Detecting Inconsistencies in Large Influence Networks with Answer Set Programming

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

[1] Universität Potsdam, Institut für Informatik, August-Bebel-Str. 89, D-14482 Potsdam
[2] Max-Planck-Institut für Molekulare Pflanzenphysiologie, Am Mühlenberg 1, D-14476 Golm

Abstract. Siegel and colleagues recently proposed a principled definition of consistency between biochemical/genetic reactions and high-throughput profiles of cell activity. Following this work, we present an approach based on Answer Set Programming to check the consistency of large-scale datasets. Furthermore, we extend this approach 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 missing reactions.
Correspondence to: philippe.veber@googlemail.com

1 Introduction

This paper deals 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 in a given condition, leaving to human experts the task of making biological sense out of tens of gene identifiers. 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 that is 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 model by Siegel and colleagues [4], later on referred to as the *Sign Consistency Model* (SCM for short), for developing declarative techniques based on *Answer Set Programming* (ASP for short) [6] to detect and explain inconsistencies in large datasets.

The SCM imposes constraints between experimental measurements and a graph representation of cellular interactions, named an *influence* (or interaction) *graph* [7]. These constraints, later on referred to as *sign consistency constraints*, are described in Section 2. Section 3 provides an intuitive introduction to ASP, a logic-programming paradigm popular due to its declarativeness. 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. For both problems, we report preliminary experimental results on randomly generated instances.

Section 6 concludes this paper with a brief discussion and an outlook on future progression.

2 Influence Graphs and Sign Consistency Constraints

Influence graphs [7] (also called interaction graphs in the literature) are a common representation for a wide class of dynamical systems. Basically, an *influence graph* is a directed graph whose vertices are the input and state variables of a system and whose edges express the effect of variables on each other. Informally, an edge $j \to i$ means that the rate of variation of j in time influences the level of i. Each edge $j \to i$ of an influence graph is labeled with a sign, either **+** or **–**, denoted by $\sigma(j, i)$. Sign **+** (resp., **–**) indicates that j tends to increase (resp., decrease) i. An example of influence graph is given in Fig. 1; it represents a simplified model for the operon lactose in *E. coli*.

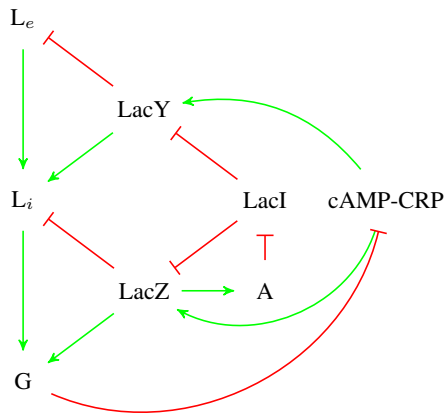

Fig. 1. Simplified model of operon lactose in *E. coli*, represented as an influence graph. The vertices represent either genes, metabolites or proteins, while the edges indicate the regulations among them. Green arrows with normal head stand for positive regulations (activations) while red arrows with tee heads stand for negative regulations (inhibitions). Vertices G and L_e are considered to be inputs of the system, that is they are unconstrained.

In the field of genetic networks, influence graphs have been investigated under various classes of dynamical systems – from ordinary differential equations [8], to synchronous [9] and asynchronous [10] Boolean networks. Influence graph have also been introduced in the field of qualitative reasoning [11], to describe physical systems where a detailed quantitative description is not available. This has been the main motivation for using influence graphs for knowledge representation in the context of biological systems.

In the SCM, *experimental profiles* are supposed to come from steady state shift experiments where, initially, the system is at steady state, then perturbed using control pa-

rameters, and eventually, it settles into another steady state (after a while). It is assumed that the data measures the differences between the initial and the final state. Thus, for each gene, protein, or metabolite, we know whether its concentration has increased or decreased, while quantitative values are unavailable, unessential, or unreliable. By $\mu(i)$, we denote the sign, again + 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 [11].

We above introduced influence graphs (as a representation of cellular interactions) and labelings of their vertices with signs (as a representation of experimental profiles). We now describe the constraints that relate both. Informally, for any 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 the multiplication on real 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 any $i \in V$, the sign $\mu(i)$ of i is consistent, if there is some edge $j \to i$ in E such that $\mu(i) = \mu(j)\sigma(j,i)$.*

Table 1 shows four different vertex labelings of the influence graph given in Fig. 1. The labeling μ_1 is consistent with the influence graph: the variation of each vertex (apart from the inputs G, and L_e, see Fig. 1) can be explained by the effect of one of its regulators. For instance LacY receives one positive influence from cAMP-CRP, and one negative influence from LacI, which accounts for the variation of LacY. The second labeling, μ_2 is not consistent: this time LacY receives only negative influences from cAMP-CRP and LacI and its variation cannot be explained.

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

Table 1. Some labelings for the influence graph depicted in Fig. 1.

The notion of (sign) consistency is extended to whole influence graphs in the natural way, requiring the sign of each vertex to be consistent. Furthermore, 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. For instance, vertex labeling μ_3 is consistent with the influence graph given in Fig. 1, as setting the signs +, –, –, –, + to L_i, LacY, LacZ, A and cAMP-CRP respectively extends μ_3 in a consistent labeling. On the other hand, μ_4 cannot be consistently extended.

3 Answer Set Programming

This section provides a brief, informal introduction to ASP (see [6] for formal details). ASP is an attractive declarative paradigm for knowledge representation and reasoning, offering a rich modeling language [12,13] along with highly efficient inference engines based on Boolean constraint solving technology [14,15,16]. The basic idea of ASP is to encode a problem as a logic program such that its answer sets represent solutions.

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 [17]. A *disjunctive logic program* is a finite set of rules of the form

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

where a_i, b_j, c_k are *atoms* for $0 < i \leq l < j \leq m < k \leq n$. A rule r as in (1) is called a *fact*, if $l = n = 1$, and an *integrity constraint*, if $l = 0$. Intuitively, a rule amounts to an implication, with ',' standing for '$\wedge$' and ';' for '$\vee$'; however, standard transformations valid in classical logic, e.g., contraposition, are not valid under the answer set semantics. In fact, the answer sets of a program are particular classical models of the program satisfying a certain stability criterion (cf. [6]). Roughly, a set X of atoms is an *answer set* of a program, if for each program rule of form (1), X contains a minimum number of atoms among $a_1, \ldots, a_l$ when $b_{l+1}, \ldots, b_m$ 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.

Though answer sets are usually defined on ground (i.e., variable-free) programs, the elegance of ASP comes from the possibility to provide non-ground *problem encodings*, where schematic rules amount to their ground instantiations. Grounders, like *lparse* [13], are capable of combining a problem encoding and an instance (typically a set of ground facts) into an equivalent ground program, which is then 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 in polynomial time. However, as soon as the experimental profile has missing values (which is very likely in practice), the problem becomes NP-hard [18].

Here, we present an encoding of the problem as a logic program such that each of its answer sets matches a consistent extension of vertex and edge labelings. Our program is composed of three parts, which we describe next.

Problem Instance The influence graph as well as the 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). Furthermore, if the variation s (either **+** or **–**) of a species i is given in the profile, it is modeled by a fact *obs_vlabel*(i,t), where $t = p$ if $s =$ **+** and $t = n$ if $s =$ **–**, and if the sign s of an edge $j \rightarrow i$ is known, it is expressed by a fact *obs_elabel*(j,i,t).

Generating Solution Candidates As stated 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:

$$\begin{array}{c} \mathit{vlabel}(V,p)\,;\,\mathit{vlabel}(V,n) \leftarrow \mathit{vertex}(V)\ , \\ \mathit{elabel}(U,V,p)\,;\,\mathit{elabel}(U,V,n) \leftarrow \mathit{edge}(U,V)\ . \end{array} \tag{2}$$

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

$$\begin{array}{c} \mathit{vlabel}(V,S) \leftarrow \mathit{obs\_vlabel}(V,S)\ , \\ \mathit{elabel}(U,V,S) \leftarrow \mathit{obs\_elabel}(U,V,S)\ . \end{array} \tag{3}$$

Note that the stability criterion for answer sets implies that a known label derived via rules in (3) is also derived via rules in (2), thus, excluding the opposite label.

Testing Solution Candidates Finally, we check whether the generated total labelings satisfy the sign consistency constraints stated in Definition 1, stipulating an influence of sign s for each vertex i with variation s. We thus define *infl*(i,s) to indicate that i receives an influence of sign s, where the encoding contains facts *sign*(p) and *sign*(n):

$$\begin{array}{l} \mathit{infl}(V,p) \leftarrow \mathit{edge}(U,V), \mathit{elabel}(U,V,S), \mathit{vlabel}(U,S), \mathit{sign}(S)\ , \\ \mathit{infl}(V,n) \leftarrow \mathit{edge}(U,V), \mathit{elabel}(U,V,S), \mathit{vlabel}(U,T), \mathit{sign}(S), \mathit{sign}(T), S \neq T\ . \end{array} \tag{4}$$

Inconsistent labelings are then ruled out by integrity constraints of the following form:

$$\leftarrow \mathit{vertex}(V), \mathit{vlabel}(V,S), \mathit{sign}(S), \mathit{not}\ \mathit{infl}(V,S)\ . \tag{5}$$

Benchmarks We assessed the efficiency of our approach on artificially generated instances. Each instance is composed of a graph, a complete labeling of its edges with signs, and a partial labeling of its vertices. Our random generator of instances has three parameters: the number of vertices in the influence graph n, the average degree in the graph β and the proportion of observed nodes γ. To generate one instance, we compute a random graph under the model by Erdős-Rényi [19], where each pair of vertices has equal probability to be an edge. The parameter β is fixed to 2.5, and varying between 2 and 3 (which are usual values in biological networks [20]) does not change the results significantly (data not shown). The labels on edges are chosen independently with probability $\frac{1}{2}$ for each sign. Then $\lfloor \gamma n \rfloor$ vertices are chosen with uniform probability, and assigned a label with probability $\frac{1}{2}$ for each sign.

The instances were solved using the grounder *lparse* [13] and the solver *cmodels* [21]. These programs were run on an Intel Core 2 Duo 2.4 GHz processor, with 4GB of memory under Linux. All reported times correspond to Unix user time.

The results are presented in Fig. 2: we separated execution time into grounding time and solving time to show their relative contribution. The grounding stage transforms the original logic program into an equivalent variable-free program. This step is required because only ground programs can be solved efficiently in practice. Both graphics display the time needed for the corresponding phase as a function of the number of vertices in the instance. For each size, we generate 50 instances distributed in 5 groups having a different γ value (here: $\frac{1}{100}$, $\frac{1}{50}$, $\frac{1}{30}$, $\frac{1}{20}$, $\frac{1}{10}$).

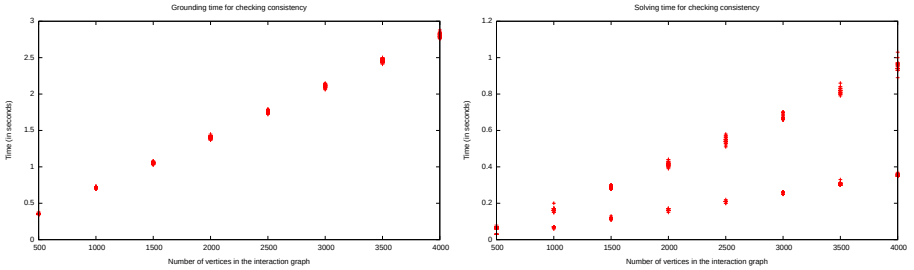

Fig. 2. Execution time for checking the consistency of an influence graph with an expression profile. The execution time is separated into its two contributions, namely grounding (on the left hand) and actual solving (on the right hand).

Grounding time increases linearly with the size of the instance, and does not vary significantly for instances of equal size. Solving time also increases linearly with the size of the instance, though in a more sophisticated way: for a given size of the instances, one can distinguish two clusters of instances having well-separated solving time. Interestingly, all "easy" instances are inconsistent (or equivalently, all instances which are consistent are in the "hard" cluster). Now for each one of these two types of instances, the solving time grows linearly with the size of the instance.

From these results, we can see that checking consistency on data of realistic size (*i.e.* influence graphs of several thousands of vertices) takes no more than a couple of seconds on a standard PC.

5 Identifying Minimal Inconsistent Cores

Once it is proved that an experimental profile is inconsistent with a given influence graph (i.e., if the logic program given in the previous section has no answer set), due to the amount of data, it is crucial to provide explanations that are as concise as possible. Here, we adopt a strategy that was successfully applied on real biological data in [22], where the basic idea is to isolate minimal subgraphs of an influence graph such that the vertices and edges cannot be labeled in a consistent way. This task is closely related to extracting Minimal Unsatisfiable Cores (MUCs) [23] in the context of Boolean Satisfiability (SAT) [14]. In allusion, we call a minimal subgraph of the influence graph whose vertices and edges cannot be labeled consistently a *Minimal Inconsistent Core* (MIC).

As in the previous section, we present a disjunctive program such that its answer sets match MICs. We assume that a problem instance, that is, an influence graph along with an experimental profile, is represented by facts as specified in Section 4. The remainder of the logic program is the problem encoding, consisting of three parts: the first generating MIC candidates, the second asserting inconsistency, and the third verifying minimality. Thereby, the generating part comprises the rules in (2) and (3) as well as:

$$\mathit{active}(V)\,;\,\mathit{inactive}(V) \leftarrow \mathit{vertex}(V)\ .$$

The purpose of this additional rule is to permit guessing a set of vertices to be marked as active. The subgraph of the influence graph induced by the active vertices forms a MIC candidate, which is tested via the two encoding parts described next.

Testing for Inconsistency By adapting a methodology used in [24], the following subprogram makes sure that the active vertices belong to a subgraph that cannot be labeled consistently, taking into account all labelings of the residual vertices and edges:

$$
\begin{aligned}
\mathit{op}(U,V) &\leftarrow \mathit{active}(V), \mathit{vlabel}(V,n), \mathit{edge}(U,V), \mathit{elabel}(U,V,S), \mathit{vlabel}(U,S), \mathit{sign}(S)\ ,\\
\mathit{op}(U,V) &\leftarrow \mathit{active}(V), \mathit{vlabel}(V,p), \mathit{edge}(U,V), \mathit{elabel}(U,V,S), \mathit{vlabel}(U,T), \mathit{sign}(S),\\
&\qquad \mathit{sign}(T), S \neq T\ ,\\
\mathit{bottom} &\leftarrow \mathit{active}(V), \mathit{vertex}(V), \mathit{op}(U,V) : \mathit{edge}(U,V)\ ,^{3}\\
&\leftarrow \mathit{not}\ \mathit{bottom}\ ,\\
\mathit{vlabel}(V,S) &\leftarrow \mathit{bottom}, \mathit{vertex}(V), \mathit{sign}(S)\ ,\\
\mathit{elabel}(U,V,S) &\leftarrow \mathit{bottom}, \mathit{edge}(U,V), \mathit{sign}(S)\ .
\end{aligned}
$$

In this (part of the) encoding, $\mathit{op}(U,V)$ indicates that the influence of vertex U on active vertex V is opposite to the variation of V. If all regulators of V have such an opposite influence, the sign consistency constraint for V is violated. In this case, atom *bottom* is produced, along with all labels for vertices and edges. Here, the stability criterion for an answer set X imposes that *bottom* and all labels can only belong to X if the given problem instance does not permit consistent labelings. Finally, integrity constraint $\leftarrow \mathit{not}\ \mathit{bottom}$ necessitates the inclusion of *bottom* in any answer set, thus, stipulating an inevitable violation of the sign consistency constraint for some vertex that is active.

Testing for Minimality The second test is based on the idea that, picking any active vertex, the sign consistency constraints for all other active vertices should be satisfied by appropriate labelings. This conception is implemented in the following subprogram:

$$
\begin{aligned}
\mathit{vlabel'}(W,V,p)\,;\mathit{vlabel'}(W,V,n) &\leftarrow \mathit{active}(W), \mathit{vertex}(W), \mathit{vertex}(V)\ ,\\
\mathit{elabel'}(W,U,V,p)\,;\mathit{elabel'}(W,U,V,n) &\leftarrow \mathit{active}(W), \mathit{vertex}(W), \mathit{edge}(U,V)\ ,\\
\mathit{vlabel'}(W,V,S) &\leftarrow \mathit{active}(W), \mathit{vertex}(W), \mathit{obs\_vlabel}(V,S)\ ,\\
\mathit{elabel'}(W,U,V,S) &\leftarrow \mathit{active}(W), \mathit{vertex}(W), \mathit{obs\_elabel}(U,V,S)\ ,\\
\mathit{infl'}(W,V,p) &\leftarrow \mathit{active}(W), \mathit{vertex}(W), \mathit{active}(V), V \neq W, \mathit{edge}(U,V),\\
&\qquad \mathit{elabel'}(W,U,V,S), \mathit{vlabel'}(W,U,S), \mathit{sign}(S)\ ,\\
\mathit{infl'}(W,V,n) &\leftarrow \mathit{active}(W), \mathit{vertex}(W), \mathit{active}(V), V \neq W, \mathit{edge}(U,V),\\
&\qquad \mathit{elabel'}(W,U,V,S), \mathit{vlabel'}(W,U,T), \mathit{sign}(S), \mathit{sign}(T), S \neq T\ ,\\
&\leftarrow \mathit{active}(W), \mathit{vertex}(W), \mathit{active}(V), V \neq W, \mathit{vertex}(V),\\
&\qquad \mathit{vlabel'}(W,V,S), \mathit{sign}(S), \mathit{not}\ \mathit{infl'}(W,V,S)\ .
\end{aligned}
$$

This subprogram is similar to the consistency check encoded via the rules in (2–5). However, sign consistency constraints are here only checked for active vertices, and they must be satisfiable for all but one arbitrary active vertex W. Since W ranges over all vertices of the given influence graph, each active vertex is taken into consideration.

[3] In the language of *lparse* [13], $\mathit{op}(U,V) : \mathit{edge}(U,V)$ is used to refer to all ground atoms $\mathit{op}(j,i)$ for which $\mathit{edge}(j,i)$ holds, with the respective ground atoms connected by ','.

Benchmarks We assess the scalability of this approach within the setting given in the previous section. There again, we distinguish between grounding time and solving time. The results are presented in Fig. 3 (note that X and Y axis are in log-scale). For grounding, we found a nearly perfect linear relationship (in log scale) between the size of the instance and the grounding time, and the slope of the line is 2. In other words, the grounding stage is here in $O(n^2)$, which is absolutely consistent with the fact that our encoding for finding MIC grows also quadratically with the number of vertices (see for instance the predicate *vlabel'*).

Concerning the solving time, we also obtain linear relationships, in the following sense. For each size, the 50 instances are distributed into two groups. Most of them are easily solvable, that is in less than a minute. However, a couple of instances are strikingly more difficult, and may be between 100 and 1000 times longer to solve. Nevertheless, we observe that the time needed to solve the easiest (resp., the hardest) instances grows linearly (in log-scale) with the size of the instances. This time, the slope of the line is slightly greater than 2, that is 2.3 and 3.2 for the easiest and the hardest instances respectively. Unfortunately, we could not characterize the hardest instances further.

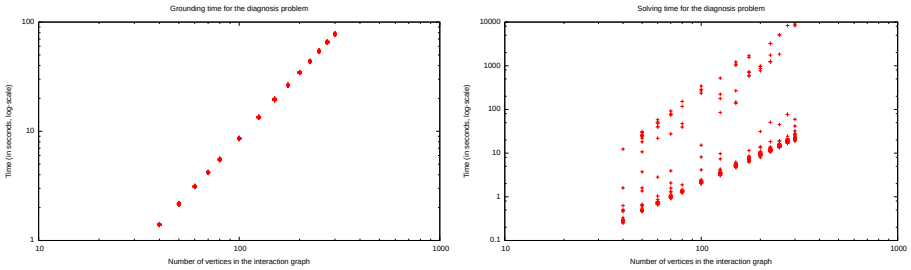

Fig. 3. Execution time for finding one MIC in an inconsistent instance. The execution time is separated into its two contributions, namely grounding (on the left hand) and actual solving (on the right hand). Note that on both cases, X and Y-axis are in log-scale.

These results suggest that finding a MIC in an inconsistent instance is computationally harder than checking consistency. This would not be surprising as extracting MUCs from unsatisfiable set of clauses is provably complete for the second level of the polynomial hierarchy [23]. However it should be noted that our instances are most often very easily solved, and it is still an open question whether instances coming real data fall into this category.

6 Discussion

We have provided an approach based on ASP to check 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. Such MICs can also be determined by means of ASP, and we have provided an encoding for this purpose. 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, we however argue for our technique based on ASP, as it allows for an elegant 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. An empirical assessment of them (on random as well as real data), along with a comparison to existent methods not based on ASP, is defered to an extended version of this paper. If the ASP approach computationally scales well, its elegance and flexibility in problem modeling might make it attractive for biological questions beyond the ones addressed here. It will also be interesting to investigate how far the performance of ASP tools can be tuned by varying and optimizing encodings.

Acknowledgments Ph. Veber is supported by a grant from the Deutscher Akademischer Austausch Dienst (DAAD). This work was partially funded by the Federal Ministry of Education and Research within the GoFORSYS project (http://www.goforsys.org/; grant 0313924).

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. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
7. Soulé, Christophe: Graphic Requirements for Multistationarity, Complexus **1**(3) (2003) 123–133
8. Soulé, Christophe: Mathematical approaches to differentiation and gene regulation. Comptes Rendus Biologies **329** (2006) 13–20
9. Remy, Élisabeth, Ruet, Paul, Thieffry, Denis: Graphic requirements for multistability and attractive cycles in a Boolean dynamical framework To appear in Advances in Applied Mathematics (2008)
10. Richard, Adrien, Comet, Jean-Paul: Necessary conditions for multistationarity in discrete dynamical systems Discrete Applied Mathematics (2007)
11. Kuipers, B.: Qualitative reasoning: Modeling and simulation with incomplete knowledge. MIT Press (1994)
12. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2) (2002) 181–234
13. Syrjänen, T.: Lparse 1.0 user's manual. http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz

14. Mitchell, D.: A SAT solver primer. Bulletin of the European Association for Theoretical Computer Science **85** (2005) 112–133
15. Giunchiglia, E., Lierler, Y., Maratea, M.: Answer set programming based on propositional satisfiability. Journal of Automated Reasoning **36**(4) (2006) 345–377
16. 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
17. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. Freeman and Co. (1979)
18. 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
19. Erdős, P., Rényi, A.: On Random Graphs. I. Publicationes Mathematicae **6** (1959) 290–297
20. Jeong, H., Tombor, B., Albert, R., Oltvai, Z.N., Barabási, A.L.: The large-scale organization of metabolic networks Nature **407** (2000) 651–654
21. Lierler, Yu.: Cmodels for Tight Disjunctive Logic Programs Proceedings of the Workshop on Constraint Logic Programming (2005)
22. 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
23. 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
24. 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