d). \quad obs\_v(p_2,d,1). \quad obs\_v(p_2,c,-1). \quad obs\_v(p_2,e,-1).\} \quad (4)$$

Figure 2: Facts representing the influence graph and experimental profiles from Figure 1 in Π_g, Π_{p_1}, and Π_{p_2}, respectively.

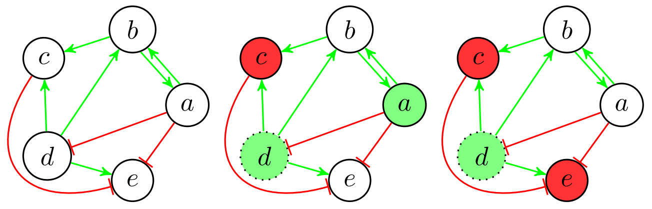

Figure 1: An influence graph (left) along with two experimental profiles (middle and right), in which increases (decreases) have been observed for vertices colored green (red), and vertex d is an input.

Repair Operations

Repairs are operations on an influence graph (*model*) or experimental profiles (*data*) that can be applied to make model and data mutually consistent. Consistency of the repaired model/data is then witnessed by consistent total labelings of vertices and edges.

To begin with, we use the rules in (5)–(9), shown in Figure 3, to define admissible repair operations. Note that particular operations are identified with *function terms* inside of predicate rep, which enables us to deal with repairs in a general way whenever knowing particular repair types is unnecessary. The function terms $add\_e(U,V)$, $flip\_e(U,V,S)$, and $inp\_v(V)$ stand for model repairs, where $add\_e(U,V)$ introduces an edge from U to V, $flip\_e(U,V,S)$ flips the sign S of an existing edge from U to V, and $inp\_v(V)$ treats vertex V as an input in all experimental profiles. For data repair, the function term $inp\_v(P,V)$ treats vertex V as an input in experimental profile P, and $flip\_v(P,V,S)$ flips the sign S of vertex V in P. These repair operations are inspired by existing biological use cases. To repair a model by adding new edges makes sense when the model is incomplete (which is often the case in practice). Flipping the sign of an edge is a way to curate the model; it means that in some experiment the effect of a regulator (activator or inhibitor) should be corrected. Turning a vertex into an input can be used to indicate missing (unknown) regulations or oscillations of regulators. Revising experimental observations puts the dataset into question and may help to identify aberrant measurements (frequent in microarray data).

Which repair operations ought to be permitted or omitted requires background knowledge about the model and data at hand. By offering a variety of operations, our framework is flexible and may be adjusted to particular situations. In rules (5)–(9), the declaration of admissible repair operations is governed by atoms $rep\_a, \ldots, rep\_v$. Depending on the requested repair types, such atoms are to be provided as facts. It would also be possible to restrict repair operations to particular edges or vertices, respectively, based on the availability of biological expert knowledge.

Finally, note that rules (5)–(9) filter some redundant repairs. An edge between distinct vertices can be introduced only if there is none in the model. Flipping the sign of an edge or vertex is possible only if a sign is provided in the model or data, respectively. Making a vertex input, globally or in a particular experimental profile, requires it to not already be input in an arbitrary or the considered profile.

Repair Encoding

With admissible repairs at hand, the rules in (10)–(12) encode the choice of operations to apply. While (10) and (11) do not explicitly refer to types of repair operations, the integrity constraint in (12) denies repair applications where a vertex is made input both globally and also in a particular experimental profile. In such a case, the latter operation would be redundant. In general, the question of declaring a vertex as input either globally or local to an experiment depends on the intention whether to repair the model or data; however, simultaneously applying similar operations is futile.

The rest of the repair encoding is about identifying witnesses for the consistency of the repaired model/data. To this end, we first declare available signs and their complement relation in (13).[1] Rules (14)–(17) take care of labeling edges and also incorporate repairs on them. In fact, Rule (14) propagates (known) signs of edges if not flipped by a repair; otherwise, Rule (15) is used to derive the opposite sign instead. For edges introduced by repairs and unlabeled edges in the model, respectively, rules (16) and (17) encode the choice of a sign, making sure that any answer set comprises a total edge labeling given by ground atoms over predicate lab_e.

Using the same methodology as with edges, but now relative to experimental profiles, rules (18)–(20) deal with vertex labels and repairs on them. In analogy to (14), Rule (18) maintains signs given in experimental profiles, while Rule (19) applies repairs flipping such signs. Ground instances of Rule (20) direct choosing signs of unobserved vertices, not yet handled by Rule (18) or (19). As a consequence, the instances of lab_v in an answer set provide a total vertex labeling.

[1]Note that *gringo* interprets arithmetic functions like '$-$'. For instance, it evaluates $--1$ to 1.

$$\mathit{rep}(\mathit{add\_e}(U,V)) \leftarrow \mathit{rep\_a}, \mathit{vtx}(U), \mathit{vtx}(V), U \neq V, {\sim}\mathit{edge}(U,V). \tag{5}$$
$$\mathit{rep}(\mathit{flip\_e}(U,V,S)) \leftarrow \mathit{rep\_e}, \mathit{edge}(U,V), \mathit{obs\_e}(U,V,S). \tag{6}$$
$$\mathit{rep}(\mathit{inp\_v}(V)) \leftarrow \mathit{rep\_g}, \mathit{vtx}(V), \mathit{exp}(P), {\sim}\mathit{inp}(P,V). \tag{7}$$
$$\mathit{rep}(\mathit{inp\_v}(P,V)) \leftarrow \mathit{rep\_i}, \mathit{vtx}(V), \mathit{exp}(P), {\sim}\mathit{inp}(P,V). \tag{8}$$
$$\mathit{rep}(\mathit{flip\_v}(P,V,S)) \leftarrow \mathit{rep\_v}, \mathit{vtx}(V), \mathit{exp}(P), \mathit{obs\_v}(P,V,S). \tag{9}$$

$$\mathit{app}(R) \leftarrow \mathit{rep}(R), {\sim}\overline{\mathit{app}}(R). \tag{10}$$
$$\overline{\mathit{app}}(R) \leftarrow \mathit{rep}(R), {\sim}\mathit{app}(R). \tag{11}$$
$$\leftarrow \mathit{app}(\mathit{inp\_v}(V)), \mathit{app}(\mathit{inp\_v}(P,V)). \tag{12}$$

$$\mathit{opp}(S,-S) \leftarrow \mathit{sig}(S). \qquad \mathit{sig}(1). \quad \mathit{sig}(-1). \tag{13}$$

$$\mathit{lab\_e}(U,V,S) \leftarrow \mathit{edge}(U,V), \mathit{obs\_e}(U,V,S), {\sim}\mathit{app}(\mathit{flip\_e}(U,V,S)). \tag{14}$$
$$\mathit{lab\_e}(U,V,T) \leftarrow \mathit{app}(\mathit{flip\_e}(U,V,S)), \mathit{opp}(S,T). \tag{15}$$
$$\mathit{lab\_e}(U,V,S) \leftarrow \mathit{app}(\mathit{add\_e}(U,V)), \mathit{opp}(S,T), {\sim}\mathit{lab\_e}(U,V,T). \tag{16}$$
$$\mathit{lab\_e}(U,V,S) \leftarrow \mathit{edge}(U,V), \mathit{opp}(S,T), {\sim}\mathit{lab\_e}(U,V,T). \tag{17}$$

$$\mathit{lab\_v}(P,V,S) \leftarrow \mathit{vtx}(V), \mathit{exp}(P), \mathit{obs\_v}(P,V,S), {\sim}\mathit{app}(\mathit{flip\_v}(P,V,S)). \tag{18}$$
$$\mathit{lab\_v}(P,V,T) \leftarrow \mathit{app}(\mathit{flip\_v}(P,V,S)), \mathit{opp}(S,T). \tag{19}$$
$$\mathit{lab\_v}(P,V,S) \leftarrow \mathit{vtx}(V), \mathit{exp}(P), \mathit{opp}(S,T), {\sim}\mathit{lab\_v}(P,V,T). \tag{20}$$

$$\mathit{rec}(P,V,S{*}T) \leftarrow \mathit{lab\_e}(U,V,S), \mathit{lab\_v}(P,U,T), {\sim}\mathit{inp}(P,V). \tag{21}$$
$$\leftarrow \mathit{lab\_v}(P,V,S), {\sim}\mathit{rec}(P,V,S), {\sim}\mathit{inp}(P,V), {\sim}\mathit{app}(\mathit{inp\_v}(V)), {\sim}\mathit{app}(\mathit{inp\_v}(P,V)). \tag{22}$$

$$\#\mathit{minimize}\{\mathit{app}(R) : \mathit{rep}(R)\}. \tag{23}$$

Figure 3: The repair encoding, consisting of the rules in (5)–(22), along with a cardinality-minimization statement in (23).

Finally, we need to check whether the variations of all non-input vertices are explained by the influences of their regulators. This is accomplished via the rules in (21) and (22). First, observe that the influence of a regulator U on V is simply the product of the signs of the edge and of U. Based on this, the integrity constraint in (22) denies cases where a non-input vertex V, neither a given input of a profile P nor made input globally or in P by any repair, receives no influence matching its variation S. That is, a non-input vertex must not be unexplained in a profile. Conversely, any answer set comprises consistent total vertex and edge labelings wrt the repaired model/data.

Example 2 *Reconsider the influence graph, described by Π_g in (2), and experimental profiles, represented as Π_{p_1} and Π_{p_2} in (3) and (4), shown in Figure 1. Let Π be the encoding consisting of the rules in (5)–(22). Then, $\Pi \cup \Pi_g \cup \Pi_{p_1} \cup \{\mathit{rep\_v}.\}$ admits two answer sets comprising $\mathit{app}(\mathit{flip\_v}(p_1, c, -1))$ as single repair operation to apply, viz., the sign of c is flipped to 1. Edge labels are as determined in Π_g, and the consistent total vertex labelings given by ground atoms over predicate $\mathit{lab\_v}$ are shown in Row I:*

		a	b	c	d	e
I	p_1	1	1	1	1	$1 \mid -1$
II	p_2	$1 \mid -1$	$1 \mid -1$	1	1	-1
III	p_1	1	1	-1	1	$1 \mid -1$
	p_2	1	1	-1	1	-1

So witnessing labelings require increases of a, b, c, and d, while e may either increase or decrease.

Similarly, the two answer sets of $\Pi \cup \Pi_g \cup \Pi_{p_2} \cup \{\mathit{rep\_v}.\}$ flipping c to 1 are shown in Row II. Here, c as well as d must increase and e decrease, and the variations of a and b are variable but must comply with each other.

When looking at model repairs using program $\Pi \cup \Pi_g \cup \Pi_{p_1} \cup \Pi_{p_2} \cup \{\mathit{rep\_g}.\}$, where we may globally make vertices input, we get two answer sets applying only the repair operation expressed by $\mathit{app}(\mathit{inp\_v}(c))$. The witnesses for p_1 and p_2 are shown in Row III. We have that p_1 and p_2 necessitate the same signs for a, b, c, and d wrt the repaired model. Moreover, with p_1, e can either increase or decrease, while it must decrease with p_2. $\diamondsuit$

Minimal Repairs

Typically, plenty of repairs are possible, in particular, if several repair operations are admitted by adding multiple control atoms $\mathit{rep\_a}, \dots, \mathit{rep\_v}$ as facts. However, one usually is only interested in repairs that make few changes on the model and/or data.

Repairs that re-establish consistency by applying a minimum number of operations can easily be selected among candidate repairs by using the $\#\mathit{minimize}$ directive available in *lparse*'s and *gringo*'s input languages (Syrjänen; Gebser et al. 2009). The $\#\mathit{minimize}$ statement in (23) means that the number of instances of predicate app in answer sets, with argument R ranging over the ground instances of "domain predicate" rep, is subject to minimization. Note that (23) does not explicitly refer to the types of repair operations whose application is to be minimized.

Example 3 *As discussed in Example 1, the experimental profiles in Π_{p_1} and Π_{p_2} are inconsistent with the influence graph represented by Π_g. When augmenting program $\Pi \cup \Pi_g \cup \Pi_{p_1} \cup \Pi_{p_2} \cup \{rep\_g.\}$ from Example 2 with the statement in (23), the answer sets comprising $app(inp\_v(c))$ as the only repair operation to apply, along with the corresponding witnesses given in Example 2, yield a cardinality-minimal and thus optimal repair.* ♢

Although we do not detail them here, we note that alternative minimality criteria, such as weighted sum or subset inclusion, can also be used with the repair encoding in (5)–(22). While augmenting the #*minimize* statement in (23) with weights for atoms is straightforward, encoding subset-based minimization is more sophisticated. An encoding of the subset-minimality test is thus deferred to the appendix.

In fact, cardinality-minimal repairs may sometimes be too coarse and suppress further structurally interesting repairs being subset-minimal.

Example 4 *The previous example resulted in cardinality-minimal answer sets comprising $app(inp\_v(c))$ as the single repair operation to apply. As a consequence, answer sets containing both $app(inp\_v(a))$ and $app(inp\_v(b))$ as applied repair operations are ignored because they fail to be cardinality-minimal. However, such answer sets exist, and the following vertex labelings are their witnesses:*

	a	b	c	d	e
p_1	1	-1	-1	1	$1 \vert -1$
p_2	1	-1	-1	1	-1

Note that the increase of a and the decrease of b are both unexplained by these witnesses. Hence, neither $app(inp\_v(a))$ nor $app(inp\_v(b))$ can be dropped without losing consistency, so that the repair at hand is subset-minimal. ♢

Example 4 shows that minimizing cardinality can miss subset-minimal repairs. In fact, any cardinality-minimal repair is also subset-minimal, while the converse does not hold in general. Regarding computational complexity, we have that cardinality-minimization is usually accomplished with algorithms devised for problems in Δ_2^P (see, e.g., (Simons, Niemelä, and Soininen 2002)), while algorithms usable for subset-minimization typically still handle Σ_2^P-hard problems (cf. (Leone et al. 2006; Drescher et al. 2008)). The different complexities of solving methods suggest that cardinality-minimization is presumably more efficient in practice than subset-minimization. This is also confirmed by our experiments, whose results based on cardinality-minimization are presented below, while subset-minimization turned out to be too costly for an extensive empirical investigation.

Prediction under Repairs

For large real-world biological networks and measurements, there can be plenty witnessing vertex and edge labelings after re-establishing consistency via repairs. To not get lost in manifold scenarios, it is thus reasonable or even mandatory to focus on their common consequences. We therefore present the identification of consequences shared by all consistent vertex and edge labelings under minimal repairs as the ultimate application of our method, and we call this task *prediction*. Due to the capability of repairing, our approach enables prediction even if model and data are mutually inconsistent, which is often the case in practice. Importantly, enumerating all consistent total labelings is unnecessary. In fact, cautious reasoning (Gebser, Kaufmann, and Schaub 2009), supported by ASP solver *clasp* (Gebser et al. 2007), allows for computing the intersection of all (optimal) answer sets while investigating only linearly many of them.

For prediction, an input program Π is composed of an instance (cf. Figure 2), a definition of admissible repair operations ($rep\_a, \ldots, rep\_v$), the repair encoding in (5)–(22), and the #*minimize* statement in (23) (or alternatively the subset-minimality test in the appendix). Predicted signs for edges and vertices are then simply read off from instances of predicates lab_e and lab_v in the intersection of all optimal answer sets of Π, that is, answer sets comprising a minimum number of instances of predicate app. Though we mainly target at prediction wrt mutually inconsistent model and data, we note that prediction under consistency, where the unique minimal set of repair operations to apply is empty, is merely a particular case of prediction under repairs.

Experiments

For validating our approach, we use the transcriptional network of *Escherichia coli*, obtained from RegulonDB (Gama-Castro et al. 2008). In the corresponding influence graph, the label of an edge depends on whether the interaction was determined as activation, inhibition, dual, or complex in RegulonDB. Overall, the influence graph consists of 5150 interactions between 1914 genes. We confront this model with datasets corresponding to the Exponential-Stationary growth shift study in (Bradley et al. 2007) and the Heatshock experiment in (Allen et al. 2003). For each of them, the extracted data yields about 850 significant variations (1 or -1) of *Escherichia coli* genes. Since the data is highly noisy, not surprisingly, it is inconsistent with the RegulonDB model. For enabling a qualitative assessment of predictions (see Table 2), we generated data samples by randomly selecting 3%, 6%, 9%, 12%, or 15% of the whole data (about 850 variations with either experiment). We use these samples for testing both our repair modes as well as prediction (of omitted experimental data).[2] All experiments were run with grounder *gringo* (2.0.3) and solver *clasp* (1.2.1) on a Linux PC equipped with AthMP+1900 processor and 4GB main memory, imposing a maximum time of 600 seconds per run. Below, we first report runtime results for (cardinality-minimal) repair and prediction; afterwards, we analyze the obtained predictions both quantitatively and qualitatively.

Feasibility of Repairs

We first tested the feasibility of our repair modes on consistent as well as inconsistent samples (depending on the random selection). Table 1 provides average runtimes and

[2]Instances and encodings available at: `http://www.cs.uni-potsdam.de/bioasp/`

Exponential-Stationary growth shift					
Repair	3%	6%	9%	12%	15%
Repair Times					
`e`	6.58 (0)	8.44 (0)	11.60 (0)	14.88 (0)	26.20 (0)
`i`	2.18 (0)	2.15 (0)	2.21 (0)	2.23 (0)	2.21 (0)
`v`	1.41 (0)	1.40 (0)	1.40 (0)	1.41 (0)	1.37 (0)
`e i`	73.16 (6)	202.66 (23)	392.97 (87)	518.50(143)	574.85(179)
`e v`	28.53 (0)	85.17 (0)	189.27 (12)	327.98 (33)	470.48 (88)
`i v`	2.09 (0)	2.14 (0)	2.45 (0)	3.08 (0)	6.06 (0)
`e i v`	133.84 (8)	391.60 (76)	538.93(151)	593.33(193)	600.00(200)
Prediction Times					
`e`	13.27 (0)	12.19 (0)	14.76 (0)	15.34 (0)	25.90 (1)
`i`	6.18 (0)	5.26 (0)	4.77 (0)	4.60 (0)	4.42 (0)
`v`	4.64 (0)	4.45 (0)	4.39 (0)	4.40 (0)	4.30 (0)
`e i`	35.25 (0)	97.66 (1)	293.80 (3)	456.55 (3)	550.33 (1)
`e v`	14.35 (0)	26.17 (0)	90.17 (3)	200.25 (13)	363.36 (16)
`i v`	6.43 (0)	5.75 (0)	6.27 (0)	6.69 (0)	8.61 (0)
`e i v`	42.51 (0)	248.30 (1)	468.71 (2)	579.58 (0)	— (0)

Heatshock					
Repair	3%	6%	9%	12%	15%
Repair Times					
`e`	25.54 (4)	42.76 (8)	50.46 (5)	69.23 (6)	84.77 (6)
`i`	2.10 (0)	2.13 (0)	2.13 (0)	2.05 (0)	2.08 (0)
`v`	1.41 (0)	1.47 (0)	1.42 (0)	1.37 (0)	1.39 (0)
`e i`	120.91(21)	374.69 (91)	553.00(169)	593.20(197)	595.99(198)
`e v`	67.92 (3)	236.05 (31)	465.92(107)	579.88(179)	596.17(197)
`i v`	2.27 (0)	4.94 (0)	60.63 (8)	257.68 (56)	418.93(123)
`e i v`	232.29(26)	542.48(152)	593.88(195)	600.00(200)	600.00(200)
Prediction Times					
`e`	25.77 (0)	37.18 (0)	29.09 (0)	36.23 (0)	41.88 (0)
`i`	6.57 (0)	5.93 (0)	5.17 (0)	4.86 (0)	4.54 (0)
`v`	4.86 (0)	5.06 (0)	5.34 (0)	5.42 (0)	5.52 (0)
`e i`	85.47 (0)	293.28 (1)	524.19 (3)	591.81 (0)	594.74 (0)
`e v`	23.32 (0)	111.99 (0)	338.95 (0)	545.56 (2)	591.23 (0)
`i v`	6.91 (0)	6.63 (0)	30.33 (0)	176.14 (1)	371.95 (0)
`e i v`	101.82 (1)	466.91 (0)	585.64 (0)	— (0)	— (0)

Table 1: Repair and Prediction Times.

numbers of timeouts in parentheses over 200 samples per percentage of selected measurements; timeouts are included as 600s in average runtimes. We ran experiments admitting the following repair operations and combinations thereof: flipping edge labels denoted by `e` (*flip_e*), making vertices input denoted by `i` (*inp_v*), and flipping preassigned variations denoted by `v` (*flip_v*). Note that the two modes to make vertices input (globally or locally) fall together here, and we used only the local repair operation in `i` while skipping tests with the other, equivalent one. Moreover, we do not include results on the adding edges repair (*add_e*), where the bottleneck is grounding since the potential addition of arbitrary edges turns the influence graph into a huge clique at the encoding level. To avoid this, edges that can possibly be added by repairs should be restricted to a (smaller) set of reasonable ones, which requires biological knowledge, e.g., regulations known for organisms related to the investigated one.

In view that *clasp* applies a branch-and-bound approach to search for a cardinality-minimal repair, we observe that the average runtimes shown in Table 1 are directly correlated to the number of admissible repair operations. The fewest repairs are admitted with `v`, given that only about 25–130 observed variations are included in the samples of varying percentage. All vertices of the influence graph can potentially be made input with `i`, but when run in isolation or together with `v`, it still performs relatively well. Finally, as labels are available for almost all edges of the influence graph, permitting to flip each of them in `e` explains long runtimes and many timeouts obtained with it, in particular, on its combinations with `i`. However, the tight correlation between number of admitted repairs and runtime indicates that our method could significantly benefit from the inclusion of biological knowledge to restrict scopes of repairs.

In addition to running on samples, we computed cardinality-minimal repairs on the full datasets (containing about 850 variations per experiment). As regards individual operations (viz., `e`, `i`, and `v`), we observed that both finding a (cardinality-minimal) candidate repair as well as proving its optimality are accomplished easily with either `i` or `v`, respectively, within split seconds. With `e`, incurring a greater scope of repairs than `i` and `v`, proving an optimum turned out to be harder, the order of magnitude however depending on the considered experiment. On Exponential-Stationary growth shift, mode `e` still completed within seconds, while proving the optimum appeared to be very hard on the Heatshock data. However, some explorations also indicated that the performance of *clasp* can be significantly improved by selecting appropriate solver settings. For instance, options `--restart-on-model` and `--opt-heu` proved to be particularly helpful, but more systematic exploration would be needed to suitably fix remaining parameters (e.g. restart policy). Regarding the minimum number of repairs needed with different operations, we found that flipping variations (`v`) required fewer applications than either flipping edge labels or making vertices input (`e` or `i`). Minimums of the former are 40 and 34 on Exponential-Stationary growth shift and Heatshock, respectively, while they are 42 and 94 for the latter two modes. Here, the difference of 60 (!) repair applications on Heatshock is particularly remarkable, as it shows that repairing with either `e` or `i` incurs by far more modifications (of the network) than data corrections needed with `v`.

Prediction under Repairs

In the second step, done after computing the minimum number of repairs needed, we performed prediction by computing the intersection of all answer sets comprising a cardinality-minimal (and sometimes empty) repair. To this end, we used the cautious reasoning capacities of *clasp* (option `--cautious`) along with options `--opt-value` and `--opt-all` for initializing the objective function with the minimum number of repairs and enumerating all optimal answer sets, respectively. Runtime results are presented in Table 1, using the same notations for repair operations as before, but taking average runtimes only over those of the 200 samples per percentage where a cardinality-minimal repair was computed before the timeout (as the optimum is not known otherwise). We observe that the runtimes for prediction are in line with the ones for computing a cardinality-minimal repair, and maximum time is not often exceeded on the samples with known optimum. This shows that pre-

Exponential-Stationary growth shift					
Repair	3%	6%	9%	12%	15%
Prediction **Rate**					
`e`	15.00	18.51	20.93	22.79	23.94
`i`	15.00	18.51	20.93	22.79	23.93
`v`	14.90	18.37	20.86	22.73	23.77
`e i`	14.92	18.61	20.55	21.96	22.80
`e v`	14.89	18.33	21.07	22.52	23.74
`i v`	14.89	18.33	20.79	22.59	23.66
`e i v`	14.58	19.00	20.29	21.13	—
Prediction **Accuracy**					
`e`	90.93	91.98	92.42	92.70	92.81
`i`	90.93	91.98	92.42	92.70	92.81
`v`	90.99	92.05	92.44	92.73	92.89
`e i`	91.09	91.90	92.57	93.03	93.19
`e v`	90.99	92.03	92.50	92.82	92.94
`i v`	90.99	92.03	92.42	92.71	92.87
`e i v`	91.35	92.29	92.52	93.04	—

Heatshock					
Repair	3%	6%	9%	12%	15%
Prediction **Rate**					
`e`	15.47	19.54	21.87	23.17	24.78
`i`	15.48	19.62	21.89	23.20	24.80
`v`	15.32	19.59	21.37	22.13	23.79
`e i`	15.37	19.62	22.83	23.44	24.05
`e v`	15.33	19.21	21.00	22.65	24.90
`i v`	15.41	19.47	21.36	21.81	23.55
`e i v`	15.01	19.11	22.52	—	—
Prediction **Accuracy**					
`e`	91.87	92.93	92.92	92.83	92.71
`i`	91.93	92.90	92.94	92.87	92.76
`v`	92.29	93.27	93.88	94.27	94.36
`e i`	91.99	92.49	91.16	93.62	94.44
`e v`	92.30	93.37	93.66	94.36	94.35
`i v`	92.24	93.34	93.90	94.26	94.38
`e i v`	92.26	93.04	91.78	—	—

Table 2: Prediction Rate and Accuracy.

diction is successfully applicable if computing a cardinality-minimal repair is feasible.

In what follows, we analyze quantity and quality of the predictions we obtained. To this end, we determined the following numbers for each run: N vertices without variation given in the sample, P newly predicted vertices (variation not given in the sample), V newly predicted vertices having the same variation as available in the whole dataset, and W newly predicted vertices having the opposite variation in the whole dataset. Based on this, the *prediction rate* is obtained via the formula $(P{*}100)/N$, and the *prediction accuracy* is given by $(V{*}100)/(V{+}W)$. That is, the prediction rate reflects the percentage of newly predicted vertices, while the prediction accuracy measures in how many cases variations available in the whole dataset (but not in the sample) have been recovered. Average prediction rates and accuracies over samples where both repair and prediction terminated are shown in Table 2. Note that some averages result from few instances only (many timeouts reported in Table 1); such results should be interpreted with care.

We first notice that in both experiments, Exponential-Stationary growth shift and Heatshock, the prediction rates are significant, varying from about 15 to 25 percent. As it can be expected, prediction rates increase with the size of samples, in particular, for the transition from 3% to 6% of preassigned variations. However, while the size of the samples increases linearly, the prediction rates do not. This may mean that prediction rates reach a plateau related to the topology of the network, as also observed in (Veber et al. 2008). Interestingly, we do not notice any significant decrease of prediction rates when admitting multiple repair operations simultaneously. This suggests that predicted variations are rather insensitive to the repair operations used for re-establishing consistency, given that the application of repairs is governed by cardinality-minimality. Comparing individual operations, we note that flipping variations (`v`) yields a slightly lower prediction rate than the others, in particular, on the larger samples of Heatshock. This could be related to the above observation that repairing wrt the full dataset requires fewer applications of `v` than needed with the other modes.

Considering prediction accuracies, they are consistently higher than 90 percent, meaning that predicted variations and experimental observations correspond in most cases.[3] As with prediction rates, accuracies increase with sample size, while the choice of admissible repair operations does not exhibit much impact. This indicates that filtering repairs by cardinality-minimality makes the qualitative results largely independent of repair types, at least on the datasets we consider here. Nonetheless, we still observe that individual operation `v` yields higher accuracies than either flipping edge labels or making vertices input (`e` or `i`). Interestingly, the gap is greatest on the larger samples of Heatshock, and fewer modifications needed with `v` on the full dataset provide a reasonable explanation for it.

Finally, we compared prediction accuracies to the ones obtained when using a different kind of repair method: iteratively removing (subset-minimal) inconsistent subnetworks (cf. (Gebser et al. 2008)) until the remaining network is found to be consistent. Repair results, that is, reduced networks, obtained in such a way depend on the order of removing inconsistent subnetworks, while the technique presented here is fully declarative. The alternative repair method achieved prediction accuracies between 65 and 73 percent on Exponential-Stationary growth shift and from 76 to 80 percent on Heatshock data. The higher accuracies of our declarative technique show that a firm repair concept pays off in prediction quality.

Discussion

We have introduced repair-based reasoning techniques for computing minimal modifications of biological networks and experimental profiles to make them mutually consis-

[3]Note that edge labels (activation or inhibition) are well-curated in RegulonDB, which both enables and explains the obtained high accuracies. In fact, the description of RegulonDB in (Baumbach, Tauch, and Rahmann 2009) includes the following: "The amount of manually curated knowledge on the gene regulatory network of *E. coli* is the largest, currently available for any living organism."

tent. As final application, we provided an approach based on ASP to predict unobserved data even in case of inconsistency under SCM. We evaluated our approach on a real biological example and showed that predictions on the basis of cardinality-minimal repairs were, for one, feasible and, for another, highly accurate. This is of practical relevance because genetic profiles from microarray data tend to be noisy and available biological networks far from being complete.

Given that our framework is configurable, it can be adjusted to varying tasks that occur in practice. As illustrated in the experiments, it enables a meaningful analysis of partially unreliable experimental data, and reasonable conclusions from it can be drawn automatically. Another application scenario stems from the utilization of data-driven methods to derive biological models (Joyce and Palsson 2006; Faith et al. 2007; Bansal et al. 2007). In short, such approaches are based on probabilistic frameworks determining most likely models (typically, transcriptional regulatory networks or perturbation targets) given the data. The underlying optimization problem is usually non-convex, and finding a global optimum is not guaranteed in practice. Hence, existing algorithms report a local optimum that should be interpreted with care: errors can occur, and there may not be a consensual model (Ferrazzi et al. 2007). Formal methods provide means to deal with these issues, as shown in (Gutiérrez-Ríos et al. 2003; Covert et al. 2008) for checking the consistency between biological models and experimental data, a functionality partially automated in a Cytoscape plug-in (Baumbach and Apeltsin 2008). However, to our knowledge, this work provides the first approach to automatically and globally reason over a whole biological network in order to identify minimal repairs on the network and/or data. Our method could thus be used both to indicate and to correct spurious parts of models generated from data.

Our approach is fully declarative and assigns a clear semantics to (minimal) repair and prediction, so decoupling intended results from phenomena of their computation. The explicit representation of repair operations allows for reasoning over corrections on either or both of data and model. The latter distinguishes our approach to repair from the one applied in the area of databases (Arenas, Bertossi, and Chomicki 1999), which is limited to data repair. Moreover, alternative notions can be used for repair minimality, e.g., based on weighted sum or subset inclusion. In fact, a (disjunctive) ASP encoding of subset-minimality testing is provided in the appendix, but tentative experiments indicated that computing subset-minimal repairs is harder than cardinality minimization. However, available biological knowledge could be incorporated in our framework to improve the validity of results as well as computational efficiency.

As an important aspect, we want to stress the advantages of using ASP as paradigm for realizing our application. On the knowledge representation side, ASP fosters uniform problem specifications in terms of a separate encoding and instances given as facts. This accelerates the development of a problem solution, viz., of the encoding, and it keeps solutions comprehensible in view of the fact that encodings are usually compact. To see this, note that our repair encoding contains not more than 20 rules, in which we specified five kinds of repair operations with different targets (model or data). One additional #*minimize* statement enabled us to compute cardinality-minimal repairs using off-the-shelf tools. Notably, the task of prediction benefits from the cautious reasoning capacities of *clasp*, intersecting all answer sets by enumerating only linearly many of them. This illustrates the second major advantage of ASP, namely, the availability of powerful off-the-shelf inference engines. Except for instance translators used to convert large biological networks and experimental profiles into ASP facts, no proprietary imperative code was needed for deploying and solving our sophisticated application.

Appendix: Subset-Minimal Repairs

Example 4 shows that minimizing cardinality can miss subset-minimal repairs, which could be of interest too. An alternative is then to identify *all* subset-minimal repairs. Unlike cardinality-minimal repair, which can be accomplished via a branch-and-bound approach, subset-minimal repair deals with the question of whether a set of applied repair operations has no proper subset that is also a repair. In other words, subset-minimality of a candidate repair is refuted if one finds a counterexample, i.e., a strictly smaller set of repair operations along with total consistent vertex and edge labelings; in turn, subset-minimality holds if no such counterexample exists. The task of finding subset-minimal repairs thus combines two subproblems: first, providing a candidate repair and, second, excluding the existence of a counterexample. Such a pattern is typical for Σ_2^P-complete problems (cf. (Schaefer and Umans 2002) for a compendium), among which we find, e.g., brave reasoning in disjunctive ASP (Eiter and Gottlob 1995; Leone et al. 2006). Even though we have not established Σ_2^P-hardness of decision problems related to subset-minimal repair, it is convenient to encode it in disjunctive ASP, in view that (disjunctive) rules for testing subset-minimality can simply be added to the rules in (5)–(22).

In fact, one can replace the #*minimize* statement in (23) with the encoding shown in Figure 4 to check whether a candidate repair, generated via the rules in (5)–(22), is subset-minimal. The rules in Figure 4 aim at the construction of a counterexample comprising a strictly smaller set of applied repair operations and consistent total labelings of edges and vertices, which are now given by instances of predicates lab_E and lab_V. Following the methodology in (Eiter and Gottlob 1995), the idea is to derive an inflationary amount of atoms from an error-indicating atom bot, expressing failure to construct a counterexample. The requirement that all atoms in an answer set must necessarily be derived makes sure that bot can only be true if there is no counterexample for a candidate repair. Finally, an integrity constraint stipulates bot to hold, so that only answer sets comprising a subset-minimal repair (not admitting any counterexample) remain.

Before describing the rules in Figure 4, note that the syntax and semantics introduced in the background are generalized to disjunctive programs as follows. Let $a = h_1\,;\ldots\,;h_l$ for the head a of a rule of form (1), where each h_j is an atom for $1 \leq j \leq l$, and '$\,;$' denotes disjunction. According to this,

$$\begin{array}{rcll}
\mathit{drop}(R)\,;\mathit{keep}(R) & \leftarrow & \mathit{app}(R). & (24)\\
\mathit{keep}(R) & \leftarrow & \mathit{rep}(R), {\sim}\mathit{app}(R). & (25)\\
\mathit{bot} & \leftarrow & \mathit{keep}(R) : \mathit{rep}(R). & (26)\\[1ex]
\mathit{lab\_E}(U,V,S) : \mathit{sig}(S) & \leftarrow & \mathit{edge}(U,V). & (27)\\
\mathit{lab\_E}(U,V,S) : \mathit{sig}(S) & \leftarrow & \mathit{app}(\mathit{add\_e}(U,V)), \mathit{keep}(\mathit{add\_e}(U,V)). & (28)\\
\mathit{lab\_E}(U,V,S) & \leftarrow & \mathit{edge}(U,V), \mathit{obs\_e}(U,V,S), {\sim}\mathit{app}(\mathit{flip\_e}(U,V,S)). & (29)\\
\mathit{lab\_E}(U,V,T) & \leftarrow & \mathit{app}(\mathit{flip\_e}(U,V,S)), \mathit{keep}(\mathit{flip\_e}(U,V,S)), \mathit{opp}(S,T). & (30)\\
\mathit{lab\_E}(U,V,S) & \leftarrow & \mathit{drop}(\mathit{flip\_e}(U,V,S)). & (31)\\[1ex]
\mathit{lab\_V}(P,V,S) : \mathit{sig}(S) & \leftarrow & \mathit{vtx}(V), \mathit{exp}(P). & (32)\\
\mathit{lab\_V}(P,V,S) & \leftarrow & \mathit{vtx}(V), \mathit{exp}(P), \mathit{obs\_v}(P,V,S), {\sim}\mathit{app}(\mathit{flip\_v}(P,V,S)). & (33)\\
\mathit{lab\_V}(P,V,T) & \leftarrow & \mathit{app}(\mathit{flip\_v}(P,V,S)), \mathit{keep}(\mathit{flip\_v}(P,V,S)), \mathit{opp}(S,T). & (34)\\
\mathit{lab\_V}(P,V,S) & \leftarrow & \mathit{drop}(\mathit{flip\_v}(P,V,S)). & (35)\\[1ex]
\mathit{con}(P,U,V) & \leftarrow & \mathit{lab\_E}(U,V,S), \mathit{lab\_V}(P,U,T), \mathit{lab\_V}(P,V,Y), Y \neq S{*}T, {\sim}\mathit{inp}(P,V). & (36)\\
\mathit{con}(P,U,V) & \leftarrow & \mathit{rep}(\mathit{add\_e}(U,V)), \mathit{exp}(P), {\sim}\mathit{app}(\mathit{add\_e}(U,V)), {\sim}\mathit{inp}(P,V). & (37)\\
\mathit{con}(P,U,V) & \leftarrow & \mathit{drop}(\mathit{add\_e}(U,V)), \mathit{exp}(P), {\sim}\mathit{inp}(P,V). & (38)\\
\mathit{mis}(P,V) & \leftarrow & \mathit{vtx}(V), \mathit{exp}(P), \mathit{con}(P,U,V) : \mathit{edge}(U,V), \mathit{con}(P,W,V) : \mathit{rep}(\mathit{add\_e}(W,V)), {\sim}\mathit{inp}(P,V). & (39)\\
\mathit{bot} & \leftarrow & \mathit{mis}(P,V), {\sim}\mathit{app}(\mathit{inp\_v}(V)), {\sim}\mathit{app}(\mathit{inp\_v}(P,V)). & (40)\\
\mathit{bot} & \leftarrow & \mathit{mis}(P,V), \mathit{drop}(\mathit{inp\_v}(V)). & (41)\\
\mathit{bot} & \leftarrow & \mathit{mis}(P,V), \mathit{drop}(\mathit{inp\_v}(P,V)). & (42)\\[1ex]
\mathit{drop}(R) & \leftarrow & \mathit{bot}, \mathit{rep}(R). & (43)\\
\mathit{keep}(R) & \leftarrow & \mathit{bot}, \mathit{rep}(R). & (44)\\
\mathit{lab\_E}(U,V,S) & \leftarrow & \mathit{bot}, \mathit{sig}(S), \mathit{edge}(U,V). & (45)\\
\mathit{lab\_E}(U,V,S) & \leftarrow & \mathit{bot}, \mathit{sig}(S), \mathit{rep}(\mathit{add\_e}(U,V)). & (46)\\
\mathit{lab\_V}(P,V,S) & \leftarrow & \mathit{bot}, \mathit{sig}(S), \mathit{vtx}(V), \mathit{exp}(P). & (47)\\
 & \leftarrow & {\sim}\mathit{bot}. & (48)
\end{array}$$

Figure 4: Disjunctive encoding of testing repairs' subset-minimality, which can be used in place of #*minimize* statement (23).

the notion of a model X is adapted by requiring $\{h_1, \dots, h_l, a_{m+1}, \dots, a_n\} \cap X \neq \emptyset$ or $\{a_1, \dots, a_m\} \not\subseteq X$ for every rule of form (1) belonging to the ground instantiation $\mathbf{\Pi}$ of a disjunctive program Π; remaining concepts stay unchanged.

Looking at Figure 4, Rule (24) describes the choice of applied repair operations to drop in a counterexample, while Rule (25) immediately declares non-applied operations as kept (or not dropped, respectively). This is used in Rule (26), deriving error-indicating atom *bot* if all repair operations are kept.[4] In order to avoid deriving *bot*, a counterexample must thus drop at least one applied repair operation.

The rules in (27)–(31) direct choosing labels of edges in the construction of a witness for re-established consistency wrt a counterexample. In fact, rules (27) and (28) express that a label needs to be chosen for each edge of the given influence graph as well as for any edge added by a repair operation that is kept. The remaining rules take care of known edge labels and respective repair operations: (29) propagates available labels not subject to flipping, while (30) and (31) derive the opposite or the original label, respectively, depending on whether an applied flip operation is kept or dropped.

A total vertex labeling wrt a counterexample is constructed in a similar fashion as with edges, using rules (32)–(35) to handle variations (within experimental profiles) and potential flips. Note that (32) is analog to (27), and (33)–(35) to (29)–(31), while there is no counterpart of (28) because our repair operations do not introduce any new vertices.

The rules in (36)–(42) formalize incompatible influences wrt the vertex and edge labelings of a counterexample, deriving *bot* if a non-input vertex is unexplained in some experimental profile. To this end, Rule (36) compares the sign of a non-input vertex V in an experimental profile P to the influence it receives from a regulator U, and $con(P,U,V)$ is derived if the influence is contrary to the sign of V. For edges that can potentially be added by repair operations, Rule (37) and (38) also derive a contrary influence on V if such an operation is not applied or dropped, respectively. Given this, Rule (39) identifies non-input vertices V receiv-

[4]Note that *gringo* expands ':' in a rule body to a conjunction of atoms on the left-hand side, where ground instances are obtained from "domain predicates" on the right-hand side. Likewise, an occurrence of ':' in a rule head is expanded to a disjunction (cf. (Gebser et al. 2009)).

ing exclusively contrary influences in a profile P, thereby, taking the conjunction over all edges in the given influence graph targeting V as well as all potential regulators from which edges to V can be added by repairs. The remaining rules derive bot if a non-input vertex V missing an explanation is not made input by any repair operation: (40) is used if neither the global nor the local operation to make V input is applied, while (41) and (42) check for whether either of these operations is dropped. (Recall that, in view of the integrity constraint in (12), repair applications making the same vertex input both globally and locally are impossible.)

Before we proceed, let us briefly summarize the encoding parts described so far. The rules in (24)–(26) direct guessing applied repair operations to drop in a counterexample. In order to witness re-established consistency, total labelings of edges and vertices are constructed via the rules in (27)–(31) and (32)–(35), respectively, taking kept and dropped repairs likewise into account. Given such labelings, the rules in (36)–(42) check for whether some non-input vertex is unexplained in an experimental profile. Importantly, if a counterexample under construction is invalid because it does not drop any applied repair operation or leaves some non-input vertex unexplained, it is indicated by deriving bot from a ground instance of Rule (26) or rules (40)–(42), respectively. Conversely, a valid counterexample manifests itself in having a representation as set of atoms such that the rules in (24)–(42) are satisfied without requiring bot to be true.

The final technical part consists of from bot deriving all possible options for repair operations, that is, dropping and keeping them, as well as all labels for edges (possibly added by a repair) and vertices. This is accomplished via the rules in (43)–(47). As atoms in an answer set must be derived necessarily from a program, their inflationary inclusion whenever bot is true makes sure that bot belongs to an answer set only if there is no counterexample for a candidate repair. Finally, the integrity constraint in (48) stipulates bot to hold. This eliminates potential answer sets consisting of a candidate repair along with a counterexample, given by labelings witnessing re-established consistency under fewer repair operations. Every remaining answer set must thus comprise a repair from which no operation can be dropped without sacrificing consistency, so that bot is necessarily derived. Hence, a candidate repair (represented in terms of predicate app) passes the test in Figure 4 iff it is subset-minimal.

Example 5 *As discussed in Example 4, the following vertex labelings witness that mutual consistency between the influence graph and experimental profiles shown in Figure 1 can be re-established by making vertices a and b inputs:*

	a	b	c	d	e
p_1	1	-1	-1	1	$1 \vert -1$
p_2	1	-1	-1	1	-1

Note that the witnesses agree on the increases of a and d, which are the regulators of b, as well as on the decrease of b, the only regulator of a. Given that the variations of a and b are both unexplained (in each of the experimental profiles p_1 and p_2), the repair that makes a and b inputs is subset-minimal. Thus, answer sets containing $app(inp\_v(a))$ and $app(inp\_v(b))$ as applied repair operations should pass the test in Figure 4 by producing bot. In view of the fact that c, e, and at least one of a and b cannot jointly be explained (increases of input d in p_1 and p_2 need not be explained), bot is derived via Rule (40) or (41) if choosing $drop(inp\_v(a))$ or $drop(inp\_v(b))$ to satisfy a respective ground instance of Rule (24). To avoid such a derivation of bot, we must thus choose $keep(inp\_v(a))$ and $keep(inp\_v(b))$ to satisfy (24). Since $keep(inp\_v(c))$ and $keep(inp\_v(e))$ must be true in view of Rule (25), while $rep(inp\_v(d))$ cannot be derived via Rule (7) in Figure 3, predicate $keep$ then holds for all derivable ground instances of (domain) predicate rep. This again implies bot in view of Rule (26). We have thus checked that bot and all atoms following from it must necessarily be derived, so that the subset-minimality test succeeds for answer sets containing $app(inp\_v(a))$ and $app(inp\_v(b))$. $\diamond$

By using the rules in Figure 4 in place of the #*minimize* statement in (23), one can compute and also perform prediction wrt subset-minimal repairs instead of cardinality-minimal ones, but the resulting tasks appear to be computationally much harder. In fact, we tried disjunctive ASP solver *claspD* (Drescher et al. 2008) on part of the data used for the experiments with *clasp* and observed many timeouts. In view of the expected runtime, we thus refrained from conducting (and reporting results of) an extensive empirical investigation. In practice, the choice of whether to apply cardinality (or likewise weighted sum) or subset inclusion as minimality measure certainly depends on the considered network and data, experience, and scalability. From the viewpoint of scalability, our tentative experiments indicated that cardinality-minimal repairs and prediction wrt them are much easier to compute.

Acknowledgments Philippe Veber was supported by DAAD. This work was partially funded by GoFORSYS (Grant 0313924). The authors are grateful to Roland Kaminski for helpful comments on encodings and to Sylvain Blachon for useful discussions on the subject of this paper.

References

Allen, T.; Herrgård, M.; Liu, M.; Qiu, Y.; Glasner, J.; Blattner, F.; and Palsson, B. 2003. 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.

Arenas, M.; Bertossi, L.; and Chomicki, J. 1999. Consistent query answers in inconsistent databases. In *Proceedings of the Eighteenth Symposium on Principles of Database Systems*, 68–79. ACM Press.

Bansal, M.; Belcastro, V.; Ambesi-Impiombato, A.; and di Bernardo, D. 2007. How to infer gene networks from expression profiles. *Molecular Systems Biology* 3(78).

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Baumbach, J., and Apeltsin, L. 2008. Linking Cytoscape and the corynebacterial reference database CoryneRegNet. *BMC Genomics* 9(184).

Baumbach, J.; Tauch, A.; and Rahmann, S. 2009. Towards the integrated analysis, visualization and reconstruction of microbial gene regulatory networks. *Briefings in Bioinformatics* 10(1):75–83.

Bradley, M.; Beach, M.; de Koning, A.; Pratt, T.; and Osuna, R. 2007. Effects of Fis on Escherichia coli gene expression during different growth stages. *Microbiology* 153:2922–2940.

Covert, M.; Xiao, N.; Chen, T.; and Karr, J. 2008. Integrating metabolic, transcriptional regulatory and signal transduction models in Escherichia coli. *Bioinformatics* 24(18):2044–2050.

Drescher, C.; Gebser, M.; Grote, T.; Kaufmann, B.; König, A.; Ostrowski, M.; and Schaub, T. 2008. Conflict-driven disjunctive answer set solving. In *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning*, 422–432. AAAI Press.

Eiter, T., and Gottlob, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence* 15(3-4):289–323.

Erdem, E.; Lin, F.; and Schaub, T., eds. 2009. *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer.

Ernst, J.; Beg, Q.; Kay, K.; Balázsi, G.; Oltvai, Z.; and Bar-Joseph, Z. 2008. A semi-supervised method for predicting transcription factor-gene interactions in Escherichia coli. *PLoS Computational Biology* 4(3).

Faith, J.; Hayete, B.; Thaden, J.; Mogno, I.; Wierzbowski, J.; Cottarel, G.; Kasif, S.; Collins, J.; and Gardner, T. 2007. Large-scale mapping and validation of Escherichia coli transcriptional regulation from a compendium of expression profiles. *PLoS Computational Biology* 5(1).

Ferrazzi, F.; Magni, P.; Sacchi, L.; Nuzzo, A.; Petrovič, U.; and Bellazzi, R. 2007. Inferring gene regulatory networks by integrating static and dynamic data. *International Journal of Medical Informatics* 76:462–475.

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.; and Collado-Vides, J. 2008. 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.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007. Conflict-driven answer set solving. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, 386–392. AAAI Press/MIT Press.

Gebser, M.; Schaub, T.; Thiele, S.; Usadel, B.; and Veber, P. 2008. Detecting inconsistencies in large biological networks with answer set programming. In *Proceedings of the Twenty-fourth International Conference on Logic Programming*, 130–144. Springer.

Gebser, M.; Kaminski, R.; Ostrowski, M.; Schaub, T.; and Thiele, S. 2009. On the input language of ASP grounder gringo. In Erdem et al. (2009), 502–508.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2009. The conflict-driven answer set solver clasp: Progress report. In Erdem et al. (2009), 509–514.

Gutiérrez-Ríos, R.; Rosenblueth, D.; Loza, J.; Huerta, A.; Glasner, J.; Blattner, F.; and Collado-Vides, J. 2003. Regulatory network of Escherichia coli: Consistency between literature knowledge and microarray profiles. *Genome Research* 13(11):2435–2443.

Ideker, T. 2004. A systems approach to discovering signaling and regulatory pathways—or, how to digest large interaction networks into relevant pieces. *Advances in Experimental Medicine and Biology* 547:21–30.

Joyce, A., and Palsson, B. 2006. The model organism as a system: Integrating 'omics' data sets. *Nature Reviews Molecular Cell Biology* 7(3):198–210.

Kauffman, S. 1993. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press.

Kauffman, S.; Peterson, C.; Samuelsson, B.; and Troein, C. 2003. Random Boolean network models and the yeast transcriptional network. *Proceedings of the National Academy of Sciences of the USA* 100(25):14796–14799.

Kuipers, B. 1994. *Qualitative reasoning. Modeling and simulation with incomplete knowledge*. MIT Press.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic* 7(3):499–562.

Schaefer, M., and Umans, C. 2002. Completeness in the polynomial-time hierarchy: A compendium. *ACM SIGACT News* 33(3):32–49.

Siegel, A.; Radulescu, O.; Le Borgne, M.; Veber, P.; Ouy, J.; and Lagarrigue, S. 2006. Qualitative analysis of the relation between DNA microarray data and behavioral models of regulation networks. *Biosystems* 84(2):153–174.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence* 138(1-2):181–234.

Soulé, C. 2006. Mathematical approaches to differentiation and gene regulation. *Comptes rendus biologies* 329(1):13–20.

Syrjänen, T. Lparse 1.0 user's manual.

Thomas, R., ed. 1979. *Kinetic Logic: A Boolean Approach to the Analysis of Complex Regulatory Systems*. Springer.

Veber, P.; Guziolowski, C.; Le Borgne, M.; Radulescu, O.; and Siegel, A. 2008. Inferring the role of transcription factors in regulatory networks. *BMC Bioinformatics* 9(228).