Proceedings of the 12th International Workshop on Constraint-Based Methods for Bioinformatics (WCB'16)

Held in conjunction with the 22th International Conference on Principles and Practice of Constraint Programming (CP-2016)

September 5, 2016
Toulouse Business School, Toulouse, France

Organized by
Alessandro Dal Palù, Agostino Dovier
and Simon de Givry

Preface

The 12th International Workshop on Constraint-Based Methods for Bioinformatics (WCB'16) continues the series of workshops on bioinformatics that were held alternately in conjunction with the previous CP and ICLP conferences. The aim of this workshop is to provide a forum where researchers in this area can exchange ideas, discuss new developments and explore possible future directions.

During the last years, biology has become a source of challenging problems for the entire field of computer science in general, and for the areas of computational logic and constraint programming in particular. Successful approaches to these problems are likely to have significant applications in several fields of research, such as medicine, agriculture, and industry. The topic of interest are all those concerning bioinformatics and constraints, and related techniques such as SAT, Answer Set Programming, Logic Programming, and Integer Linear Programming:

- RNA prediction and motif search
- protein structure and functional prediction
- genetic linkage analysis and haplotype inference
- phylogenetic tree reconstruction
- pedigree reconstruction and diagnosis
- genomic selection design
- gene regulatory network inference and analysis
- biochemical network simulation and visualization
- solvers for problems in biology
- metabolic pathway analysis
- DNA sequence assembly
- contig scaffolding
- multiple sequence alignment
- machine learning and big data
- ontologies in biology
- constraint databases in bioinformatics
- logical interfaces to relational biological databases
- web bioinformatics services

Regular WCB submissions were reviewed by up to three reviewers. In addition to this, the workshop chairs have invited this year bioinformatics papers submitted to CP2016 Biology Track and to ICLP2016. Papers 6 and 8 were accepted at CP2016. The ten papers reflected upon the diversity of the active topics that have been actively pursued, especially on systems biology and sequencing data analysis. We hope that these papers can promote future research on constraints and bioinformatics.

August 2016

Alessandro Dal Palù
Agostino Dovier
Simon de Givry

WCB'16 Workshop Organization

Program Chairs

Alessandro Dal Palù (Univ. of Parma, Italy),
Agostino Dovier (Univ. of Udine, Italy),
and Simon de Givry (INRA - MIAT, France)

Program Committee

Rolf Backofen (Albert-Ludwigs-University Freiburg, Germany),
Pedro Barahona (Universidade Nova de Lisboa, Portugal),
Alexander Bockmayr (Freie Universität Berlin, Germany),
Mats Carlsson (SICS, Sweden),
Francois Fages (Inria Paris-Rocquencourt, France),
Ines Lynce (INESC-ID/IST, University of Lisbon, Portugal),
Nigel Martin (Birkbeck, University of London, UK),
Alberto Policriti (University of Udine, Italy),
Enrico Pontelli (New Mexico State University, USA),
Sylvain Soliman (Inria Paris-Rocquencourt, France),
Sebastian Will (University Leipzig, Germany),
and Matthias Zytnicki (MIA-Toulouse, INRA, France)

Table of Contents

Logic programming applied to genome evolution in cancer*

A. Dal Palù[1], A. Dovier[2], A. Formisano[3], A. Policriti[2], and E. Pontelli[4]

[1] Dipartimento di Matematica e Informatica, Università degli Studi di Parma, Italy
[2] Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine, Italy
[3] Dipartimento di Matematica e Informatica, Università degli Studi di Perugia, Italy
[4] Department of Computer Science, New Mexico State University

Abstract. As often observed in the literature, cancer evolution follows a path that is unique to each patient; therefore, classical analysis based on the identification of typical mutations, provides little insight in the understanding of the general rules that drive cancer genesis and evolution. Recent genome sequencing pipelines allow researchers to retrieve rich genetic and epigenetic information from sampled tissues. Analyzing and comparing the evolution of cancer cells for each patient over a large time span can provide some accurate information and relationships. This paper presents a project for a logic programming based analysis that processes time-related genomic information.

Keywords: Cancer evolution, Genome analysis, ASP

1 Introduction

Modern sequencing techniques applied to genomic studies are now capable of producing high-throughput data related to specific individuals. With fast and inexpensive methods, it is possible to retrieve accurate information about a DNA sequence, its methylation (used for epigenetic studies), histones modifications, and gene and protein expression. The process can be repeatedly applied to the same sample over years, for instance, before and after a set of pharmacological therapies. The evolution of an organism and/or a specific sample of cells at genomic scale can be tracked when observing such biological properties. The cancer cells include features such as fast changing genome and cross combination of different offsprings of tumoral cells.

The classical theory of gene mutation, used since the 70s, defines the cancer evolution as a Darwinian process, where the cells compete for survival and the mutations accumulated over time may produce the insurgence of a tumor. However, the search for specific markers and pathways did not produce a clear understanding for many cases. More flexible models could capture the large variability of DNA mutations observed in the same type of tumors among patients.

* The work is partially supported by INdAM GNCS 2016 project.

Compared to previous models, where a simple gene mutation was assumed during cancer evolution, new data allows a more precise investigation and suggests new models based on evolution principles. In particular, the temporal dimension is taken into account in the genomic and epigenomic analysis [29]. This novel paradigm is reflected in the growing literature on *Cancer genome evolution*[18]: this research direction considers the genetic material as a global and detailed source of information. The changes among cells generations during the development of a tumor can be tracked and explained by looking at the global properties over time.

The goal of our study is to employ *Answer Set Programming (ASP)* [26, 24] to model new mining techniques, that search for relevant time-dependent relationships. In particular, differently from classical algorithms, where statistical analysis is used to identify strong peaks over a noise threshold, we focus on mixing evolutionary analysis and mutation analysis. The combination of the two techniques allows us to produce a rich and flexible model. The use of logic programming helps in the definition of a declarative model that merges two distinct aspects: the haplotype identification problem and phylogenetic reconstruction. The literature has already offered separate logic programming models of these two problems. In our case, the evolution of cancer genome can provide uniform input to both problems, namely the search for descriptors of mutations that are correlated over time.

Along with the modeling of this novel perspective, another challenge is the size of the data to be analyzed, requiring the use of modern ASP solving technologies and motivating the exploration of novel resolution models, such as those based on the use of parallel programming techniques (e.g., GPU programming, as recently explored in [27, 5, 7, 2, 3]). This paper provides a preliminary report describing the activities of an ongoing GNCS-2016 project, focused on the analysis of genome evolution in cancer, and outlining the potential of this research.

2 Background

We assume that the reader is familiar with Answer Set Programming (see, e.g., [26]). In this section, we briefly introduce the formalization of two well-known problems in bioinformatics. The first problem is the *haplotype inference* problem [16], i.e., the problem of identifying the minimal set of mutations that explain those observed on a population-wide genome sequencing. The second problem considered is the classical problem of *phylogenetic inference*: the reconstruction of a tree that summarizes the mutations over time for a set of species.

ASP is particularly suited to the modeling and resolution of these classes of problems, because of its flexibility in the modeling phase, its elaboration tolerance, and the fast prototyping cycle. In the literature, there are examples of ASP encoding of the haplotyping problem [10] and phylogenetic tree reconstruction problem [25, 9] (along with other uses of ASP to support phylogenetic data, e.g., to support complex queries on phylogenetic repositories [4]). These problems have also been addressed using alternative logic-based and constraint-based

paradigms—the readers are referred to, e.g., [1, 28] for additional references. However there are no applications nor combinations of these techniques in the study of genome evolution in cancer.

2.1 Phylogenetic Inference

Phylogenies are artifacts that describe the relationships among entities (e.g., proteins or genomes) derived from a process of evolution. We often refer to the entities studied in a phylogeny as *taxonomic units (TUs)* or *taxa*.

The field of *Phylogenetics* developed from the domain of biology as a powerful instrument to investigate similarities and differences among entities as a result of an evolutionary process. Evolutionary theory provides a powerful framework for comparative biology, by converting similarities and differences into events reflecting causal processes. As such, evolutionary-based methods provide more reliable answers than the traditional similarity-based methods, as they employ a theory (of evolution) to describe changes instead of relying on simple pattern matching. Indeed, evolutionary analyses have become the norm in a variety of areas of biological analysis. Evolutionary methods have proved successful, not merely in addressing issues of interest to evolutionary biologists, but in regard to practical problems of structural and functional inference [32]. Evolutionary inference of pairing interactions determining ribosomal RNA structure [35] is a clear case in which progress was made by the preferential use of an evolutionary inference method, even when direct (but expensive and imprecise) experimental alternatives were available. Eisen and others [31, 8] have shown how an explicitly evolutionary approach to protein "function" assignment eliminates certain categories of error that arise from gene duplication and loss, unequal rates of evolution, and inadequate sampling. Other inference problems that have been addressed through evolutionary methods include studies of implications of SNPs in the human population [31], identification of specificity-determining sites [14], inference of interactions between sites in proteins [34], interactions between proteins [33], and inferences of categories of sets of genes that have undergone adaptive evolution in recent history [23].

Phylogenetic analysis has also found applications in domains that are outside of the realm of biology; for example, a rich literature has explored the evolution of languages (e.g., [12, 30, 6]). The definitions and techniques employed are the same; of course the notion of "observable property" can be different. Starting from genes one notices differences using string matching algorithms. But differences (to be analyzed and explained) can be more macroscopic such as the presence/absence of a tail in an animal or the way one say "father" in a language.

Modeling. Let us consider the problem of *phylogenetic tree reconstruction*, namely: given a set of data characterizing the entities being studied (e.g., species, genes, languages), we wish to identify a phylogeny that accurately describes the evolutionary lineages among the given entities. We start with the notion of phylogenetic tree and then we give the notion of compatibility of characters.

A *phylogenetic tree* (or simply a *phylogeny*) is typically a labeled binary tree $(V, E, L, \mathcal{T}, \mathcal{L})$ where:

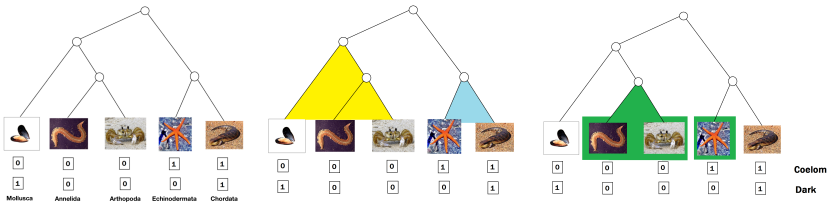


Fig. 1. A Sample Phylogeny (left), compatible (center–Coelom) and incompatible (right–Dark) characters

- The leaves L represent the taxonomic units being compared;
- The internal nodes $V \setminus L$ represent the (hypothetical) ancestral units; in rare cases, the internal nodes correspond to concrete entities (e.g., fossils);
- The edges E of the tree describe evolutionary relationships; the structure of the edges describe the processes that hypothetically led to the evolution of the TUs, e.g., biological processes of *speciation*, *gene duplication*, and *gene loss*;
- Commonly, each TU is described by a collection of finite domain properties, referred to as *characters*. In the formalization, $\mathcal{T} = (C, D, f)$ is the description of such properties, where
  - $C = \{c_1, \ldots, c_k\}$ is a finite set of characters;
  - $D = (D_{c_1}, \ldots, D_{c_k})$ associates a finite domain to each character;
  - $f : L \times C \to \bigcup_{c \in C} D_c$ is a function that provides the value of each character for each TU being studied.
- We are often interested in the length of the branches of a phylogeny and/or the assignment of *dates* to the internal nodes of the phylogeny; if this feature is present, then we will describe it as a function $\mathcal{L} : E \to \mathbb{R}^+$.

Whenever we do not have information about the length of the branches, we omit the component $\mathcal{L}$ from the description of the phylogeny.

For presentation simplicity, we focus on one example with macroscopic observable properties. Fig. 1 (left) shows a phylogenetic tree for the TUs $L = \{\mathit{Mollusca}, \mathit{Annelida}, \mathit{Arthopoda}, \mathit{Echinodermata}, \mathit{Chordata}\}$. In this example, the set of characters is $C = \{\mathit{Coelom}, \mathit{Dark}\}$—*Coelom* denotes the presence/absence of coelom (a body cavity between the intestine and the body walls), while *Dark* denotes the phenotypical character of having dark color. In this example, these are both binary characters, i.e., $D_{\mathit{Coelom}} = D_{\mathit{Dark}} = \{0, 1\}$. The function f describing the five TUs is given by the table underneath each TU in the figure—e.g., $f(\mathit{Annelida}, \mathit{Coelom}) = 0$ and $f(\mathit{Annelida}, \mathit{Dark}) = 0$.

The key point in the phylogenetic tree reconstruction problem is how to define what does it mean to "accurately describe", i.e., what measure of accuracy is used to compare plausible trees. A variety of measures have been proposed, and various phylogenetic reconstruction methods have been proposed based on the specific measure being used to assess quality of the phylogeny. A common method

used in deriving phylogenies is based on the idea of *character compatibility*—a principle derived from Le Quesne's idea of uniquely derived characters [21, 22].

The intuitive idea of compatibility is as follows: a character c is compatible with a phylogeny if the TUs that present the same value for such character are connected by a subtree within the phylogeny. More formally, given a phylogeny $\mathcal{P} = (V, E, L, \mathcal{T}, \mathcal{L})$, with $\mathcal{T} = (C, D, f)$, a character $c \in C$ is compatible with $\mathcal{P}$ if there is a mapping $h_c : V \to D_c$ such that:

- For each $t \in L$ we have that $h_c(t) = f(t, c)$;
- For each $i \in D_c$, the projection of the graph (V, E) on the set of nodes $V_i^c = \{t \in V \mid h_c(t) = i\}$ has a subgraph that has V_i^c as nodes and it is a rooted tree.

A character that is not compatible with a phylogeny $\mathcal{P}$ is said to be *incompatible*. The above (sub-tree) requirement implicitly states that when a character changes (during evolution) it never goes back to the previous value. This is referred to as the *Camin-Sokal* requirement; moreover, it also accounts for the requirement that the "change" occurs in a unique place, known as the *Dollo* requirement.

In the example of Fig. 1, the character *Coelom* is compatible with the given phylogeny—as shown in Fig. 1(middle). On the other hand, the character *Dark* is not compatible with this phylogeny (as shown in Fig. 1(right)).

The goal, in phylogeny reconstruction, is to determine a phylogeny that maximizes the number of characters that are compatible with it. This problem has been often referred to as the *k-incompatibility problem* [11]. Formally, the k-incompatibility problem is the problem of deciding, given a set L of TUs, a character description $\mathcal{T} = (C, D, f)$ of L, and an integer $n \in \mathbb{N}$, whether there is a phylogeny $(V, E, L, \mathcal{T})$ that has at most k incompatible characters.

2.2 Haplotype Inference

The differences between two organisms of the same species are derived from differences in some peculiar points of their DNA sequences. We present here the problem of reconstructing the connection between a set of *diploid* organisms (such as humans), given some information about such specific DNA locations.

The DNA of diploid organisms is organized in pairs of not completely identical copies of *chromosomes*. The sequence of nucleotides from a single copy is called *haplotype*, while the conflation of the two copies constitutes a *genotype*. Each person inherits one of the two haplotypes from each parent. The most common variation between two haplotypes is a difference in a single nucleotide. Using statistical analysis within a population, it is possible to describe and analyze the typical points where these mutations occur. Each of such differences is called a *Single Nucleotide Polymorphism (SNP)*. In other words, a SNP is a single nucleotide site, in the DNA sequence, where more than one type of nucleotide (usually two) occur with a non-negligible population frequency. We refer to such sites as *alleles*.

Considering a specific genotype, a SNP site where the two haplotypes have the same nucleotide is called an *homozygous* site, while it is *heterozygous* otherwise. Research has confirmed that SNPs are the most common and predominant

form of genetic variation in DNA. Moreover, SNPs can be linked to specific traits of individuals and with their phenotypic variations within their population. Consequently, haplotype information in general, and SNPs in particular, are relevant in several contexts, such as, for instance, in the study and diagnosis of genetic diseases, in forensic applications, etc. This makes the identification of the haplotype structure of individuals, as well as the common part within a population, of crucial importance. In practice, biological experiments are used to collect genotype data instead of haplotype data, mainly due to cost or technological limitations. To overcome such limitations, accurate computational methods for inferring haplotype information from genotype data have been developed during the last decades (for a review, the reader is referred to [17, 15, 16]).

Modeling. The haplotype inference problem can be formulated as follows. First, we apply an abstraction and represent genotypes and haplotypes by focusing on the collection of ambiguous SNPs sites in a population. Moreover, let us denote, for each site, the two possible alleles using 0 and 1, respectively. Hence, an haplotype will be represented by a sequence of n components taken from $\{0, 1\}$. Each genotype g, being a conflation of two (partially) different haplotypes h_1 and h_2, will be represented as a sequence of n elements taken from $\{0, 1, 2\}$, such that 0 and 1 are used for its *homozygous* sites, while 2 is used for the *heterozygous* sites. More specifically, following [20], let us define the conflation operation $g = h_1 \oplus h_2$ as follows:

$$g[i] = \begin{cases} h_1[i] & \text{if } h_1[i] = h_2[i] \\ 2 & \text{otherwise} \end{cases}$$

where $g[i]$ denotes the i^{th} element of the sequence g, for $i = 1, \ldots, n$.

We say that a genotype g is *resolved* by a pair of haplotypes h_1 and h_2 if $g = h_1 \oplus h_2$. A set H of haplotypes *explains* a given set G of genotypes, if for each $g \in G$ there exists a pair of haplotypes $h_1, h_2 \in H$ such that $g = h_1 \oplus h_2$.

Given a set G of m genotypes, the *haplotype inference problem* consists of determining a set H of haplotypes that explains G. The cardinality of H is bound by $2m$ but, in principle, each genotype having $k \leq n$ ambiguous sites, can be explained by 2^{k-1} different pairs of haplotypes. For instance, the singleton $G = \{212\}$ (i.e., $k = 2$) can be explained in two ways, namely by choosing $H = \{011, 110\}$ or $H = \{010, 111\}$ (see also Fig. 2). Hence, in general, there might be an exponential number of explanations for a given set G. All of them are, from the combinatorial point of view, "equivalent" and a blind algorithm—not exploiting any biological insights—may result in inaccurate, i.e., biologically improbable, solutions. What is needed is a genetic model of haplotype evolution to guide the algorithm in identifying the "right" solution(s).

Several approaches have been proposed, relying on the implicit or explicit adoption of assumptions reflecting general properties of an underlying genetic model. We focus on one of such formulations, namely *parsimony*. The main underlying idea is the application of a variant of *Ockham's principle of parsimony:* the *minimum-cardinality* possible set H of haplotypes is the one to be chosen as explanation for a given set of genotypes G. For instance the set G in Fig. 2

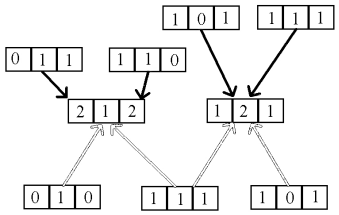


Fig. 2. The set $G = \{212, 121\}$ and two possible explanations

admits two explanations. The one at the bottom, i.e., $\{010, 111, 101\}$, is preferable by the parsimony principle. In this formulation, the haplotype inference problem has been shown in [20] to be APX-hard, through a reduction from the node-covering problem.

3 Methods

The basic idea is to use ASP to model the genome analysis. In particular, as first approximation of the problem, we focus on mutations that took place in specific locations of the DNA (Single Nucleotide Polymorphism). These mutations are tracked at different moments in time for the same individual and tissue, opposed to traditional techniques that search for these mutations across a large set of individuals. Since the data is enriched by time information, it is possible to integrate haplotype search with phylogenetic structure of tumoral fingerprints. In fact, cell offspring relationships are strongly related to an evolutionary tree for species. In our case, it is possible to model different snapshots of the genome at different points in time, and correlate mutations over time as in the classical phylogenetic inference. The algorithms for the construction of a phylogenetic tree need to be modified to capture the evolutionary properties of the various genomes collected from the same patient. Similar approaches have appeared in the literature (e.g., [13]), though not based on logic programming. The goal is to use the combination of haplotyping and phylogenetic tree reconstruction to reconstruct the mutations over time, and provide an evolutionary map of cancer haplotypes. The ASP framework allows us to prototype the models and have a fast feedback about their quality.

3.1 Modeling

The *evolutionary haplotype inference problem* can be formulated by extending the formalization presented for the haplotype inference problem. We define a linear timeline $T = t_0, t_1, \ldots, t_{k-1}$, whose time-steps are associated to each input genotype. Formally a *timed genotype* is a pair (g, t_i) made of a genotype g and a time-step $t_i \in T$. A *timed haplotype* is a haplotype associated to a time step: formally (h, t_i) where h is a haplotype and $t_i \in T$.

We say that a timed genotype (g, t_i) is *resolved* by a pair of timed haplotypes (h_1, t_j) and (h_2, t_k) if $g = h_1 \oplus h_2$, $t_i \geq t_j$ and $t_i \geq t_k$. A set H of timed haplotypes *explains* a given set G of timed genotypes, if for each $g \in G$ there exists a pair of timed haplotypes such that they resolve g.

We need to introduce the notion of haplotype *persistence*: given a set H of timed haplotypes, $(h, t_i) \in H$ is persistent if for any t_j, such that $t_i \leq t_j$, $(h, t_j) \in H$. In other words, persistent haplotypes in H are defined at specific time-steps and they will explain any timed genotypes at times greater or equal to t_i. A set H of timed haplotypes is persistent if every haplotype in H is persistent.

The last notion we introduce is the *preference* over two persistent sets $H_1 \preceq H_2$. Intuitively, we prefer timed haplotypes that appear as late as possible: this reflects the fact that the occurrence of an haplotype cannot be delayed anymore and therefore captures some relevant properties in the timed genomes (e.g., consequences of a therapy). On the other hand, any haplotype at a certain time t_i can be introduced at previous time-steps, without violating any properties. Therefore, a preference that captures the late occurrence of haplotypes reflects a more accurate characterization of the set H. Note that any solution for the original haplotype inference problem can be extended to a timed haplotype solution by adding the time step t_0 to each haplotype.

Formally, given two persistent haplotypes $(h, t_i) \in H_1$ and $(h, t_j) \in H_2$, we say that $(h, t_i) \preceq (h, t_j)$ if $t_i \geq t_j$. We extend the preference to persistent haplotype sets: $H_1 \preceq H_2$ reflects the fact that the set H_1 is preferred to H_2, namely there is no pair $(h, t_i) \in H_1$ and $(h, t_j) \in H_2$ such that $(h, t_i) \not\preceq (h, t_j)$.

Given a set G of timed genotypes, the *evolutionary haplotype inference problem* consists of determining sets H of persistent timed haplotypes that explain G such that there is no other solution $H_1 \preceq H$.

This model introduces only time information to available genotype. It is possible to extend it to other facts that are annotated with the samples. For example, the clinical condition of the patient can provide information about therapies and other physiological parameters. The timed genotypes can be enriched by a tuple of properties that could help in the comparison between solutions of evolutionary haplotype inference problem for different patients. This information can be retrieved from public/controlled access databases (see, e.g., the cancer genome atlas cancergenome.nih.gov).

4 Conclusion

In this work-in-progress paper, we briefly discussed the initial modeling of the evolutionary haplotype inference problem; the problem is tied to investigation of genome evolution in cancer (e.g., as result of pharmacological interventions). The problem is combinatorial in nature, and suitable for modeling and analysis using logic programming techniques. The project is in its infancy and will proceed through the integration of the proposed haplotype inference with techniques to reconstruct an associate evolutionary tree (with techniques borrowed from phylogenetic analysis).

References

1. P. Barahona, L. Krippahl, and O. Perriquet. Bioinformatics: A challenge to constraint programming. *Hybrid Optimization*, 45:463–487, 2011.
2. Federico Campeotto, Agostino Dovier, Ferdinando Fioretto, and Enrico Pontelli. A GPU implementation of large neighborhood search for solving constraint optimization problems. In Proc of *ECAI* 2014 - 21st European Conference on Artificial Intelligence, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 189–194. IOS Press, 2014.
3. Federico Campeotto, Agostino Dovier, and Enrico Pontelli. A declarative concurrent system for protein structure prediction on GPU. *J. Exp. Theor. Artif. Intell.*, 27(5):503–541, 2015.
4. B. Chisham, B. Wright, T. Le, T. Son, and E. Pontelli. CDAO-Store: Ontology-driven Data Integration for Phylogenetic Analysis. *BMC Bioinformatics*, 12:98, 2011.
5. Alessandro Dal Palù, Agostino Dovier, Andrea Formisano, and Enrico Pontelli. CUD@SAT: SAT solving on GPUs. *J. Exp. Theor. Artif. Intell.*, 27(3):293–316, 2015.
6. A.J. Dobson. Lexicostatistical grouping. *Anthropological Linguistics*, 11:216–221, 1969.
7. Agostino Dovier, Andrea Formisano, Enrico Pontelli, and Flavio Vella. A GPU implementation of the ASP computation. In Marco Gavanelli and John H. Reppy, editors, *Proc of Practical Aspects of Declarative Languages - 18th International Symposium, PADL 2016,*, volume 9585 of *Lecture Notes in Computer Science*, pages 30–47. Springer, 2016.
8. J.A. Eisen and P.C. Hanawalt. A phylogenomic study of DNA repair genes, proteins, and processes. *Mutation Research - DNA Repair*, 435(3):171–213, 1999.
9. Esra Erdem. Applications of answer set programming in phylogenetic systematics. In Marcello Balduccini and Tran Cao Son, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, volume 6565 of *Lecture Notes in Computer Science*, pages 415–431. Springer, 2011.
10. Esra Erdem, Ozan Erdem, and Ferhan Türe. HAPLO-ASP: Haplotype inference using answer set programming. In Esra Erdem, Fangzhen Lin, and Torsten Schaub, editors, *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings*, volume 5753 of *Lecture Notes in Computer Science*, pages 573–578. Springer, 2009.
11. G.F. Estabrook. Ancestor-descendant relations and incompatible data: Motivation for research in discrete mathematics. In *Mathematical Hierarchies and Biology*, volume 27 of *DIMAS Series in Discrete Mathematics*, pages 1–28. American Mathematical Society, 1997.
12. H.A. Gleason. Counting and calculating for historical reconstruction. *Anthropological Linguistics*, 1:22–32, 1959.
13. C. Greenman et al. Estimation of rearrangement phylogeny for cancer genomes. *Genome Res.*, 22(2):346–361, 2012.
14. T. Gruber. Toward principles for the design of ontologies used for knowedge sharing. *International Journal of Human Computer Studies*, 43(5-6), 1995.
15. Dan Gusfield. An overview of combinatorial methods for haplotype inference. In Istrail et al. [19], pages 9–25.
16. Dan Gusfield and Steven Hecht Orzack. Haplotype inference. In Srinivas Aluru, editor, *Handbook of Computational Molecular Biology*, Computer & Information Science, chapter 18. Chapman & Hall/CRC, 2005.

17. Bjarni V. Halldórsson, Vineet Bafna, Nathan Edwards, Ross Lippert, Shibu Yooseph, and Sorin Istrail. A survey of computational methods for determining haplotypes. In Istrail et al. [19], pages 26–47.
18. S Horne, C Ye, B Abdallah, G Liu, and H Heng. Cancer genome evolution. *Transl Cancer Res*, 4(3):303–313, 2015.
19. Sorin Istrail, Michael S. Waterman, and Andrew G. Clark, editors. *Computational Methods for SNPs and Haplotype Inference, DIMACS/RECOMB Satellite Workshop, Piscataway, NJ, USA, November 21-22, 2002, Revised Papers*, volume 2983 of *Lecture Notes in Computer Science*. Springer, 2004.
20. Giuseppe Lancia, Maria Cristina Pinotti, and Romeo Rizzi. Haplotyping populations by pure parsimony: Complexity of exact and approximation algorithms. *INFORMS Journal on Computing*, 16(4):348–359, 2004.
21. W.J. Le Quesne. A Method of Selection of Characters in Numerical Taxonomy. *Syst. Zool.*, 18:201–205, 1969.
22. W.J. Le Quesne. Further Studies Based on the Uniquely Derived Character Concept. *Syst. Zool.*, 21:281–288, 1972.
23. D.A. Liberles and M.L. Wayne. Tracking adaptive evolutionary events in genomic sequences. *Genome Biol.*, 3(6), 2002.
24. Victor W. Marek and Miroslaw Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm*, pages 375–398. Springer Verlag, 1999.
25. N. Moore and P. Prosser. The ultrametric constraint and its application to phylogenetics. *J. Artif. Intell. Res.*, 32:901–938, 2008.
26. Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999.
27. J. D. Owens et al. GPU computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.
28. A. Dal Palù, A. Dovier, A. Formisano, and E. Pontelli. Exploring Life through Logic Programming: Answer Set Programming in Bioinformatics. In Michael Kifer and Annie Liu, editors, *Declarative Logic Programming: Theory, Systems, and Applications*. Springer, To appear, available as TR-CS-NMSU-2014-10-24 New Mexico State University.
29. O. Podlaha, M. Riester, S. De, and F. Michor. Evolution of the cancer genome. *Trends Genet.*, 28(4):155–163, 2012.
30. D. Ringe, T.Warnow, and A. Taylor. Indo-European and computational cladistics. *Transactions of the Philological Society*, 100(1):59–129, 2002.
31. E.A. Stone and A. Sidow. Physicochemical constraint violation by missense substitutions mediates impairment of protein function and disease severity. *Genome Res.*, 15(7):978–986, 2005.
32. J.L. Thorne. Models of protein sequence evolution and their applications. *Curr. Opin. Genet. Dev.*, 10(6):602–605, 2000.
33. E.R. Tillier, L. Biro, G. Li, and D. Tillo. Codep: maximizing co-evolutionary interdependencies to discover interacting proteins. *Proteins*, 63(4):822–831, 2006.
34. E.R. Tillier and T.W. Lui. Using multiple interdependency to separate functional from phylogenetic correlations in protein alignments. *Bioinformatics*, 19(6):750–755, 2003.
35. C.R. Woese and N.R. Pace. Probing RNA Structure, Function, and History by Comparative Analysis. In *The RNA World*, pages 91–117. Cold Spring Harbor Laboratory Press, 1993.

Answer Set Programming for Logical Analysis of Data

Katinka Becker[1], Martin Gebser[2], Torsten Schaub[2], and Alexander Bockmayr[1]

[1] Freie Universität Berlin, FB Mathematik und Informatik, Arnimallee 6,
D-14195 Berlin, Germany
{katinka.becker,alexander.bockmayr}@fu-berlin.de
[2] Universität Potsdam, Institut für Informatik, August-Bebel-Str. 89,
D-14482 Potsdam, Germany
{gebser,torsten}@cs.uni-potsdam.de

Abstract. New experimental technologies in medicine and biology lead to the availability of increasingly large data sets. Extracting information from those data sets is a major challenge in current life sciences. Logical Analysis of Data (LAD) is a method combining ideas from optimization and combinatorics in order to explore large data sets by deriving logical patterns with certain properties. These patterns can be seen as building blocks of implicit information. In the case of biological data, their examination can serve as a step towards a deeper understanding of the underlying biology. We present a new approach to LAD using Answer Set Programming and demonstrate how it can be used for biological applications.
Keywords: Answer Set Programming, Logical Analysis of Data, Patterns, Classification

1 Introduction

Logical Analysis of Data (LAD) is a method for data analysis using concepts from machine learning, combinatorics and Boolean functions. It was first described by Peter L. Hammer, Yves Crama and Toshihide Ibaraki in 1988 [6] and further developed in the following decades. LAD is a method for data analysis that aims for an understanding of the central information carried by a data set. Our aim is to apply this approach to biological data sets such as phosphorylation measurements of protein networks or gene expression data. To make use of the method, we develop a tool box for LAD and implement it by means of Answer Set Programming (ASP) [13], a form of declarative programming based on the stable model semantics of logic programming.

The paper is organized as follows. Sect. 2 gives an introduction to the LAD method and describes the main steps of data analysis, namely the generation of patterns and the formation of a theory. The basic ASP methodology is described in Sect. 3. In Sect. 4, we present our implementation, and Sect. 5 illustrates how it can be used for biological applications. Sect. 6 concludes the paper and points out future directions.

2 Logical Analysis of Data

Logical Analysis of Data (LAD) [6] is a methodology for data analysis which combines concepts from combinatorics and Boolean functions as well as from machine learning and optimization. LAD finds applications in various research areas and attracted particular interest in the biomedical field [1].

x_0	x_1	x_2	x_3	x_4	x_5	
1	1	0	1	1	1	Ω^+
1	0	0	0	1	1	
1	1	1	1	0	1	
1	1	1	1	0	0	
1	1	0	0	1	0	
1	0	0	1	0	1	
1	1	1	1	1	1	
0	1	0	1	0	0	Ω^-
0	1	0	0	0	1	
0	0	0	1	1	0	

Table 1. Binary data set Ω partitioned into positive observations Ω^+ and negative observations Ω^- by decision variable x_0

2.1 Basic concepts and notations

Originally LAD was designed for binary data sets. Let $\Omega \subseteq \{0,1\}^n$ be a set of *observations* that are divided into two subsets by a *decision variable* x_0, so that $\Omega = \Omega^+ \uplus \Omega^-$ is the disjoint union of Ω^+ and Ω^-, called *positive* and *negative observations*, respectively (see Tab. 1). Note that the decision variable could be any of the given variables, each leading to a corresponding division.

A *Boolean function* of $n \in \mathbb{N}$ variables is a mapping $f\colon \{0,1\}^n \to \{0,1\}$. A data set Ω as described above can be seen as a *partially defined Boolean function* (pdBf), meaning that the function f is only given for $\Omega = \Omega^+ \uplus \Omega^-$. Every function $e\colon \{0,1\}^n \to \{0,1\}$ with $e(x) = f(x)$ for all $x \in \Omega$ is called an *extension* of f. A major goal of LAD is to find such an extension for a given pdBf.

2.2 Patterns

The key concept of LAD are patterns consisting of literals. For a Boolean variable x, we denote its negation by $\overline{x}$. Both x and $\overline{x}$ are *literals*. A *term* is a conjunction or product of literals. The *degree* of a term is the number of literals in it. Every term t can be seen as a Boolean function. For a vector $v \in \{0,1\}^n$, we denote by $t(v)$ the binary value that results from applying the Boolean function t to v, where $t(v)$ is defined even if the degree of t is less than n. We say that a term t *covers* a point $v \in \{0,1\}^n$ if $t(v) = 1$.

Definition 1 (Positive (negative) pattern). *Let f be a pdBf defined on a set $\Omega = \Omega^+ \uplus \Omega^-$ of observations. A term t is called a* positive (negative) pattern *of f if it covers at least one positive (negative) observation and no negative (positive) observation.*

Note that every positive observation corresponds to a positive pattern, and every negative observation to a negative pattern. Given the data set in Tab. 1 (partitioned by x_0), the observations $x_1\overline{x_2}x_3x_4x_5$ and $x_1\overline{x_2}x_3\overline{x_4}\,\overline{x_5}$ thus provide a positive and a negative pattern, respectively. Other (arbitrary) examples of positive patterns include $\overline{x_1}\,\overline{x_3}$, x_2 and x_4x_5. For instance, x_4x_5 covers the positive observations $x_1\overline{x_2}x_3x_4x_5$,

$\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4x_5$ and $x_1x_2x_3x_4x_5$, while it evaluates to 0 for the remaining observations, particularly considering the three which are negative.

Various types of patterns have been studied and their relative efficiency has been analyzed [10]. A pattern can be preferred to another pattern based on different criteria.

Definition 2 (Simplicity preference). *A pattern P_1 is* simplicity-wise preferred *to a pattern P_2 if and only if the set of literals in P_1 is contained in the set of literals in P_2.*

Definition 3 (Selectivity preference). *A pattern P_1 is* selectivity-wise preferred *to a pattern P_2 if and only if the set of literals in P_2 is contained in the set of literals in P_1.*

Definition 4 (Evidential preference). *A pattern P_1 is* evidentially preferred *to a pattern P_2 if and only if the set of observations covered by P_1 includes the observations covered by P_2.*

Based on these preferences, various types of patterns can be described and analyzed. A special type of patterns of particular interest are *prime patterns* [1].

Definition 5 (Prime patterns). *A pattern with an inclusion-wise minimal set of literals is called* prime*.*

In other words, a pattern is prime if the removal of any of its literals results in a term which is not a pattern anymore. For the data set in Tab. 1, the positive pattern x_2 clearly is a prime pattern as it consists of one literal only. On the other hand, $x_3x_4x_5$ is not prime because x_3 can be removed to obtain the smaller positive pattern x_4x_5. This positive pattern x_4x_5 is prime given that neither x_4 nor x_5 is a positive pattern by itself.

2.3 Non-pure patterns - Homogeneity and prevalence

The development of patterns in LAD can be seen as the development of a simple language which can easily be understood and communicated across various fields of applications. Talking about applications apart from purely mathematical problems, one is always confronted with data sets which are not perfect in the sense of having one unique explanation. There might be errors coming from experiments or discretization. Taking this into account, parameters have been introduced to allow for some latitude in the definition of patterns. Two of them which we will use in the sequel are called *homogeneity* and *prevalence* [2].

Definition 6 (Homogeneity). *The* homogeneity $Hom^+(P)$ *of a positive pattern P is given by*

$$Hom^+(P) = \frac{Cov_{pos}(P)}{Cov(P)}, \tag{1}$$

where $Cov_{pos}(P)$ is the number of positive observations covered by P and $Cov(P)$ is the number of observations covered in total. The homogeneity $Hom^-(P)$ of a negative pattern P is defined analogously.

Definition 7 (Prevalence). *The* prevalence $Prev^+(P)$ *of a positive pattern* P *is given by*

$$Prev^+(P) = \frac{Cov_{pos}(P)}{|\Omega^+|}. \tag{2}$$

The prevalence $Prev^-(P)$ *of a negative pattern* P *is defined analogously.*

2.4 Theories and predictions

As mentioned before, a goal of LAD is to find a suitable extension e for a given pdBf f. Such extensions are called *theories* in LAD and allow us to make predictions for unobserved vectors.

In practice, there are several steps on the way to building a theory where decisions have to be made and different concepts have been proposed (e.g. [4]). The first step is to select a representative subset of patterns. On the one hand, this subset should be large enough to capture all features of the data set. On the other hand, the subset should not be too large, because it might become hard to understand and might lead to uncertain classifications. To ensure that positive observations are assigned to be positive by the theory and negative observations are assigned to be negative, the subset of patterns is chosen such that every positive (negative) observation is covered by at least one positive (negative) pattern. An observation is classified as positive (negative) if it is covered by some of the positive (negative) patterns in the theory and by no negative (positive) pattern.

To classify observations that are covered by positive and negative patterns, LAD constructs a *discriminant* that assigns relative weights to the patterns. The discriminant Δ for $v \in \{0,1\}^n$ is given by

$$\Delta(v) = \sum_k w_k^+ P_k^+(v) + \sum_l w_l^- P_l^-(v), \tag{3}$$

where $P_1^+, \ldots, P_k^+$, $k \in \mathbb{N}$ are the positive patterns with their assigned positive weights $w_1^+, \ldots, w_k^+$ and $P_1^-, \ldots, P_l^-$, $l \in \mathbb{N}$ are the negative patterns with negative weights $w_1^-, \ldots, w_l^-$. For classification, a threshold t has to be chosen. A vector v is classified as positive if $\Delta(v) > t$ and negative if $\Delta(v) \leq t$.

3 Answer Set Programming

Answer Set Programming (ASP) [13] is a declarative programming paradigm. In contrast to imperative programming, the task for the user is to give a detailed description of *what the problem is* rather than explaining *how the problem should be solved*. The implementation of a problem in ASP consists thus of a concise representation of the problem by logical rules, which are then instantiated by an ASP grounder and solved by an ASP solver.

Various systems for grounding and solving were developed in the past years. We use *clingo*, a combination of the grounder *gringo* and the solver *clasp* [8]. In the following, we give a short introduction into the syntax and semantics of ASP.

ASP is based on the stable model semantics of logic programming [13]. Search problems are reduced to the computation of stable models which are found by ASP solvers. Problems are formulated as logic programs which are finite sets of rules. A *rule* r is of the form

$$A_0 \text{ :- } A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n, \tag{4}$$

where $n \geq m \geq 0$, each A_i, $0 \leq i \leq n$, is an *atom* and 'not' stands for negation by default. We call the left side of the rule the *head* and the right side the *body*. In the following, we explain the input language used by ASP systems [8]. A rule, as in (4), is a conditional constraint, meaning that the head must be true if the body is true. If $n = 0$, rule (4) is called a *fact* and denoted by

$$A_0.$$

Such a fact expresses that the atom A_0 is always true. Omitting A_0 in (4) amounts to taking A_0 to be false, and rule (4) represents an *integrity constraint*. Accordingly, the resulting rule

$$\text{:- } A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n.$$

expresses that a stable model must not satisfy the body. Integrity constraints are thus often used to eliminate model candidates of a program.

To facilitate the use of ASP in practice, several extensions have been developed. First of all, rules with variables are viewed as shorthands for the set of their ground instances. Further language constructs include *conditional literals* and *cardinality constraints* [16]. The former are of the form $A : B_1, \ldots, B_m$, the latter can be written as $s\ \{C_1; \ldots; C_n\}\ t$, where A and B_i are possibly default negated literals and each C_j is a conditional literal; s and t provide lower and upper bounds on the number of satisfied literals in a cardinality constraint. The practical value of both constructs becomes apparent when used in conjunction with variables. For instance, a conditional literal like $a(X) : b(X)$ in a rule's body expands to the conjunction of all instances of $a(X)$ for which the corresponding instance of $b(X)$ holds. Similarly, $2\ \{a(X) : b(X)\}\ 4$ holds whenever between two and four instances of $a(X)$ (subject to $b(X)$) are true. In addition to cardinality constraints (using `#count` in full detail), the input language of ASP provides further aggregates such as `#min`, `#max` and `#sum`. Similarly, objective functions minimizing the sum of weights w_i of literals B_i are expressed as `#minimize` $\{w_1 : B_1; \ldots; w_n : B_n\}$. Specifically, we rely in the sequel on the input language of the ASP system *clingo* [8], as detailed in the corresponding user's guide [7].

The first step in the process of finding a solution to a problem is the grounding (e.g. by *gringo*) of rules that include variables, meaning that those variables are replaced by constants in ground instances. The grounded program is then passed to the solver (e.g. *clasp*) which computes the stable models of the program.

Definition 8 (Reduct). *The* reduct P^X *of a program* P *relative to a set* X *of atoms is defined by*

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

```
1  i(1,1,1,1). i(1,1,2,0). i(1,1,3,1). i(1,1,4,1). i(1,1,5,1).
2  i(1,2,1,0). i(1,2,2,0). i(1,2,3,0). i(1,2,4,1). i(1,2,5,1).
3  i(1,3,1,1). i(1,3,2,1). i(1,3,3,1). i(1,3,4,0). i(1,3,5,1).
4  i(1,4,1,1). i(1,4,2,1). i(1,4,3,1). i(1,4,4,0). i(1,4,5,0).
5  i(1,5,1,1). i(1,5,2,0). i(1,5,3,0). i(1,5,4,1). i(1,5,5,0).
6  i(1,6,1,0). i(1,6,2,0). i(1,6,3,1). i(1,6,4,0). i(1,6,5,1).
7  i(1,7,1,1). i(1,7,2,1). i(1,7,3,1). i(1,7,4,1). i(1,7,5,1).
8  i(0,8,1,1). i(0,8,2,0). i(0,8,3,1). i(0,8,4,0). i(0,8,5,0).
9  i(0,9,1,1). i(0,9,2,0). i(0,9,3,0). i(0,9,4,0). i(0,9,5,1).
10 i(0,10,1,0).i(0,10,2,0).i(0,10,3,1).i(0,10,4,1).i(0,10,5,0).
```

Fig. 1. Input instance derived from the data set given in Tab. 1

where $body^+(r)$ is the set of all positive atoms of the body and $body^-(r)$ is the set of all negative atoms of the body.

Definition 9 (Stable Model). *A set X of atoms is a* stable model *of a program P if the inclusion-wise minimal model of the reduct P^X of P relative to X is equal to X.*

4 Implementation of LAD in ASP

Over the past decades several implementations of LAD have been developed. These include a `C++` tool implemented by Mayoraz [14], the `LAD-WEKA` software by Bonates and Gomes [3] written in `WEKA`, which is a data mining software package of `Java`, and `ladoscope` by Lemaire [12] in `OCaml`. Most of these tools are no longer maintained. Although some of them like `ladoscope` offer a wide range of functions, new functionalities of LAD might be of interest, which are hard to add to the existing code.

Our goal is to develop an implementation of LAD which is as clear and succinct as possible. This makes ASP a natural choice. The structure of the problem, i.e., enumerating patterns of Boolean expressions, is another important reason for choosing the ASP framework. In this section, we describe our encoding of pattern generation and theory formation. For alternative approaches using Mixed-Integer Linear Programming (MILP), we refer e.g. to [15, 9].

An instance of our problem is a binary data set with the properties defined in Sect. 2 (see Tab. 1). It is parsed into the format given in Fig. 1. Each entry of the table is given by a single predicate `i(sign,name,variable,value)`, where `sign` is the sign of the observation (1 for positive and 0 for negative observation), `name` is the identifier of the observation and `value` is the binary value of a `variable`.

4.1 Pattern generation

The basic encoding for pattern generation is given in Fig. 2. Before running the program the user has to define the constants `degree`, `sign`, `homogeneity` and `prevalence`. A stable model provides a pattern of given degree and sign with homogeneity and prevalence greater than or equal to the chosen constants.

```
1  % GENERATE
2  degree { pat(S,B) : i(sign,_,S,B) } degree.
3
4  % DEFINE
5  not_covered(W,X) :- i(W,X,_,_), pat(S,B), not i(W,X,S,B).
6  covered(W,X) :- not not_covered(W,X), i(W,X,_,_).
7
8  % TEST
9  :- pat(S,B), pat(S,Q), Q<B.
10 :- #sum{ homogeneity-100,X : covered(W,X), W=sign;
11          homogeneity,X : covered(W,X), W!=sign } > 0.
12 :- nbrrightobs(C),
13    #sum{ 100,X : covered(W,X), W=sign } < prevalence*C.
```

Fig. 2. Basic encoding for pattern generation

The program follows the general methodology of ASP which means that it is organized in three parts, called *generate, define* and *test*. We want a stable model to contain a pattern of a given degree. Any stable model thus includes `degree` many atoms over literals `pat(S,B)`, where `S` is the variable name and `B` its Boolean value. The first rule given in line 2 is a choice rule generating solution candidates. We know from the definition of a pattern that it must cover at least one observation of its sign. Because of this constraint, we choose literals from the observations of the same sign.

At this point, it is not certain that the set of literals belongs to an actual pattern. It could be any set of single literals included in different observations of the right sign, but such that the whole set of literals is not covering a single observation. The *define* part is used to specify predicates narrowing stable models down. We want to count how many observations (divided by their sign) are covered by a set of atoms `pat(S,B)`. To do this, we introduce in line 6 the predicate `covered(W,X)` which is true if observation `X` having sign `W` is covered by the generated literals. For defining this predicate, we use an auxiliary predicate `not_covered(W,X)` in line 5 which is true for an observation `W` if one of the literals `pat(S,B)` does not appear in `W`. Then `covered(W,X)` is true if `not_covered(W,X)` is not true.

In the next part, indicated by *test*, we can make use of the defined predicate and test whether the choice of literals fulfills the definition of a pattern. Line 9 is a general test forbidding that a pattern contains the same variable with different assignments. In line 10 and 11, we test whether the set of literals fulfills the homogeneity condition (1). In addition, line 12 and 13 make sure that the prevalence condition (2) is met by a generated pattern. The unary predicate `nbrrightobs` is calculated in advance and counts the number of observations having the same sign as the pattern wanted.

4.2 Prime patterns

By adding only one more test to the encoding presented above, we can focus the patterns to generate on those which are prime. The encoding in Fig. 3 is based on the definition of prime patterns, namely that a pattern is prime if and only if the deletion of any of

```
1 % DEFINE
2 del_covered(W,X,S) :- i(W,X,S,_),
3           #count{ T : pat(T,C), i(W,X,T,C), T != S } = degree-1.
4
5 % TEST
6 :- pat(S,B),
7    #sum{ homogeneity-100,X : del_covered(W,X,S), W = sign;
8           homogeneity,X : del_covered(W,X,S), W != sign } <= 0.
```

Fig. 3. Additional lines to the encoding in Fig. 2 for the generation of prime patterns

its literals results in a term which is not a pattern. We determine the coverage for all terms obtained by the deletion of a single literal (lines 2-3) and test whether the term satisfies the homogeneity condition (lines 6-8). Note that it is not necessary to test the prevalence condition as a term obtained by leaving out one of the literals of a pattern cannot cover less observations than the original pattern.

4.3 Theory formation

The formation of a theory is based on the selection of a subset of patterns fulfilling certain properties. To this end, a set of (prime) patterns of interest is given by the predicate `pat(sign,name,variable,value)`, specifying the literals of candidate patterns similar to those of observations. The program in Fig. 4 encodes the task of choosing some of the patterns such that every positive observation is covered by at least `pos` many positive patterns and every negative observation is covered by at least `neg` many negative patterns, where `pos` and `neg` are user-defined constants. In the basic case, both constants are set to 1, while a greater value results in higher coverage of positive or negative observations, respectively. The `#minimize` statement in line 15 is used to choose a minimum-cardinality subset of patterns that covers the whole set of observations according to the specified constants.

Once a minimal pattern set has been found, it can be used to form a theory by assigning appropriate weights to each of the patterns. The resulting discriminant can then be applied to make predictions for binary vectors that do not belong to the given set of observations.

5 Biomedical Application

One of the main advantages of the analysis with LAD over other well-known machine learning methods, such as *Support Vector Machines* (see [17]), is the generation of patterns. They extract the important information from the given data and are easier to interpret than, e.g., hyperplanes.

The method as well as our implementation is independent from the context of the data and thus possible applications are widely spread. Here we give one example of how our implementation can be used in the biomedical field.

```
% GENERATE
{ minimalcover(W,M) } :- pat(W,M,_,_).

% DEFINE
not_all_entries_cov(W,M,X) :- i(W,X,_,_),
     pat(W,M,S,B), not i(W,X,S,B).
all_entries_covered(W,M,X) :- not not_all_entries_cov(W,M,X),
     minimalcover(W,M), i(W,X,_,_).

% TEST
:- i(1,X,_,_), #count{ M : all_entries_covered(1,M,X) } < pos.
:- i(0,X,_,_), #count{ M : all_entries_covered(0,M,X) } < neg.

% OPTIMIZE
#minimize{ 1,W,M : minimalcover(W,M) }.
```

Fig. 4. Encoding for finding a minimal pattern set that can be used for theory formation

5.1 Identifying protein interactions from phosphorylation measurements

Assume we are given a data set consisting of phosphorylation measurements of proteins in a cellular signaling network, as in Tab. 2. This example is taken from [11] and shows an extract of the perturbation measurements of the EGFR-signaling pathway after discretization. In Fig. 5 an idea of the known underlying protein network is given.

The phosphorylation of MEK, AKT, ERK and S6K was measured under different combinations of a stimulus at one of the growth factors TGF_α and IGF, and inhibitions at MEK or PI3K. In Tab. 2 the active stimulus is represented by a 1 entry and the absence of a stimulus is represented by a 0 entry. The same holds for the presence of an inhibition (1 entry) or its absence (0 entry).

We divided the table into positive and negative observations by the protein S6K. Note here that this could be any of the observed proteins. We now search the data set for prime patterns which explain the outcome of the phosphorylation of the downstream protein S6K. For simplicity, we restrict ourselves to patterns with perfect homogeneity and no constraint on their prevalence.

Positive prime patterns We first investigate the positive prime patterns. Recall that prime patterns are patterns such that the removal of any literal results in a non-pattern. We divide our analysis into patterns including a stimulus at TGF_α and those including a stimulus at IGF.

The search for positive prime patterns including TGF_α leads to a single solution, namely `pat(1,1)`, a degree-one prime pattern. The active stimulus at TGF_α itself suggests that a stimulus at TGF_α can lead to a phosphorylation of S6K independently from the use of additional inhibitions.

We then look for positive prime patterns including a stimulus at IGF. In total, we find four different prime patterns having this property, as reflected in the output of the ASP system:

TGF_α	IGF	MEK inh	PI3K inh	MEK	AKT	ERK	S6K
1	0	0	0	0	0	0	1
0	1	0	0	0	1	1	1
1	0	1	0	1	1	0	1
0	1	1	0	1	1	0	1
1	0	0	1	1	0	1	1
0	0	1	0	1	0	0	0
0	0	0	1	0	0	1	0
0	1	0	1	1	1	1	0

Table 2. Discretized phosphorylation measurements in the EGFR pathway [11]

```
Answer: 1
pat(2,1) pat(4,0)
Answer: 2
pat(2,1) pat(5,0)
Answer: 3
pat(2,1) pat(3,1)
Answer: 4
pat(2,1) pat(7,0)
```

We can nicely interpret those stable models. The fact that we do not find a degree-one pattern is due to the structure of the network. The stimulation of IGF does not guarantee that S6K is phosphorylated as the downstream can be cut by the inhibition of the pathway going through PI3K. Answer 2 illustrates the observation that we can have a positive readout at S6K while stimulating IGF without seeing a phosphorylation of MEK. Answer 4 is a similar observation saying that we do not have a phosphorylation of ERK when we see a phosphorylation of S6K under the given stimulus. Those two stable models indicate that MEK and ERK might not lie in the pathway from IGF to S6K. The same interpretation can be made regarding answer 3, which says that an inhibition at MEK does not prevent a phosphorylation of S6K. Answer 1 is the only stable model with a higher prevalence of 0.4, where the additional prohibition of the inhibition at PI3K represents the path from IGF along PI3K.

Negative prime patterns In the next step, we analyze the negative prime patterns of the data set in the same manner. As we never observe a negative readout at S6K when the stimulus at TGF_α is active, there is no negative pattern including TGF_α. There are also no degree-one negative prime patterns including IGF and only a single degree-two prime pattern which consists of `pat(2,1)` and `pat(4,1)`, standing for the active stimulus at IGF and the inhibition at PI3K.

6 Conclusion and Future Directions

We presented a new approach to Logical Analysis of Data using Answer Set Programming. The rich and simple declarative modeling language of ASP allows for solving

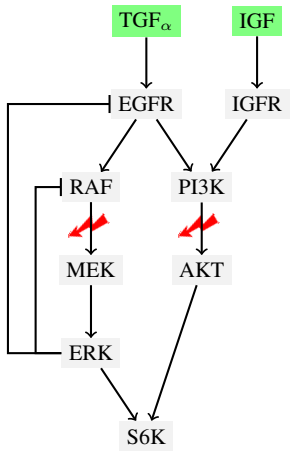


Fig. 5. The EGFR-signaling pathway [11]

hard search problems by the reduction to the computation of stable models. We made use of this advantage for the problem of pattern generation according to LAD. Because of the simple description of the problem, it is easy to communicate the solutions across research fields and to give an intuitive interpretation. Utilization of the ASP solver *clingo* [8] makes it possible to apply the program to large data sets and still obtain results in a reasonable amount of time.

In Sect. 5 we described one possible application in the biomedical field. We explained the application to discretized perturbation measurements, where we can see that prime patterns offer an opportunity for the deeper understanding of information carried in biological data sets. In fact, Logical Analysis of Data has been a powerful tool for data analysis especially in the biomedical field within the last decades. We believe that using ASP for LAD and possible extensions may greatly enhance its applicability.

References

1. Alexe, G., Alexe, S., Bonates, T., Kogan, A.: Logical analysis of data – the vision of Peter L. Hammer. *Annals of Mathematics and Artificial Intelligence*, 49(1):265–312, (2007)
2. Alexe, G., Hammer, P.: Spanned patterns for the logical analysis of data. *Discrete Applied Mathematics*, 154(7):1039–1049, (2006)
3. Bonates, T., Gomes, V.: LAD-WEKA tutorial. `https://lia.ufc.br/~tiberius/lad/` (2014)
4. Boros, E., Hammer, P., Ibaraki, T., Kogan, A., Mayoraz, E., Muchnik, I.: An implementation of logical analysis of data. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):292–306, (2000)
5. Chikalov, I., Lozin, V., Lozina, I., Moshkov, M., Nguyen, H., Skowron, A., Zielosko, B.: Logical analysis of data: Theory, methodology and applications. Three Approaches to Data Analysis: 147–192. Springer (2013)

6. Crama, Y., Hammer, P., Ibaraki, T.: Cause-effect relationships and partially defined Boolean functions. *Annals of Operations Research*, 16:299–326, (1988)
7. Gebser, M., Kaminski, R., Kaufmann, B., Lindauer, M., Ostrowski, M., Romero, J., Schaub, T., Thiele, S.: Potassco user guide. `http://potassco.sourceforge.net` (2015)
8. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124, (2011)
9. Guo, C., Ryoo H.: Compact MILP models for optimal and Pareto-optimal LAD patterns. *Discrete Applied Mathematics*, 160(16-17):2339–2348, (2012)
10. Hammer, P., Kogan, A., Simeone, B., Szedmak, S.: Pareto-optimal patterns in logical analysis of data. *Discrete Applied Mathematics*, 144(1-2):79–102, (2004)
11. Klinger, B., Sieber, A., Fritsche-Guenther, R., Witzel, F., Berry, L., Schumacher, D., Yan, Y., Durek, P., Merchant, M., Schäfer, R., Sers, C., Blüthgen, N.: Network quantification of EGFR signaling unveils potential for targeted combination therapy. *Molecular Systems Biology*, 9:673, (2013)
12. Lemaire, P.: Ladoscope. `http://www.kamick.org/lemaire/software.html` (2015)
13. Lifschitz, V.: What is answer set programming? Proceedings of AAAI'08: 1594–1597. AAAI (2008)
14. Mayoraz, E.: C++ tools for logical analysis of data. `http://rutcor.rutgers.edu/pub/LAD/man.pdf` (1998)
15. Ryoo, H., Jang, I.: MILP approach to pattern generation in logical analysis of data. *Discrete Applied Mathematics*, 157(4):749–761, (2009)
16. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, (2002)
17. Vapnik, V.: The Nature of Statistical Learning Theory. Springer (1995)

Identification of Bifurcations in Biological Regulatory Networks using Answer-Set Programming

Louis Fippo Fitime[1], Olivier Roux[1], Carito Guziolowski[1]§, and Loïc Paulevé[2]§

[1] LUNAM Université, École Centrale de Nantes, IRCCyN UMR CNRS 6597
(Institut de Recherche en Communications et Cybernétique de Nantes)
1 rue de la Noë – B.P. 92101 – 44321 Nantes Cedex 3, France.
[2] LRI UMR CNRS 8623, Univ. Paris-Sud – CNRS,
Université Paris-Saclay, 91405 Orsay, France

Abstract Aiming at assessing differentiation processes in complex dynamical systems, this paper focuses on the identification of states and transitions that are crucial for preserving or pre-empting the reachability of a given behaviour. In the context of non-deterministic automata networks, we propose a static identification of so-called bifurcations, i.e., transitions after which a given goal is no longer reachable. Such transitions are naturally good candidates for controlling the occurrence of the goal, notably by modulating their propensity. Our method combines Answer-Set Programming with static analysis of reachability properties to provide an under-approximation of all the existing bifurcations. We illustrate our discrete bifurcation analysis on several models of biological systems, for which we identify transitions which impact the reachability of given long-term behaviour. In particular, we apply our implementation on a regulatory network among hundreds of biological species, supporting the scalability of our approach.

1 Introduction

The emerging complexity of dynamics of biological networks, and in particular of signalling and gene regulatory networks, is mainly driven by the interactions between the species, and the numerous feedback circuits it generates [43,31,35,30]. One of the prominent and fascinating features of cells is their capability to differentiate: starting from a multi-potent state (for instance, a stem cell), cellular processes progressively confine the cell dynamics in a narrow state space, an attractor. Deciphering those decisional processes is a tremendous challenge, with important applications in cell reprogramming and regenerative medicine.

Discrete models of network dynamics, such as Boolean and multi-valued networks [42,4], have been designed with such an ambition. These frameworks model nodes of the network by variables with small discrete domains, typically

§ Corresponding authors: carito.guziolowski@irccyn.ec-nantes.fr, loic.pauleve@lri.fr

Boolean. Their value changes over time according to the state of their parent nodes. Exploring the dynamical properties of those computational models, such as reachability (the ability to evolve to a particular state) or attractors (long-run behaviors), allows to understand part of important cellular processes [36,1,5].

In classical control theory the reachability and safety control of discrete systems is an important topic which is both critical and challenging. Safety verification or reachability analysis aims at showing that starting from some initial conditions a system cannot evolve towards to some unsafe region in the state space. In a stochastic setting the different trajectories originating from one initial state have a different likelihood and one can then evaluate what is the probability that the system reaches the assigned set of states starting from a given initial distribution of the set of initial states. In safety problems, where the evolution of the system can be influenced by some control input, one should select it appropriately so as to minimize the probability that the state of the system will move to an unsafe set of states. In this work we address the computation of sets of states from which the system can evolve into an undesired set of states given a model represented as a discrete finite-state of interacting components, such as an Automata Network. This topic is in particular highly relevant for systems biology and systems medicine, since it can suggest intervention sites or therapeutic targets in Biological Regulatory Networks (BRNs) that may counteract pathological behavior.

Contributions. In this work we introduce the notion of *bifurcation* in Automata Networks (ANs) and provide a scalable method for their identification relying on declarative programming with Answer-Set Programming (ASP) [3]. A bifurcation corresponds to a transition after which the system looses the capability to reach a given goal state. Identifying bifurcations extensively relies on the reachability problem, which is PSPACE-complete in ANs and related frameworks [8]. In order to obtain an approach tractable on large biological networks, we show how to combine techniques from static analysis of ANs dynamics, from concurrency, and from constraint programming in order to relax efficiently the bifurcation problem. Our method identifies correct bifurcations only (no false positives), but, due to the embedded approximations, is incomplete (false negatives may exist). To our knowledge, this is the first integrated method to extract such decisive transitions from models of interaction networks.

Related work. Many works have been devoted to the control theory and system verification; in particular in reachability analysis and safety control for system verification. We limit here to those developed for discrete systems.

In the case of deterministic systems, a large number of methods have been proposed to aid in the verification of safety-critical computer software and hardware. Due to the inherently discrete nature of these systems, early attempts in this area have concentrated on purely discrete systems [6]. A verification procedure has been proposed for discrete event dynamic systems in [25]. They use a finite state machine representation to model discrete event dynamic systems, temporal logic statements (to represent specifications imposed on their operations) and a model-checking verification method introduced by Clarke et al. [10]

to test if the given relationship between the model and the specifications holds. The same method has been applied to the verification of programmable logic controllers [27] and control procedures for batch plants [26].

For systems with complex behavior, the development of modeling frameworks and verification methods has been done among others by [20]. Because of the significant increase of the state space, many approximation methods were proposed for reachability computation and can be grouped in two main approaches. The first one is based on an approximate simulation relation to obtain an abstraction of the original system [19]. In the second approach, static analysis methods inspired from Cousot et al. [11] analyze the system dynamic and cope with the state-space explosion. In this group we can cite the development of static analysis of reachability properties based on an over- and under-approximation using the Process Hitting [30] and extended for the ANs. Besides methods that approximated the computation of reachability, authors in [33,34] proposed methods for safety verification that do not require the computation of reachable sets, but instead relied on the notion of barrier certificates [32].

A key notion underlying network behavior is that state-space is organized into a number of basins of attraction, connecting states according to their transitions, and summing up the network's global dynamics. For systems having many attractors, some of them may correspond to desired behaviors, while others, to unsafe regions. In the global network dynamic a system can move towards an unsafe region. Therefore, beyond ensuring the properties reachability and safety verification, proposing strategies to interfere on a system and force a desired behavior is important. Previous works introduced the concept of Minimal Intervention Sets (MISs) [23] for biological networks and later generalized in [37] for addressing Boolean models of signaling networks. This concept allowed one to choose an intervention strategy to provoke a desired/observed response in certain target nodes. Compared to the approach present of this paper, they do not take into account the transient dynamics of the system, which limits considerably the domain of predictions. [28,2] proposed approaches based on cut sets to identify nodes/reactions whose perturbation would prevent the occurrence of a given state/reaction. Whereas those prediction can help to control the reachability of an attractor, they do not allow to capture the differentiations processes, as do bifurcations.

Outline Section 2 introduces the Automata Network formalisms on which our method relies. The notion of bifurcation in ANs is introduced in section 3. In section 4, we give a brief overview of methods for checking reachability properties which is a core task for the characterization of bifurcations. In section 5, we combine static analysis, dynamics unfolding, and ASP, aiming at providing a scalable identification of the bifurcation transitions for a given goal. In section 6, we evaluate a prototype implementation on several large biological models in order to support the scalability of our approach. Section 7 concludes the paper.

2 Automata Networks

We consider an Automata Network (AN) as a finite set of finite-state machines having transitions between their local states conditioned by the state of other automata in the network. An AN is defined by a triple (Σ, S, T) (definition 1) where Σ is the set of automata identifiers; S associates to each automaton a finite set of local states: if $a \in \Sigma$, $S(a)$ refers to the set of local states of a; and T associates to each automaton the list of its local transitions. Each local state is written of the form a_i, where $a \in \Sigma$ is the automaton in which the state belongs to, and i is a unique identifier; therefore given $a_i, a_j \in S(a)$, $a_i = a_j$ if and only if a_i and a_j refer to the same local state of the automaton a. For each automaton $a \in \Sigma$, $T(a)$ refers to the set of transitions of the form $t = a_i \xrightarrow{\ell} a_j$ with $a_i, a_j \in S(a)$, $a_i \neq a_j$, and ℓ the enabling condition of t, formed by a (possibly empty) set of local states of automata different than a and containing at most one local state of each automaton.

Definition 1 (Automata Network (Σ, S, T)). *An* Automata Network *(AN) is defined by a tuple (Σ, S, T) where*

- *Σ is the finite set of automata identifiers;*
- *For each $a \in \Sigma$, $S(a) = \{a_i, \ldots, a_j\}$ is the finite set of local states of automaton a; $S \triangleq \prod_{a \in \Sigma} S(a)$ is the finite set of global states;*
  $\mathbf{LS} \triangleq \bigcup_{a \in \Sigma} S(a)$ denotes the set of all the local states.
- *$T = \{a \mapsto T_a \mid a \in \Sigma\}$, where $\forall a \in \Sigma, T_a \subseteq S(a) \times 2^{\mathbf{LS} \setminus S(a)} \times S(a)$ with $(a_i, \ell, a_j) \in T_a \Rightarrow a_i \neq a_j$ and $\forall b \in \Sigma, |\ell \cap S(b)| \leq 1$, is the mapping from automata to their finite set of local transitions.*

We write $a_i \xrightarrow{\ell} a_j \in T \triangleq (a_i, \ell, a_j) \in T(a)$.

At any time, each automaton is in one and only one local state, forming the global state of the network. Assuming an arbitrary ordering between automata identifiers, the set of global states of the network is referred to as S as a shortcut for $\prod_{a \in \Sigma} S(a)$. Given a global state $s \in S$, $s(a)$ is the local state of automaton a in s, i.e., the a-th coordinate of s.

A local transition $t = a_i \xrightarrow{\ell} a_j \in T$ is applicable in a global state $s \in S$ when a_i and all the local states in ℓ are in s. The application of the local transition, noted $s \cdot t$, replaces the local state of a with a_j (definition 2). It results in a (global) transition $s \xrightarrow{t} s'$ where $s' = s \cdot t$. In this paper, we consider the *asynchronous* semantics of ANs: only one local transition can be applied at a time, meaning only one automaton changes its local state by the transitions between two global states. In this semantics, different local transitions may be applicable to a same state, which may potentially lead to very different dynamics. The choice of the transition is *non-deterministic*. A global state s' is reachable from s, noted $s \rightarrow^* s'$, if and only if there exists a (possibly empty) sequence of transitions leading from s to s'.

Definition 2 (Transition, reachabilility). *Given a state $s \in S$ and a local transition $t = a_i \xrightarrow{\ell} a_j \in T$ such that $s(a) = a_i$ and $\forall b_k \in \ell$, $s(b) = b_j$, $s \cdot t$ is the state s where a_i has been replaced by a_j:*

$$\forall b \in \Sigma, (s \cdot t)(b) = \begin{cases} a_j & \text{if } b = a \\ s(b) & \text{otherwise} \end{cases}$$

We then write $s \xrightarrow{t} s'$ where $s' = s \cdot t$. The reachability *binary relation $\rightarrow^* \subseteq S \times S$ satisfies*

$$s \rightarrow^* s' \overset{\Delta}{\Leftrightarrow} s = s' \vee \exists t \in T : s \xrightarrow{t} s'' \wedge s'' \rightarrow^* s'$$

Figure 1 represents an AN (Σ, S, T) of 3 automata ($\Sigma = \{a, b, c\}$), with $S(a) = \{a_0, a_1, a_2\}$, $S(b) = \{b_0, b_1\}$, $S(c) = \{c_0, c_1, c_2\}$, and 8 local transitions defined as follows:

$$T(a) = \{t_1 = a_1 \xrightarrow{\emptyset} a_0, t_2 = a_0 \xrightarrow{b_0} a_1, t_3 = a_0 \xrightarrow{b_0, c_0} a_2\}$$
$$T(b) = \{t_4 = b_0 \xrightarrow{\emptyset} b_1, t_5 = b_1 \xrightarrow{a_0} b_0\}$$
$$T(c) = \{t_6 = c_0 \xrightarrow{a_1} c_1, t_7 = c_1 \xrightarrow{b_1} c_0, t_8 = c_1 \xrightarrow{b_0} c_2\}$$

From the given initial state $s_0 = \langle a_0, b_0, c_0 \rangle$, 3 transitions can be applied: t_2, t_3, and t_4; the application of the latter results in $s_0 \cdot t_4 = \langle a_0, b_1, c_0 \rangle$ (automaton b is now in state b_1).

3 Bifurcations

From an initial state s_0 and a *goal* local state, we call a *bifurcation* a transition from a state where the goal is reachable to a state where the goal is not reachable, i.e., there exists no sequence of transition that lead to a state containing the goal local state.

Let us consider the AN of figure 1, with $s_0 = \langle a_0, b_0, c_0 \rangle$ and the goal a_2. Figure 2 shows all the possible transitions from s_0. The states with a gray background are connected to a state containing a_2 (in thick/blue). The transitions in thick/red are bifurcations: once in a white state, there exist no sequence of transitions leading to a_2. In other words, bifurcations are the transitions from a gray state to a white state. In this example, t_8 is the (unique) local transitions responsible for bifurcations from s_0 to a_2.

In this paper, given an AN (Σ, S, T), we are interested in identifying the local transitions $t_b \in T$ that trigger a bifurcation from a state reached from $s_0 \in S$ for a given goal, describing a set of states $S_g \subseteq S$. We call s_b a global state where a bifurcation occurs, and s_u the global state after the bifurcation: $s_u = s_b \cdot t_b$. The goal is reachable from s_b but not from s_u. This is illustrated by figure 3. Note that, as illustrated, s_b is not inevitably reached: we allow the existence of alternative paths of transitions to the goal.

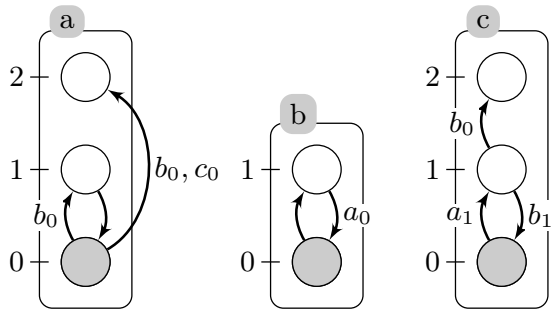


Figure 1. An example of Automata Network (AN).‘ Automata are represented by labelled boxes, and local states by circles where ticks are their identifier within the automaton – for instance, the local state a_0 is the circle ticked 0 in the box a. A transition is a directed edge between two local states within the same automaton. It can be labelled with a set of local states of other automata. Grayed local states stand for the global state $\langle a_0, b_0, c_0 \rangle$.

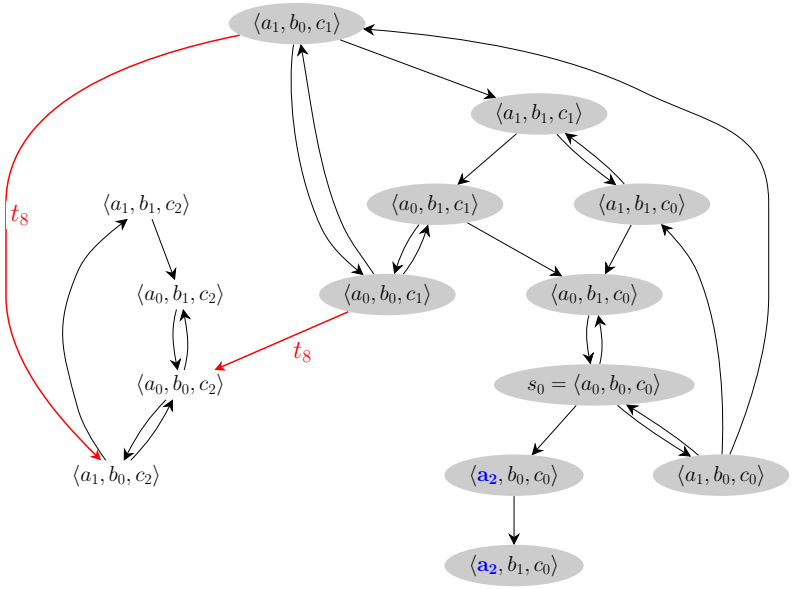


Figure 2. Transition graph of the AN in figure 1 from the initial state $s_0 = \langle a_0, b_0, c_0 \rangle$. The goal a_2 is in thick/blue; the states connected to the goal are in gray; the bifurcations to the goal in thick/red, labelled with the local transitions in the AN definition.

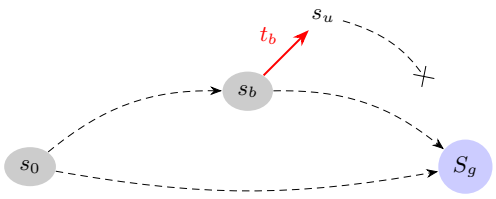


Figure 3. General illustration of a bifurcation. s_0 is the initial state, S_g is a set of states in which the goal local state is present. The dashed arrows represent a sequence (possibly empty) of transitions. The plain red arrow is a bifurcation from a global state s_b to s_u, and t_b is the associated local transition.

Definition 3 formalizes the notion of bifurcation, where the goal is specified by a local state g_1 (hence $S_g = \{s \in S \mid s(g) = 1\}$). Note that this goal specification does not loose generality, as one can build an automaton g with local states g_0 and g_1, and with a local transitions from g_0 to g_1 conditioned by each desired goal state.

Definition 3 (Bifurcation). *Given an AN (Σ, S, T), a global state $s_0 \in S$ and a goal local state g_1 with $g \in \Sigma$ and $g_1 \in S(g)$, a* bifurcation *is a transition $s_b \xrightarrow{t_b} s_u$ of the AN with $s_b, s_u \in S$ and $t_b \in T$, such that $s_0 \to^* s_b$ and $\forall s' \in S$ where $s_u \to^* s'$, $s'(g) \neq g_1$.*

4 Reachability and formal approximations

The identification of a bifurcation as presented in the previous section can be decomposed in three steps: we want to find $s_b \in S$ and $t_b \in T$ such that (1) s_b is reachable from s_0 ($s_0 \to^* s_b$); (2) g_1 is reachable from s_b ($\exists s_g \in S : s_g(g) = g_1 \wedge s_b \to^* s_g$); (2) g_1 is not reachable from $s_u = s_b{\cdot}t_b$ ($\forall s \in S, s_u \to^* s \Rightarrow s(g) \neq g_1$). Therefore, alongside the enumeration of candidate s_b and t_b, reachability is at the core of our bifurcation identification.

In this section, we give a brief overview of the basics of reachability checking, stressing the methods we use in this paper.

4.1 State graph and partial order reductions

Given two states s, s' of an AN (or an equivalent Petri net), verifying $s \to^* s'$ is a PSPACE-complete problem [8].

The common approach for reachability checking is to build the (finite) set of all the states reachable from s until finding s', by exploring all the possible transitions. However, such a set can be rapidly intractable with large models. Techniques relying on symbolic representations, notably using Binary Decision Diagrams (BDDs) and variants [21] can improve the scalability of this approach by several orders of magnitude [9].

In many cases, numerous transitions modelled by ANs are *concurrent*: their application is independent from each other. For instance, if t_1 and t_2 are concurrent in a state s, one can apply indifferently $s \cdot t_1 \cdot t_2$ and $s \cdot t_2 \cdot t_1$. Such features can be exploited to provide very compact representations of the reachable states in a concurrent system, taking into account the partial order of transition applications. Unfoldings, and more precisely their complete finite prefixes [13], allow to compute efficiently such compact representations.

In this paper, part of our method uses complete finite prefixes of unfoldings to compute the states that are reachable from the fixed initial state s_0. Indeed, because biological networks are typically very large, but also very sparse (each node/automaton interacts with a few others, compared to the size of the network), they exhibit a high degree of concurrency for their transitions, making unfolding approaches very effective.

4.2 Formal approximations

When facing a large AN, it may turn out that the reachable state space is too large for the aforementioned exact verification of reachability. Moreover, the complexity of the reachability problem can be prohibitive when numerous verifications have to be done, for instance when enumerating candidate initial states (such as s_b in our case for checking goal reachability from it).

In this paper, we rely on the reachability approximations for ANs introduced in [29,15]. We will use both *over-approximations* (OA) and *under-approximations* (UA) of the reachability problem: $s \to^* s'$ is true only if $\mathsf{OA}(s \to^* s')$ is true and $s \to^* s'$ is true if $\mathsf{UA}(s \to^* s')$ is true; but the converses do not hold in general:

$$\mathsf{UA}(s \to^* s') \Rightarrow s \to^* s' \Rightarrow \mathsf{OA}(s \to^* s')$$

The approximations rely on static analysis by abstract interpretation of AN dynamics. We give here the basic explanations for the over- and under-approximation. The analysis rely on the decomposition of the systems dynamics in automata, in order to derive necessary or sufficient conditions for a reachability property of the form $s \to^* s'$.

The core objects are the *local paths* within two local states a_i, a_j of a same automaton a. We call $a_i \rightsquigarrow a_j$ an *objective* and define $\mathsf{local\text{-}paths}(a_i \rightsquigarrow a_j)$ the set of the acyclic paths of local transitions between a_i and a_j. Definition 4 gives the formalization of $\mathsf{local\text{-}paths}$ where we use the following notations: for a local transition $t = a_i \xrightarrow{\ell} a_j \in T$, $\mathsf{orig}(t) \triangleq a_i$, $\mathsf{dest}(t) \triangleq a_j$, $\mathsf{enab}(t) \triangleq \ell$; ε denotes the empty sequence, and $|\eta|$ is the length of sequence η.

Definition 4 ($\mathsf{local\text{-}paths}$)**.** *Given an objective $a_i \rightsquigarrow a_j$,*

- *if $i = j$, $\mathsf{local\text{-}paths}(a_i \rightsquigarrow a_i) \triangleq \{\varepsilon\}$;*
- *if $i \neq j$, a sequence η of transitions in $T(a)$ is in $\mathsf{local\text{-}paths}(a_i \rightsquigarrow a_j)$ if and only if $\mathsf{orig}(\eta^1) = a_i$, $\mathsf{dest}(\eta^{|\eta|}) = a_j$, $\forall n, 1 \leq n < |\eta|$, $\mathsf{dest}(\eta^n) = \mathsf{orig}(\eta^{n+1})$, and $\forall n, m, |\eta| \geq n > m \geq 1, \mathsf{dest}(\eta^n) \neq \mathsf{orig}(\eta^m)$.*

We write $t \in \eta \overset{\Delta}{\Leftrightarrow} \exists n, 1 \leq n \leq |\eta| : \eta_n = t$. *Given a local path* η, $\widetilde{\eta}$ *denotes the union of the conditions of all the local transitions composing it:*

$$\widetilde{\eta} \overset{\Delta}{=} \bigcup_{n=1}^{|\eta|} \mathsf{enab}(\eta_n)$$

In the AN of figure 1, $\mathsf{local\text{-}paths}(a_0 \rightsquigarrow a_2) = \{a_0 \xrightarrow{b_0, c_0} a_2\}$; $\mathsf{local\text{-}paths}(c_2 \rightsquigarrow c_1) = \emptyset$.

Focusing on the reachability of a single local state g_1 from a state s where $s(g) = 0$, the analyzes essentially start with the local paths in $\mathsf{local\text{-}paths}(g_0 \rightsquigarrow g_1)$: if g_1 is reachable, then at least of the local path η has to be realizable, meaning that all the local states of its conditions ($\widetilde{\eta}$) should be reachable. This lead to a recursive reasoning by repeating the procedure with the objectives from s to the local states in $\widetilde{\eta}$.

The dependence relationships between the local paths of the different automata can be represented as a graph, where the nodes are all the local states, all the possible objectives, and all their local paths. Such a graph is called a *Local Causality Graph* (LCG), and abstracts all the executions of the AN.

From a complexity point of view, local paths are computed for each pair of local states within every automata; the length of a local path being at most the number of local states within the automaton, the number of local paths is at most polynomial in the number of local transitions and exponential in the size of the single automaton. In practice, the automata are small, typically between 2 and 4 states for biological models. Therefore, LCGs turn out to be very small compared to the reachable state space of biological networks. They have been successfully applied for analyzing dynamics of ANs with hundreds or thousands of automata, which were intractable with standard model-checking approaches [29,28].

The over-approximation and under-approximation reduce to finding subgraphs of LCGs that satisfy some particular structural properties, which have been proven to be necessary or sufficient for the reachability property, respectively. Whereas the over-approximation can be verified in a time linear with the LCG [29], the under-approximation we consider in this paper requires to enumerate over many possible sub-LCGs, but checking if a sub-LCG satisfies the sufficient condition is linear in its size.

Note that further refinements of $\mathsf{local\text{-}paths}$ have been considered for the mentioned approximations, but for the sake of simplicity, we stick to this coarse-grained presentation in the scope of this paper.

Appendix A gives examples of LCGs satisfying necessary or sufficient conditions for reachability properties in the AN of figure 1.

5 Identification of bifurcations using ASP

Among the states reachable from s_0, we want to find a state s_b from which (1) the goal is reachable and (2) there exists a transition to a state from which the goal is not reachable. Putting aside the complexity of reachabilities checking, the

enumeration of candidate states s_b is a clear bottleneck for the identification of bifurcations in an AN.

Our approach combines the formal approximations and (optionally) unfoldings introduced in the previous section with a constraint programming approach to efficiently identify bifurcations. As discussed in the previous section, checking the over-/under-approximations from candidate states and sub-LCGs is easy. For the case of unfolding, checking if a state s belongs to the state space represented by a complete finite prefix is NP-complete [14]. Therefore, a declarative approach such as Answer-Set Programming (ASP) [3] is very well suited for specifying admissible s_b and t_b, and obtaining efficient enumerations of solutions by a solver.

We first present the general scheme of our method, and then given details on its implementation with ASP.

5.1 General scheme

A sound and complete characterization of the local transitions $t_b \in T$ triggering a bifurcation from state s_0 to the goal g_1 would be the following: t_b is a bifurcation transition if and only if there exists a state $s_b \in S$ such that

$$\text{(C1)}\ s_u \not\rightarrow^* g_1 \qquad \text{(C2)}\ s_b \rightarrow^* g_1 \qquad \text{(C3)}\ s_0 \rightarrow^* s_b$$

where $s_u = s_b \cdot t_b$, $s \not\rightarrow^* g_1 \overset{\Delta}{\Leftrightarrow} \forall s' \in S, s \rightarrow^* s' \Rightarrow s'(g) \neq g_1$ and $s \rightarrow^* g_1 \overset{\Delta}{\Leftrightarrow} \exists s_g \in S : s_g(g) = g_1 \wedge s \rightarrow^* s_g$.

However, in an enumeration scheme for s_b candidates, checking reachability and non-reachability of the goal from each s_b candidate ((C1) and (C2)) is prohibitive. Instead, we relax the above constraints as follows:

$$\text{(I1}^\#\text{)}\ \neg\mathsf{OA}(s_u \rightarrow^* g_1) \qquad \text{(I2}^\#\text{)}\ \mathsf{UA}(s_b \rightarrow^* g_1) \qquad \begin{array}{r}\text{(I3)}\ s_b \in \mathsf{unf\text{-}prefix}(s_0)\\ \text{(I3}^\#\text{)}\ \mathsf{UA}(s_0 \rightarrow^* s_b)\end{array}$$

where $\mathsf{unf\text{-}prefix}(s_0)$ is the set of all reachable states from s_0 represented as the prefix of the unfolding of the AN which has to be pre-computed (section 4.1). Either (I3) or (I3$^\#$) can be used, at discretion. From OA and UA properties (section 4.2), we directly obtain:

$$\text{(I1}^\#\text{)} \Rightarrow \text{(C1)} \qquad \text{(I2}^\#\text{)} \Rightarrow \text{(C2)} \qquad \begin{array}{r}\text{(I3)} \Leftrightarrow \text{(C3)}\\ \text{(I3}^\#\text{)} \Rightarrow \text{(C3)}\end{array}$$

Therefore, our characterization is sound (no false positive) but incomplete: some t_b might be missed (false negatives). Using (I3) instead of (I3$^\#$) potentially reduces the false negatives, at the condition that the prefix of the unfolding is tractable. When facing a model too large for the unfolding approach, we should rely on (I3$^\#$) which is much more scalable but may lead to more false negatives.

Relying on the unfolding from s_b ($\mathsf{unf\text{-}prefix}(s_b)$) is not considered here, as it would require to compute a prefix from each s_b candidate, whereas $\mathsf{unf\text{-}prefix}(s_0)$ is computed only once before the bifurcation identification.

5.2 ASP syntax and semantics

We give a very brief overview of ASP syntax and semantics that we use in the next section. Please refer to [18,3,16] for an in-depth introduction to ASP.

An ASP program consists in a set of predicates, and logic rules of the form:

```
1 a_0 ← a_1, ..., a_n, not a_{n+1}, ..., not a_{n+k}.
```

where a_i are atoms, terms or predicates in first order logic.

Essentially, such a logical rules states that when all $a_1, \ldots, a_n$ are true and all $a_{n+1}, \ldots, a_{n+k}$ are false, then a_0 has to be true as well. a_0 can be $\bot$ (or simply omitted): in such a case, the rule is satisfied only if the right hand side of the rule is false (at least one of $a_1, \ldots, a_n$ is false or at least one of $a_{n+1}, \ldots, a_{n+k}$ is true). A solution consists in a set of true atoms/terms/predicates with which all the logical rules are satisfied.

ASP allows to use variables (starting with an uppercase) instead of terms/predicates: these *pattern* declarations will be expanded to the corresponding propositional logic rules prior to the solving. For instance, the following ASP program has as unique (minimal) solution `b(1) b(2) c(1) c(2)`.

```
2 c(X) ← b(X).
3 b(1).
4 b(2).
```

In the following, we also use the notation n `{ a(X) : b(X) }` m which is true when at least n and at most m `a(X)` are true where `X` ranges over the true `b(X)`.

5.3 ASP implementation

We present here the main rules for implementing the identification of bifurcation transitions with ASP. A significant part of ASP declarations used by (I1$^{\#}$), (I2$^{\#}$), (I3), and (I3$^{\#}$) are generated from the prior computation of local-paths and, in the case of (I3), of the prefix of the unfolding. Applied on figure 1, our implementation correctly uncovers t_8 as a bifurcation for a_2.

Declaration of local states, transitions, and states Every local state $a_i \in S(a)$ of each automaton $a \in \Sigma$ are declared with the predicate `ls(`a`,`i`)`. We declare the local transitions of the AN and their associated conditions by the predicates `tr(`id`,`a`,`i`,`j`)` and `trcond(`id`,`b`,`k`)`, which correspond to the local transition $a_i \xrightarrow{\{b_k\} \cup \ell} a_j \in T$. States are declared with the predicate `s(ID,A,I)` where `ID` is the state identifier, and `A`, `I`, the automaton and local state present in that state. Finally, the goal g_1 is declared with `goal(`g`,1)`.

For instance, the following instructions declare the automaton a of figure 1 with its local transitions, the state $s_0 = \langle a_0, b_0, c_0 \rangle$, and the goal being a_2:

```
1 ls(a,0). ls(a,1). ls(a,2).
2 tr(1,a,1,0).
3 tr(2,a,0,1). trcond(2,b,0).
4 tr(3,a,0,2). trcond(3,b,0). trcond(3,c,0).
5 s(0,a,0). s(0,b,0). s(0,c,0).        goal(a,2).
```

Declaration of s_b, t_b, and s_u The bifurcation transition t_b, declared as `btr(`b`)`, is selected among the declared transitions identifiers (line 6). If $a_i \xrightarrow{\ell} a_j$ is the selected transition, the global state s_u (recall that $s_u = s_b \cdot t_b$) should satisfy $s_u(a) = a_j$ (line 7) and, $\forall b_k \in \ell$, $s_u(b) = b_k$ (line 8). The state s_b should then match s_u, except for the automaton a, as $s_b(a) = a_i$ (lines 9 and 10).

```
6  1 { btr(ID) : tr(ID,_,_,_) } 1.
7  s(u,A,J) ← btr(ID),tr(ID,A,_,J).
8  s(u,B,K) ← btr(ID),trcond(ID,B,K).
9  s(b,A,I) ← btr(ID),tr(ID,A,I,_).
10 s(b,B,K) ← s(u,B,K),btr(ID),tr(ID,A,_,_),B!=A.
```

(I1#) declaration of $\neg$OA($s_u \rightarrow^* g_1$) This part aims at finding the state s_u from which g_1 is not reachable. For that, we designed an ASP implementation of the reachability over-approximation presented in section 4.2. It consists in building a Local Causality Graph (LCG) from pre-computed local-paths. A predicate `oa_valid(G,ls(A,I))` is then defined upon G to be true when the local state a_i is reachable from the initial state s_G. The full implementation is given in appendix B.1.

We instantiate a LCG u with the initial state s_u by declaring that the goal is not reachable (`oa_valid`) (line 11).

```
11 ← oa_valid(u,ls(G,I)),goal(G,I).
```

(I2#) declaration of UA($s_b \rightarrow^* g_1$) This part aims at finding the state s_b from which g_1 is reachable. Our designed ASP implementation of the reachability under-approximation (section 4.2) consists in finding a sub-LCG G with the satisfying properties for proving the sufficient condition. The edges of this sub-LCG are declared with the predicate `ua_lcg(G,Parent,Child)`. The graph is parameterized by (1) a *context* which specifies a set of possible initial states and (2) an edge from the node `root` to the local state(s) for which the simultaneous reachability has to be decided from the supplied context. The full implementation is given in appendix B.2.

We instantiate the under-approximation for building a state s_b from which the goal g_1 is reachable: g_1 is a child of the `root` node of graph b (line 12). The context is subject to the same constraints as s_b from s_u (lines 13 and 14 reflect lines 9 and 10). Then, s_b defines one local state per automaton among the context from which the reachability of g_1 is ensured (line 15), and according to lines 9 and 10.

```
12 ua_lcg(b,root,ls(G,I)) ← goal(G,I).
13 ctx(b,A,I) ← btr(ID),tr(ID,A,I,_).
14 ctx(b,B,K) ← s(u,B,K),btr(ID),tr(ID,A,_,_),B!=A.
15 1 { s(b,A,I) : ctx(b,A,I) } 1 ← ctx(b,A,_).
```

(I3) declaration of $s_b \in$ unf-prefix(s_0) Given a prefix of an unfolding from s_0 (section 4.1), checking if s_b is reachable from s_0 is an NP-complete problem [14] which can be efficiently encoded in SAT [22] (and hence in ASP). A synthetic description of the ASP implementation of reachability in unfoldings is given in appendix C. The interested reader should refer to [13]. Our encoding provides a predicate `reach(a,i)` which is true if a reachable state contains a_i. Declaring s_b reachable from s_0 is done simply as follows:

```
16 reach(A,I) ← s(b,A,I).
```

(I3$^\#$) declaration of UA$(s_0 \rightarrow^* s_b)$ An alternative to (I3) which does not require to compute a complete prefix of the unfolding is to rely on the under-approximation of reachability similarly to (I2$^\#$). The under-approximation is instantiated for the reachability of s_b from s_0 with the following statements:

```
17 ua_lcg(0,root,ls(A,I)) ← s(b,A,I).
18 ctx(0,A,I) ← s(0,A,I).
```

6 Experiments

We evaluate our method for the computation of bifurcation transitions in models of actual biological networks showing differentiation capabilities. We have selected three networks showing at least two attractors reachable from a same initial state. In these cases, by supplying an adequate goal state representing one of the attractor, we expect to identify transitions causing a bifurcation from this attractor. We start by introducing our three case studies.

Immunity control in bacteriophage lambda (Lambda phage) . A number of bacterial and viral genes take part in the decision between lysis and lysogenization in temperate bacteriophages. In the lambda case, at least five viral genes (refered to as cI, cro, cII, N and cIII) and several bacterial genes are involved. We apply our method on an automata network equivalent to the model introduced in [41] and with two different goals, corresponding to lysis or lysogenization.

Epidermal Growth Factor & Tumor Necrosis Factor$_\alpha$ (EGF/TNF) is a model that combines two important mammalian signaling pathways induced by the Epidermal Growth Factor (EGF) and Tumor Necrosis Factor alpha (TNF$_\alpha$) [24,7]. EGF and TNF$_\alpha$ ligands stimulate ERK, JNK and p38 MAPK cascades, the PI3K/AKT pathways, and the NFkB cascade. This network encompasses cross-talks between these pathways, as well as two negative feedback loops.

T-helper cell plasticity (Th) has been studied in [1] in order to investigate switches between attractors subsequent to changes of input conditions. It is a cellular network regulating the differentiation of T-helper (Th) cells, which orchestrate many physiological and pathological immune responses. T-helper (CD4+) lymphocytes play a key role in the regulation of the immune response. By APC activation, native CD4 T cells (Th0) differentiate into specific Th subtypes producing different cytokines which influence the activity of immune effector cell

Automata Network	Goal	(I3)			(I3$^{\#}$)	
		$\lvert\mathsf{unf\text{-}prefix}(s_0)\rvert$	$\lvert t_b\rvert$	Time	$\lvert t_b\rvert$	Time
Lambda phage	CI_2	45	6	$0.14s$	0	$0.34s$
$\lvert\Sigma\rvert = 4 \quad \lvert T\rvert = 11$	Cro_2		3	$0.15s$	2	$0.44s$
EGF/TNF	$NFkB_0$	52	4	$0.07s$	2	$0.15s$
$\lvert\Sigma\rvert = 28 \quad \lvert T\rvert = 55$	IKB_1		3	$0.07s$	2	$0.13s$
Th_th1	$BCL6_1$	444	6	$19.6s$	5	$25.82s$
$\lvert\Sigma\rvert = 101 \quad \lvert T\rvert = 381$	$TBET_1$		5	$13.08s$	4	$26.4s$
Th_th2	$GATA3_0$	3264	7	$28.7s$	4	$27.5s$
$\lvert\Sigma\rvert = 101 \quad \lvert T\rvert = 381$	$BCL6_1$		5	$28.4s$	4	$28.01s$
Th_th17	$RORGT_1$	2860	9	$23.9s$	8	$29.04s$
$\lvert\Sigma\rvert = 101 \quad \lvert T\rvert = 381$	$BCL6_1$		5	$26.2s$	4	$26.64s$
Th_HTG	$BCL6_1$	OT	-	-	6	$61.9s$
$\lvert\Sigma\rvert = 101 \quad \lvert T\rvert = 381$	$GATA3_1$		-	-	7	$34.16s$

Table 1. Experimental results for the identification of bifurcations depending if (I3) or (I3$^{\#}$) is used. Models Th_th1, Th_th2, Th_th17, Th_HTG are the same automata network but have different initial state. For each model, two different goals have been tested. $\lvert\Sigma\rvert$ is the number of automata, and $\lvert T\rvert$ the number of transitions; $\lvert \inf(s_0)\rvert$ is the size (number of events in the partial order structure) of the prefix of the unfolding from the initial state of the model; $\lvert t_b\rvert$ is the number of identified bifurcation transitions. Computation times have been obtained on an Intel® Core™ i7-4600M 2.90GHz CPU with 7.7GiB of RAM. OT indicates an out-of-time execution (more than one hour).

types. Several subtypes (Th1, Th2, Th17, Treg, Tfh, Th9, and Th22) have been well established. We report in this paper four (Pro Th1, Pro Th2, Pro Th17, Pro HTG(Th1 + Th17)) different initial state from which different attractors are reachable. The network is composed of 101 nodes and 221 interactions.

Method. For each selected model with initial state and goal, we performed the bifurcation identification following either (I1$^{\#}$), (I2$^{\#}$), (I3) (unfolding from s_0); or (I1$^{\#}$), (I2$^{\#}$), (I3$^{\#}$) (reachability under-approximation). We use clingo 4.5.3 [17] as ASP solver, and Mole [38] for the computation of the unfolding ((I3)). The computation times correspond to the total toolchain duration, and includes the local-paths computation, unfolding, ASP program generation, ASP program loading and grounding, and solving. Note that the local-paths computation (and ASP program generation) is almost instantaneous for each case. Source code and models are provided in the supplementary material file.

Results. Table 1 summarizes the results of the experiments. The last experiment shows the limit of the exact analysis of the reachable state space: the

computation of the prefix is not tractable on this model. However, the alternative approach (I3$^{\#}$) allows to identify bifurcation transitions in this large model. Following section 5.1, (I3$^{\#}$) always results in less bifurcations transitions than (I3) with our models. It can be explained with the additional approximation for the reachability of s_b from s_0, using the notations of section 3. One can finally remark that when (I3) is tractable, (I3$^{\#}$) shows a slightly slower solving time, albeit of the same order of magnitude. This suggest that checking if a state belongs to an unfolding is more efficient than checking its under-approximation. Finally, because there exists to our knowledge no other method for identifying bifurcations, we cannot compare our results with different methods, and in particular with an exact method in order to appreciate the false negative rate obtained by the (I1$^{\#}$)-(I2$^{\#}$)-(I3) scheme.

7 Conclusion

This paper presents an original combination of computational techniques to identify transitions of a dynamical system that can remove its capability to reach a (set of) states of interest. Our methodology combines static analysis of Automata Networks (ANs) dynamics, partial order representations of the state space, and constraint programming to efficiently enumerate those bifurcations. To our knowledge, this is the first intregated approach for deriving bifurcation transitions from concurrent models, and ANs in particular.

Bifurcations are key features of biological networks, as they model decisive transitions which control the *differentiation* of the cell: the bifurcations decide the portions of the state space (no longer) reachable in the long-run dynamics. Providing automatic methods for capturing those differentiations steps is of great interest for biological challenges such as cell reprogramming [12,1], as they suggest targets for modulating undergoing cellular processes.

Given an initial state of the AN and a goal state, our method first computes static abstractions of the AN dynamics and (optionnaly) a symbolic representation of the reachable state space with so-called unfoldings. From those prior computations, a set of constraints are issued to identify bifurcation transitions. We used Answer-Set Programming to declare the admissible solutions and the solver clingo to obtain their efficient enumerations. For large models, the unfolding may be intractable: in such a case, the methods relies only on reachability over- and under-approximations. By relying on those relaxations which can be efficiently encoded in ASP, our approach avoids costly exact checking, and is tractable on large models, as supported by the experiments.

Further work will consider the complete identification of bifurcation transitions, by allowing false positives (but no false negatives). In combination with the under-approximation of the bifurcations presented in this paper, it will provide an efficient way to delineate *all* the transitions that control the reachability of the goal attractor. Future work will also focus on exploiting the identified bifurcations for driving estimations of the probability of reaching the goal at steady state, in the scope of hybrid models of biological networks [39,40].

References

1. W. Abou-Jaoudé, P. T. Monteiro, A. Naldi, M. Grandclaudon, V. Soumelis, C. Chaouiya, and D. Thieffry. Model checking to assess t-helper cell plasticity. *Frontiers in Bioengineering and Biotechnology*, 2(86), 2015.
2. V. Acuna, F. Chierichetti, V. Lacroix, A. Marchetti-Spaccamela, M.-F. Sagot, and L. Stougie. Modes and cuts in metabolic networks: Complexity and algorithms. *Biosystems*, 95(1):51–60, 2009.
3. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.
4. G. Bernot, F. Cassez, J.-P. Comet, F. Delaplace, C. Müller, and O. Roux. Semantics of biological regulatory networks. *Electronic Notes in Theoretical Computer Science*, 180(3):3 – 14, 2007.
5. L. Calzone, E. Barillot, and A. Zinovyev. Predicting genetic interactions from boolean models of biological networks. *Integr. Biol.*, 7(8):921–929, 2015.
6. C. G. Cassandras. *Discrete event systems: modeling and performance analysis*. CRC, 1993.
7. C. Chaouiya, D. Bérenguier, S. M. Keating, A. Naldi, M. P. van Iersel, N. Rodriguez, A. Dräger, F. Büchel, T. Cokelaer, B. Kowal, B. Wicks, E. Gonçalves, J. Dorier, M. Page, P. T. Monteiro, A. von Kamp, I. Xenarios, H. de Jong, M. Hucka, S. Klamt, D. Thieffry, N. Le Novère, J. Saez-Rodriguez, and T. Helikar. SBML qualitative models: a model representation format and infrastructure to foster interactions between qualitative modelling formalisms and tools. *BMC Systems Biology*, 7(1):1–15, 2013.
8. A. Cheng, J. Esparza, and J. Palsberg. Complexity results for 1-safe nets. *Theor. Comput. Sci.*, 147(1&2):117–136, 1995.
9. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 241–268. Springer Berlin / Heidelberg, 2002.
10. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.
11. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'77)*, pages 238–252, New York, NY, USA, 1977. ACM.
12. I. Crespo and A. del Sol. A general strategy for cellular reprogramming: The importance of transcription factor cross-repression. *STEM CELLS*, 31(10):2127–2135, Oct 2013.
13. J. Esparza and K. Heljanko. *Unfoldings – A Partial-Order Approach to Model Checking*. Springer, 2008.
14. J. Esparza and C. Schröter. Unfolding based algorithms for the reachability problem. *Fund. Inf.*, 47(3-4):231–245, 2001.
15. M. Folschette, L. Paulevé, M. Magnin, and O. Roux. Sufficient conditions for reachability in automata networks with priorities. *Theoretical Computer Science*, 608, Part 1:66 – 83, 2015. From Computer Science to Biology and Back.
16. M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012.

17. M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Clingo* = ASP + control: Preliminary report. In M. Leuschel and T. Schrijvers, editors, *Technical Communications of the Thirtieth International Conference on Logic Programming (ICLP'14)*, volume arXiv:1405.3694v1, 2014. Theory and Practice of Logic Programming, Online Supplement.
18. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, pages 1070–1080, 1988.
19. A. Girard, A. A. Julius, and G. J. Pappas. Approximate simulation relations for hybrid systems. *IFAC Analysis and Design of Hybrid Systems, Alghero, Italy*, 2006.
20. R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel. *Hybrid systems*. Springer-Verlag, 1993.
21. A. Hamez, Y. Thierry-Mieg, and F. Kordon. Building efficient model checkers using hierarchical set decision diagrams and automatic saturation. *Fundam. Inf.*, 94(3-4):413–437, 2009.
22. K. Heljanko. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-Safe petri nets. *Fundamenta Informaticae*, 37(3):247–268, 1999.
23. S. Klamt and E. D. Gilles. Minimal cut sets in biochemical reaction networks. *Bioinformatics*, 20(2):226–234, 2004.
24. A. MacNamara, C. Terfve, D. Henriques, B. P. Bernabé, and J. Saez-Rodriguez. State–time spectrum of signal transduction logic models. *Physical Biology*, 9(4):045003, 2012.
25. I. Moon. Automatic verification of discrete chemical process control systems. 1992.
26. I. Moon and S. Macchietto. Formal verification of batch processing control procedures. In *Proc. PSE'94*, page 469. 1994.
27. I. Moon, G. J. Powers, J. R. Burch, and E. M. Clarke. Automatic verification of sequential control systems using temporal logic. *AIChE Journal*, 38(1):67–75, 1992.
28. L. Paulevé, G. Andrieux, and H. Koeppl. Under-approximating cut sets for reachability in large scale automata networks. In N. Sharygina and H. Veith, editors, *Computer Aided Verification*, volume 8044 of *Lecture Notes in Computer Science*, pages 69–84. Springer Berlin Heidelberg, 2013.
29. L. Paulevé, M. Magnin, and O. Roux. Static analysis of biological regulatory networks dynamics using abstract interpretation. *Mathematical Structures in Computer Science*, 22(04):651–685, 2012.
30. L. Paulevé and A. Richard. Static analysis of boolean networks based on interaction graphs: a survey. *Electronic Notes in Theoretical Computer Science*, 284:93 – 104, 2011. Proceedings of The Second International Workshop on Static Analysis and Systems Biology (SASB 2011).
31. E. Plathe, T. Mestl, and S. Omholt. Feedback loops, stability and multistationarity in dynamical systems. *Journal of Biological Systems*, 3:569–577, 1995.
32. S. Prajna. Barrier certificates for nonlinear model validation. *Automatica*, 42(1):117–126, 2006.
33. S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *Hybrid Systems: Computation and Control*, pages 477–492. Springer, 2004.
34. S. Prajna, A. Jadbabaie, and G. J. Pappas. A framework for worst-case and stochastic safety verification using barrier certificates. *Automatic Control, IEEE Transactions on*, 52(8):1415–1428, 2007.
35. A. Richard. Negative circuits and sustained oscillations in asynchronous automata networks. *Advances in Applied Mathematics*, 44(4):378–392, 2010.

36. O. Sahin, H. Frohlich, C. Lobke, U. Korf, S. Burmester, M. Majety, J. Mattern, I. Schupp, C. Chaouiya, D. Thieffry, A. Poustka, S. Wiemann, T. Beissbarth, and D. Arlt. Modeling ERBB receptor-regulated G1/S transition to find novel targets for de novo trastuzumab resistance. *BMC Systems Biology*, 3(1), 2009.
37. R. Samaga, A. V. Kamp, and S. Klamt. Computing combinatorial intervention strategies and failure modes in signaling networks. *Journal of Computational Biology*, 17(1):39–53, 2010.
38. S. Schwoon. Mole. `http://www.lsv.ens-cachan.fr/~schwoon/tools/mole/`.
39. I. Shmulevich, E. R. Dougherty, S. Kim, and W. Zhang. Probabilistic boolean networks: a rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18(2):261–274, feb 2002.
40. G. Stoll, E. Viara, E. Barillot, and L. Calzone. Continuous time boolean modeling for biological signaling: application of gillespie algorithm. *BMC Syst Biol*, 6(1):116, 2012.
41. D. Thieffry and R. Thomas. Dynamical behaviour of biological regulatory networks–ii. immunity control in bacteriophage lambda. *Bulletin of mathematical biology*, 57:277–97, 1995 Mar 1995.
42. R. Thomas. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, 42(3):563 – 585, 1973.
43. R. Thomas and R. d'Ari. *Biological Feedback*. CRC Press, 1990.

A Examples of Local Causality Graphs

Figure 4 gives examples of Local Causality Graphs (section 4.2) for approximation reachability of a_2 in the AN of figure 1. The left LCG does not satisfy the necessary condition (no local paths from c_2 to c_0), hence a_2 is not reachable from the given initial state$\langle a_1, b_0, c_2\rangle$. The middle LCG does satisfy the necessary condition. And the right LCG is a valid sub-LCG for the sufficient condition for a_2 reachability.

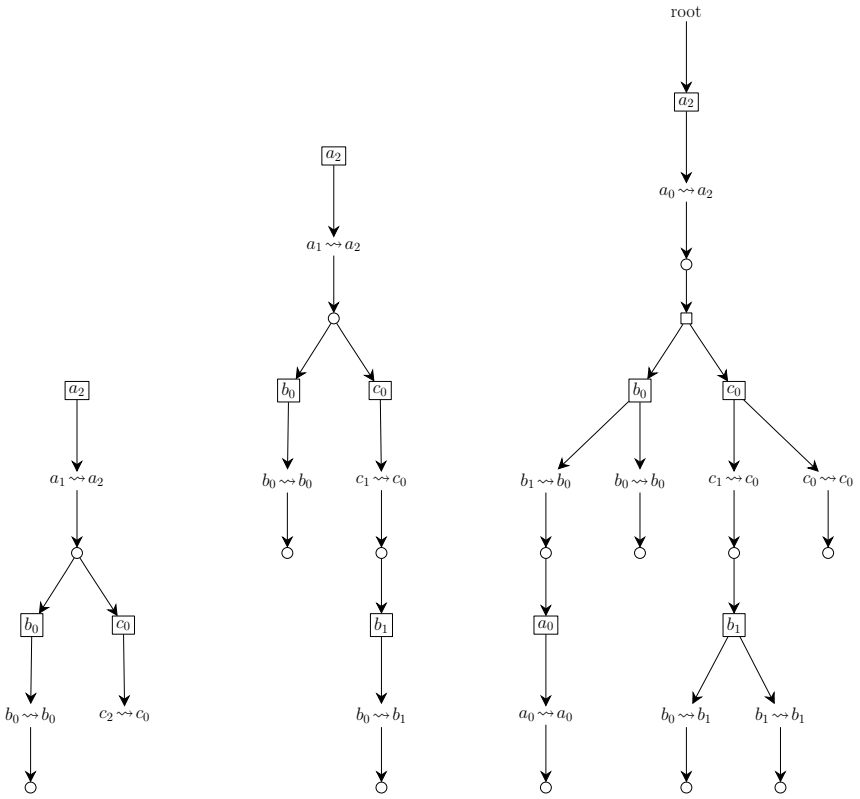


Figure 4. Exemples of Local Causality Graphs for (left) over-approximation of a_2 reachability from $\langle a_1, b_0, c_2\rangle$ (middle) over-approximation of a_2 reachability from $\langle a_1, b_0, c_1\rangle$ (right) under-approximation of a_2 reachability from $\langle a_0, b_1, c_1\rangle$. The small circles represent the local paths.

B ASP implementation of Over- and Under-approximation of Reachability

B.1 OA($s \to^* s'$): necessary condition for reachability

We propose here a possible encoding of the necessary condition for reachability in ANs outlined in section 4.2 and introduced in [29]. Starting from $s_u(g) = g_0$, the analysis starts with the *local paths* of the *objective* $g_0 \rightsquigarrow g_1$: g_1 is reachable only if all the conditions of the transitions of at least one local path $\eta \in \mathsf{local\text{-}paths}(g_0 \rightsquigarrow g_1)$ are reachable. This recursive reasoning can be modelled with a graph relating dependencies between objectives, local paths, and local states.

The local paths computed *a priori* are used to generate the template declaration of the directed edges of the LCG `oa_lcg(G,Parent,Child)` from each possible objective $a_i \rightsquigarrow a_j$. If $\mathsf{local\text{-}paths}(a_i \rightsquigarrow a_j) = \emptyset$, the objective $a_i \rightsquigarrow a_j$ is linked to a node `bottom`:

1 `oa_lcg(G,obj(`a`,`i`,`j`),bottom)` ← `oa_lcg(G,_,obj(`a`,`i`,`j`)).`

otherwise, for $\mathsf{local\text{-}paths}(a_i \rightsquigarrow a_j) = \{\eta^1, \ldots, \eta^n\}$, we declare a node `lpath` for each different local path $m \in \{1, \ldots, n\}$ as a child of $a_i \rightsquigarrow a_j$:

2 `oa_lcg(G,obj(`a`,`i`,`j`),lpath(obj(`a`,`i`,`j`),`m`))` ← `oa_lcg(G,_,obj(`a`,`i`,`j`)).`

then, for each different local state $b_k \in \widetilde{\eta^m}$ in the conditions of the local transitions of η^m, we add an edge from the `lpath` node to `ls(`b`,`k`)`:

3 `oa_lcg(G,lpath(obj(`a`,`i`,`j`),`m`),ls(`b`,`k`))` ← `oa_lcg(G,_,obj(`a`,`i`,`j`)).`

In the case when the local path requires no condition ($\widetilde{\eta^m} = \emptyset$, this can happen when the objective is trivial, i.e., $a_i \rightsquigarrow a_i$, or when the local transitions do not dependent on the other automata), we link the `lpath` to a node `top`:

4 `oa_lcg(G,lpath(obj(`a`,`i`,`j`),`m`),top)` ← `oa_lcg(G,_,obj(`a`,`i`,`j`)).`

A LCG G for over-approximation is parameterized with a state s_G: if a local path has a local state a_j in its transition conditions, the node `ls(`a`,`j`)` is linked, in G, to the node for the objective $a_i \rightsquigarrow a_j$ (line 5), with $a_i = s_G(a)$. It is therefore required that state s_G defines a (single) local state for each automaton referenced in G (line 6).

5 `oa_lcg(G,ls(A,I),obj(A,J,I))` ← `oa_lcg(G,_,ls(A,I)), s(G,A,J).`
6 `1 { s(G,A,J) : ls(A,J) } 1` ← `oa_lcg(G, _, ls(A, _)).`

The necessary condition for reachability is then declared using the predicate `oa_valid(G,N)` which is true if the node `N` satisfies the following condition: it is not `bottom` (line 7); and, in the case of a local state or objective node, one of its children is `oa_valid` (lines 8 and 9; or in the case of a local path, either `top` is its child, or all its children (local states) are `oa_valid` (lines 10 and 11).

7 ← `oa_valid(G,bottom).`
8 `oa_valid(G,ls(A,I))` ← `oa_lcg(G,ls(A,I),X),oa_valid(G,X).`
9 `oa_valid(G,obj(A,I,J))` ← `oa_lcg(G,obj(A,I,J),X),oa_valid(G,X).`
10 `oa_valid(G,N)` ← `oa_lcg(G,N,top).`
11 `oa_valid(G,lpath(obj(`a`,`i`,`j`),`m`))` ← $\bigwedge_{b_k \in \widetilde{\eta^m}}$ `oa_valid(G,ls(`b`,`k`)).`

B.2 UA($s \rightarrow^* s'$): sufficient condition for reachability

We give here a declarative implementation of the sufficient condition for reachability in ANs outlined in section 4.2 and introduced in [15]. The under-approximation consists in building a graph relating objectives, local paths, and local states which satisfies several constraints. If such a graph exists, then the related reachability property is true. Similarly to (I1$^{\#}$), we give template declarations for the edges with the predicate `ua_lcg(G,Parent,Child)`. We assume that the reachability property is specified by adding an edge from `root` to `ls(`a`,`i`)` for each local state to reach.

The graph `ua_lcg` is parameterized with a *context* which is a set of local states, declared with the predicate `ctx(G,A,J)`. Every local states a_i of the graph that are not part of the reachability specification belong to that context (line 12); and are linked to the objective $a_j \rightsquigarrow a_i$ for each a_j in the context (line 13).

```
12 ctx(G,A,I) ← ua_lcg(G,N,ls(A,I)), N != root.
13 ua_lcg(G,ls(A,I),obj(A,J,I)) ← ua_lcg(G,_,ls(A,I)), ctx(G,A,J).
```

A first constraint is that each objective in the graph is linked to one and only one of its local path. Therefore, objectives without local paths ($\mathsf{local\text{-}paths}(a_i \rightsquigarrow a_j) = \emptyset$) cannot be included (line 14), for the others, a choice has to be made among $\mathsf{local\text{-}paths}(a_i \rightsquigarrow a_j) = \{\eta^1, \ldots, \eta^n\}$ (line 15).

```
14 ← ua_lcg(G,_,obj(a,i,j)).
15 1 { ua_lcg(G,obj(a,i,j),lpath(obj(a,i,j),1..n)) } 1 ← ua_lcg(G,_,obj(a,i,j)).
```

As for `oa_lcg`, each local path is linked to all the local states composing its transition conditions: for each $m \in \{1, \ldots, n\}$, for each $b_k \in \widetilde{\eta^m}$,

```
16 ua_lcg(G,lpath(obj(a,i,j),m),ls(b,k)) ← ua_lcg(G,_,obj(a,i,j)).
```

The graph has to be acyclic. This is declared using a predicate `conn(G,X,Y)` which is true if the node `X` is connected (there is a directed path) to `Y` (line 17). A graph is cyclic when `conn(G,X,X)` (line 18).

```
17 conn(G,X,Y) ← ua_lcg(G,X,Y).  conn(G,X,Y) ← ua_lcg(G,X,Z), conn(G,Z,Y).
18 ← conn(G,X,X).
```

Then, if the node for an objective $a_i \rightsquigarrow a_j$ is connected to a local state a_k, the under-approximation requires $a_i \rightsquigarrow a_j$ to be connected with $a_k \rightsquigarrow a_j$ (assuming that a has at least 3 local states, definition not shown):

```
19 ua_lcg(G,obj(A,I,J),obj(A,K,J)) ← not boolean(A), conn(G,obj(A,I,J),ls(A,K)).
```

When a local transition is conditioned by at least two other automata (for instance $c_\circ \xrightarrow{a_i,b_j} c_\bullet$), the under-approximation requests that reaching b_j does not involve other local states from a others that a_i. This is stated by the `indep(G,Y,`a`,`i`,ls(`b`,`j`))` which cannot be true if b_j is connected to a local state a_k with $k \neq i$ line 20. Then, the under-approximation requires that at most one `indep` predicate is false, for a given LCG G and a given local path Y (line 21). Such an independence should also hold between the local states of the reachability specification (line 22).

```
20 indepfailure(Y,ls(A,I)) ← indep(G,Y,A,I,N), conn(G,N,ls(A,K)), K!=I.
21 ← indepfailure(Y,N),indepfailure(Y,M),M!=N.
22 indep(G,root,A,I,ls(B,J)) ← ua_lcg(G,root,ls(A,I)),ua_lcg(G,root,ls(B,J)),B != A.
```

For $\eta^m \in \mathsf{local\text{-}paths}(a_i \rightsquigarrow a_j)$, for each local transition $a_\circ \xrightarrow{\ell} a_\bullet \in \eta^m$, for each couple of different local states in its condition $b_k, c_l \in \ell, b_k \neq c_l$:

```
23 indep(G,lpath(obj(a,i,j),m),b,k,ls(c,l)) ← ua_lcg(G,_,lpath(obj(a,i,j),m)).
```

C ASP implementation of reachability in unfoldings

A (prefix of an) unfolding is an acyclic bipartite digraph where nodes are either *events* (application of a transition) or *conditions* (change of local state) [13]. We use the predicate `post(X,Y)` to denote an edge from X to Y; and `h(C,ls(A,I))` to denote that the condition C corresponds to the local state a_i. Figure 5 shows an example of unfolding.

A state s belongs to the prefix if it is possible to build a *configuration* such that all the local states in s have a unique corresponding condition on the *cut* of the configuration (line 1).

A configuration is a set of events, and we use `e(E)` to denote that the event E belongs to the configuration. By definition, if E is in a configuration, all its parent events are in the configuration (line 2). There should be no *conflicts* between two events of a configuration: two events are in conflict if they share a common parent condition (line 3).

A condition is on the cut if its parent event is in the configuration (line 4), and none of its children event is in the configuration (line 5).

```
1 1 { cut(C)  : h(C,ls(A,I)) } 1 ← reach(A,I).
2 e(F) ← post(F,C),post(C,E),e(E).
3 ← post(C,E),post(C,F),e(E),e(F),E != F.
4 e(E) ← cut(C),post(E,C).
5 ← cut(C),post(C,E),e(E).
```

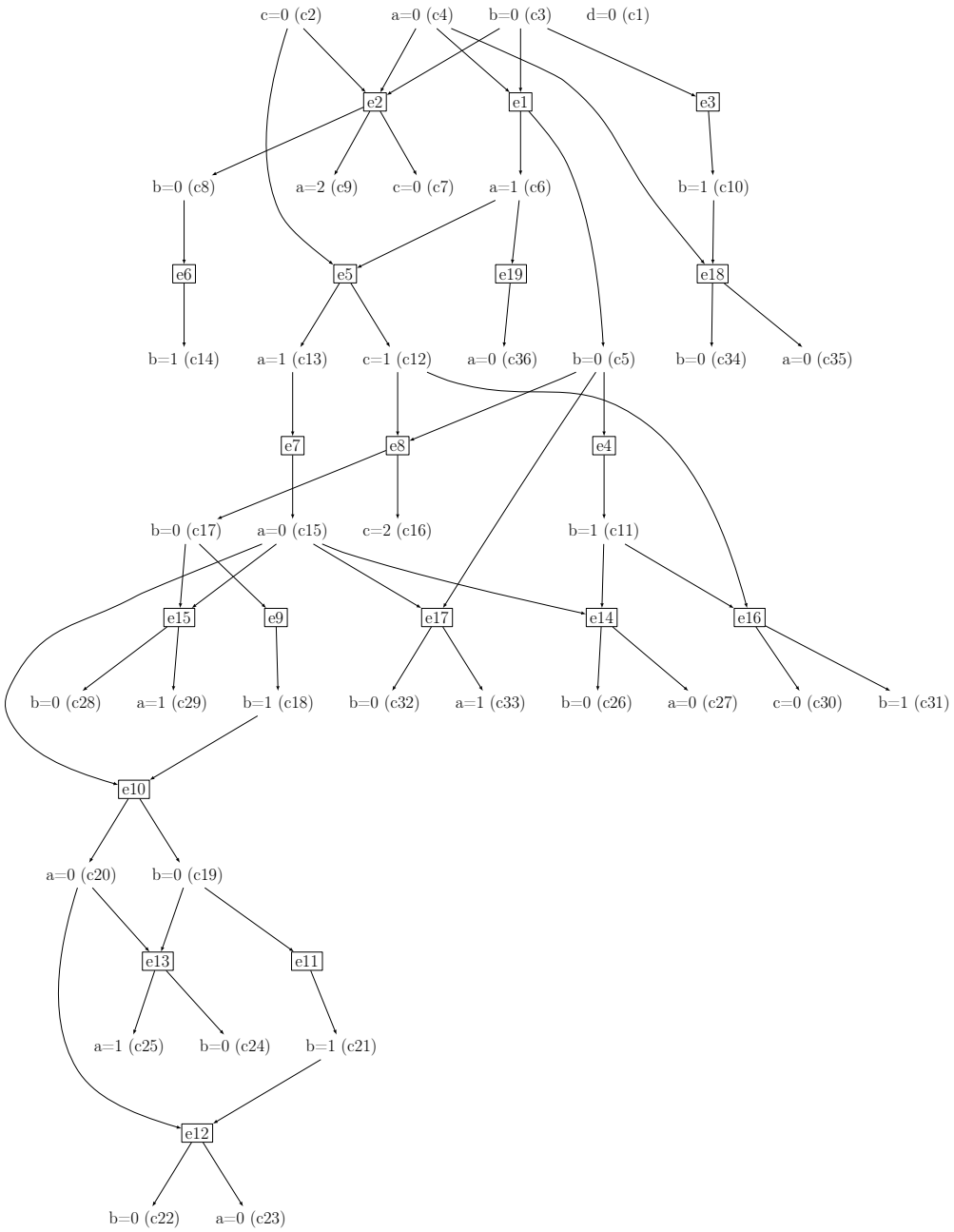


Figure 5. Unfolding of the AN of figure 1. Events are boxed nodes, conditions have no borders and indicate both the automata local state and the condition identifier.

Metabolic Pathways as Temporal Logic Programs

Jean-Marc Alliot[1] and Martín Diéguez[2⋆] and Luis Fariñas del Cerro[1]

[1]Institut de recherche en informatique de Toulouse, Toulouse University
[2]Centre International de Mathématiques et d'Informatique, Toulouse University

Abstract. Metabolic Networks, formed by series of metabolic pathways, are made of intracellular and extracellular reactions that determine the biochemical properties of a cell and by a set of interactions that guide and regulate the activity of these reactions. Cancers, for example, can sometimes appear in a cell as a result of some pathology in a metabolic pathway. Most of these pathways are formed by an intricate and complex network of chain reactions, and they can be represented in a human readable form using graphs which describe the cell signaling pathways. In this paper we present a logic, called Molecular Equilibrium Logic, a nonmonotonic logic which allows representing metabolic pathways. We also show how this logic can be presented in terms of a syntactical subset of Temporal Equilibrium Logic, the temporal extension of Equilibrium Logic, called Splittable Temporal Logic Programs.

1 Introduction

Molecular Interaction Maps [20], formed by a series of metabolic pathways, are made of intracellular and extracellular reactions that determine the biochemical properties of a cell by consuming and producing proteins, and by a set of interactions that guide and regulate the activity of these reactions. These reactions are at the center of a cell's existence, and are regulated by other proteins, which can either activate these reactions or inhibit them. These pathways form an intricate and complex network of chain reactions, and can be represented using graphs. *Molecular Interaction Maps* (MIM's) [1] are such a representation, and it is possible to write these graphs using editors such as Pathvisio [17] (which outputs its own XML representation) or System Biology Markup Language (SBML) [2] editors.

These graphs can become extremely large, and although essential for knowledge capitalization and formalization, they are difficult to use because: (1) Reading is complex due to the very large number of elements, and reasoning is even more difficult; (2) Using a graph to communicate goals is only partially suitable because the representation formalism requires expertise; (3) Graphs often contain implicit knowledge, that is taken for granted by one expert, but is missed by another one.

⋆ Martín Diéguez was supported by the Centre international de mathématiques et d'informatique (contract ANR-11-LABX-0040-CIMI)

Our aim consists in providing a logical framework that helps users to detect possible inconsistencies as well as reasoning on such kind of maps. We have chosen to use Pathvisio and its XML representation as our editor/representation of choice for representing these graphs, but this work could be extended to SBML and SBML editors. In [11], we modelled a restricted subclass of MIM's in terms of first-order logic with equality. This work was simplified into propositional logic in [6], which enabled to use all propositional calculus tools such as solving abductive queries on MIM's. Unfortunately that representation was unable to express the temporal properties of MIM, which are implicit in the formalisations. So we extended our work with temporal logic in [5]. This representation was enhanced with a naive approach to abductive temporal reasoning by assuming bounded time and the so-called *closed world assumption* [27], a concept tightly connected with Logic Programming and Non-Monotonic reasoning[1]. The use of non-monotonicity allows us to use defaults and inertia rules to express things like "a protein remains in the environment if it is not used in one reaction", which greatly enhances our temporal descriptions.

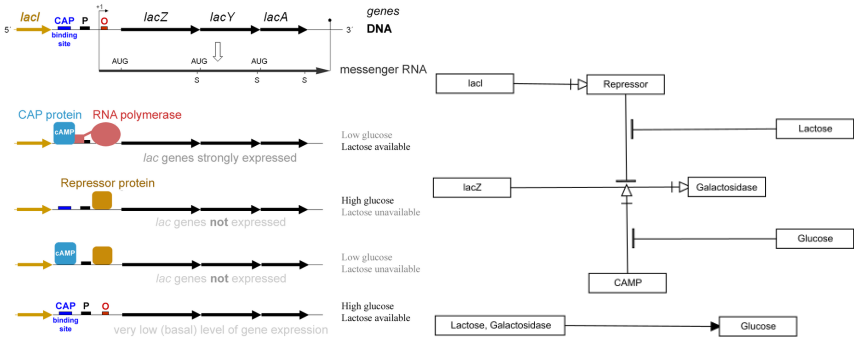
In order to incorporate such kind of defaults and to justify the use of the closed world assumption in [5], we present in this paper Molecular Equilibrium Logic (MEL), a reformulation of the temporal version of Molecular Interaction Logic [5] in terms of Equilibrium Logic [25], a well-known logical characterisation of *Stable Models* [15] and *Answer Sets* [9]. Moreover we show the existence of a connection between MEL and Temporal Equilibrium Logic (TEL) [3], the temporal extension of the Equilibrium Logic. By going one step further, we show that MEL can be encoded in a syntactic subclass of TEL called splittable temporal logic programs (STLP's) [4], which allows us to capture the set of Molecular Equilibrium Models (explained in Section 4) in terms of a Linear Time Temporal Logic (LTL) formula [22] (see Section 7).

The rest of this paper is organized as follows: Section 2 presents several biological concepts used along this paper as well as describes the problems to solve in layman's words and with a simple example. Section 3 describes the concepts of production and regulation which are the basic operations present in a MIM. Sections 4 and 5 respectively describe two different semantics based on equilibrium logic: Molecular Equilibrium Logic and Temporal Equilibrium Logic. The former is capable of describing general pathways while the latter is the best-known temporal extension of Equilibrium Logic. In Section 6 we stablish the relation between the two aforementioned formalisms, which is studied in detail in Section 7 where we prove that the Equilibrium Models of our temporal theories can be expressed in Linear Time Temporal Logic [22] via Temporal Completion [4,10].

[1] Regarding non-monotonic approaches to model biological systems, there are several contributions in the area of *Answer Set Programming* [9,28], *action languages* [29] or *Inductive Logic Programming* [12]. In these contributions the temporal behaviour is considered in [29] but both representation and query languages are different.

2 A simple classical example

In this section we introduce the example of the regulation of the *lac* operon [19][23], which will be used and developed in the rest of this paper . The lac operon (lactose operon) is an operon required for the transport and metabolism of lactose in many bacteria. Although glucose is the preferred carbon source for most bacteria, the lac operon allows for the effective digestion of lactose when glucose is not available. The lac operon is a sequence of three genes (lacZ, lacY and lacA) which encode 3 enzymes. Then, these enzyms carry the transformation of lactose into glucose. We will concentrate here on lacZ. LacZ encodes the β-galactosidase which cleaves lactose into glucose and galactose. The lac operon uses a two-part control mechanism to ensure that the cell expends energy producing the enzymes encoded by the lac operon only when necessary. First, in the absence of lactose, the lac repressor halts production of the enzymes encoded by the lac operon. Second, in the presence of glucose, the catabolite activator protein (CAP), required for production of the enzymes, remains inactive.



(a) The Lac Operon (b) MIM representing the Lac Operon

Fig. 1. Graphical and MIM representation of the Lac Operon.

Figure 1(a) describes this regulatory mechanism. The expression of lacZ gene is only possible when RNA polymerase (pink) can bind to a promotor site (marked P, black) upstream the gene. This binding is aided by the cyclic adenosine monophosphate (cAMP in blue) which binds before the promotor on the CAP site (dark blue). The lacI gene (yellow) encodes the repressor protein

[2] The Nobel prize was awarded to Monod, Jacob and Lwoff in 1965 partly for the discovery of the lac operon by Monod and Jacob [18], which was the first genetic regulatory mechanism to be understood clearly, and is now a "standard" introductory example in molecular biology classes.

[3] A less formal explanation can be found in `https://en.wikipedia.org/wiki/Lac_operon`

LacI (yellow) which binds to the promotor site of the RNA polymerase when lactose is not available, preventing the RNA polymerase to bind to the promoter and thus blocking the expression of the following genes (lacZ, lacY and lacA): this is a *negative regulation*, or *inhibition*, as it blocks the production of the proteins. When lactose is present, the repressor protein LacI binds with lactose and is converted to allolactose, which is not able to bind to the promotor site, thus enabling RNA polymerase to bind to the promotor site and to start expressing the lacZ gene if cAMP is bound to CAP. cAMP is on the opposite a *positive regulation*, or an *activation*, as its presence is necessary to express the lacZ gene. However, cAMP is itself regulated negatively by glucose: when glucose is present, the concentration of cAMP becomes low, and thus cAMP does not bind to the CAP site, blocking the expression of lacZ. In this figure, we have three kinds of entities which have different initial settings and temporal dynamics:

- lacI, lacZ and cAMP are initial external conditions of the model and they do not evolve in time.
- galactosidase and the repressor protein can only be produced inside the graph, and are always absent at the start (time 0) of the modeling. Their value will then evolve in time according to the processes described by the graph.
- glucose and lactose also evolve in time (like galactosidase and the repressor protein) according to the processes described by the graph, but they are also initial conditions of the system, and can either be present or absent at time 0, like lacI, lacZ and cAMP.

So, an entity must be classified according to two main characteristics: **C1:** It can evolve in time according to the cell reactions (appear and disappear), or it can be fixed, such as a condition which is independent of the cell reactions (temperature, protein always provided in large quantities by the external environment, etc...). **C2:** It can be an initial condition of the cell model (present *or* absent at the beginning of the modeling), or can *only* be produced by the cell. There are thus three kinds of entities, which have three kind of behaviour:

Exogenous entities: an *exogenous* entity satisfies $C1$ and $\neg C2$; their status *never* change through time: they are set once and for all by the environment or by the experimenter at the start of the simulation; the graph never modifies their value, and if they are used in a reaction, the environment will always provide "enough" of them.

Pure endogenous entities: on the opposite, a *pure endogenous* entity satisfies $\neg C1$ and $C2$; their status evolves in time and is set *only* by the dynamic of the graph. They are absent at the beginning of the reaction, and can only appear if they are produced inside the graph.

Weak endogenous entities: *weak endogenous* entities satisfy $C2$ and $C1$; they can be present or absent at the beginning of the process (they are initial conditions of the model), however their value after the start of the process is entirely set by the dynamic of the graph. So they roughly behave like *pure endogenous* entities, but the initial condition can be set by the experimenter.

The status of a protein/condition is something which is set by the biologist, regarding his professional understanding of the biological process described by the graph[4]. However a rule of thumb is that exogenous entities are almost never produced inside the graph (they never appear at the right side of a production arrow), while endogenous entities always appear on the right side of a production arrow (but they can also appear on the left side of a production rule, especially weak endogenous entities). These distinctions are fundamental, because the dynamics of these entities are different and they will have to be formalized differently.

3 Fundamental operations

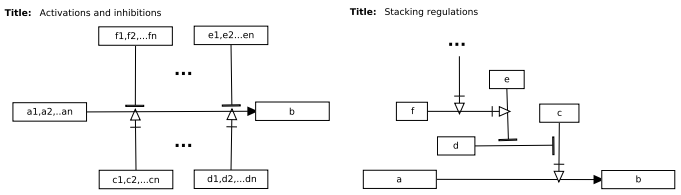
The mechanism described in the previous section is summarized in the simplified graph in Figure 1(b). This example contains all the relationship operators that will be used in the rest of this document. In order to make their presentation clearer, we will distinguish between productions and regulations:

Productions can take two different forms, depending on whether the reactants are consumed by the reactions or not: In Figure 1(b), lactose and galactosidase produce glucose, and are consumed while doing so, which is thus noted $(galactosidase, lactose \twoheadrightarrow glucose)$. On the opposite, the expression of the lacZ gene to produce galactosidase (or of the lacl gene to produce the Lacl repressor protein) does not consume the gene, and we have thus $(lacZ \twoheadrightarrow galactosidase)$. Generally speaking, If the reaction consumes completely the reactant(s) we write: $a_1, a_2, \cdots, a_n \twoheadrightarrow b$ while if the reactants are not consumed by the reaction, we write $a_1, a_2, ...a_n \twoheadrightarrow b$. In the former representation the production of b completely consumes $a_1, a_2...a_n$ whereas in the latter $a_1, a_2...a_n$ are not consumed when b is produced.

Regulations can also take two forms: every reaction can be either *inhibited* or *activated* by other proteins or conditions. In the Diagram of Figure 1(b), the production of galactosidase from the expression of the lacZ gene is activated by cAMP (we use $cAMP \twoheadrightarrow$ to express activation). At the same time the same production of galactosidase is blocked (or inhibited) by the Lacl repressor protein (noted $Repressor \dashv$).

Generally speaking, we write $a_1, a_2, ...a_n \twoheadrightarrow$ if the simultaneous presence of $a_1, a_2, ...a_n$ activates a production or another regulation. Similarly we write $a_1, a_2, ...a_n \dashv$ if the simultaneous presence of $a_1, a_2, ...a_n$ inhibits a production or another regulation. On Figure 2(a), we have a summary of basic inhibitions/activations on a reaction: the production of b from $a_1, \cdots, a_n$ is activated by the simultaneous presence of $c_1, \cdots, c_n$ **or** by the simultaneous presence of $d_1, \cdots, d_n$, and inhibited by the simultaneous presence of $e_1, \cdots, e_n$ **or** by the simultaneous presence of $f_1, \cdots, f_n$. These regulations are often "stacked", on many

[4] It is important here to notice that lactose can be either considered as a weak endogenous variable, or as an exogenous variable if we consider that the environment is always providing "enough" lactose. It is a simple example which shows that variables in a graph can be interpreted differently according to what is going to be observed.



(a) Activations/Inhibitions (b) Stacking

Fig. 2. Examples of activations and inhibitions and stacking contexts.

levels (see Figure 2(b)). For example in Figure 1(b), the inhibition by the LacI repressor protein of the production of galactosidase can itself be inhibited by the presence of lactose, while the activation of the same production by cAMP is inhibited by the presence of glucose.

A final word of warning is necessary. Graphs pragmatically describe sequences of operations that biologists find important. They are only a model of some of the biological, molecular and chemical reactions that take place inside the cell; they can also be written in many different ways, depending on the functional block or operations that biologists want to describe, and some relationships are sometimes simply left out because they are considered not important for the function which is described in a particular graph.

4 Molecular Equilibrium Logic

In this section we introduce *Molecular Equilibrium Logic*. The syntax of this logic consists of two elementary building blocks: *pathway context* and *pathway formula*. The former corresponds to the representation of the activation and inhibition conditions while the latter allows representing the production of new substances (see Section 3). A *pathway context* is formed by expressions defined by the following grammar:

$$\alpha ::= \langle \{\alpha_1, \cdots, \alpha_n\} P \twoheadrightarrow, \{\alpha_{n+1}, \cdots, \alpha_{n+m}\} Q \dashv \rangle,$$

where P and Q are sets (finite and possibly empty) of propositional variables representing the conditions of activation ($\twoheadrightarrow$) and inhibition ($\dashv$) of the reaction. Every context can be associated with a (possibly empty) set of activation (α_i, with $1 \leq i \leq n$) and inhibition (α_j, with $1 \leq j \leq m$) contexts. One or both sets can be empty. Broadly speaking, the context associated with a pathway formula represents the set of substances that must be present or absent in order to make the reaction possible. As an example of context, let us consider the example of the Lac Operon, whose graph is displayed in Figure 1(b). The context associated with the production rule $lacZ \twoheadrightarrow Galactosidase$ corresponds to the following expression:

$$\gamma = \langle\{\langle\varnothing\twoheadrightarrow, \{Glucose\}\dashv\rangle\}\{CAMP\}\twoheadrightarrow,\{\langle\varnothing\twoheadrightarrow,\{Lactose\}\dashv\rangle\}\{Repressor\}\dashv\rangle. \quad (1)$$

A *Pathway formula* is a rule built from the grammar $F ::= [\alpha](P^{\wedge} \multimap q) \mid F \wedge F$, where α represents a context, $\multimap \in \{\rightarrow, \twoheadrightarrow\}$, $P^{\wedge}$ stands for a conjunction of all atoms in the finite set P and q corresponds to a propositional variable. Regarding our running example, which is shown in Figure 1(b), consists of three different pathways, each of them corresponds to one of the following pathway formulas:[5]

$$[\langle\varnothing\varnothing \twoheadrightarrow, \varnothing\varnothing \dashv\rangle](Lactose, Galactosidase \rightarrow Glucose) \quad (2)$$
$$[\langle\varnothing\varnothing \twoheadrightarrow, \varnothing\varnothing \dashv\rangle](lacI \twoheadrightarrow Repressor) \quad (3)$$
$$[\gamma](lacZ \rightarrow Galactosidase). \quad (4)$$

From a biological point of view, substances can be created or destroyed by reactions that might take place in parallel. Therefore, we must take into account situations where a protein is produced and consumed at the same time or where a protein remains present because it was not involved in a reaction which would have consumed it. We model this aspect by extending the set of propositional variables Σ to the set $\widehat{\Sigma} = \Sigma \cup \{\mathbf{Pr}(p_1), \cdots, \mathbf{Pr}(p_n)\} \cup \{\mathbf{Cn}(p_1), \cdots, \mathbf{Cn}(p_n)\}$, where $p_1, \cdots, p_n$ are either a weak or pure endogenous variables. Informally speaking, every atom of the form $\mathbf{Pr}(p)$ means that p is produced as a result of a chemical reaction while $\mathbf{Cn}(p)$ means that the reactive p is consumed in a reaction. Regarding our running example, we notice that the production of *Glucose* implies that *Galactosidase* is consumed. However, *Lactose*, as an exogenous variable is never consumed. From now on, we will use the symbols Σ and $\widehat{\Sigma}$ referring to, respectively, the signature (set of entities occurring in a MIM) and its corresponding extension.

The semantics of MEL is based on the monotonic logic of Molecular Here and There (MHT) plus a minimisation criterion among the Here and There models. Given a set of propositional variables Σ we define a *Molecular Here and There*[6] *interpretation* $\mathbf{M}$ as an infinite sequence of pairs $m_i = \langle H_i, T_i\rangle$ with $i = 0, 1, 2, \ldots$ where $H_i \subseteq T_i \subseteq \widehat{\Sigma}$ satisfying the following properties: for all endogenous variable $p \in \Sigma$ and for all exogenous variable $q \in \Sigma$ and for all $i \geq 0$,

(A) if $\mathbf{Pr}(p) \in H_i$ then $p \in H_{i+1}$;
(B) if $p \in H_i$ and $\mathbf{Cn}(p) \notin T_i$ then $p \in H_{i+1}$;
(C) if $\mathbf{Pr}(p) \in T_i$ then $p \in T_{i+1}$;
(D) if $p \in T_i$ and $\mathbf{Cn}(p) \notin T_i$ then $p \in T_{i+1}$;
(E) if $q \in H_i$ then $q \in H_{i+1}$;
(F) if $q \in T_i$ then $q \in T_{i+1}$.

[5] Notice that only the pathway formula associated with the production of *Galactosidase* has an associated context, defined in (1), while the rest of pathway formulas have an empty context .

[6] Here and There [16] is an intermediate logic which severs as a monotonic basis for the Equilibrium Models [25], a logical characterisation of the Stable Model semantics [15]

For simplicity, given a MHT interpretation, we write $\mathbf{H}$ (resp. $\mathbf{T}$) to represent the sequence of pair components $H_0, H_1, \ldots$ (resp. $T_0, T_1, \ldots$). Using this notation, we will sometimes abbreviate the interpretation as $\mathbf{M} = \langle \mathbf{H}, \mathbf{T} \rangle$. If $\mathbf{H} = \mathbf{T}$, we will call $\mathbf{M}$ *total* model.

Before presenting the satisfaction relation we introduce the *activation* ($\mathcal{A}(\alpha)$) and *inhibition* ($\mathcal{I}(\alpha)$) expressions associated with a pathway context $\alpha = \langle \{\alpha_1, \cdots, \alpha_n\} P \rightarrow, \{\beta_{n+1}, \cdots, \beta_{n+m}\} Q \dashv \rangle$. Informally speaking, $\mathcal{A}(\alpha)$ characterizes when the context α is active while $\mathcal{I}(\alpha)$ describes when it is inhibited. These expressions, which will be used in the definition of the satisfaction relation, are defined as follows:

$$\mathcal{A}(\alpha) = \bigwedge_{p \in P} p \wedge \bigwedge_{i=1}^{n} \mathcal{A}(\alpha_i) \wedge \left(\bigvee_{q \in Q} \neg q \vee \bigwedge_{j=n+1}^{m} \mathcal{I}(\beta_j) \right)$$

$$\mathcal{I}(\alpha) = \bigvee_{p \in P} \neg p \vee \bigvee_{i=1}^{n} \mathcal{I}(\alpha_i) \vee \left(\bigwedge_{q \in Q} q \wedge \bigwedge_{j=n+1}^{m} \mathcal{A}(\beta_j) \right).$$

If one part of the context α is empty, then the corresponding part is of course absent in $\mathcal{A}(\alpha)$ and $\mathcal{I}(\alpha)$. For instance, the activation and inhibition expressions of the context γ described in (1) correspond to the Boolean expressions: $\mathcal{A}(\gamma) = CAMP \wedge \neg Glucose \wedge (\neg Repressor \vee Lactose)$ and $\mathcal{I}(\gamma) = \neg CAMP \vee Glucose \vee (Repressor \wedge \neg Lactose)$.

Given a MHT interpretation $\mathbf{M}$, $i \geq 0$ and a pathway formula F on Σ, we define the satisfaction relation ($\mathbf{M}, i \vDash F$) as follows:

- $\mathbf{M}, i \vDash p$ iff $p \in H_i$, for any variable $p \in \Sigma$;
- $\mathbf{M}, i \vDash \neg p$ iff $p \notin T_i$, with $p \in \Sigma$;
- disjunction and conjunction are satisfied in usual way;
- $\mathbf{M}, i \vDash [\alpha](P^{\wedge} \twoheadrightarrow q)$ iff for all $\mathbf{H}' \in \{\mathbf{H}, \mathbf{T}\}$ and $j \geq i$, if $\langle \mathbf{H}', \mathbf{T}' \rangle, j \vDash \mathcal{A}(\alpha)$ and $P \subseteq H'_j$, then $\{\mathbf{Pr}(q), \mathbf{Cn}(p) \mid p \in P \text{ an endogenous variable}\} \subseteq H'_j$;
- $\mathbf{M}, i \vDash [\alpha](P^{\wedge} \twoheadrightarrow q)$ iff for all $\mathbf{H}' \in \{\mathbf{H}, \mathbf{T}\}$ and $j \geq i$, if $\langle \mathbf{H}', \mathbf{T}' \rangle, j \vDash \mathcal{A}(\alpha)$ and $P \subseteq H'_j$, then $\mathbf{Pr}(q) \in H'_j$;

As in other equilibrium logic extensions, we relate two MHT models $\mathbf{M} = \langle \mathbf{H}, \mathbf{T} \rangle$ and $\mathbf{M}' = \langle \mathbf{H}', \mathbf{T}' \rangle$ as follows: $\mathbf{M}' \leq \mathbf{M}$ iff $\mathbf{T} = \mathbf{T}'$ and for all $i \geq 0$ $H'_i \subseteq H_i$. $\mathbf{M}' < \mathbf{M}$ if $\mathbf{M}' \leq \mathbf{M}$ and $\mathbf{M}' \neq \mathbf{M}$. We say that a MHT interpretation $\mathbf{M}$ is a *Molecular Equilibrium Model* of a set of pathway formulas Γ iff $\mathbf{M}$ is total, $\mathbf{M}, 0 \vDash \Gamma$ an there is no $\mathbf{M}'$ such that $\mathbf{M}' < \mathbf{M}$ such that $\mathbf{M}', 0 \vDash \Gamma$.

5 Temporal Equilibrium Logic

Temporal Equilibrium Logic (TEL) [3] extends Equilibrium Logic [25][7] with temporal operators from Linear Time Temporal Logic [22]. TEL can also be seen

[7] Modal extensions of Equilibrium Logic and the logic of Here and There can be considered as promising lines of research which lead to several remarkable results, among others, [7,13].

as a temporal extension of the stable models semantics [15] for logic programming. This formalism is very suitable for representing the temporal behaviour of biological systems, since the use of the laws of inertia allows us to avoid the specification of the large number of frame axioms [24] that should be considered in the representation. The TEL formulas we will consider along this paper are built from the following grammar:

$$\varphi ::= \bot \mid p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \rightarrow \varphi_2 \mid \bigcirc\varphi_1 \mid \Box\varphi_1 \mid \Diamond\varphi_2,$$

where φ_1 and φ_2 are also temporal formulas. Regarding the modal operators, $\bigcirc$ is read "next", $\Box$ is read "forever" and $\Diamond$ stands for "eventually" or "at some future point".

The semantics of TEL is defined, in the same spirit as in Equilibrium Logic, in terms of a temporal extension of the logic of Here and There [16], called Temporal Here and There (THT), plus a minimisation criterion among the THT models. We define a *Temporal Here and There interpretation* $\mathbf{M}$ as an infinite sequence of pairs $m_i = \langle H_i, T_i \rangle$ with $i = 0, 1, 2, \ldots$ where $H_i \subseteq T_i \subseteq \widehat{\Sigma}$. For simplicity, given a temporal interpretation, we write $\mathbf{H}$ (resp. $\mathbf{T}$) to represent the sequence of pair components $H_0, H_1, \ldots$ (resp. $T_0, T_1, \ldots$). Using this notation, we will sometimes abbreviate the interpretation as $\mathbf{M} = \langle \mathbf{H}, \mathbf{T} \rangle$. An interpretation $\mathbf{M} = \langle \mathbf{H}, \mathbf{T} \rangle$ is said to be *total* when $\mathbf{H} = \mathbf{T}$. The satisfaction relation $\models$ is interpreted as follows on THT models ($\mathbf{M}$ is a THT model and $k \in \mathbb{N}$):

1. $\mathbf{M}, k \models p$ iff $p \in H_k$, for any $p \in \widehat{\Sigma}$.
2. $\mathbf{M}, k \models \varphi \wedge \psi$ iff $\mathbf{M}, k \models \varphi$ and $\mathbf{M}, k \models \psi$.
3. $\mathbf{M}, k \models \varphi \vee \psi$ iff $\mathbf{M}, k \models \varphi$ or $\mathbf{M}, k \models \psi$.
4. $\mathbf{M}, k \models \varphi \rightarrow \psi$ iff for all $\mathbf{H}' \in \{\mathbf{H}, \mathbf{T}\}$, $\langle \mathbf{H}', \mathbf{T} \rangle, k \not\models \varphi$ or $\langle \mathbf{H}', \mathbf{T} \rangle, k \models \psi$.
5. $\mathbf{M}, k \models \bigcirc\varphi$ iff $\mathbf{M}, k+1 \models \varphi$.
6. $\mathbf{M}, k \models \Box\varphi$ iff for all $j \geq k$, $\mathbf{M}, j \models \varphi$.
7. $\mathbf{M}, k \models \Diamond\varphi$ iff there is $j \geq k$ such that $\mathbf{M}, j \models \varphi$.
8. never $\mathbf{M}, k \models \bot$.

Note that, as happens in Equilibrium logic, $\neg\varphi \stackrel{\text{def}}{=} \varphi \rightarrow \bot$.

Proposition 1. *Let* $\mathbf{M}$ *be a model. For all pathway context* α *and for all* $i \in \mathbb{N}$*, (a)* $\mathbf{M}, i \models_{MHT} \mathcal{A}(\alpha)$ *iff* $\mathbf{M}, i \models_{THT} \mathcal{A}(\alpha)$*; (b)* $\mathbf{M}, i \models_{MHT} \mathcal{I}(\alpha)$ *iff* $\mathbf{M}, i \models_{THT} \mathcal{I}(\alpha)$*;*

Proof. First note that $\mathcal{A}(\alpha)$ and $\mathcal{I}(\alpha)$ are build on the language p, $\neg p$ (with $p \in \Sigma$), $\wedge$ and $\vee$. Second, remark that, regarding the aforementioned language, MHT and THT have the same satisfaction relation (note that when negation only affects to atoms of Σ, $\mathbf{M}, i \models_{\text{THT}} \neg p$ iff $p \notin T_i$ iff $\mathbf{M}, i \models_{\text{MHT}} \neg p$). From all those facts it is easy to prove, by induction, (a) and (b). ∎

A formula φ is THT-*valid* if $\mathbf{M}, 0 \models \varphi$ for any $\mathbf{M}$. An interpretation $\mathbf{M}$ is a THT-*model* of a theory Γ, written $\mathbf{M} \models \Gamma$, if $\mathbf{M}, 0 \models \varphi$, for all formula $\varphi \in \Gamma$. Notice that when we disregard temporal operators, we obtain the logic of HT. On the other hand, if we restrict the semantics to total interpretations, $\langle \mathbf{T}, \mathbf{T} \rangle \models \varphi$ corresponds to satisfaction of formulas $\mathbf{T} \models \varphi$ in LTL. Given two interpretations

$\mathbf{M} = \langle \mathbf{H}, \mathbf{T} \rangle$ and $\mathbf{M}' = \langle \mathbf{H}', \mathbf{T}' \rangle$ we say that $\mathbf{M}'$ is *lower or equal than* $\mathbf{M}$, written $\mathbf{M}' \leq \mathbf{M}$, when $\mathbf{T}' = \mathbf{T}$ and for all $i \geq 0$, $H'_i \subseteq H_i$. As usual, $\mathbf{M}' < \mathbf{M}$ stands for $\mathbf{M}' \leq \mathbf{M}$ and $\mathbf{M}' \neq \mathbf{M}$. Finally, an interpretation $\mathbf{M}$ is said to be a *temporal equilibrium model* of a theory Γ if $\mathbf{M}$ is a total model of Γ and there is no other $\mathbf{M}' < \mathbf{M}$, such that $\mathbf{M}' \vDash \Gamma$.

6 From Molecular Equilibrium Logic to Temporal Equilibrium Logic

In this section we first show how MEL can be embedded in TEL by providing a translation between their monotonic basis, MHT and THT. The reader might have noticed that the only differences between MHT and THT interpretations are the constraints (A)-(F), which are imposed on the MHT. Those restrictions can be captured in THT by adding the following rule of *inertia*, for any variable $p \in \Sigma$ as follows:

$$inertia(p) \stackrel{\text{def}}{=} \begin{cases} \Box\left((\mathbf{Pr}(p) \vee (p \wedge \neg \mathbf{Cn}(p))) \rightarrow \bigcirc p\right) & if\ p \text{ is an endogenous variable} \\ \Box(p \rightarrow \bigcirc p) & if\ p \text{ is an exogenous variable.} \end{cases}$$

Informally speaking, endogenous variables are true in the next state if they are produced or if they are present and not consumed. On the other hand, exogenous variables are automatically passed to the next state since they are never produced or consumed. They are just present in the environment.

Proposition 2. *Given a signature Σ, let $\mathbf{M}$ be a THT interpretation on $\widehat{\Sigma}$. $\mathbf{M}$ is a MHT model (that is, $\mathbf{M}$ satisfies conditions (A)-(F)) iff $\mathbf{M}, 0 \vDash_{THT} inertia(p)$, for all variable $p \in \Sigma$.*

Proof. From right to left, let us assume that $\mathbf{M}, 0 \vDash_{\text{THT}} inertia(p)$. It follows for all $i \geq 0$ $\mathbf{M}, i \vDash_{\text{THT}} \mathbf{Pr}(p) \vee (p \wedge \neg \mathbf{Cn}(p)) \rightarrow \bigcirc p$, if p is an endogenous variable or $\mathbf{M}, i \vDash_{\text{THT}} (p \rightarrow \bigcirc p)$ if p is exogenous. Using the THT satisfaction relation, it can be easily seen that satisfying both implications implies to meet conditions (A)-(F). Therefore $\mathbf{M}$ is a MHT model.

Conversely, if $\mathbf{M}$ is an MHT model, $\mathbf{M}$ satisfies conditions (A)-(F). It is easy to prove, by using the satisfiability of THT, that conditions (A)-(D) imply that $\mathbf{M}, 0 \vDash_{\text{THT}} inertia(p)$ for endogenous variables while conditions (E)-(F) imply that $\mathbf{M}, 0 \vDash_{\text{THT}} inertia(p)$ for exogenous. ∎

Given a pathway formula F, we define the THT formula $tr(F)$ (on $\widehat{\Sigma}$) as:

$$tr([\alpha](P^{\wedge} \twoheadrightarrow q)) = \Box\left(\mathcal{A}(\alpha) \wedge P^{\wedge} \rightarrow \left(\mathbf{Pr}(q) \wedge \bigwedge_{p \in P} \mathbf{Cn}(p)\right)\right);$$
$$tr([\alpha](P^{\wedge} \twoheadrightarrow q)) = \Box(\mathcal{A}(\alpha) \wedge P^{\wedge} \rightarrow \mathbf{Pr}(q));$$
$$tr(F_1 \wedge F_2) = tr(F_1) \wedge tr(F_1),$$

where F_1 and F_2 are arbitrary pathway formulas and both p and q are endogenous variables. Going back to our running example, the temporal theory associated with (2)-(4) would correspond to the following THT formula:

$$\Box\left(Lactose \wedge Galactosidase \rightarrow (\mathbf{Pr}\left(Glucose\right) \wedge \mathbf{Cn}\left(Galactosidase\right))\right) \quad (2)$$
$$\wedge \Box\left(lacl \rightarrow \mathbf{Pr}\left(Repressor\right)\right) \quad (3)$$
$$\wedge \Box\left(\mathcal{A}(\gamma) \wedge lacZ \rightarrow \mathbf{Pr}\left(Galactosidase\right)\right) \quad (4)$$

Theorem 1 (Correspondence). *Let F be a pathway formula built on Σ and $\mathbf{M}$ be a THT interpretation on $\widehat{\Sigma}$. It holds that:*

(a) $\mathbf{M}, 0 \vDash_{MHT} F$ *iff* $\mathbf{M}, 0 \vDash_{THT} tr\left(F\right) \wedge \bigwedge_{p \in \Sigma} inertia(p)$;
(b) $\mathbf{M}, 0 \vDash_{MEL} F$ *iff* $\mathbf{M}, 0 \vDash_{TEL} tr\left(F\right) \wedge \bigwedge_{p \in \Sigma} inertia(p)$.

Proof. We first consider Case (a). Thanks to Proposition 2, we can reduce the whole proof to the claim: $\mathbf{M}, 0 \vDash_{\text{MHT}} F$ iff $\mathbf{M}, 0 \vDash_{\text{THT}} tr\left(F\right)$. It is easily to check that this claim for elements of Σ as well as conjuntion and disjunction of elements of Σ (see Proposition 1). For the case of pathway formulas we proceed by induction on the form of the pathway formulas: base cases, $[\alpha]\left(P^{\wedge} \twoheadrightarrow q\right)$ and $[\alpha]\left(P^{\wedge} \nrightarrow q\right)$, are proved by means of the satisfaction relation of THT and MHT, Condition (A)-(F) and Proposition 1. The conjunction of pathway formulas follows directly from the induction hypothesis. Finally, Case (b) follows from (a) since the minimisation used for computing the equilibrium models is the same in both formalisms. ■

7 MIM's as splittable temporal logic programs

In this section we show how $tr\left(F\right)$ can be turned into an *splittable temporal logic program* (STLP), a syntactical subset of TEL which has been studied in detail in [4]. A *Temporal Logic Program* Π on $\widehat{\Sigma}$ is said to be *splittable* if Π consists of rules of any of the forms:

(1) $B \wedge N \rightarrow H$;
(2) $B \wedge \bigcirc B' \wedge N \wedge \bigcirc N' \rightarrow \bigcirc H'$;
(3) $\Box(B \wedge N \rightarrow H)$;
(4) $\Box(B \wedge \bigcirc B' \wedge N \wedge \bigcirc N' \rightarrow \bigcirc H')$,

where B and B' are conjunctions of atoms, N and N' are conjunctions of negative literals like $\neg p$ with $p \in \widehat{\Sigma}$, and H and H' are disjunctions of atoms. The *(positive) dependency graph*, of an STLP Π, noted $G(\Pi)$, is a graph whose nodes are the atoms of Π and the edges are defined by the expression below:

$$E = \{(p,p) \mid p \in E\} \cup \{(p,q) \mid \exists B \rightarrow H \in \Pi \text{ s.t. } p \in H \text{ and } q \in B\}.$$

A set of atoms L is called a *loop* of a logic program Π iff the subgraph of $G(\Pi)$ induced by L is strongly connected. Notice that reflexivity of $G(\Pi)$ implies that for any atom p, the singleton $\{p\}$ is also a loop. Every loop of $G(\Pi)$

generates an implication which is called *loop formula*[21,14,4][8]. By $LF(\Pi)$ we refer to the conjunction of all loop formulas of Π.

Theorem 2 (from [4]). *Let Π be an STLP and* $\mathbf{T}$ *an LTL model of Π. Then* $\langle \mathbf{T}, \mathbf{T} \rangle \vDash_{TEL} \Pi$ *iff* $\mathbf{T} \vDash_{LTL} \Pi \wedge LF(\Pi)$. ■

$tr(\Gamma) \wedge \bigwedge_{p \in \Sigma} inertia(p)$ can be expressed as a STLP, thanks to the following THT equivalences:

1) $\Box(((\varphi_1 \vee \varphi_2) \wedge \psi) \leftrightarrow ((\varphi_1 \wedge \psi) \vee (\varphi_2 \wedge \psi)))$;
2) $\Box(((\varphi_1 \wedge \varphi_2) \vee \psi) \leftrightarrow ((\varphi_1 \vee \psi) \wedge (\varphi_2 \vee \psi)))$;
3) $\Box((\psi \rightarrow (\varphi_1 \wedge \varphi_2)) \leftrightarrow ((\psi \rightarrow \varphi_1) \wedge (\psi \rightarrow \varphi_2)))$;
4) $\Box(((\varphi_1 \vee \varphi_2) \rightarrow \psi) \leftrightarrow ((\varphi_1 \rightarrow \psi) \wedge (\varphi_2 \rightarrow \psi)))$;
5) $\Box(\varphi_1 \wedge \varphi_2) \leftrightarrow (\Box\varphi_1 \wedge \Box\varphi_2)$.

For example, the following STLP corresponds to rules (2)-(4) plus the rules of inertia for the atoms *Glucose*, *Repressor*, *Lac*, *LacZ*, *Galactosidase*, *Lactose* and $CAMP$:

$$\left.\begin{array}{l}\Box(Lactose \wedge Galactosidase \rightarrow \mathbf{Pr}(Glucose)) \\ \Box(Lactose \wedge Galactosidase \rightarrow \mathbf{Cn}(Galactosidase))\end{array}\right\}(2)$$

$$\Box(lacl \rightarrow \mathbf{Pr}(Repressor)) \quad \}(3)$$

$$\left.\begin{array}{l}\Box(CAMP \wedge \neg Glucose \wedge \neg Repressor \wedge lacZ \rightarrow \mathbf{Pr}(Galactosidase)) \\ \Box(CAMP \wedge \neg Glucose \wedge Lactose \wedge lacZ \rightarrow \mathbf{Pr}(Galactosidase))\end{array}\right\}(4)$$

$$\left.\begin{array}{l}\Box(\mathbf{Pr}(Glucose) \rightarrow \bigcirc Glucose) \\ \Box(Glucose \wedge \neg\mathbf{Cn}(Glucose) \rightarrow \bigcirc Glucose)\end{array}\right\} inertia(Glucose)$$

$$\left.\begin{array}{l}\Box(\mathbf{Pr}(Repressor) \rightarrow \bigcirc Repressor) \\ \Box(Repressor \wedge \neg\mathbf{Cn}(Repressor) \rightarrow \bigcirc Repressor)\end{array}\right\} inertia(Repressor)$$

$$\left.\begin{array}{l}\Box(\mathbf{Pr}(Galactosidase) \rightarrow \bigcirc Galactosidase) \\ \Box(Galactosidase \wedge \neg\mathbf{Cn}(Galactosidase) \rightarrow \\ \qquad \bigcirc Galactosidase)\end{array}\right\} inertia(Galactosidase)$$

$$\Box(Lactose \rightarrow \bigcirc Lactose) \quad \} inertia(Lactose)$$

$$\Box(CAMP \rightarrow \bigcirc CAMP) \quad \} inertia(CAMP)$$

$$\Box(Lacl \rightarrow \bigcirc Lacl) \quad \} inertia(Lacl)$$

$$\Box(LacZ \rightarrow \bigcirc LacZ) \quad \} inertia(LacZ)$$

Observation 1 *Let Γ be a set of pathway formulas and $\Pi = tr(\Gamma) \wedge \bigwedge_{p \in \Sigma} inertia(p)$, expressed as an STLP. Then*

[8] We refer the reader to [4] for details about the computation of such loop formulas.

(a) $G(\Pi)$ *has only unitary loops;*

(b) Since $G(\Pi)$ *has only unitary loops, Temporal Equilibrium Models of* Π *coincide with the LTL models of the temporal extension of Clark's completion [10,4], denoted by* $COMP(\Pi)$ *[4].* ■

Temporal Completion consists in specifying, along time, that the truth value of an atom $p \in \widehat{\Sigma}$ must be logically equivalent to the disjunction of all its possible causes (see [4] for details). More precisely, $COMP(\Pi)$ corresponds, in our case, to the following expression:

$$COMP(\Pi) = \Box(\bigcirc p \leftrightarrow (\mathbf{Pr}(p) \vee (p \wedge \neg \mathbf{Cn}(p)))) \wedge \Box\left(\mathbf{Pr}(p) \leftrightarrow \bigvee_{[\alpha](P^{\wedge} \multimap p) \in F} P^{\wedge} \wedge \mathcal{A}(\alpha)\right)$$
$$\wedge \Box\left(\mathbf{Cn}(p) \leftrightarrow \bigvee_{[\alpha](P^{\wedge} \rightarrow p) \in F} P^{\wedge} \wedge \mathcal{A}(\alpha)\right).^{9}$$

Theorem 3 (Main result). *Let* Γ *be a set of pathway formulas,* $\Pi = tr(\Gamma) \wedge \bigwedge_{p \in \Sigma} inertia(p)$ *and* $\mathbf{T}$ *be an LTL model of* Π*.*

$$\langle \mathbf{T}, \mathbf{T} \rangle \vDash_{MEL} \Gamma \text{ iff } \mathbf{T} \vDash_{LTL} COMP(\Pi).$$

Proof. From Theorem 1 we get that $\langle \mathbf{T}, \mathbf{T} \rangle \vDash_{\text{MEL}} \Gamma$ iff $\langle \mathbf{T}, \mathbf{T} \rangle \vDash_{\text{TEL}} \Pi$. From Theorem 2 it follows that $\langle \mathbf{T}, \mathbf{T} \rangle \vDash_{\text{TEL}} \Pi$ iff $\mathbf{T} \vDash_{LTL} \Pi \wedge LF(\Pi)$. Finally, regarding Observation 1 we can reduce $\Pi \wedge LF(\Pi)$ to $COMP(\Pi)$ so, therefore $\mathbf{T} \vDash_{LTL} COMP(\Pi)$. ■

8 Conclusion and Future Work

In this paper we gave a formal representation of MIM's in terms of Temporal Equilibrium Logic. To do so, we first defined Molecular Equilibrium Logic, a nonmonotonic logic for dealing with general pathways. Then we showed that this logic can be captured by an LTL formula via a translation into Splittable Temporal Logic Programs under TEL semantics.

As a follow up, we are looking for a way to solve abductive temporal queries on MIM's. Abductive query express important properties; for example the abductive solution to $\Diamond\Box \bigwedge_{p \in \Sigma} (p \leftrightarrow \bigcirc p)$ is the set of all possible conditions that make the cell reach a stable state. An idea to undertake this problem is to combine the works on abduction in Equilibrium Logic [26] and in modal logic [23] in order to define a procedure for abduction in Temporal Equilibrium Logic. Furthermore, finding the complexity of our fragment of temporal equilibrium logic is an open problem. Although in the general case it is known to be EXPSPACE [8], this bound might be lower in our case as the problem is restricted to STLP's with only unitary loops.

[9] We omitted the completion at time step 0 since the formula at the initial state depends on the extensional database, which is not considered here.

References

1. Molecular interaction maps site. `http://discover.nci.nih.gov/mim/`, accessed: August 9, 2016
2. The systems biology markup language site. `http://sbml.org/Documents`, accessed: August 9, 2016
3. Aguado, F., Cabalar, P., Diéguez, M., Pérez, G., Vidal, C.: Temporal equilibrium logic: a survey. Journal of Applied Non-Classical Logics 23(1-2), 2–24 (2013)
4. Aguado, F., Cabalar, P., Pérez, G., Vidal, C.: Loop formulas for splitable temporal logic programs. In: LPNMR'11. vol. 6645, pp. 80–92. springer, Vancouver, Canada (2011)
5. Alliot, J.M., Demolombe, R., Diéguez, M., Fariñas del Cerro, L., Favre, G., Faye, J.C., Obeid, N., Sordet, O.: Temporal modeling of biological systems. In: Akama, S. (ed.) Towards Paraconsistent Engineering: From Pure Logic to Applied Logic. Springer (2016), to appear
6. Alliot, J.M., Demolombe, R., Fariñas del Cerro, L., Diéguez, M., Obeid, N.: Abductive reasoning on molecular interaction maps. In: 7th European Symposium on Computational Intelligence and Mathematics. Springer, Cádiz, Spain (2015)
7. Balbiani, P., Diéguez, M.: Temporal here and there. (2016), unpublished
8. Bozzelli, L., Pearce, D.: On the complexity of temporal equilibrium logic. In: LICS'15. pp. 645–656. IEEE, Kyoto, Japan (2015)
9. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Commun. ACM 54(12), 92–103 (2011)
10. Clark, K.L.: Negation as Failure. In: Logic and Databases, pp. 293–322. Plenum Press, (1978)
11. Demolombe, R., Fariñas del Cerro, L., Obeid, N.: A logical model for molecular interactions maps. In: Fariñas del Cerro, L., Inoue, K. (eds.) Logical Modeling of Biological Systems, pp. 93–123. John Wiley & Sons, (2014)
12. Doncescu, A., Yamamoto, Y., Inoue, K.: Biological systems analysis using inductive logic programming. In: AINA'07. pp. 690–695. , Niagara Falls, Canada (2007)
13. Fariñas del Cerro, L., Herzig, A., Su, E.I.: Epistemic equilibrium logic. In: IJCAI'15. pp. 2964–2970. AAAI Press, Buenos Aires, Argentina (2015)
14. Ferraris, P., Lee, J., Lifschitz, V.: A generalization of the lin-zhao theorem. Annals of Mathematics and Artificial Intelligence 47(1-2), 79–101 (2006)
15. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP'88, pp. 1070–1080. MIT Press, Cambridge, MA (1988)
16. Heyting, A.: Die formalen Regeln der intuitionistischen Logik. In: Sitzungsberichte der Preussischen Akademie der Wissenschaften, Physikalisch-mathematische Klasse. pp. 42–56. , (1930)
17. Iersel, M.V., Kelder, T., Pico, A., Hanspers, K., Coort, S., Conklin, B., Evelo, C.: Presenting and exploring biological pathways with pathvisio. BMC Bioinformatics (September 2008), doi: 10.1186/1471-2105-9-399
18. Jacob, F., Monod, J.: Genetic regulatory mechanisms in the synthesis of proteins. Journal of Molecular Biology 3, 318–356 (1961)
19. Kennell, D., Riezman, H.: Transcription and translation initiation frequencies of the Escherichia coli lac operon. Journal of Molecular Biology 114(1), 1–21 (1977)
20. Kohn, K.W., Pommier, Y.: Molecular interaction map of the p53 and Mdm2 logic elements, which control the off-on swith of p53 response to DNA damage. Biochemical and Biophysical Research Communications 331(3), 816–827 (2005)

21. Lin, F., Zhao, Y.: ASSAT: Computing answer sets of a logic program by sat solvers. Artificial Intelligence 157(1-2), 112–117 (2002)
22. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer-Verlag, New York, NY, USA (1991)
23. Mayer, M.C., Pirri, F.: Propositional abduction in modal logic. Logic Journal of the IGPL 3(6), 907–919 (1995)
24. McCarthy, J., Hayes, P.: Some philosophical problems from the standpoint of artificial intelligence. Machine Intelligence Journal 4, 463–512 (1969)
25. Pearce, D.: A new logical characterisation of stable models and answer sets. In: NMELP'96, Lecture Notes in Artificial Intelligence, vol. 1216, pp. 57–70. Springer, Bad Honnef, Germany (1996)
26. Pearce, D., Valverde, A.: Abduction in equilibrium logic. In: ASP'01 Workshop. , Stanford, USA (2001)
27. Reiter, R.: On closed world data bases. In: Logic and Data Bases. pp. 55–76 (1977)
28. Schaub, T., Thiele, S.: Metabolic network expansion with answer set programming. In: ICLP'09. pp. 312–326. Springer, Pasadena, CA, USA (2009)
29. Tran, N., Baral, C.: Reasoning about non-immediate triggers in biological networks. Annals of Mathematics and Artificial Intelligence 51(2-4), 267–293 (2007)

Minimality of Metabolic Flux Modes under Boolean Regulation Constraints

Martin Morterol[1], Philippe Dague[1,2], Sabine Peres[1,2], and Laurent Simon[2]

[1] LRI, Université Paris-Sud, CNRS, Université Paris-Saclay, 91405 Orsay, France
[2] MaIAGE, INRA, Université Paris-Saclay, 78350 Jouy-en-Josas, France
[3] LaBRI, Université Bordeaux, France

Abstract. We consider the problem of characterizing metabolic flux modes in steady state, which satisfy some given arbitrary Boolean regulation constraint rc. We show that minimal-support solutions, called $MCFMs$ (minimal constrained flux modes), constitute a (in general strict) superset of those $EFMs$ (elementary flux modes, which are the minimal-support solutions in absence of regulation constraint) that satisfy rc (called $EFM^{rc}s$), the only ones computed by the existing tools. We propose a general SMT (Satisfiability Modulo Theory)-based algorithm with two different minimization processes allowing the computation of all $MCFMs$, resp. all $EFM^{rc}s$. The modeling language offers a great expressiveness allowing one to query the network with Boolean constraint, bounded solutions size and metabolites maximization. Results of our $MCFTool$ about $EFM^{rc}s$ are compared to those of $RegEfmtool$ for various types of Boolean constraints. We improve the state-of-the-art for enough specific queries, *i.e.* when the problem is sufficiently constrained to get a greatly reduced solution space, offering the perspective of scalability at the size of the genome for those queries, while characterizing the whole solution space, both things that are inaccessible to nowadays tools.

1 Introduction

The analysis of metabolic networks has become a major topic in biotechnology in recent years. Applications range from the enhanced production of selected outputs to the prediction of genotype-phenotype relationships. The notion of elementary flux mode (EFM) is a key concept derived from the analysis of metabolic networks from a pathway-oriented perspective. An EFM is defined as a smallest (for reactions set inclusion) sub-network that enables the metabolic system to operate in steady state with all irreversible reactions proceeding in the appropriate direction [17, 18]. EFM analysis is a useful metabolic pathway analysis tool to identify the structure of a metabolic network that links the cellular phenotype to the corresponding genotype. Applications of network-based pathway analysis have been presented for predicting functional properties of metabolic networks, measuring different aspects of robustness and flexibility, and even assessing gene regulatory features [20, 7]. In the context of cellular

metabolism, robustness is defined as the ability of cells to achieve the optimal performance even under perturbations imposed by a gene knockout. The robustness of cellular metabolism is mainly due to the redundancy of pathway options that the wild type can choose from to function in order to achieve similar performance. With this concept, it is also possible to compare the pathways taken in different tissues and in different physiological conditions. Actually, $EFMs$ can only be enumerated in small to medium-scale metabolic networks because the number of $EFMs$ increases exponentially with the network size [12]. The huge number of $EFMs$ associated with large biochemical networks prevents from drawing simple conclusions from their analysis. Studies have been carried out on the analysis of sub-networks. The method to calculate these fluxes is based on the double description algorithm [13, 6] which enumerates the extremal rays in a pointed polyhedral cone. Being a computationally demanding task, several approaches to parallel or distributed computation of elementary modes have been proposed through parallelization techniques [8] or algorithmic reformulations [22, 11, 21]. To speed up the computation of $EFMs$, gene regulatory information has been taken into account to eliminate mathematically possible $EFMs$ [10]. The authors added Boolean expressions to reduce the solution space. In adding 58 Boolean functions to the *E. coli* core model [15] they divided by three the number of $EFMs$. Despite $RegEfmTool$ [9] allows a very fast computation of $EFMs$, it relies for most of the cases on a post processing treatment to take into account added constraints. The double description algorithm [13, 6] does not allow a Just in Time calculation of $EFMs$.

Moreover, the very output of classical EFM tools may be questionable. On most of interesting biological models, current tools can easily generate millions of solutions, making the answer simply untractable for any human being. It is thus essential to offer an efficient navigation into the set of solutions to be able to test and exploit any biological scenario. We propose in this work to *navigate* in the solution space by allowing the user to add/remove constraints on the solutions he wants to focus on. As we will show in this paper, this is not only about filtering out subsets of the initial set of $EFMs$. By adding constraints, the minimality criterion of $EFMs$ in the original problem does not hold anymore and minimal solutions in the enriched theory may need to merge together a large number of the original $EFMs$, making any process built upon $EFMs$ calculation unefficient.
In a previous work [16], we proposed a method based on *SAT solvers* [4] to calculate $EFMs$. We were inspired by the seminal paper on *Knowledge Base Compilation Techniques* [5] but, instead of explicitly compute all the $EFMs$, we only answered a set of queries on the network. We proposed to develop a method similar to Just in Time Knowledge Compilation [1], based on *SAT solving*, to determine network properties. In this paper, we propose to add Boolean regulatory constraints directly in our method of $EFMs$ computation. Moreover, we characterize other metabolic flux modes in steady state $MCFMs$ (minimal constrained flux modes), which satisfy some given arbitrary Boolean regulation constraints. We show that these minimal-support solutions constitute a superset

of the $EFMs$ which satisfy the same constraints. We propose a general SMT (Satisfiability Modulo Theory)-based algorithm [2, 14] with two different minimization processes allowing the computation of all $MCFMs$, resp. all $EFMs$ satisfying given arbitrary Boolean constraints.

2 EFM^{rc}s vs MCFMs

A metabolic network model is given by r enzymatic reactions $ER = \{R_1, ..., R_r\}$, with possibly a subset of those reactions known as irreversible, $Irr \subseteq ER$, $m \leq r$ internal metabolites $IM = \{M_1, ..., M_m\}$ and a stoichiometric matrix $S \in \mathbb{R}^{m \times r}$ with full rank m. A flux distribution $v \in \mathbb{R}^r$ is defined by given flux rates for all reactions. Its support is the set of reactions it is made of: $Supp(v) = \{R_i | v_i \neq 0\}$.

Definition 1 (EFM). *The set C of flux distributions enabling the network to operate at steady state with all irreversible reactions proceeding in the appropriate direction is:*

$$C = \{v \in \mathbb{R}^r \mid Sv = 0 \wedge \forall R_i \in Irr \ v_i \geq 0\} \tag{1}$$

The elementary flux modes ($EFMs$) are the solutions of minimal support:

$$E = \{e \in C \mid \nexists e' \in C \ Supp(e') \subset Supp(e)\} \tag{2}$$

C is a convex polyhedral cone. It is pointed (in 0) if $Irr = ER$, which can be always realized by splitting any reversible reaction R into two irreversible reactions R and R_{rev}, which will be assumed in the following. An EFM is characterized up to a positive scalar by its support, i.e. two $EFMs$ having the same support are equal within a positive scalar factor and will be identified. An EFM is thus a ray of E and will be noted $< e >= \{\lambda e | \lambda > 0\}$ for any representative e, its support being $Supp(e)$. The $EFMs$ are the extremal rays of C and any element of C can be expressed as a linear combination of $EFMs$ with nonnegative coefficients.
Let rc be any Boolean formula on the set of variables ER. For $v \in C$, we denote by $rc(v)$ the truth value of rc for the valuation given by $R_i = True$ iff $v_i \neq 0$, i.e. $R_i \in Supp(v)$. Thus $rc(v)$ depends only on $Supp(v)$. But notice there is in general no monotonic relation between the satisfiability of the constraint rc and the supports inclusion, i.e., if $Supp(v) \subset Supp(v')$, it may happen that $rc(v) \wedge \neg rc(v')$ as well as $\neg rc(v) \wedge rc(v')$.

Definition 2 (MCFM). *The set C^{rc} of flux distributions at steady state satisfying the direction of irreversible reactions and Boolean regulation constraint rc is:*

$$C^{rc} = \{v \in C \mid rc(v)\} \tag{3}$$

The minimal constrained flux modes ($MCFMs$) are the solutions of minimal support:

$$M^{rc} = \{e \in C^{rc} \mid \nexists e' \in C^{rc} \ Supp(e') \subset Supp(e)\} \tag{4}$$

Again, we will be in general interested only in the supports of the $MCFMs$, as for a given support, it is easy to recover the flux coefficients from the equation $Sv = 0$. We will see that an $MCFM$ is characterized up to several positive scalars by its support and can be noted $< e_1, ..., e_k >= \{\lambda_1 e_1 + ... + \lambda_k e_k | \lambda_1, ..., \lambda_k > 0\}$ for given representatives $e_1, ..., e_k$, its support being $Supp(e_1) \cup ... \cup Supp(e_n)$.

Lemma 1 *Let $E^{rc} = \{e \in E \mid rc(e)\}$ the subset of $EFM^{rc}s$, i.e. those $EFMs$ satisfying rc. Then $E^{rc} \subseteq M^{rc}$, i.e. any EFM^{rc} is an $MCFM$ (the converse is generally false).*

This result points out that the operations of Boolean constraint satisfaction and support-minimality do not commute. If a flux distribution has a minimal support among all non constrained flux distributions in C, i.e. is an EFM, and happens to satisfy rc, then it has obviously a minimal support among all constrained flux distributions in C^{rc}, i.e. is an $MCFM$. But it may happen that an $MCFM$ owns a strict subset, which is a flux distribution in C (that obviously does not satisfy rc) and thus is not an EFM. The consequence is that the subset E^{rc} of the $EFM^{rc}s$, obtained by filtering the $EFMs$ by the constraint rc, which is the only one presently computed by the state-of-the-art tool $RegEfmtool$ [9], does not characterize the minimal-support elements of the solution space of our problem, i.e. the flux distributions satisfying rc. Our objective is precisely to compute these $MCFMs$.

Consider for example the very simple metabolic network in Figure 1, with $ER = Irr = \{R1, R1_{rev}, T1, T2, T3, T4\}$ (the reactions Ti, which involve both internal and external metabolites, are called transporters), $IM = \{A, B\}$ and stoichiometric coefficients equal to 1. I.e., the first line of S is (-1 1 1 0 -1 0) and the second line (1 -1 0 1 0 -1). Notice that we split the reversible reaction $R1$ into two irreversible reactions $R1$ and $R1_{rev}$. So, we are already in a case where a Boolean constraint applies, i.e. $rc0 = \neg R1 \vee \neg R1_{rev}$, expressing that the two irreversible reactions issued from this splitting cannot occur together in one solution. Such constraints are handled implicitly in an ad-hoc way by existing tools, such as $Efmtool$, by suppressing a posteriori solutions of E containing a cycle such as $\{R1, R1_{rev}\}$ to obtain E^{rc0}. At the opposite, our method will handle them explicitly as any other Boolean constraint during the computation itself. In this very particular case, results are the same, i.e. $M^{rc0} = E^{rc0}$, as, if a flux distribution satisfies $rc0$, so does any subset of it ($rc0$ satisfies a monotonic property with regards to support inclusion). The $EFM^{rc0}s$ are thus given by $E^{rc0} = \{e1, e2, e3, e4\}$ with $e1 =< R1 + T1 + T3 >$, $e2 =< R1_{rev} + T2 + T4 >$, $e3 =< T1 + T4 >$, $e4 =< T2 + T3 >$.

Now, suppose we are only interested in flux distributions containing the two reactions $T1$ and $T2$, i.e. satisfying the constraint $rc = T1 \wedge T2$, and we look for the minimal-support ones. A classical tool as $RegEfmtool$ will not return any solution after filtering of $EFMs$ of E^{rc0} by rc as $E^{rc0 \wedge rc} = \emptyset$. But actually, there are three $MCFMs$: $M^{rc0 \wedge rc} = \{m1, m2, m3\}$ with $m1 =< R1 + T1 + T3, T2 + T3 >= e1 + e4$, $m2 =< R1_{rev} + T2 + T4, T1 + T4\} = e2 + e3$, $m3 =< T1 + T4, T2 + T3 >= e3 + e4$.

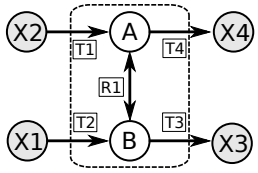


Fig. 1. Toy metabolic network.

The solution space C^{rc} is a subcone of the cone C but, unlike C, it is not convex in general. For example, with $e1$ and $e2$ as above, $e1+e2 \notin C^{rc0}$. This is general for any constraint which is a disjunction of at least two negative literals, $rc = \neg R_{i_1} \vee \neg R_{i_2} \vee ... \vee \neg R_{i_n}$. Because if $\exists v_j, v_k \in C$ with $\{R_{i_1}, ..., R_{i_n}\} \backslash \{R_{i_j}\} \subseteq Supp(v_j)$, $R_{i_j} \notin Supp(v_j)$,$\{R_{i_1}, ..., R_{i_n}\} \backslash \{R_{i_k}\} \subseteq Supp(v_k)$, $R_{i_k} \notin Supp(v_k)$, then $v_j, v_k \in C^{rc}$ and $v_j + v_k \notin C^{rc}$ as $\{R_{i_1}, ..., R_{i_n}\} \subseteq Supp(v_j + v_k)$.
Nevertheless, for particular types of constraints, the convexity is ensured. This is the case for products, i.e. conjunctions of literals. Actually, if $rc = R_{i_1} \wedge ... \wedge R_{i_k} \wedge \neg R_{i_{k+1}} \wedge ... \wedge \neg R_{i_n}$ and $v, v' \in C^{rc}$, then $\{R_{i_1}, ..., R_{i_k}\} \subseteq Supp(v)$, $\forall j$ $i_{k+1} \leq j \leq i_n$ $R_j \notin Supp(v)$, $R_j \notin Supp(v')$, thus, $\forall \lambda, \lambda' > 0, \lambda + \lambda' = 1$, as $Supp(\lambda v + \lambda' v') = Supp(v) \cup Supp(v')$, one gets $\{R_{i_1}, ..., R_{i_k}\} \subseteq Supp(\lambda v + \lambda' v')$, $\forall j$ $i_{k+1} \leq j \leq i_n$ $R_j \notin Supp(\lambda v + \lambda' v')$. So, if rc is a product, then C^{rc} is a convex polyhedral cone but, unlike C, it is not in general topologically closed (neither open). As any Boolean constraint rc is equivalent to a DNF (Disjunctive Normal Form), C^{rc} is actually a union of convex polyhedral cones. This union can be assumed to be disjoint by choosing the products in the DNF two by two unsatisfiable together. Each convex polyhedral cone of the union is included in a face of C (defined by the negative literals of the corresponding product, where $\neg R_i$ defines the hyperplane $v_i = 0$).

Any element of C^{rc}, as for C, is a nonnegative linear combination of undecomposable solutions. But, unlike the $EFMs$, which are at the same time minimal-support and undecomposable solutions, these two properties no longer coincide in presence of rc. An $MCFM$ is obviously undecomposable in C^{rc} but in general not any undecomposable solution is an $MCFM$. Actually it can be shown that non minimal undecomposable solutions are obtained by combining an $MCFM$ and an EFM in $E \backslash E^{rc}$, and can thus be computed in this way if needed. We will now present our SMT-based method to compute M^{rc} and E^{rc}, i.e. the $MCFMs$ and the $EFM^{rc}s$.

3 SMT encoding

As shortly mentioned in the introduction, we previously worked with a SAT encoding representation of metabolic networks for $EFMs$ computation [16]. However, despite good results on some specific hard networks, ensuring that found pathways were steady state solutions forced us to enrich the SAT solver with

an *ad hoc* linear algebra engine, making the final tool closer to a naive SMT solver rather than a plain SAT engine. Most of the time was spent in exploring branches that could have been cut if we had more expression power. In this paper, we propose to completely reformulate the initial problem into SMT in order to unleash the full power of SMT solvers, while keeping our initial idea intact. We use the SMT CVC4 [3] for its performance and documentation. As we will show in this section, this will allow many kinds of additional constraints to be added.

3.1 Stoichiometric and irreversibility constraints

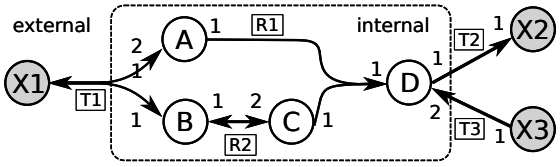


Fig. 2. Toy example of a metabolic network.

Let us now describe how we propose to encode a metabolic pathway into SMT constraints. As a running example, we will follow the encoding of the toy example of Figure 2. We first split the reversible reactions $R2$ and $T1$ into two irreversible ones, $R2$, $R2_{rev}$ and $T1$, $T1_{rev}$ respectively. The network is thus given by $ER = Irr = \{R1, R2, R2_{rev}, T1, T1_{rev}, T2, T3\}$, $IM = \{A, B, C, D\}$ and stoichiometric matrix S with lines (-1 0 0 2 -2 0 0), (0 -1 1 1 -1 0 0), (-1 2 -2 0 0 0 0), (1 0 0 0 0 -1 2). A metabolic pathway can now be encoded with the following rules (real variable R is used for the flux of the pathway in the reaction R and real variable M for the quantity of the internal metabolite M):

1. Since all reactions are now irreversible, they must have a nonnegative flux: $\forall R \in ER,\ R \geq 0$. In the given example, this gives the following rules: $T1 \geq 0,\ T1_{rev} \geq 0,\ R1 \geq 0,\ R2 \geq 0,\ R2_{rev} \geq 0,\ T2 \geq 0,\ T3 \geq 0$.
2. Two irreversible reactions issued from the splitting of one reversible reaction are mutually exclusive: $\forall R, R_{rev} \in ER, (R > 0) \implies (R_{rev} = 0)$ and $(R_{rev} > 0) \implies (R = 0)$. On the example, this gives the following rules: $(T1 > 0) \implies (T1_{rev} = 0)$, $(T1_{rev} > 0) \implies (T1 = 0)$, $(R2 > 0) \implies (R2_{rev} = 0)$, $(R2_{rev} > 0) \implies (R2 = 0)$.
3. Internal metabolites must be nonnegative: $\forall M \in IM,\ M \geq 0$. On our example, this is expressed as: $A \geq 0,\ B \geq 0,\ C \geq 0,\ D \geq 0$.
4. At least one internal metabolite must be used: $\sum_{M \in IM} M > 0$. Our example produces the following constraint: $A + B + C + D > 0$.

5. A positive internal metabolite value is equal to the sum, weighted by the stoichiometry, of fluxes of all reactions that produce it, respectively that consume it: $\forall M \in IM, (M > 0) \implies (M = \sum_{R|S(M,R)>0} S(M,R) \times R \wedge M = -\sum_{R|S(M,R)<0} S(M,R) \times R)$. Again, on our example, this produces the following constraints: $(A > 0) \implies (A = 2 \times T1 \wedge A = R1 + 2 \times T1_{rev})$, $(B > 0) \implies (B = T1 + R2_{rev} \wedge B = R2 + T1_{rev})$, $(C > 0) \implies (C = 2 \times R2 \wedge C = R1 + 2 \times R2_{rev})$, $(D > 0) \implies (D = R1 + 2 \times T3 \wedge D = T2)$.
6. If an internal metabolite is null, fluxes of all reactions that produce or consume it are null too: $\forall M \in IM, (M = 0) \implies \forall R \in ER\ (S(M,R) \neq 0 \implies R = 0)$. This gives on our illustrative example: $(A = 0) \implies (T1 = 0 \wedge R1 = 0 \wedge T1_{rev} = 0)$, $(B = 0) \implies (T1 = 0 \wedge R2 = 0 \wedge T1_{rev} = 0 \wedge R2_{rev} = 0)$, $(C = 0) \implies (R1 = 0 \wedge R2 = 0 \wedge R2_{rev} = 0)$, $(D = 0) \implies (R1 = 0 \wedge T2 = 0 \wedge T3 = 0)$.

3.2 Boolean constraints

Any Boolean constraint in CNF (Conjunctive Normal Form) can be expressed in our tool. More precisely, nonnegative real-valued variables are used for reactions and the regulation constraints are written as follows :

1. The positive literal R, i.e. the reaction R must be active, is given by $R > 0$.
2. The negative literal $\neg R$, i.e. the reaction R must not be active, is given by $R = 0$.
3. Any conjunction of clauses made up of literals as above can be expressed.
4. An upper boundary on the size of the solution can be added, by summing active reactions and asserting a constraint that the sum is below a given constant.

This formalism is very convenient to express a lot of biological constraints of interest, for example:

- *Constraining metabolites.* Since metabolites are encoded in SMT like reactions, the same formalism can be used to express Boolean constraints on metabolites or on both reactions and metabolites.
- *Maximization of metabolite production.* Finding the $EFM^rcs/MCFMs$ that maximize the production of an output metabolite given an input, while satisfying every other constraint. This can be achieved by finding an initial solution then searching another one with a higher production ratio and iterate the process.
- *Independent pathways.* Finding all $EFM^rcs/MCFMs$ producing a given metabolite such that each solution does not share any reaction with the others. This can be easily done by iteratively forbidding all reactions found in a solution and restarting the search.
- *Robustness.* Finding all fluxes producing a given metabolite such as each flux does not share at least X reactions with others. For doing so, from an initial solution, we ensure with a constraint that at least X reactions present in this solution are inactive for the search of the next solution.
- *Bounding set.* Adding a boundary over a specific set. This can be done in the same way as we do with global boundary.

4 Minimization

With the encoding presented in Section 3, the SMT solver computes an initial solution v_0 satisfying stoichiometric and irreversibility constraints and Boolean constraint rc if any, i.e. an element of C^{rc}. We know that v_0, as any element of C, is a nonnegative linear combination of $EFMs$. But, as we are interested in computing $MCFMs$ and $EFM^{rc}s$, we will define minimization procedures focused on finding such minimal solutions included in v_0. Then, after having blocked any possible solution whose support would contain the support of a minimal solution found so far, the process is repeated iteratively from a new initial solution v_1 given by the solver, up to the moment where either no more initial solution is found (and thus all minimal solutions have been exhaustively found) or the number of minimal solutions found reaches a given upper bound.

We describe in algorithm 1 one step of this iteration, i.e. the minimization procedure which provides (some, not all in general in one step) minimal solutions included in a SMT solution *sol*. The corresponding algorithms differ if we are looking for $MCFMs$ or $EFM^{rc}s$ (we call them Minimize_MCFM and Minimize_EFM respectively), but the main idea is the same: either the solution *sol* is minimal or it contains at least one reaction R that can be removed, while ensuring to stay inside the cone C, for continuing recursively the search for a minimal sub-solution. To gain efficiency by finding several minimal sub-solutions instead of just one, the recursion is actually double, on the solution sub-flux $subf1$ extracted from *sol* by removing R and on the sub-flux $subf2$ obtained by "subtracting" $subf1$ to *sol*. As $subf2$, which belongs to C, is not guaranteed to satisfy the constraint rc, we cannot suppose in the successive recursive calls that *sol* is a solution in C^{rc}.

Algorithm 1 Minimize_X(sol) (where X is MCFM or EFM)
Input: a flux sol $\in C$. Output: a decomposition of sol into $MCFMs$ or $EFM^{rc}s$

1: $subf1 = extract\_\mathrm{X}(sol)$
2: **if** $subf1 == \{\}$ **then**
3: **if** $check\_constraint(sol)$ **then**
4: $SMT.addConstraint(\bigvee_{R \in Supp(sol)} R = 0)$
5: **return** $\{sol\}$
6: **else**
7: **if** $(X == EFM)$ **then**
8: $SMT.addConstraint(\bigvee_{R \in Supp(sol)} R = 0)$
9: **end if**
10: **return** $\{\}$
11: **end if**
12: **end if**
13: $subf2 = subtract(sol, subf1)$
14: **return** $Minimize\_X(subf1) \cup Minimize\_X(subf2)$

We first (line 1) extract a strict sub-flux $subf1$ of sol that belongs to C. It is the only procedure which differs for $MFMs$ and $EFM^{rc}s$ computation and it is presented below in algorithms 3 and 4. If sol is minimal, this procedure always returns {}. In this latter case (line 2), we check (line 3) if sol satisfies the constraint rc, i.e. $\in C^{rc}$ (this checking cannot be done in general higher than in a leaf of the recursion tree as a flux may not satisfy rc and contain a sub-flux that satisfies rc). If it is the case, we block for the future (line 4) the exploration of any flux whose support would contain $Supp(sol)$, by adding to the SMT the constraint expressing that the flux in at least one reaction of $Supp(sol)$ has to be null, and we return sol as a minimal solution (line 5). If it is not the case (line 6), we return {} (line 10) and, as it means that $sol \in E \backslash E^{rc}$, i.e. is an EFM that does not satisfy rc, we can before that, but only in the Minimize_EFM algorithm (it is not possible for Minimize_MCFM as an $MCFM$ may contain such an EFM), block the exploration of any flux that would contain this EFM (lines 7-9). We then subtract $subf1$ (within a positive scalar) from sol (line 13) to get a positive linear decomposition of sol into proper sub-fluxes $subf1$ and $subf2$. We finally proceed to a double recursive call of the Minimize procedure on these two sub-fluxes and return the union of the results (line 14).

Algorithm 2 subtract(sol, subf1) Input: sol, subf1 $\in C$ with subf1 sub-flux of sol. Output: subf2 $\in C$ with sol positive linear combination of subf1 and subf2

1: $min = min_{\{i|R_i \in Supp(subf1)\}}(sol_i/subf1_i)$
2: **return** $sol - min \times subf1$

The subtract procedure is described in algorithm 2. This procedure returns the difference between the flux sol and a well chosen positive multiple of the flux $subf1$ (line 2). The choice of this positive scalar is done in order to ensure that the flux obtained is a proper sub-flux of sol and $\in C$, i.e. verifies irreversibility constraints, which means has nonnegative coefficients. This is guaranteed by choosing the scalar min (line 1) so that a coefficient corresponding to $Supp(subf1)$ is canceled while the others remain nonnegative.

4.1 Extraction for MCFM

The extraction procedure extract_MCFM for $MCFMs$ computation is described in algorithm 3.

We simply force the SMT to search for a solution (of all constraints, so including rc) whose support is strictly included in $Supp(sol)$. For this, we temporarily (which is ensured by storing the input state of the SMT, line 1, and restoring it before returning the result, line 9) add constraints expressing that the flux in at least one reaction of $Supp(sol)$ has to be null (line 2) while the fluxes in all reactions out of $Supp(sol)$ have to be null (line 3). If there is a solution (line 4), then the solution found (line 5) is returned (line 10). If not (line 6) then {} (line 7), used by the algorithm 1, is returned (line 10).

Algorithm 3 extract_MCFM(sol) Input: a flux sol $\in C$.
Output: a proper sub-flux subf1 of sol $\in C^{rc}$ or {} if it does not exist

1: $SMT.checkPoint()$
2: $SMT.addConstraint(\bigvee_{R \in Supp(sol)} R = 0)$
3: $SMT.addConstraint(\bigwedge_{R \notin Supp(sol)} R = 0)$
4: **if** $SMT.isSAT()$ **then**
5: $subf1 = SMT.getSolution()$
6: **else**
7: $subf1 = \{\}$
8: **end if**
9: $SMT.RestorecheckPoint()$
10: **return** $subf1$

4.2 Extraction for EFM

The extraction procedure extract_EFM for $EFM^{rc}s$ computation is described in algorithm 4.

Algorithm 4 extract_EFM(sol) Input: a flux sol $\in C$.
Output: a proper sub-flux subf1 of sol $\in C$ or {} if it does not exist

1: **if** $dim(kernel(S^{Supp(sol)})) = 1$ **then**
2: **return** {}
3: **else**
4: $ker = getKernel(S, sol)$
5: $min = min_{\{i|R_i \in Supp(sol)\}}(ker_i/sol_i)$
6: **if** $min < 0$ **then**
7: **return** $sol - (1/min) \times ker$
8: **else**
9: $max = max_{\{i|R_i \in Supp(sol)\}}(ker_i/sol_i)$
10: **return** $sol - (1/max) \times ker$
11: **end if**
12: **end if**

Unlike the extract_MCFM procedure that relies entirely on the SMT solver, the extract_EFM procedure relies on specific properties of $EFMs$ expressed with linear algebra. The key property that characterizes an EFM e, i.e. an extremal ray of the cone C, is that the kernel of $S^{Supp(e)}$, the sub-matrix of the stoichiometric matrix S made up of the column vectors corresponding to the reactions in $Supp(S)$, has dimension one. So we begin to check this property for sol. If it is satisfied (line 1), this means that sol is an EFM and thus does not contain any proper sub-flux $\in C$ and so we return {} (lines 2 and 13). If not (line 3), i.e. this dimension is at least two, we pick up any non null vector ker of the kernel of S (thus such that $Sker = 0$) with $Supp(ker) \subseteq Supp(sol)$ and ker not colinear to sol (this is what does the function $getKernel(S, sol)$

line 4) and we build a linear combination $subf1$ of this vector and sol in order to cancel at least one coefficient and thus decrease the dimension. As both sol and $ker \in Kernel(S)$, we are sure it is also the case for $subf1$. To ensure that $subf1 \in C$, we have to guarantee that all its coefficients are nonnegative. This is achieved by adding to sol (whose coefficients are nonnegative) a well chosen multiple of ker such that all the coefficients remain nonnegative, except at least one that becomes null. As coefficients of ker may be of any sign ($ker \notin C$ in general), there are two cases: if ker contains at least one negative coefficient, it is added to sol after multiplication by the positive scalar $-(1/min)$ (lines 5-7); if all coefficients of ker are nonnegative, it is added to sol after multiplication by the negative scalar $-(1/max)$ (lines 8-10). In both cases the vector $subf1$ obtained is returned (line 13).

Notice that if we are interested in computing both $MCFMs$ and $EFM^{rc}s$, instead of running the two algorithms Minimize_MCFM and Minimize_EFM, it is more efficient, taking into account that $EFM^{rc}s$ are included in $MCFMs$ to run only Minimize_MCFM, that provides all $MCFMs$, and enrich it at leaves (line 5 of algorithm 1) by a test checking if the found $MCFM$ sol is an EFM^{rc}. The test is the same as line 1 of algorithm 4, i.e. the kernel of $S^{Supp(sol)}$ has dimension one.

5 Experimental results

We focus in this section on the experimental study of $MCFTool$ (for Minimal Constrained Flux tool), the SMT-based Metabolic Network Analysis tool we implemented. It is important to notice that, despite its impressive performances, $RegEfmtool$ does not allow the diversity of constraints $MCFTool$ allows. It is thus not trivial to report an empirical study as fair as possible, each of the approaches being not always comparable. We will thus first focus on typical run cases where both $RegEfmtool$ and $MCFTool$ can be used. Then, in a second part, we will try to report how $MCFTool$ behaves on some typical run cases scenarios.

Moreover, $MCFTool$ does not include any network compression for now, so for fair comparison we decided to use compressed networks (computed by $RegEfmTool$) as the input for both $RegEfmTool$ and $MCFTool$. Of course, we turned off compression in $RegEfmTool$ since the network is already compressed.

As a last remark, let us point out that our tool is able to compute both $MCFMs$ and $EFMs$. Thus, as an aside information, we will also report $MCFMs$ computations time in all figures. As we will see, there are generally slightly more $MCFMs$ than $EFMs$ and thus the computation time is slightly larger. We however think it is a good information for consolidate the reader understanding of $EFMs$ vs $MCFMs$ since the first figures. We will focus on the experimental study of $MCFMs$ computation in the last subsection below.

We used a cluster of bi-pro Intel® Xeon® E5-2609v2 2.5GHz, 8 Core, 64 GB, ubuntu 64 bit machine for this experiment.

Problem	Time	$RegEfmTool$	$MCFTool$
aradidopsis-1	339s	665,324	30,585
aradidopsis-2	1400s	1,504,145	84,483
i AS253	Timeout (6h)	None	135,400

Table 1. Comparison of $RegEfmTool$ and $MCFTool$ performances on classical EFMs generation. Timeout was set to 6 hours.

We will first compare $MCFTool$ and $RegEfmTool$ without additional constraints, then we will explain the Boolean constraints generation and compare them with constraints made up of products of positive then negative literals, after that we will compare them with constraints that are well processed by $RegEfmTool$.

5.1 Plain Calculation (no constraints)

$RegEfmTool$, which is based on $Efmtool$, is a very optimized and fast tool to produce huge numbers of $EFMs$. As we pointed out above in this paper, we took an orthogonal approach. In this section, we want to stress how fast $MCFTool$ is on $RegEfmTool$ favorite field. In general our approach is one (and sometimes several) order(s) of magnitude slower than $RegEfmTool$. Despite this, we wanted to report how far our approach can go in the time needed by $RegEfmTool$ to complete (recall that our approach is anytime). We used two metabolic pathway problems derived from arabidopsis such as the calculation time is less than 6 and 24 minutes with $RegEfmTool$. We added a third (more challenging) problem, called "i AS253" [19], to complete the comparison.

Even if the set of problems we chose is very small, it is sufficient to point out typical run cases. As reported in table 1, we can see that our tool cannot easily cope with so many $EFMs$. Thus, when no additional constraints are added, $RegEfmTool$ is clearly the method of choice. However, the anytime aspect of $MCFTool$ can be crucial on large networks, where we can report at least some answers when $RegEfmTool$ is not able to terminate.

5.2 Calculation with constraints

Constraints generation Measuring how adding constraints (by queries) impacts the final result is not trivial. In order to build a set of queries, the result of which will characterize a typical query with less than 5,000 $EFMs$ answers, we adopted a *generate and test* strategy. We chose an upper bound of 5,000 $EFMs$ as an estimation of how many $EFMs/MCFMs$ a human being may handle (otherwise the user will have to add constraints to filter the answer). Again, we took the well studied arabidopsis network (containing 1,504,145 $EFMs$) and created a set of queries by randomly picking reactions until the set of $EFMs$ (obtained by $RegEfmTool$) is less than 5,000. In order to consolidate our CPU time, all reported CPU times are the median over 15 runs.

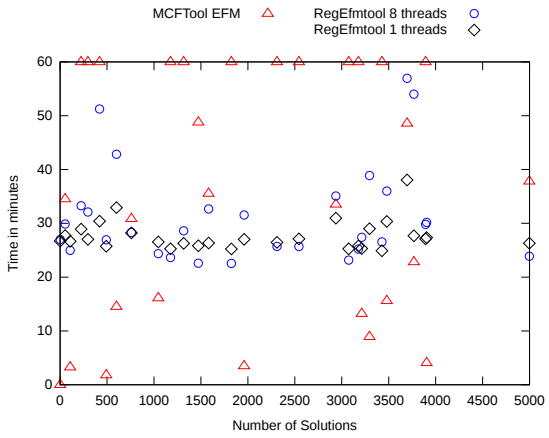


Fig. 3. Execution time with positive reactions selection.

Calculation with positive and negative selection Figure 3 does not display $MCFM$'s execution times because in this case $MCFMs$ and $EFMs$ are not equal so comparing them does not make sense. Figures 3 and 4 reports execution times on positive (conjunct of positive literals) and negative (conjunct of negative literals) selections, respectively. As expected, $RegEfmTool$ is able to handle all the queries but, as this is done for the major part in post-processing, most $EFMs$ computation requires around 25 minutes. There is no substantial gain. At the opposite, we observed a very important discrepancy in $MCFTool$ performances depending on the positive/negative aspect of the query. Our approach seems to be very irregular (figure 3 shows a lot of timeouts while figure 4 is much more in favor of our approach). This can be probably explained if we carefully look at our queries. Positive queries, in average, select around 4 reactions when negative select in average 14 reactions. Indeed, when a reaction is randomly selected for a positive query, if this reaction is a transporter this will decrease drastically the number of $EFMs$ in the network. However, this will not increase as much the number of rules that can be used for propagation. At the opposite, a negative query needs to remove several transporters before decreasing significantly the number of $EFMs$. In other words, a positive selection query does not sufficiently reduce the combinatorial aspect of pathways. A negative selection query implies a larger number of constraints leading to more propagations in the SMT engine.

Calculation with constraints well handled by *RegEfmTool* $RegEfmTool$ restricts constraints to be used for filtering during the iteration process in the double description method to the rules "only require input reactions with the value 1 and an output reaction with the value 0" which can be expressed by "If reaction X is active then reaction Y is not". So, in this section, we generated this

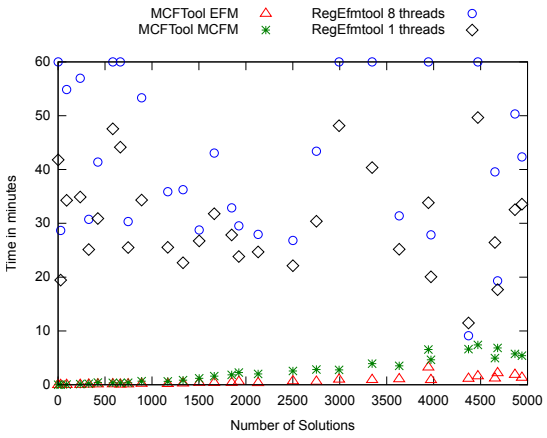


Fig. 4. Execution time with negative reactions selection.

kind of queries. In average these queries contain around 59 clauses. As we can see in Figure 5, $MCFTool$ is faster than $RegEfmTool$ on queries where additional constraints strongly reduces the number of solutions. However, $RegEfmTool$ scales better and is faster when the solution contains more than 4000 $EFMs$. As we initially targetted problems with few solutions (the user will add constraints to reduce the set of solutions to explore), this shows that our approach is very competitive, even on queries that $RegEfmTool$ can easily handle.

5.3 Comparing EFMs and MCFMs

Previously, we compared $RegEfmTool$ and $MCFTool$, by focusing only on queries that were handled by $RegEfmTool$. However, our tool can handle more complex queries or constraints, not handled by $RegEfmTool$. We fairly did not report results on such queries: we wanted to focus on the comparison between the two approaches. In the same spirit, we did not focused on practical aspects of $MCFMs$ computation whereas this is typically the strong point of our approach (our tool is the only one capable of this, so an experimental comparison is not possible). As it is reported on the above figures, we did however reported $MCFMs$ computation on the queries we computed, when $MCFMs$ and $EFMs$ are equal .

6 Conclusion

The calculation of metabolic fluxes based on an SMT allows a direct querying of the network without enumerating all the solutions of the initial problem and then

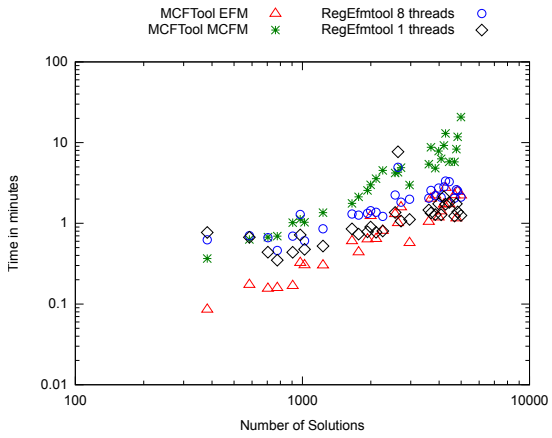


Fig. 5. Execution time with well working request

filtering them with added constraints and allows the expression of a large type of constraints: regulation of enzymes, maximization/minimization of metabolites production/uptake, selection of specific fluxes distribution, size of fluxes distribution, robustness. The major advantage of this method is that it enables a *just in time* computation whereas the state-of-the-art methods, based on Double Description, need the complete iterative computation before producing first solutions and, when constraints are added, often cannot much filter intermediate solutions during the iteration process, most of the filtering having to be done at the end when all non constrained solutions have been found (this is in particular the case for Boolean regulation constraints, due to the absence of monotonicity property between constraints satisfaction and support inclusion, except for very special types of constraints). In addition our method is able to compute the minimal solutions of the problem with constraints and not just those minimal solutions of the problem without constraints filtered by the constraints, which is not done by any tool. However our approach lacks efficiency when the solution space is not enough constrained by the query and we have to face a large enumeration. In this case, the approaches based on Double Description, like $RegEfmTool$, outperform us. But it is the opposite when the problem is sufficiently constrained. Moreover, other kinds of constraints than Boolean ones can be easily taken into account inside the SMT framework, such as thermodynamics or kinetics. Consequently, using our method for selecting $EFMs/MCFMs$ under constraints should now allow studying large metabolic networks almost instantly and open the possibility to calculate them at the genome scale.

References

[1] G. Audemard, J. Lagniez, and L. Simon. "Just-In-Time Compilation of Knowledge Bases". In: *23rd International Joint Conference on Artificial Intelligence(IJCAI'13)*. 2013, pp. 447–453.

[2] C. Barrett et al. "Satisfiability Modulo Theories". In: ed. by A. Biere et al. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009. Chap. 26, pp. 825–885. ISBN: 978-1-58603-929-5.

[3] Clark Barrett et al. "CVC4". In: *Proceedings of the 23rd International Conference on Computer Aided Verification*. CAV'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 171–177. ISBN: 978-3-642-22109-5.

[4] A. Biere et al., eds. *Handbook of Satisfiability*. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, p. 980. ISBN: 978-1-58603-929-5.

[5] A. Darwiche and P. Marquis. "A knowledge compilation map". In: *J. of AI Research* (2002), pp. 229–264.

[6] K. Fukuda and A. Prodon. "Double Description Method Revisited". In: *Combinatorics and Computer Science*. Vol. 1120. LNCS. Springer, 1996, pp. 91–111.

[7] J. Gagneur and S. Klamt. "Computation of elementary modes : a unifying framework and the new binary approach". In: *BMC Bioinformatics* 5.175 (2004).

[8] D. Jevremovic et al. "Parallelization of Nullspace Algorithm for the computation of metabolic pathways". In: *Parallel Computing* 37.6-7 (2011), pp. 261–278. ISSN: 0167-8191. DOI: 10.1016/j.parco.2011.04.002. URL: http://www.sciencedirect.com/science/article/pii/S0167819111000378.

[9] C. Jungreuthmayer, D.E. Ruckerbauer, and J. Zanghellini. "regEfmtool: Speeding up elementary flux mode calculation using transcriptional regulatory rules in the form of three-state logic". In: *Biosystems* 113.1 (2013), pp. 37–39.

[10] Christian Jungreuthmayer et al. "Avoiding the enumeration of infeasible elementary flux modes by including transcriptional regulatory rules in the enumeration process saves computational costs". In: *PloS one* 10.6 (2015), e0129840.

[11] S. Klamt, J. Saez-Rodriguez, and E. Gilles. "Structural and functional analysis of cellular networks with CellNetAnalyzer". In: *BMC Systems Biology* 1.1 (2007), p. 2. ISSN: 1752-0509. DOI: 10.1186/1752-0509-1-2. URL: http://www.biomedcentral.com/1752-0509/1/2.

[12] S. Klamt and J. Stelling. "Combinatorial complexity of pathway anaysis in metabolic networks". In: *Mol Bio Rep* 29 (2002), pp. 233–236.

[13] T.S. Motzkin et al. "The Double Description Method". In: *Contributions to the Theory of Games II*. Ed. by H. W. Kuhn and A. W. Tucker. Princeton University Press, 1953.

[14] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. "Solving SAT and SAT Modulo Theories: From an Abstract Davis–Putnam–Logemann–

Loveland Procedure to DPLL(T)". In: *J. ACM* 53.6 (Nov. 2006), pp. 937–977.

[15] J.D. Orth et al. "A Comprehensive Genome-Scale Reconstruction of Escherichia Coli metabolism–2011". In: *Molecular Systems Biology* 535.7 (2011).

[16] S. Peres, M. Morterol, and L. Simon. "SAT-Based Metabolics Pathways Analysis without Compilation". In: *Lecture Note in Bioinformatics*. Ed. by P. Mendes et al. (Eds.): CMSB. Vol. 8859. Springer International Publishing, 2014, pp. 20–31.

[17] S. Schuster, T. Dandekar, and D.A. Fell. "Detection of elementary modes in biochemical networks : A promising tool for pathway analysis and metabolic engineering". In: *Trends Biotechnol.* 17 (1999), pp. 53–60.

[18] S. Schuster, D.A. Fell, and T. Dandekar. "A general definition of metabolic pathways useful for systematic organization and analysis of complex metabolic networks". In: *Nat. Biotechnol.* 18 (2000), pp. 326–332.

[19] Anthony C. Smith and Alan J. Robinson. "A metabolic model of the mitochondrion and its use in modelling diseases of the tricarboxylic acid cycle". In: *BMC Systems Biology* 5.1 (2011), pp. 1–13. ISSN: 1752-0509. DOI: 10.1186/1752-0509-5-102. URL: http://dx.doi.org/10.1186/1752-0509-5-102.

[20] J. Stelling et al. "Metabolic network structure determines key aspect of functionnality and regulation". In: *Nature* 420 (2002), pp. 190–193.

[21] M. Terzer and J. Stelling. "Large-scale computation of elementary flux modes with bit pattern trees". In: *Bioinformatics* 24.19 (2008), pp. 2229–2235.

[22] A. Von Kamp and S. Schuster. "Metatool 5.0: Fast and flexible elementary modes analysis". In: *Bioinformatics* 22 (2006), pp. 1930–1931.

Guaranteed Weighted Counting for Affinity Computation: Beyond Determinism and Structure

Clément Viricel[1−2], David Simoncini[1], Sophie Barbe[2], and Thomas Schiex[1]

[1] MIAT, Université de Toulouse, INRA UR 875, Castanet-Tolosan, France
firstname.lastname@toulouse.inra.fr
[2] LISBP, Université de Toulouse, CNRS, INRA, INSA, Toulouse, France
firstname.lastname@insa-toulouse.fr

Abstract. Computing the constant Z that normalizes an arbitrary distribution into a probability distribution is a difficult problem that has applications in statistics, biophysics and probabilistic reasoning. In biophysics, it is a prerequisite for the computation of the binding affinity between two molecules, a central question for protein design. In the case of a discrete stochastic Graphical Model, the problem of computing Z is equivalent to weighted model counting in SAT or CSP, known to be #P-complete [38]. SAT solvers have been used to accelerate guaranteed normalizing constant computation, leading to exact tools such as `cachet` [33], `ace` [8] or `minic2d` [28]. They exploit determinism in the stochastic model to prune during counting and the dependency structure of the model (partially captured by tree-width) to cache intermediary counts, trading time for space. When determinism or structure are not sufficient, we consider the idea of discarding sufficiently negligible contributions to Z to speedup counting. We test and compare this approach with other solvers providing deterministic guarantees on various benchmarks, including protein binding affinity computations, and show that it can provide important speedups.

1 Introduction

Graphical models [12] are sparse representations of highly dimensional multivariate distributions that rely on a factorization of the distribution in small factors. When variables are discrete, graphical models cover a variety of mathematical models that represent joint discrete distributions (or functions) that can be either Boolean functions (*e.g.,* in propositional satisfiability SAT and constraint satisfaction CSP), cost functions (as in partial weighted MaxSAT and Cost Function Networks [9]) or probability distributions (in stochastic models such as Markov Random Fields and Bayesian networks).

Typical queries on such graphical models are either optimization or counting queries (or a mixture of these). In optimization queries, we look for an assignment that maximizes the joint function, *i.e.,* a model in SAT, a solution in CSP or a Maximum a posteriori assignment (MAP) in a Markov Random Field (MRF). All these problems have an associated NP-complete decision problem.

Counting problems are central in stochastic graphical models because they capture the computation of marginal probabilities on subsets of variables and the computation of the normalizing constant Z that is required to define a probability distribution from the

non-normalized distribution of Markov Random Fields. This difficult problem requires a summation over an exponential number of elementary terms and is #P-complete [38]. As shown by [37], one call to a #-P oracle suffices to solve any problem in the Meyer-Stockmeyer polynomial hierarchy in deterministic polynomial time, an indication that it could be outside of the PH.

Computing Z is a central problem in statistics (*e.g.,* for parameter estimation in MRFs), for Bayesian network processing (to account for evidence) and is also crucial in statistical physics where it is called the partition function. A typical domain where partition function computation can be extremely useful is computational protein design. Indeed, the affinity of a protein for a specific target molecule can be estimated by modeling both molecules as MRFs representing physics force fields and by computing the two partition functions: one for the bound protein and target and another for the same molecules in unbound state [35].

For these reasons, various approaches have been designed to tackle this problem. The Mean-Field algorithm [17], Tree-reweighted Belief Propagation [40] as well as more recent proposals [25] have been proposed, but they do not offer any formal guarantee on the quality of the approximation they produce, except in very special cases. Monte-Carlo methods including Markov Chain Monte Carlo methods [16] offer asymptotic convergence /s, but convergence is impractically slow. Indeed, there are recent significant examples showing that the time needed for Monte Carlo methods to converge can be easily under-estimated [36]. Practical MCMC based tools also rely on heuristics that destroy these theoretical guarantees. More recent stochastic methods exploiting universal hashing functions offer "Probably Approximately Correct" (PAC) estimators [14, 7]. Here, a bound δ on the probability that the estimation does not lie within a $(1+\varepsilon)$ ratio of the true value is set and a corresponding estimation produced.

Finally, different methods, mostly based on SAT-solvers, have been defined that can perform exact weighted model counting (#SAT) with deterministic guarantees, a problem to which the problem of computing Z can be easily reduced. To avoid the exponential blowup in the number of terms to add, solvers providing deterministic guarantees rely on two independent ideas: exploiting determinism (zero weights) to prune regions of the space that do not contribute to the sum, and exploiting independence which may be detected at the graphical model structure level, as captured by its tree-width, but also at a finer level as context-sensitive independence [33][3]. Independence enables caching of intermediate counts that can be factored out and lead to exponential time savings at the cost of memory. The very same ideas are also exploited in knowledge compilers that may compile graphical models or SAT formulas to languages on which counting becomes easy [8, 28].

In this paper, we explore the possibility of preserving the *deterministic* guarantees of exact solvers and explore a new source of pruning that may be present even when determinism or independence are too limited to allow for exact counting: detecting and pruning regions for which it is possible to prove, at limited cost, that they contain an amount of weight which is too small to significantly change the computed value of Z. Instead of providing a PAC guarantee, our algorithm provides an approximation of the

[3] They may also exploit the fact that counting the number of models of a valid formula is easy. This requires to check for validity, something that modern CDCL solvers do not do anymore.

normalizing constant that is guaranteed to lie within a ratio of $1 + \varepsilon$ of the true value (with probability 1), a guarantee that none of the PAC randomized algorithms above can provide in finite time.

Our initial motivation for computing Z lies in Computational Protein Design (CPD). The aim of CPD is to design new proteins that have desirable properties which are not available in the existing catalog of known proteins. One of these properties is the *affinity* between a protein and another molecule (such as another protein, a peptide, an amino-acid, a small organic molecule, etc...). The binding affinity gives an indication of the likelihood that two molecules will prefer to bind together rather than remain dissociated and thus that a protein will be likely to bind to another molecule of interest. Proteins can be described as a set of bound atoms subjected to a number of atom scale forces captured by a pairwise force field defining a Markov Random Field [29]. From this MRF, the binding affinity can be estimated by computing the ratio of the partition functions of the molecules in bound and unbound states [35, 15].

In the rest of the paper, after introducing our notations and the binding affinity computation problem, we present the Z_ε^* algorithm, a variant of Branch and Bound targeted at counting instead of optimizing. Z_ε^* relies on the availability of a local upper bound on Z. We then consider different simple, fast, safe and incremental upper bounds on Z, integrate them in Z_ε^* and compare them to exact counting tools on two categories of benchmarks: general benchmarks extracted from the UAI and Probabilistic Inference (PIC'2011) challenges and partition function computation problems appearing as sub-problems of binding affinity computation on real proteins. Surprisingly, despite a very limited caching strategy, the resulting algorithm is able to outperform exact solvers on a variety of problems and is especially efficient on CPD-derived problems. Because Z_ε^* relies on a new source of pruning, its underlying principle and associated bounds can be immediately used to improve existing SAT-based counters using Max-SAT bounds, which are closely related to local consistencies in Cost Function Networks [23, 4, 24].

2 Background

A Markov Random Field defines a joint probability distribution over a set of variables as a factorized product of local functions, usually denoted as potential functions.

Definition 1 *A discrete Markov Random Field (MRF) is a pair (X, Φ) where $X = \{1, \ldots, n\}$ is a set of n random variables, and Φ is a set of potential functions. Each variable $i \in X$ has a finite domain D^i of values that can be assigned to it. A potential function $\phi_S \in \Phi$, with scope $S \subseteq X$, is a function $\phi_S : D^S \mapsto \mathbb{R} \cup \{\infty\}$ where D^S denotes the Cartesian product of all D^i for $i \in S$.*

The energy or potential of an assignment $t \in D^X$ is denoted as $E(t) = \sum_{\phi_S \in \Phi} \phi_S(t[S])$ where $t[S]$ is the projection (or restriction) of t to the variables in S. Notice that this definition shows that an MRF is essentially equivalent to a Cost Function Network (or WCSP [9]). A tuple $t \in D^S$ will be represented as a set of pairs $\{(i, t[i]) \mid i \in S\}$.

The probability of a tuple $t \in D^X$ is then defined as:

$$P(t) = \frac{\exp(-E(t))}{\sum_{t' \in D^X} \exp(-E(t'))}$$

The normalizing constant below the fraction is usually denoted as Z. The potential ϕ_S are called energies, in relation with statistical physics. An assignment with minimum energy has therefore maximum probability. With pairwise potentials ($|S| \leq 2$), an MRF defines a graph with variables as vertices and potential scopes S as edges. In the rest of this paper, for the mere sake of simplicity and w.l.o.g., we assume pairwise MRFs including also unary potential functions and a constant $\phi_\varnothing$ potential function. We denote by d the maximum domain size and e the number of pairwise potential functions. Using table representations, a pairwise MRF requires $O(ed^2)$ space to be represented.

Note that Bayesian networks can be seen as specific MRFs enforcing a local normalization condition of potentials and a specific DAG-base graph structure, that together guarantee that $Z = 1$. As soon as evidence (observations) change the domain of the variables however, Bayesian networks become unnormalized and computing Z becomes #P-complete in general.

2.1 Computational Protein Design and Binding Affinity

Proteins are linear chains of small molecules called "amino-acids". There are 20 natural different amino-acids. All amino-acids share a common core and the cores of all successive amino-acids in a proteins are linked together to form a linear chain, called the protein backbone. Each amino-acid also has a variable side-chain which chemical nature defined the precise amino-acid used. This lateral chain is highly flexible. The structure of a protein in 3D-space is therefore characterized by the shape of the linear chain itself (the backbone), and the specific spatial orientation of all side-chains, at each position of the chain. Proteins are universally present in the cells of all living organisms and perform a vast array of functions including catalyze, signaling, recognition, transporting, repair... Proteins differ from one another primarily in their sequence of amino-acids which usually results in protein folding into a specific 3D structure that determines its function. The characteristic of proteins that also allows their diverse set of functions is their ability to bind other molecules, with high affinity and specificity. See [5, 1] for an intrduction to proteins targeted at the CP audience.

Proteins have a relatively stable general shape. The relative stability of a molecule in a given conformation can be evaluated by computing its energy, lower energy states being more stable. This energy is derived from various molecular forces including bond angles, electrostatic forces, molecular clashes and distances. It can be computed using existing force fields such as Amber [29], the one used in our experiments. Notice that molecular clashes – interpenetrating atoms – may generate infinite energies *i.e.,* determinism.

Despite a plethora of functionalities of proteins, there is still an ever-increasing demand for proteins endowed with specific properties of interest for many applications (in biotechnology, synthetic biology, green chemistry and nanotechnology) which either do not exist in nature or have yet not been found in the biodiversity. To this end, Computational structure-based Protein Design (CPD) has become a key technology. By combining physico-chemical models governing relations between protein amino-acid composition and protein 3D structure with advanced computational algorithms, CPD seeks to identify one or a set of amino-acid sequences that fold into a given 3D structure

and possess the targeted properties. This *in silico* search for the best sequence candidates opens up new possibilities to better guide protein engineering by focusing experimentation on the relevant sequence space for the desired protein function and thereby reducing the size of mutant libraries that need to be built and screened. In recent years, CPD has experienced important success, especially in the design of therapeutic proteins [27], novel enzymes [31], protein-protein interfaces [18, 32], and large oligomeric ensembles [19]. Nevertheless, the computational design of proteins with defined affinity for a given molecule (such as a small organic, a peptide, another protein…) which is essential for large range of applications, continues to present challenges.

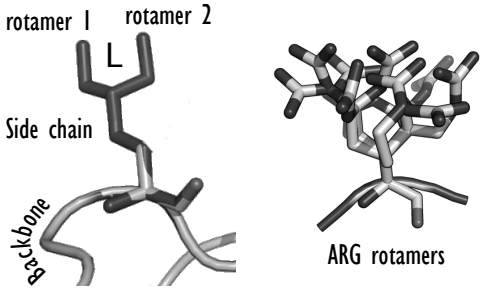


Fig. 1. A local view of a protein with a backbone and two acid side-chain reorientations (rotamers) for a given amino-acid (L = Leucine). A typical rotamer library for another amino-acid is shown on the right (ARG = Arginine).

A traditional approach to model proteins in CPD is to assume that their backbone is totally rigid and that only side-chains move, each side-chain being able to adopt a discrete set of most likely conformations defined in a so-called "rotamer" library (see Figure 1). We use the Penultimate rotamer library [26].

With one variable per side-chain, each with a domain equal to the set of available rotamers for this side-chain and a pairwise decomposable energy function such as Amber force field, a protein naturally defines a pairwise MRF with a rather dense graph. The partition function Z of this MRF captures important properties of the protein. Specifically, the association constant (or binding constant) is used to describe the affinity between a protein and a ligand (a protein or another molecule of interest). This association constant can be estimated by computing the partition function of the two molecules in bound and unbound states. The ratio of these two partition functions being proportional to their affinity.

From a computational point of view, an important property of proteins of interest is that their general shape is stable which means that the proportion of low energy (or high probability) states among the exponential number of possible states is likely to be very small. On the opposite side of the energy scale, the infinite energies created by molecular clashes means that there will be states with 0 probability. This is favorable for exact solvers that can exploit determinism to speedup Z computation. It

however means that CPD instances will exhibit unbounded *tilt* (defined in [7] as the ratio $\tau = \frac{\max_{t \in D^X} P(t)}{\min_{t \in D^X} P(t)}$). This situation is not ideal for the WeightMC PAC algorithm which requires a finite upper-bound on τ to run in finite time.

3 Guaranteed counting

Because it is rarely (if ever) needed to compute a probability or a partition function with an absolute precision (which is also inherently limited by finite representations), we consider the general problem of computing an ε- approximation $\hat{Z}$ of Z, *i.e.,* such that:

$$\frac{Z}{1+\varepsilon} \leq \hat{Z} \leq Z \tag{1}$$

Such approximation allows us to compute an estimate $\hat{P}(t) = \frac{\exp(-E(t))}{\hat{Z}}$ such that $P(t) \leq \hat{P(t)} \leq (1+\varepsilon)P(t)$. In the context of #-SAT, it has been shown that providing such relative approximations remains intractable for most of the known SAT polynomial classes [30]. As we will see, it can however be exploited to prune during polynomial space depth-first tree-search based counting and sometimes provide important speedups.

Assuming that for any MRF, and any assignment t of *some* of its variables, we can compute an upper bound $Ub(t)$ of the partition function of the MRF where variables are assigned as in t, the Depth First Branch and Bound schema used for exactly solving optimization problems on cost function networks [9, 1] can be adapted to compute Z [39].

```
   Function Z*_ε(t, V)
1  |  if V = ∅ then
2  |  |  Ẑ ← Ẑ + exp(−E(t));
   |  else
3  |  |  Choose i ∈ V;
   |  |  for a ∈ D^i do
   |  |  |  t' ← t ∪ {(i, a)};
4  |  |  |  if (U + Ub(t') + Ẑ ≤ (1 + ε)Ẑ) then
5  |  |  |  |  U ← U + Ub(t');
   |  |  |  else
   |  |  |  |  Z*_ε(t', V − {i});
```

Algorithm 1: Guaranteed approximate counting. Initial call: $Z^*_\varepsilon(\varnothing, X)$. U and $\hat{Z}$ are global variables initialized to 0.

The algorithm simply explores the tree of all possible assignments of the MRF, starting with the whole set of unassigned variables (in V), choosing an unassigned

variable (line 3), trying all possible values. When all variables are assigned (line 1), the contribution of the complete assignment t is accumulated in a running count which will eventually define the approximation $\hat{Z}$ (line 2). However, branches which provide a sufficiently small mass of probability (as estimated by $Ub(t')$) are pruned and this overestimation of the neglected mass is accumulated in U (line 5). Because pruning may occur, eventually, $\hat{Z}$ will be a lower bound of Z.

Theorem 1. *Z^*_ε terminates and returns an ε-approximation of Z.*

Proof. The termination follows from the fact that Z^*_ε explores a finite tree. We now show that the algorithm always provides a ε-approximation. When the algorithm finishes, all the assignments have either been explored (line 2) and counted or pruned (line 5). Since U is the sum of all the upper bounds on the mass of probability in all pruned branches, we have that $\hat{Z} + U \geq Z$. Initially, $\hat{Z} = U = 0$ and the invariant $\hat{Z} \geq \frac{\hat{Z}+U}{1+\varepsilon}$ holds. The test at line 4 guarantees that this invariant still holds at the end of the algorithm. Therefore $\hat{Z} \geq \frac{Z}{(1+\varepsilon)}$. □

While inspired by Depth First Branch and Bound (DFBB) that provides polynomial space complexity, this algorithm behaves differently from it. In DFBB, for a fixed order of exploration, when the local bound used for pruning (here $Ub(t)$) is tighter, less nodes are explored. This property is lost in Z^*_ε. Indeed, it is easy to imagine a scenario where a tight bound $Ub(t)$ will lead to more nodes being explored than using a weaker $Ub'(t)$: imagine that search has started and collected a mass $\hat{Z} = 1$ and $U = 0$ for either bounds. Then comes a subtree of small size for which $Ub(t) = \varepsilon$ while $Ub'(t) \gg \varepsilon$. This subtree will be pruned by $Ub(t)$ leading to $U = \varepsilon$ but instead will be enumerated with $Ub'(t)$ preserving $U = 0$. In this context, the algorithm using the tight $Ub(t)$ is not allowed to prune anymore in the immediate future: if the forthcoming leaves all have very small probability mass, it will be forced to visit all of them while the algorithm using $Ub'(t)$ preserved some margin and may be able to skip a significant fraction of them.

Indeed, similarly to what happens with the α-β algorithm [20], the order in which leaves are explored may have a major effect on the algorithm efficiency. Let us assume that we have a perfect $Ub(t)$ and that the leaves of the tree have exponentially decreasing mass of probability, the i^{th} visited leaf having a mass of $\varepsilon^{i-1}, \varepsilon < 1$ (such an extreme distribution of probability mass may seem unlikely, but corresponds to linearly increasing energies). In this case, the first leaf bears more mass than all the rest of the tree and the Z^*_ε algorithm would visit just one leaf. If the inverse ordering of leaves is assumed, the algorithm will have to explore all leaves. It seems therefore important to collect highest masses first. The polynomial space complexity of DFBB comes however with strict constraints on the order of exploration of leaves and best-first algorithms that could overcome this restriction would lead to worst-case exponential space complexity. Interesting future work would be to use the recent highly flexible any-space Branch and Bound algorithm HBFS [2] to improve the leaf ordering within bounded space.

However, contrary to what happens with optimization, even an exact upper bound and a perfect ordering does not guarantee that only one leaf needs to be explored. If we instead assume a totally flat energy landscape, with all leaves having the same energy,

Z^*_ε will have to explore a $\frac{1}{1+\varepsilon}$ fraction of the leaves just to accumulate enough mass in $\hat{Z}$ to prune.

Overall, it is important to realize that Z^*_ε needs to achieve two goals:

1. collect probability masses on a potentially very large number of complete assignments to compute a suitable approximation
2. exploit its upper bound to prune the largest possible part of the tree

The first goal could be achieved by existing algorithms producing an exhaustive list of the m-best assignments [13] or all assignment within a threshold of the optimum (a service that any DFBB-based optimization system provides for free). These algorithms use bounds on the *maximum* probability instead of the total probability mass which leads to stronger pruning and potentially higher efficiency than Z^*_ε but do not provide any guarantee since the number m of assignments that would need to be enumerated to provide ε-approximation is unknown.

Because a potentially very large number of probability masses need to be collected, a very fast search is required. To accelerate it, we equip Z^*_ε with a very simple form of "on the fly" caching: at any node during the search, we eliminate any variable which is either assigned or of bounded degree as proposed initially for optimization [22], but using sum-product variable elimination [11]. This caches all the influence of the eliminated variable in a temporary (trailed) potential function. This means that the leaves of the search tree will be sub-problems with bounded tree-width that may represent an exponential number of assignments. This naturally makes Z^*_ε related to the `vec` weighted counting algorithm, an anytime MRF counter based on w-cutsets (vertex cutset which if assigned leave a w-tree) and variable elimination over w-trees [11].

The second goal is to prune the largest possible part of the tree search. However, since the first goal requires a very fast search algorithm, using a powerful but computationally expensive bound is probably doomed to fail. For this reason, we have considered simple fast incrementally updated upper bounds by borrowing recent optimization bounds [9] which are known to work well in conjunction with Depth First Search.

3.1 Bounds for guaranteed counting

For any MRF, we define a first upper bound on Z denoted by Ub_1.

$$Z \leq Ub_1 = \left(\prod_{\phi_S, |S|<2} \sum_{t \in D^S} \exp\left(-\phi_S(t)\right) \right) \cdot \left(\prod_{\phi_S, |S| \geq 2} \exp\left(- \min_{t \in D^S} \phi_S(t) \right) \right)$$

Proof. By definition, we have that

$$Z = \sum_{t \in D^X} \left(\prod_{\phi_S, |S|<2} \exp\left(-\phi_S(t)\right) \cdot \prod_{\phi_S, |S| \geq 2} \exp\left(-\phi_S(t)\right) \right)$$

Trivially, $\exp\left(-\phi_S(t)\right) \leq \max_{t \in D^S} \left(\exp\left(-\phi_S(t)\right)\right) = \exp\left(-\min_{t \in D^S} \phi_S(t)\right)$ (by monotonicity). Applying this to the right term above, and exploiting the fact that this term now does not depend on t, we get that:

$$Z \leq \left(\sum_{t \in D^X} \prod_{\phi_S, |S|<2} \exp\left(-\phi_S(t)\right) \right) \cdot \left(\prod_{\phi_S, |S| \geq 2} \exp\left(- \min_{t \in D^S} \phi_S(t) \right) \right)$$

Since the set $\{\phi_S, |S| < 2\}$ contains only unary or constant functions, distributivity allows to swap sum and product and the result follows. Notice that this bound can be computed in linear time. □

This bound can be strengthened by selecting a subset of all pairwise potentials in Φ defining a partial spanning k-tree $T \subset \Phi$. By applying a sum-product non serial dynamic programming [11] on $T' = T \cup \{\phi_S \in \Phi \ : \ |S| < 2\}$, we can obtain the exact $Z_{T'}$ for this sub-MRF in polynomial time. We can multiply $Z_{T'}$ by $(\prod_{\phi_S \in \Phi \setminus T'} \exp(-\min_{t \in D^S} \phi_S(t)))$ and get a tighter upper bound on Z which we denote Ub_T:

$$Z \leq Ub_T = \underbrace{\left(\sum_{t \in D^X} \prod_{\phi_S \in T'} \exp\left(-\phi_S(t)\right) \right)}_{\text{Computed using non serial dynamic programming}} \cdot \left(\prod_{\phi_S \in \Phi \setminus T'} \exp\left(- \min_{t \in D^S} \phi_S(t) \right) \right)$$

Proof. The proof is essentially similar to the previous one, and obtained by just replacing the set $\{\phi_S, |S| < 2\}$ and its complement set $\{\phi_S, |S| \geq 2\}$, defining the ranges of the products by the sets $\{\phi_S \in T'\}$ (and its complement respectively). The first item can be simply computed in $O(nd^2)$ time using non serial dynamic programming. □

These bounds alone are very weak. To further strengthen them, we reformulate the MRF using soft arc-consistencies [9] on its energy representation [1]. Soft arc consistencies essentially shift energy from pairwise potential functions to unary potential functions and eventually to the constant potential function $\phi_\varnothing$ while preserving equivalence. The result of this is an equivalent MRF (defining the same distribution) with increased unary and constant $\phi_\varnothing$ potential functions and pairwise potential functions that satisfy $\min_{t \in D^S} \phi_S(t) = 0$. Besides strengthening the bounds, it removes the need to compute the right term which is always equal to 1. Ub_1 and Ub_T can then be computed in $O(nd)$ and $O(nd^2)$ instead of $O(ed^2)$ (extension to non pairwise potentials would require the use of partial k-trees instead of trees and change the d^2 into a d^{k+1}).

In the rest of the paper we consider spanning trees and try both Existential Directional Arc Consistency (EDAC) and Virtual Arc Consistency (VAC) [9] as possible ways of strengthening Ub_1 and Ub_T.

4 Experimental evaluation and comparison

To evaluate the ability of the Z^*_ε algorithm to provide guaranteed deterministic approximations to Z, we implemented it on the top of the open source `toulbar2` solver[4].

[4] `http://www.inra.fr/mia/T/toulbar2`

The variable and value ordering used are the default weighted-degree and last conflict variable ordering and the existential support value-ordering [9]. We enforce EDAC at the root node and during search as usual for optimization. When VAC is used, it is only enforced at the root node because of its computational cost. Instead of the k-way branching described in Algorithm 1, we use a binary branching that either includes or reject a chosen value a at each branching decision. At each node, all variables of degree ≤ 2 are eliminated. The upper bound Ub_T uses a fixed maximum spanning tree with maximum sum of mean cost after enforcing arc consistencies at the root node. Our implementation is limited to pairwise potentials.

We compared it to different exact weighted counting approaches in terms of efficiency and quality of our guaranteed approximation. Four different exact counters have been considered. The first one is the already described `vec` exact counter [11]. The second one is the exact SAT based weighted counting tool `cachet` [33][5]. `cachet` relies internally on the Zchaff SAT solver to enumerate models with non zero weight and uses context-sensitive independence to cache intermediate counts. We also used the `ace` 3.0 compiler [8], using the UAI competition executable provided in the `ace` distribution (always using a pseudo-random generator seed of 0). `ace` computes a tree decomposition and based on the obtained width may either perform tabular variable elimination or encode to CNF and compile in d-DNNF using `c2d`. We also tested the recent `minic2d` Sentential Decision Diagram (SDD) compilation package [28]. SDD are more constrained than d-DNNF and may therefore lead to larger compiled forms than d-DNNF, but since we do not need a compiled form and just the value of Z, we used the `-W` option of `minic2d` that performs weighted counting without compilation hoping to trade space for time. `minic2d` relies on its own internal SAT solver which is provided as a compiled binary in the distributed `minic2d` package. Because some of the compared solvers (`vec`, `cachet`) provide only a double floating representation of Z (or its logarithm), all software has been used in double floating point mode.

All executions have been performed on one core of an Intel® Xeon® CPU E5-2680 v3 @ 2.50GHz (a Q4 2014 cpu) with a limit of 60 GB on RAM usage.

4.1 MRF to #SAT encoding

If `ace` uses its own internal optimized MRF to SAT encoding, both `cachet` and `minic2d` require specific SAT encoding. Exact #SAT weighted counters use weighted literals and define the weight of a model as the product of the weights of all literals which are true in the model. They therefore rely on multiplicative potentials $\exp(-\phi_S(t))$. To transform an MRF into a literal-weighted CNF formula with a weighted count equal to the partition function, we use the ENC1 encoding of [8], originally described in [10]. This encoding is the CNF version of the so-called *local polytope*-based ILP encoding introduced in [34] for MRFs and [21] for weighted CSPs [9]. For each variable $i \in X$, we use one proposition $d_{i,r}$ for each value $r \in D^i$. This proposition is true iff variable i is assigned the value r. We encode *At Most One* (AMO) with hard clauses $(\neg d_{i,r} \vee \neg d_{i,s})$ for all $i \in X$ and all $r < s$, $r, s \in D^i$, as well as *At Least One* (f) with one hard

[5] We thank Jean-Marie Lagniez, CRIL, France for providing us with a patched version of cachet that can be compiled and run without any issue on recent systems.

clause $(\bigvee_r d_{i,r})$ for each i. These clauses ensure that the propositional encoding allows exactly one value for each variable in each model. For each potential ϕ_S, and each tuple $t \in D^S$, we have a propositional variable $p_{S,t}$. For non-zero energies $\phi_S(t)$, we have the literal $p_{S,t}$ with weight $\exp(-\phi_S(t))$. This represents the multiplicative potential to use if the tuple t is used. $\neg p_{S,t}$ is instead weighted by 1, the identity for multiplication. For every variable $i \in S$, we have a hard clause $(d_{i,t[i]} \vee \neg p_{S,t})$. These clauses enforce that if tuple t is used, its values $t[i]$ must be used. Then, for each variable $i \in S$ and each value $r \in D^i$, we have hard clauses $(\neg d_{i,r} \vee \bigvee_{t \in D^S, t[i]=r} p_{S,t})$ that enforces that if a value $r \in D^i$ is used, one of the allowed tuples $t \in D^S$ such that $t[i] = r, w_S(t) < k$ must be used.

It is interesting to notice that for pure Constraint Satisfaction Problems (MRFs having only $0/\infty$ potentials), it is known that Unit Propagation (UP) on this encoding enforces arc consistency in the original CSP [3].

We apply obvious optimization steps, explicitly forbidding local assignments with zero mass (sources of determinism). This encoding can be directly fed into `minic2d`. Large problems however could not be encoded because `minic2d` only allows to express weights in a one-line list of maximum 100,000 chars in length[6].

In `cachet`, weighted literals l are either such that l and $\bar{l}$ receive a mass of 1 that has no effect on final mass, or such that the weights of a variable and its negation sum to 1. This is sufficient and convenient to express Bayesian nets because of their local normalization constraint. For arbitrary MRFs, for every $p_{S,t}$ corresponding to a mass $m = \exp(-\phi_S(t))$ we introduce another propositional variable $n_{S,t}$ with weights m (positive) and $1 - m$ (negative) and a simple implication clause $p_{S,t} \rightarrow n_{S,t}$. This extra variable is connected to the rest of the problem only through this clause and can therefore easily be eliminated, leading to a multiplicative factor m in models where $p_{S,t}$ is true and $1 = m + (1 - m)$ in models where $p_{S,t}$ is false, as required.

4.2 Benchmarks

Two types of benchmarks have been used. The first type of benchmark is made of instances of partition function computation appearing as sub-problems of binding affinity computations on molecular systems defined by a protein interacting with a peptide or an amino-acid. The 3D model of these molecular systems were derived from crystallographic structures of the proteins in complex with their ligands, deposited in the Protein Data Bank. Missing heavy atoms in crystal structures as well as hydrogen atoms were added using the *tleap* module of the *Amber 14* software package [6]. The molecular all-atom *ff14SB* force field was used for the proteins and the ligands (peptides and amino-acids). The molecular systems were then subjected to 1000 steps of energy minimization with the *Sander* module of *Amber 14*. Next, a portion of the proteins including amino-acids at the interface between the protein and the ligand as well as surrounding amino-acids with at least one atom within 8 to 12 Å (according to the molecular system) of the interface was selected.[7]

[6] This parameter could not be changed, being in the non open-source part on `minic2d`.

[7] Each of these systems requires extensive molecular modeling expertise to be properly defined. We intend to make this benchmark together with the Z^*_ε implementation available.

To evaluate the effect of the strength of the upper bound on the algorithm efficiency, we applied Z^*_ε with $\varepsilon = 10^{-3}$ on a series of 349 systems using the four different bounds. For each system, the most complex partition function, defined on the compound system, is computed. The largest problem has 22 variables and the largest domain size is 34. The gap between our two bounds and the guaranteed approximation of Z determined by Z^*_ε is shown in Figure 2. The bound Ub_T is clearly stronger than Ub_1, as expected.

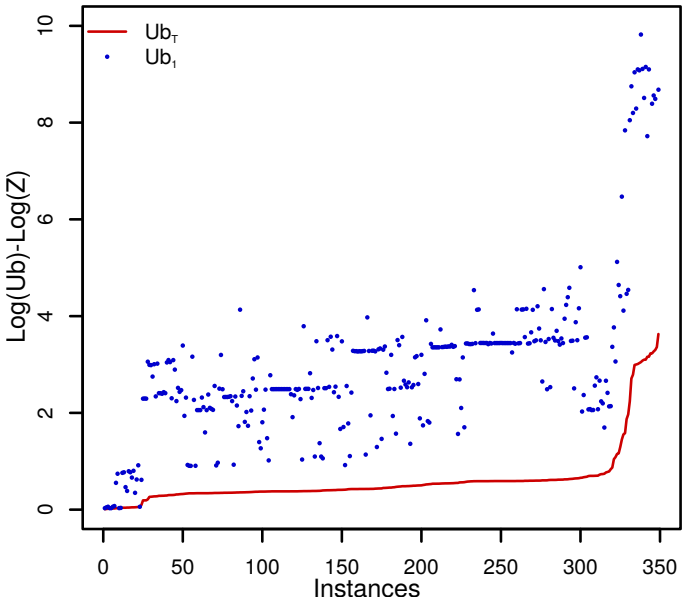


Fig. 2. Gap to Z: we represent $\log(Ub) - \log(\hat{Z})$ at the root node for Ub_1 and Ub_T using EDAC tightening (only negligible difference with VAC on these problems). The instances on the x axis are sorted in increasing gap size for the strongest Ub_T bound.

We then compare the running times of Z^*_ε using these 4 bounds in a cactus plot in Figure 3. The best bound in terms of run-time is the lightest Ub_1+EDAC bound confirming that stronger, thus more expensive, bounds may quickly become counter productive.

We represent the same information with the fastest Ub_1+EDAC and two of the three exact counting tools in Figure 4. We omit `ace` and `minic2d`. Indeed, `ace` was able to solve only 17 problems within the time limit and failed on all remaining problems with a memory exception (despite the explicit allocation of 60GB to the JAVA machine). `minic2d` was instead unable to model 294 systems out of the 349 because of its previously mentioned limitation on the length of the weight line. On the remaining 55 problems, `minic2d` solved 7 problems in less than one hour.

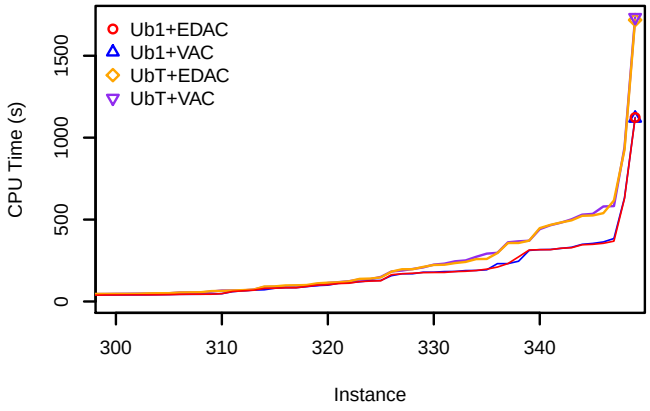


Fig. 3. Cactus plot of the running times of Z^*_ε with our four bounds. A point at (x, y) indicates that the number of solved problems is x if a deadline of y seconds is imposed on each resolution.

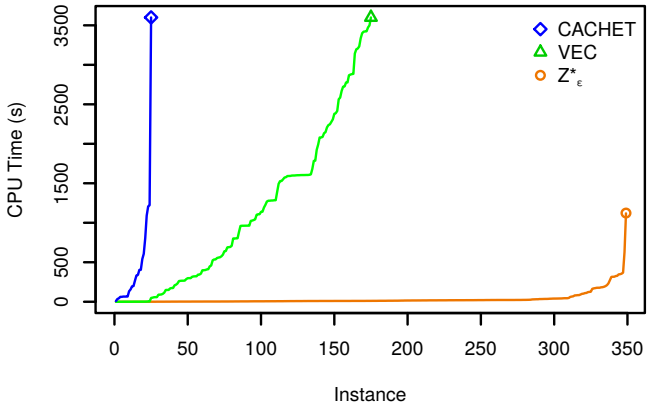


Fig. 4. Cactus plot of the running times of Z^*_ε with the Ub_1+EDAC bound together with `cachet` and `vec`. A point at (x, y) indicates that the number of solved problems is x if a deadline of y seconds is imposed on each resolution.

In the rest of the experiments we therefore use the Ub_1+EDAC upper bound which seems the most efficient bound. To see how the Z_ε^* algorithm performs on other types of problems, we used instances extracted from UAI and PIC'2011 challenge instances (PR task)[8] that use only pairwise potentials as a second set of benchmark. Using the same value of $\varepsilon = 10^{-3}$, we again compared Z_ε^* with `vec`, `cachet`, `ace` and `minic2d`.

The results clearly show there is no single winner: except for `cachet` which is always dominated by one of the other solvers, each algorithm may outperform others. Specifically, the Z_ε^* algorithm, despite its lack of sophisticated caching technology, is able to outperform its competitors in various cases. Nevertheless, `minic2d` outperformed Z_ε^* on the Grid category (probably because of the combination of Boolean variables and relatively small treewidth), itself outperformed by `vec` and further outperformed by `ace`.

Instance	Z_ε^*	`minic2d`	`ace`	`vec`	`cachet`
smokers_10	$< \mathbf{0.01}$	0.663	$< \mathbf{0.01}$	$< \mathbf{0.01}$	0.264
smokers_20	$< \mathbf{0.01}$	312.825	$< \mathbf{0.01}$	$< \mathbf{0.01}$	332.168
rbm_20	7.46	T	**3.376**	16.17	941.14
rbm_ferro_20	$< \mathbf{0.01}$	T	3.24	16.18	893.84
rbm_21	13.75	T	**6.854**	34.35	2041.64
rbm_ferro_21	$< \mathbf{0.01}$	T	6.868	34.17	1975.78
rbm_22	33.75	T	**14.411**	72.78	T
rbm_ferro_22	$< \mathbf{0.01}$	T	14.418	72.43	T
grid10x10.f10	66.32	$< \mathbf{0.01}$	$< \mathbf{0.01}$	$< \mathbf{0.01}$	T
grid20x20.f10	T	T	**13.316**	2104.57	T
grid20x20.f15	T	T	**13.665**	2099.24	T
grid20x20.f2	T	T	**13.603**	2107.92	T
grid20x20.f5	T	T	**13.609**	2102.32	T
GEOM30a_3	$< \mathbf{0.01}$	$< \mathbf{0.01}$	$< \mathbf{0.01}$	$< \mathbf{0.01}$	2.668
GEOM30a_4	7.66	78.604	$< \mathbf{0.01}$	$< \mathbf{0.01}$	43.77
GEOM30a_5	67.48	368.361	$< \mathbf{0.01}$	$< \mathbf{0.01}$	405.38
GEOM40_2	$< \mathbf{0.01}$	$< \mathbf{0.01}$	$< \mathbf{0.01}$	$< \mathbf{0.01}$	$< \mathbf{0.01}$
GEOM40_3	$< \mathbf{0.01}$	$< \mathbf{0.01}$	$< \mathbf{0.01}$	$< \mathbf{0.01}$	3.62
GEOM40_4	1.42	0.58	$< \mathbf{0.01}$	$< \mathbf{0.01}$	616.59
GEOM40_5	12.67	12.801	$< \mathbf{0.01}$	$< \mathbf{0.01}$	828.50
myciel5g_3	**37.2**	T	M	T	T
myciel5g_4	M	T	M	T	T
myciel5g_5	M	T	M	T	T
queen5_5_3	367.2	T	**83.004**	945.20	T
queen5_5_4	**2423.67**	T	M	T	T

Table 1. Time results for UAI/PIC'2011 instances. Three different categories are represented: Boltzmann machines (rbm) with attractive (ferro) and non attractive coupling, Grids, and graph problems. Running-times are given in seconds. M: Memory Out ($60GB$), T: Time out ($1h$). Bold is best.

[8] http://www.cs.huji.ac.il/project/PASCAL

Conclusion

Existing solvers providing deterministic guarantees for partition function computation exploit two sources of efficiency. This first one is caching of local counts based on context-sensitive independence [33], related to tree-decomposition. The other one is determinism *i.e.,* , the existence of zero probability assignments allowing to prune zero probability mass sub-trees during search. This second source of efficiency will provide significant speedups only when a significant fraction of the search space has 0 probability. Such distributions have very low entropy.

In this paper, motivated by the computation of statistical estimate of affinity between bio-molecules, we have proposed to build upon existing optimization technology to provide a new source of pruning for partition function computation with deterministic guarantees: a branch and bound-based schema equipped with upper bounds derived from soft local consistencies. As existing SAT-based exact approaches, our algorithm exploits determinism and a much simpler and less powerful form of caching than those based on tree-decomposition. It is however able to prune regions of proven negligible mass of probability and is therefore able to exploit relatively low entropy distributions having a much wider support, including those with no determinism. The resulting algorithm offers an adjustable deterministic guarantee on the quality of the computed partition function and, despite its limited caching strategy, may already offer interesting speedups compared to exact solvers.

Z^*_ε includes two crucial ingredients to quickly gather large number of probability masses: pruning based on very fast incremental upper bounds derived from optimization bounds and on-the-fly sum-prod elimination. An important point is that these ingredients can be easily injected into existing SAT-based counters, including knowledge-compilation based counters using SAT-solver traces. This could be achieved by defining counting upper bounds from existing Max-SAT bounds. These bounds have already been related to soft arc-consistency bounds [23, 4, 24]. This should extend their range of application to guaranteed approximate probabilistic inference on problems with limited or no determinism.

From an affinity computation point of view, the next step is now to evaluate the actual empirical quality of the association constant estimation provided by the computed ratio of partition functions. Beyond algorithmic approximations, the modeling may also have important effects on the estimated value based on different rotamer discretizations, relative positions of molecules in the complex or weights of different contributions in the energy function. To pursue this target, we intend to use available databases that provide experimental values of the association constant of various protein-ligand complexes following various mutations on one of the partners. To keep the modeling to a reasonable level of complexity, this will be preferably achieved on protein-protein complexes.

Acknowledgments : We would like to thank Simon de Givry for his help with `toulbar2`. We thank the Computing Center of Region Midi-Pyrénées (CALMIP, Toulouse, France) and the Genotoul Bioinformatics Platform of INRA-Toulouse for providing computing resources and support. C. Viricel was supported by a grant from INRA and Region Midi-Pyrénées.

References

1. Allouche, D., André, I., Barbe, S., Davies, J., de Givry, S., Katsirelos, G., O'Sullivan, B., Prestwich, S., Schiex, T., Traoré, S.: Computational protein design as an optimization problem. Artif. Intell. 212, 59–79 (2014)
2. Allouche, D., De Givry, S., Katsirelos, G., Schiex, T., Zytnicki, M.: Anytime hybrid best-first search with tree decomposition for weighted csp. In: Principles and Practice of Constraint Programming. pp. 12–29. Springer (2015)
3. Bacchus, F.: GAC via unit propagation. In: Proc. of CP. pp. 133–147 (2007)
4. Bonet, M.L., Levy, J., Manyà, F.: Resolution for max-sat. Artificial Intelligence 171(8), 606–618 (2007)
5. Campeotto, F., Dal Palu, A., Dovier, A., Fioretto, F., Pontelli, E.: A constraint solver for flexible protein model. Journal of Artificial Intelligence Research 48, 953–1000 (2013)
6. Case, D., Babin, V., Berryman, J., Betz, R., Cai, Q., Cerutti, D., Cheatham Iii, T., Darden, T., Duke, R., Gohlke, H., et al.: Amber 14 (2014)
7. Chakraborty, S., Fremont, D.J., Meel, K.S., Seshia, S.A., Vardi, M.Y.: Distribution-aware sampling and weighted model counting for sat. In: Proc. 28^{th} Conference on Artificial Intelligence. p. 1722–1730 (2014)
8. Chavira, M., Darwiche, A.: On probabilistic inference by weighted model counting. Artificial Intelligence 172(6), 772–799 (2008)
9. Cooper, M., de Givry, S., Sanchez, M., Schiex, T., Zytnicki, M., Werner, T.: Soft arc consistency revisited. Artificial Intelligence 174, 449–478 (2010)
10. Darwiche, A.: A logical approach to factoring belief networks. KR 2, 409–420 (2002)
11. Dechter, R.: Bucket elimination: A unifying framework for reasoning. Artificial Intelligence 113(1–2), 41–85 (1999)
12. Dechter, R.: Reasoning with probabilistic and deterministic graphical models: Exact algorithms. Synthesis Lectures on Artificial Intelligence and Machine Learning 7(3), 1–191 (2013)
13. Dechter, R., Flerova, N., Marinescu, R.: Search algorithms for m best solutions for graphical models. In: AAAI. Citeseer (2012)
14. Ermon, S., Gomes, C.P., Sabharwal, A., Selman, B.: Taming the curse of dimensionality: Discrete integration by hashing and optimization. arXiv preprint arXiv:1302.6677 (2013)
15. Georgiev, I., Lilien, R.H., Donald, B.R.: The minimized dead-end elimination criterion and its application to protein redesign in a hybrid scoring and search algorithm for computing partition functions over molecular ensembles. J. Comput. Chem. 29(10), 1527–42 (Jul 2008)
16. Gilks, W.R.: Markov chain monte carlo. Wiley Online Library (2005)
17. Jaakkola, T.S.: Tutorial on variational approximation methods. Advanced mean field methods: theory and practice p. 129 (2001)
18. Karanicolas, J., Kuhlman, B.: Computational design of affinity and specificity at protein–protein interfaces. Curr. Opin. Struct. Biol. 19(4), 458–463 (2009)
19. King, N.P., Bale, J.B., Sheffler, W., McNamara, D.E., Gonen, S., Gonen, T., Yeates, T.O., Baker, D.: Accurate design of co-assembling multi-component protein nanomaterials. Nature 510(7503), 103–108 (2014)
20. Knuth, D.E., Moore, R.W.: An analysis of alpha-beta pruning. Artificial intelligence 6(4), 293–326 (1976)
21. Koster, A.: Frequency assignment: Models and Algorithms. Ph.D. thesis, University of Maastricht, The Netherlands (1999), available at www.zib.de/koster/thesis.html.
22. Larrosa, J.: Boosting search with variable elimination. In: Principles and Practice of Constraint Programming - CP 2000. LNCS, vol. 1894, pp. 291–305. Singapore (Sep 2000)

23. Larrosa, J., Heras, F.: Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In: Proc. of the 19^{th} IJCAI. pp. 193–198. Edinburgh, Scotland (2005)
24. Larrosa, J., Heras, F., de Givry, S.: A logical approach to efficient max-sat solving. Artif. Intell. 172(2-3), 204–233 (2008)
25. Liu, Q., Ihler, A.T.: Bounding the partition function using holder's inequality. In: Proceedings of the 28th International Conference on Machine Learning (ICML-11). pp. 849–856 (2011)
26. Lovell, S.C., Word, J.M., Richardson, J.S., Richardson, D.C.: The penultimate rotamer library. Proteins 40(3), 389–408 (Aug 2000), http://www.ncbi.nlm.nih.gov/pubmed/10861930
27. Miklos, A.E., Kluwe, C., Der, B.S., Pai, S., Sircar, A., Hughes, R.A., Berrondo, M., Xu, J., Codrea, V., Buckley, P.E., et al.: Structure-based design of supercharged, highly thermoresistant antibodies. Chemistry & biology 19(4), 449–455 (2012)
28. Oztok, U., Darwiche, A.: A top-down compiler for sentential decision diagrams. In: Proceedings of the 24th International Conference on Artificial Intelligence. AAAI Press (2015)
29. Pearlman, D.A., Case, D.A., Caldwell, J.W., Ross, W.S., Cheatham, T.E., DeBolt, S., Ferguson, D., Seibel, G., Kollman, P.: Amber, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules. Computer Physics Communications 91(1), 1–41 (1995)
30. Roth, D.: On the hardness of approximate reasoning. Artificial Intelligence 82(1), 273–302 (1996)
31. Röthlisberger, D., Khersonsky, O., Wollacott, A.M., Jiang, L., DeChancie, J., Betker, J., Gallaher, J.L., Althoff, E.a., Zanghellini, A., Dym, O., Albeck, S., Houk, K.N., Tawfik, D.S., Baker, D.: Kemp elimination catalysts by computational enzyme design. Nature 453(7192), 190–5 (May 2008), http://www.ncbi.nlm.nih.gov/pubmed/18354394
32. Sammond, D.W., Eletr, Z.M., Purbeck, C., Kuhlman, B.: Computational design of second-site suppressor mutations at protein–protein interfaces. Proteins: Structure, Function, and Bioinformatics 78(4), 1055–1065 (2010)
33. Sang, T., Beame, P., Kautz, H.: Solving bayesian networks by weighted model counting. In: Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05). vol. 1, pp. 475–482 (2005)
34. Schlesinger, M.: Syntactic analysis of two-dimensional visual signals in noisy conditions. Kibernetika 4, 113–130 (1976)
35. Silver, N.W., King, B.M., Nalam, M.N., Cao, H., Ali, A., Kiran Kumar Reddy, G., Rana, T.M., Schiffer, C.A., Tidor, B.: Efficient computation of small-molecule configurational binding entropy and free energy changes by ensemble enumeration. J. Chem. Theory Comput. 9(11), 5098–5115 (2013)
36. Simoncini, D., Allouche, D., de Givry, S., Delmas, C., Barbe, S., Schiex, T.: Guaranteed discrete energy optimization on large protein design problems. Journal of chemical theory and computation 11(12), 5980–5989 (2015)
37. Toda, S.: On the computational power of pp and $\oplus$p. In: Foundations of Computer Science, 1989., 30th Annual Symposium on. pp. 514–519. IEEE (1989)
38. Valiant, L.G.: The complexity of computing the permanent. Theoretical computer science 8(2), 189–201 (1979)
39. Viricel, C., Simoncini, D., Allouche, D., de Givry, S., Barbe, S., Schiex, T.: Approximate counting with deterministic guarantees for affinity computation. In: Modelling, Computation and Optimization in Information Systems and Management Sciences. pp. 165–176. Springer (2015)
40. Wainwright, M.J., Jaakkola, T.S., Willsky, A.S.: A new class of upper bounds on the log partition function. Information Theory, IEEE Transactions on 51(7), 2313–2335 (2005)

Computational protein modelling based on limited sets of constraints

Maryana Wånggren[1], Martin Billeter[2], and Graham J.L. Kemp[1]

[1] Department of Computer Science and Engineering,
Chalmers University of Technology, SE-412 96 Gothenburg, Sweden
[2] Department of Chemistry and Molecular Biology, University of Gothenburg,
SE-405 30 Gothenburg, Sweden

Abstract. Molecular dynamics simulations, often combined with simulated annealing, are commonly used when calculating structural models of proteins, e.g. based on NMR experiments. However, one is often faced with limited and sometimes insufficient information for determining a well-resolved 3D structure. In addition, the type of data available for different proteins may vary: ranges for torsion angles, distance approximations, relative orientation of different molecular parts etc. We are using whatever structural information is around, together with a dynamic programming approach for searching the space of feasible conformations to rapidly determine 3D structures that are consistent with the input constraints. Time-efficiency is important for good sampling of the conformational space but also to replace expensive, complex and time consuming experiments. Our approach benefits from having both high level and low level descriptions of conformational features and constraints, and the possibility to infer new constraints from those that are given.

Keywords: constraints, dynamic programming, protein modelling, zipping and assembly

1 Introduction

Proteins are important biological macromolecules that consist of chains of amino acid residues. Knowing the three-dimensional structures of proteins is important in fully understanding the molecular basis for their function, but experimental determination of protein structures can be difficult, costly and time-consuming. Therefore there is a strong interest in using computational modelling methods to obtain model proteins structures.

In our current work we are developing a computational modelling method that can use whatever information is easy to obtain, which could be different from case to case. In different projects there could be available information from one or more kinds of experimental investigation, e.g. co-evolution information, disulphide linkage analysis results secondary structure information, data from

NMR experiments. With NMR, one can carry out many different kinds of experiment in order to collect more data that can be used in determining a structure. However, doing this has a cost. For example some information about main chain amide groups that are spatially close together can be obtained relatively easily, compared with obtaining a full list of all nuclear Overhauser effect (NOE) restraints that give distance estimates for pairs of atoms that are close together in three-dimensional space. We aim to develop a protein modelling method that can be used with whatever limited data sets are available in order to construct satisfactory models without the need for extensive additional experiments.

The rest of this paper is organised as follows. In Section 2 we describe aspects of protein structure that are needed to understand the methods described in the paper, introduce zipping and assembly method which uses dynamic programming in exploring a large confomational space, and describe the kinds of constraint that are used when building models of protein main chains. In Section 3 we describe the application of our method in modelling two proteins. We discuss our own system, related work and some future directions in Section 4. The main contributions of the work are summarised in Section 5.

2 Background

2.1 Protein structure

In this paper we are mainly concerned with modelling a protein's main chain. Part of a protein's main chain is shown in Figure 1. A protein chain is able to fold into its native conformation by rotation around two of the bonds in the main chain, designated ϕ and ψ.

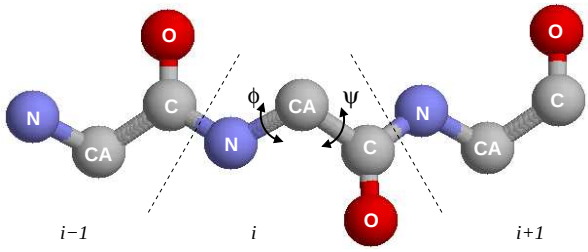


Fig. 1. The heavy (i.e. non-hydrogen) main chain atoms of three consecutive amino acid residues are represented by spheres, and the covalent bonds between these are represented by rods. Nitrogen and oxygen atoms (N and O) are shown in blue and red respectively; carbon atoms are shown in grey. The central carbon atom (the Cα carbon, labelled CA) is the main chain atom to which a side chain (not shown) is attached. Rotation can occur around the bonds labelled ϕ and ψ.

If we assume standard bond lengths and angles [8], the task of predicting the conformation of a protein's main chain reduces to predicting values for all of the

ϕ and ψ angles. Some ϕ and ψ combinations are energetically more favourable than others, and some combinations are not possible at all since these would result in atoms clashing into each other. Figure 2 shows the distribution of over 300 ϕ and ψ combinations taken from experimentally determined structures from the Protein Data Bank (PDB)[3].

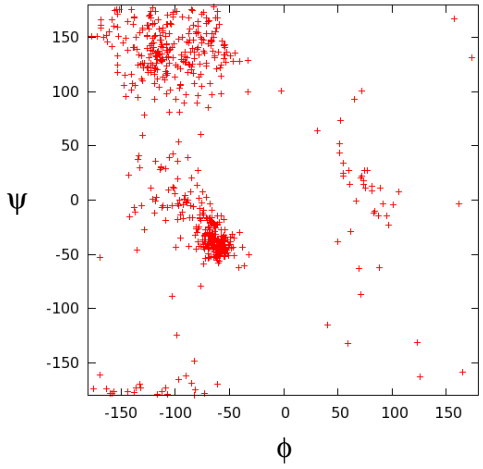


Fig. 2. Combinations of values for ϕ and ψ torsion angles from a library of proteins from the Protein Data Bank. The values of the ϕ and ψ angles are in degrees.

2.2 Zipping and assembly

In many approaches to modelling protein chains an entire chain is first constructed, then its conformation is repeatedly adjusted and evaluated in an attempt to reach the protein's native conformation. An alternative approach to exploring the search space when constructing a model is to build models of short fragments of protein chain independently of each other, and then to combine these fragments into longer fragments. This is what is done in the zipping and assembly method [7, 9].

Zipping and assembly is a dynamic programming algorithm that constructs longer fragments from pairs of shorter ones. The principle is illustrated in Figure 3. The numbers along the bottom of the figure represent residue positions within the chain. $Cell_{i,j}$ represents the part of the protein chain being modelled from residue position i to position j, for example the fragment from positions 8 to 10 is represented by $Cell_{8,10}$. Each cell contains sets of fragments that are candidates for modelling the conformation of part of the target protein. The

cells on the lowest level ($Cell_{1,1}$, $Cell_{2,2}$, etc., shaded) each contain a set of alternative conformations for the single amino acid residue at that position within the cell. The cells on level two ($Cell_{1,2}$, $Cell_{2,3}$, etc.) contain sets of two-residue fragments, each of which is formed by combining one residue conformation from the the cell to the lower left, and one residue conformation from the cell to the lower right. These are combined by superposing main chain atoms onto a reference peptide plane, as has been done previously by others e.g. [4]. Alternative conformations for longer fragments are constructed by combining shorter fragments from lower cells. Consider the five-residue fragment from positions 4 to 8. Possible conformations for this fragment will be stored in $Cell_{4,8}$ cell, which contains a question mark. These can be constructed by combining a fragment chosen from the cell labelled a1 with one from the cell labelled a2, or combining a fragment chosen from cell b1 with one chosen from cell b2, and so on. Similarly, all cells in the diagram can be filled with fragment conformations that are the result of combining a fragment chosen from one of the cells to the lower left of that cell, with a fragment from a cell to the lower right. The fragments at higher levels are successively longer than those at the levels below (this growth in chain length is illustrated in the cells on the left edge of the figure). Finally, $Cell_{1,10}$ at the apex will contain a set of possible conformations for the entire protein (consisting of 10 residues in this toy example).

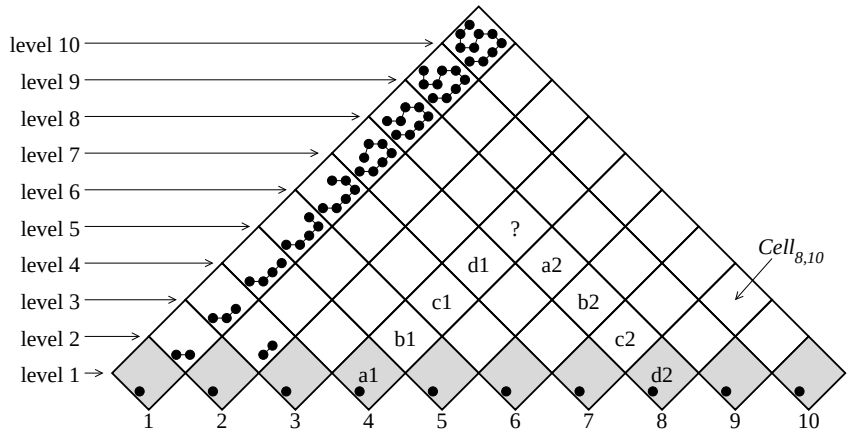


Fig. 3. Zipping and assembly of a protein with 10 residues.

Claims made for the zipping and assembly method include [7]:

— its local-first-global-later search explains quick folding, and avoidance of vast stretches of conformational space ("local" here refers to local in sequence);
— it reflects the parallel nature of physical kinetics;

— it captures the relationship between contact order (whether pairs of amino acid residues that are close together in 3-D space also tend to be close to each other along the protein chain, or tend to be distant from each other along the protein chain) and folding rate;
— it identifies slow- and fast-folding proteins, and slow- and fast-folding routes.

2.3 Conformational constraints

The constraints used in this work include upper and lower distance bounds between pairs of atoms. Distance constraints can come from a variety of sources. If it is know that two cysteine residues form a disulphide bond, then their atoms must be sufficiently close. Elements of protein secondary structure (α helices and strands in a β-sheet) are associated with tight distance constraints between main chain nitrogen and oxygen atoms that are involved in hydrogen bonds. From NMR experiments, the HN-HN distance constraints from NOEs determine the extent of the α helices as well as the relative position and orientation of strands in a β-sheet. No other short HN-HN distances are expected in a protein; these NOEs are therefore providing constraints with a very high information content. This allows, in contrast to regular structure calculations from NMR data, to rely on a very low number of constraints.

There can be upper and lower bounds on the values of main chain torsion angles. These can come from knowledge about the protein's secondary structure (e.g. based on HN-HN NOEs from NMR experiments) or torsion angle ranges predicted by TALOS+ from chemical shifts data [13].

The model structure ought to be consistent with whatever information there is about the protein (see Section 3). Further, the protein should be free from steric clashes, i.e. the chain should not fold in a way that causes one part of the protein to overlap with another part.

3 Case studies

To demonstrate the use of zipping and assembly with constraints, we have attempted to reconstruct the structure of proteins using only their amino acid residue sequences, secondary structure information (the extents of α-helix regions and information about anti-parallel bridges), and disulphide bond pairings as starting information.

The modelling process and results are presented here for two proteins: human p8MTCP1 and human defensin β-defensin 6. Both of these proteins have three disulphide bonds. Human p8MTCP1 (68 amino acid residues) has three α-helix regions, while human defensin β-defensin 6 (45 amino acid residues) has a mainly β-sheet structure with several anti-parallel bridges.

The modelling program starts by filling each cell on level 1 of the zipping and assembly data structure (Figure 3) with ϕ and ψ angle combinations from a library of real proteins taken from PDB database (Figure 2). For those residues that have angle constraints, the set of ϕ and ψ angle combinations is filtered. The

modelling program then proceeds to generate candiate models in each cell in level 2 and above in the zipping and assembly data structure. Distance constraints and steric overlap checks are used to filter the solutions that are put into each cell. The user can specify how many solutions should be created in each cell; in the modelling exercises described in this section, 1000 fragments are created in each cell.

3.1 Human p8MTCP1

The structure of human p8MTCP1 has been determined experimentally by NMR [PDB entry: 2HP8] [2] (Figure 4). This high level information is shown in Figure 5. From these facts, a large set of lower level distance and angle constraints can be derived. These are summarised in Figure 6.

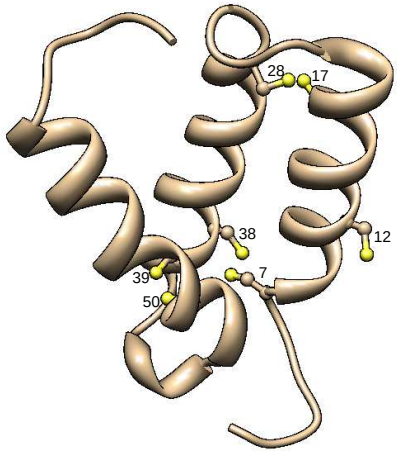


Fig. 4. Disulphide bonds in human p8MTCP1 (Protein Data Bank entry 2HP8). A disulphide bond can be formed between the sulphur atoms of a pair of spatially adjacent cysteine residues. The ribbon cartoon represents the main chain of human p8MTCP1. There are 7 Cys residues (side chains shown as ball-and-stick), 6 of which form three disulphide bonds (Cys7-Cys38, Cys17-Cys28, Cys39-Cys50).

If we have a constraint that Cα atoms of residues i and j must be within distance d from each other, this places upper distance bounds on atoms in the residues between i and j. The expected separation between the Cα atoms of two consecutive residues is 3.8 Å and the triangle inequality can be used to infer additional distance constraints involving the residues in between i and j. Prolog code for propagating distance constraints to lower cells in the zipping and assembly data structure is shown in Figure 7, and the result of applying this code is illustrated in Figure 6.

```
residue(1,'MET').
residue(2,'PRO').
residue(3,'GLN').
residue(4,'LYS').
residue(5,'ASP').  % etc.

disulphide_bond(7,38).
disulphide_bond(17,28).
disulphide_bond(39,50).

alpha_helix(8,20).
alpha_helix(29,39).
alpha_helix(48,62).
```

Fig. 5. Prolog facts describing structural features of human p8MTCP1.

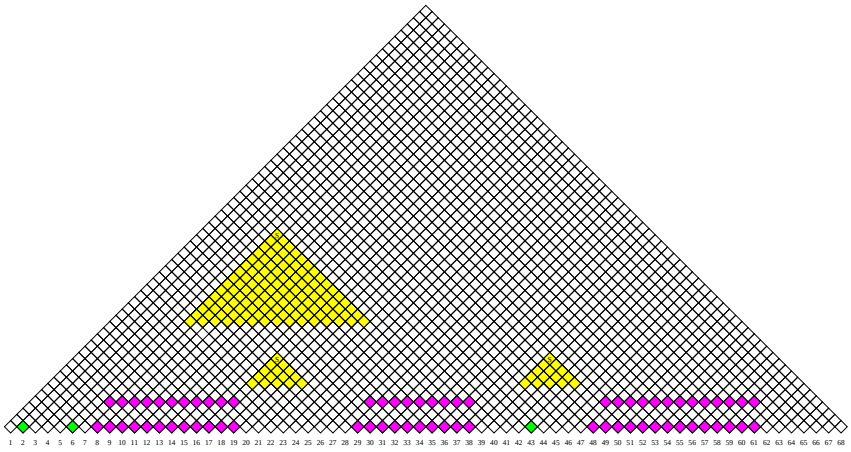


Fig. 6. Constraints used in modelling human p8MTCP. The yellow cells with letter "S" represent three disulphide bonds that provide tight distance constraints between pairs of residues at positions (7, 38), (17, 28) and (39, 50), and weaker distance constraints between pairs of residues represented by the yellow cells. Proline residues at positions 2, 6 and 43 have tight constraints on the range of possible values for their ϕ torsion angle (green cells). The helical regions have constraints on possible ϕ and ψ angle combinations (pink cells on level 1), and tight distance constraints between the main chain oxygen atom of residue i and the main chain nitrogen atom of residue $i+4$ (pink cells on level 4).

```
infer_upper_bounds :-
    upper_distance_bound((A,'CA'),(B,'CA'),Distance),
    between(I,A,B),
    between(J,I,B),
    MustNotBeMoreThan is (((B - A) - (J - I)) * 3.8) + Distance,
    CannotBeMoreThan is (J - I) * 3.8,
    MustNotBeMoreThan < CannotBeMoreThan,
    assert_upper_bound((I,'CA'),(J,'CA'),MustNotBeMoreThan),
    fail.

infer_upper_bounds.
```

Fig. 7. Prolog code for propagating distance constraints to lower cells.

The zipping and assembly method was run generating 1000 fragments in each cell in the data structure. Of the 1000 models built for the entire chain, the most similar model to the experimentally determined structure had a root mean square distance of 2.6 Å over all Cα atoms (Figure 8). Regions of greatest difference are close to the unconstrained ends of the chain. Comparing the 1000 models with each other the main differences are the orientation of the third helix which is only anchored to the second helix by a disulphide bond near one end, and the placement of the unconstrained residues near the ends of the chain.

Fig. 8. Main chain of modelled human p8MTCP, built using knowledge of the extents helical regions and its disulphide bridges superposed on the experimentally determined structure [PDB: 2HP8]. The colour gradient allows the chains to be followed easily from the N-terminal (residue 1, blue) to the C-terminal (residue 68, red).

3.2 Human β-Defensin 6

The structure of human β-defensin 6 has been determined experimentally by NMR [PDB entry: 2LWL] [6]. This structure contains four anti-parallel bridges (Figure 9).

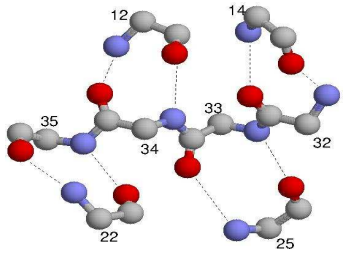


Fig. 9. Antiparallel bridges in human β-defensin 6 inferred from HN-HN NOEs.

Prolog rules are used to derive distance and torsion angles constraints from facts about the protein's structure (Figure 10). Distance constraints due to disulphide and anti-parallel bridges are illustrated in Figure 12.

```
residue(1,'PHE').
residue(2,'PHE').
residue(3,'ASP').
residue(4,'GLU').
residue(5,'LYS').  % etc.

disulphide_bond(6,33).
disulphide_bond(13,27).
disulphide_bond(17,34).

alpha_helix(4,8).

antiparallel_bridge(12,34).
antiparallel_bridge(14,32).
antiparallel_bridge(22,35).
antiparallel_bridge(25,33).
```

Fig. 10. Prolog facts describing structural features of human β-defensin 6.

Initially we were unable to obtain models that had the protein's native fold. However, there was additional information implicit in the high level description of structural features that could be used to infer some additional constraints.

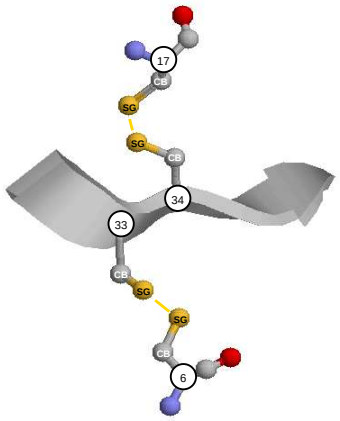


Fig. 11. The large grey arrow represents a strand in a β-sheet. Cα atoms are represented by numbered circles. Consecutive residues 33 and 34 lie within the strand, and both are involved in disulphide bonds (shown in yellow) with residues 6 and 17, respectively. This situation places upper and lower distance constraints between residues 6 and 17.

Specifically, within a strand in a β-sheet the side chains of consecutive residues are on alternate faces of the sheet, i.e. side chains of adjacent residues are oriented in opposite directions. In the case of human β-defensin 6, the residues at positions 33 and 34 are both cysteine residues involved in disulphide bonds. Since the side chains of residues 33 and 34 are oriented away from each other, the Cα atoms of their disulphide bond partners (residues 6 and 17) are expected to be between 13 Å and 15 Å apart. This situation is illustrated in Figure 11, and a rule for inferring these constraints has been implemented in Prolog (Figure 13).

The most similar model to the experimentally determined structure had a root mean square distance of 1.9 Å over Cα atoms in the core region spanning residues 6 to 35 (Figure 14). The greatest differences were the unconstrained ends of the chain. Superposed Cα traces of 50 model structures show that the models agree well in core, but vary considerably in their unconstrained regions near the ends of the chain (Figure 15).

4 Discussion

The system described in this paper is implemented in a combination of C and Prolog. The zipping and assembly method is implemented in C. Prolog is used for representing structural features at a high level (Figures 5 and 10), for generating low level distance and angle constraints from high level descriptions of structural features, for propagating distance constraints "downwards" in the zipping and assembly data structure (Figure 7), and for reasoning about high level

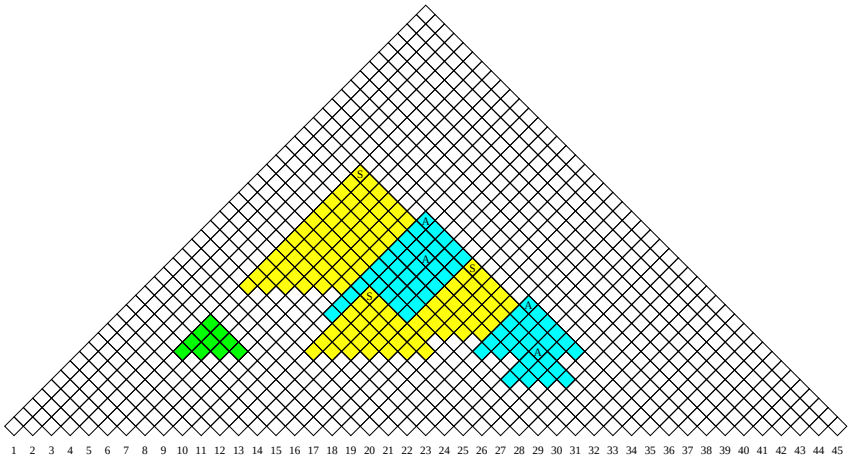


Fig. 12. Distance constraints due to disulphide and anti-parallel bridges mapped onto cells in the zipping and assembly data structure. Fragments generated in a particular cell are checked against the constraints in that cell, and only feasible fragments are retained. Disulphide (S) and anti-parallel (A) bridges in human β-defensin 6 give distance constraints between residue pairs indicated by the yellow and cyan cells. The green cells represent distance constraints imposed on residues 6 and 17, and nearby residues, by the situation shown in Figure 11.

structural information in order to infer additional constraints (Figure 13) that can be crucial in constructing a satisfactory model.

The constraints in the green cells in Figure 12 cannot be inferred from low level distance and angle constraints. These constraints depend on expert knowledge about protein structure that has been encoded as a Prolog rule (clause 2 of `disulphide_distance_constraints/0` in Figure 13).

The ability to propagate constraints downwards in the zipping and assembly data structure is important since this can improve execution times substantially, and sometimes makes it possible to find solutions that would not otherwise be found, due to many unpromising fragments "clogging up" the lower cells in the zipping and assembly data structure.

4.1 Related work

Several examples of other work on protein modelling where constraint-based methods are used are discussed here.

The PSICO system [10] constructs protein models that minimise constraint violations, where many thousands of distance constraints from NMR experiments are taken into consideration. In the work described in this paper, we attempt to build models using much smaller sets of constraints (roughly two orders of mag-

```
disulphide(A,B) :- disulphide_bond(A,B).
disulphide(A,B) :- disulphide_bond(B,A).

in_antiparallel_bridge(X) :- antiparallel_bridge(X,_).
in_antiparallel_bridge(X) :- antiparallel_bridge(_,X).

disulphide_distance_constraints :-
    disulphide_bond(A,B),
    assert(lower_distance_bound((A,'CA'),(B,'CA'),4.0)),
    assert(upper_distance_bound((A,'CA'),(B,'CA'),7.0)),
    fail.

disulphide_distance_constraints :-
    disulphide(A,B),
    disulphide(C,D),
    1 is C-B,
    in_antiparallel_bridge(B),
    in_antiparallel_bridge(C),
    assert(lower_distance_bound((A,'CA'),(D,'CA'),13.0)),
    assert(upper_distance_bound((A,'CA'),(D,'CA'),15.0)),
    fail.

disulphide_distance_constraints.
```

Fig. 13. Prolog code for inferring distance constraints between disulphide bond partners and adjacent residues. The first clause of `disulphide_distance_constraints/0` asserts upper and lower distance constraints between the Cα atoms of the two residues that are involved in the disulphide bridge. The second clause of `disulphide_distance_constraints/0` tests whether residues B and C are adjacent in the protein chain, are both present in a strand and asserts upper and lower constraints on the distance between the Cα atoms of the disulphide bond partners of residues B and C.

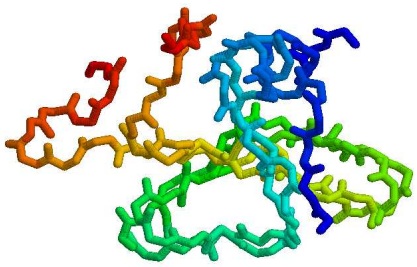


Fig. 14. Main chain model of human β-defensin 6 superposed on the main chain of experimentally determined structure from PDB entry 2LWL.

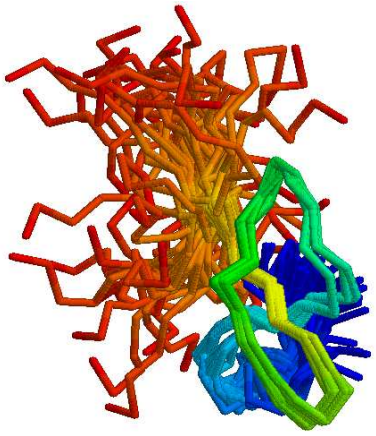


Fig. 15. Cα traces of 50 models of human β-defensin 6. The structures of the core are in good agreement, and the terminal regions are dynamic.

nitude smaller). In addition to distance constraints, we make use of constraints on torsion angle values.

Dal Palù *et al.* [5] use a CLP-based method to model proteins using fragment assembly. In their work they use a reduced representation of amino acid residues in which each residue is represented by its Cα atom and the centroid of its side chain. By using all main chain heavy atoms in the work presented here, it is easier to directly use constraints on the values of ϕ and ψ torsion angles that are predicted from chemical shifts in NMR experiments.

Backofen and Will [1] use a constraint-based approach to a lattice model of protein folding, where a sequence of hydrophobic and polar "amino acid residues" are folded onto a regular grid. The zipping and assembly method was first described in the context of lattice models [7, 9]. While lattice models are a gross over-simplification of the real protein folding problem, they provide a convenient framework for experimenting with search strategies and simplified scoring functions [11].

Traoré *et al.* [15] address a different problem where the aim is to select which side chains can be attached to a rigid protein main chain, taking into consideration predicted conformations of these side chains and pairwise energies between side chains. In contrast, the work described here attempts to model the main chain.

4.2 Future work

We are extending our program so that longer fragments that are consistent with given constraints, taken from a library of known protein structures, can be used directly when modelling, thus making better use of known structural information

that is present in the Protein Data Bank. Instead of starting with only the conformations of single residues in the cells in level 1 of the zipping and assembly data structure, we shall then be able to place some longer fragments directly into cells at higher levels. Doing this should both improve the performance of the program (the current program typically takes several minutes to run on a laptop) and to improve the quality of the models that are constructed.

Scoring functions should be added to evaluate and filter the suggested fragment conformations in the higher cells of the zipping and assembly data structure. Currently any fragment that does not violate the given constraints is accepted. We would expect better results by scoring and selecting fragments.

So far our focus has been on modelling protein main chains, and we have not placed side chains carefully. Side chain modelling could be carried out within the zipping and assembly framework, or in a separate modelling step that follows construction of the main chain (e,g, [14]).

We are working to improve the performance and memory usage of the program. Execution time is not only related to protein length (assuming sufficient main memory), but is also affected by "contact order" [12]. High contact order implies constraints "higher" in the ZAM data structure (Figure 12); low contact order structures (with "lower constraints", Figure 6) are easier and faster to model.

5 Conclusions

We have implemented a method for modelling protein main chains that uses high level declarative descriptions of protein features as its starting point. Lower level distance and angle constraints can be generated automatically from these. Declarative Prolog rules are used to propagate distance constraints so that unpromising solutions are pruned early. The zipping and assembly method for exploring the vast conformational space has several benefits [7]. We have implemented a zipping and assembly method that makes use of given and derived constraints to guide the search towards feasible solutions. Expert structural knowledge that can be crucial in finding satisfactory solutions is expressed as declarative Prolog rules that are used to infer additional constraints.

Acknowledgements

This work is supported by a Project Research Grant from the Swedish Research Council (621-2011-6171)

References

1. Backofen, R., Will, S.: A constraint-based approach to fast and exact structure prediction in three-dimensional protein models. Constraints 11(1), 5–30 (2006)

2. Barthe, P., Yang, Y.S., Chiche, L., Hoh, F., Strub, M.P., Guignard, L., Soulier, J., Stern, M.H., van Tilbeurgh, H., Lhoste, J.M., Roumestand, C.: Solution structure of human p8 MTCP1, a cysteine-rich protein encoded by the MTCP1 oncogene, reveals a new α-helical assembly motif. J. Mol. Biol. 274, 801–815 (1997)
3. Berman, H., Henrick, K., Nakamura, H.: Announcing the worldwide Protein Data Bank. Nat. Struct. Biol. 10, 980 (2003)
4. Campeotto, F., Dal Palù, A., Dovier, A., Fioretto, F., Pontelli, E.: A constraint solver for flexible protein model. J. Artif. Intell. Res. 48, 958–1000 (2013)
5. Dal Palù, A., Dovier, A., Fogolari, F., Pontelli, E.: CLP-based protein fragment assembly. Theory and Practice of Logic Programming 10, 709–724 (2010)
6. De Paula, V.S., Gomes, N.S.F., Lima, L.G., Miyamoto, C.A., Monteiro, R.Q., Almeida, F.C.L., Valente, A.P.: Structural Basis for the Interaction of Human β-Defensin 6 and Its Putative Chemokine Receptor CCR2 and Breast Cancer Microvesicles. J. Mol. Biol. 425, 4479–4495 (2013)
7. Dill, K.A., Lucas, A., Hockenmaier, J., Huang, L., Chiang, D., Joshi, A.K.: Computational linguistics: A new tool for exploring biopolymer structures and statistical mechanics. Polymer 48, 4289–4300 (2007)
8. Engh, R.A., Huber, R.: Accurate bond and angle parameters for X-ray protein structure refinement. Acta Crystallogr. A 47(4), 392–400 (1991)
9. Hockenmaier, J., Joshi, A.K., Dill, K.A.: Routes Are Trees: The Parsing Perspective on Protein Folding. Proteins: Structure, Function and Bioinformatics 66, 1–15 (2007)
10. Krippahl, L., Barahona, P.: PSICO: Solving Protein Structures with Constraint Programming and Optimization. Constraints 7, 317–331 (2002)
11. Lau, K.F., Dill, K.A.: A Lattice Statistical Mechanics Model of the Conformational and Sequence Spaces of Proteins. Macromolecules 22, 3986–3997 (1989)
12. Plaxco, K.W., Simons, K.T., Baker, D.: Contact order, transition state placement and the refolding rates of single domain proteins. J. Mol. Biol. 277, 985–994 (1998)
13. Shen, Y., Lange, O., Delaglio, F., Rossi, P., Aramini, J.M., Liu, G., Eletsky, A., Wu, Y., Singarapu, K.K., Lemak, A., Ignatchenko, A., Arrowsmith, C.H., Szyperski, T., Montelione, G.T., Baker, D., Bax, A.: Consistent blind protein structure generation from NMR chemical shift data. Proc. Natl. Acad. Sci. USA 105, 4685–4690 (2008)
14. Swain, M.T., Kemp, G.J.L.: A CLP approach to the protein side-chain placement problem. In: Walsh, T. (ed.) Principles and Practice of Constraint Programming — CP2001. pp. 479–493. Springer-Verlag (2001)
15. Traoré, S., Allouche, D., André, I., de Givry, S., Katsirelos, G., Schiex, T., Barbe, S.: A new framework for computational protein design through cost function network optimization. Bioinformatics 29(17), 2129–2136 (2013)

Constraining redundancy to improve protein docking

Ludwig Krippahl and Pedro Barahona

NOVA-LINCS, DI, FCT-NOVA,
2829-516 Caparica, Portugal
ludi@fct.unl.pt
pb@fct.unl.pt

Abstract. Predicting protein-protein complexes (protein docking) is an important factor for understanding the majority of biochemical processes. In general, protein docking algorithms search through a large number of possible relative placements of the interacting partners, filtering out the majority of the candidates in order to produce a manageable set of candidates that can be examined in greater detail. This is a six-dimensional search through three rotational degrees of freedom and three translational degrees of freedom of one partner (the probe) relative to the other (the target). The standard approach is to use a fixed step both for the rotation (typically 10^o to 15^o) and the translation (typically 1Å). Since proteins are not isotropic, a homogeneous rotational sampling can result in redundancies or excessive displacement of important atoms. A similar problem occurs in the translational sampling, since the small step necessary to find the optimal fit between the two molecules results in structures that differ by so little that they become redundant. In this paper we propose a constraint-based approach that improves the search by eliminating these redundancies and adapting the sampling to the size and shape of the proteins involved. A test on 217 protein complexes from the protein-protein Docking Benchmark Version 5 shows an increase of over 50% in the average number of non-degenerate acceptable models retained for the most difficult cases. Furthermore, for about 75% of the complexes in the benchmark, computation time is decreased by half, on average.

Keywords: protein docking; geometric search; constraints

1 Introduction

Protein interactions are crucial in any living organism, since proteins make up most of the biochemical machinery of the cell. Proteins are also the main product of genes and thus lie at the base of the phenotypic expression of the genome and of all metabolism. This is why understanding protein interactions is so important for both theoretical and practical reasons. From the elucidation of biochemical mechanisms to medicine and drug design, predicting how proteins fit together

provides useful information. Given the difficulty of studying protein-protein interactions experimentally and the progress in high-throughput methods of determining individual protein structures [5], modelling protein interactions from the known structures of the interacting partners is bound to remain an important tool for biochemical and medical research.

There are two main approaches to predicting protein-protein complexes, or protein docking. Some algorithms use local search or stochastic sampling methods, such as genetic algorithms [15] or simulated annealing [2], to maximize a scoring function that estimates how favourable the interaction is. This function takes into account several factors such as solvation effects, entropy and electrostatics, which limits the configurations that can be sampled during the search. This is why most docking algorithms use geometric complementarity as a first filtering criterion [1, 7, 11, 12, 14], postponing a more detailed evaluation to a second stage, limited to a smaller set of selected candidates. This allows a systematic search through a very large number of possible configurations – on the order of 10^{15} or more – by rotating one of the proteins, the probe, relative to the other, the target, and exploring all translations for each orientation in small steps. Each configuration is then evaluated according to geometric complementarity in order to retain the best candidates. This relatively smaller number of candidates – 10^3 to 10^4 – is then scored in more detail and ranked in the evaluation step. This paper focuses on optimizing the search in the filtering step. We use the BiGGER (Bimolecular complex Generation with Global Evaluation and Ranking) docking program, which has a geometric search algorithm based on constraint programming techniques. BiGGER uses constraint propagation to speed up the translational search by eliminating the majority of unproductive configurations, such as those with forbidden overlaps or insufficient surface contact [8] and also to restrict the translational search according to predicted or observed points of contact between the interacting partners [9]. In this paper, we describe the application of the same constraint processing ideas to optimize the search through the rotational space and the selection of the best candidate models by imposing constraints on the redundancy of the candidates.

1.1 Uniform rotational search

The standard approach for the rotational search in exhaustive search docking algorithms is to generate a set of uniformly distributed rotations by rotating the probe molecule around the three orthogonal axis in a constant step, typically ranging from 6^o to 15^o. Although the implementation details vary, one usual method is to sample the combinations of rotations around the x, y and z axis. If we rotate in steps of 15^o, this would mean a total of $12 \times 24 \times 24$ orientations, since there are 24 steps of 15^o around each axis but, for one axis, we would only need half a turn to avoid repeating orientations. This results in some degenerate orientations and can be improved by selecting 12×24 uniformly distributed axes and then rotating 23 steps around each one (the 0^o and 360^o rotations around any arbitrary axis all lead to no rotation) and then adding the original orientation, for a total for $12 \times 24 \times 23 + 1$ is 6625 orientations. This is the default uniform

rotational search algorithm used in BiGGER. It starts by creating $24 \times 24 = 576$ points uniformly spread on the surface of a sphere of radius 1, using the spiral method [13], and then selecting all those with $z \geq 0$ to define 288 rotation axes [1] Each rotation axis is then used to generate 23 quaternions specifying the 23 different rotations of 15^{o} around the axis, and finally the original orientation (no rotation) is also kept, for the total of 6625 orientations.

1.2 The search dilemma

While a finer sampling of both the rotation and translation space is desirable in order to find the best fit between the two proteins, besides the increase in computation time, it also leads to a larger number of very similar configurations. This, in turn, increases the chance that incorrect configurations push all acceptable candidates out of the set of retained models. However, a coarser sample may result in missing the most favourable ways of fitting the proteins and thus not find any acceptable models with a good enough geometric fit to be retained in the filtered set. It is this dilemma that motivates our addition of redundancy constraints. With the method described below, we adapt the rotation sampling to the size and shape of the protein in order that the atom displacement between neighbouring orientations is as close to uniform as possible. This gives us a better control of the trade-off between searching too many or too few orientations. In addition, we constrain the set of retained models to avoid keeping models that are too similar. This way we can do a translation search in small steps of 1Å, in order to better optimize the geometrical fit, but also mitigate the crowding out of acceptable candidates by groups of redundant models.

2 Method

One possibility for optimizing the rotation sampling is to take into account the shape of the probe protein and apply a different constraint to the generation of the rotations. Instead of requiring a constant step, it would be better to require that the maximum displacement of the atoms from one rotation to its nearest neighbour be constant. This would result in a distribution of orientations that is heterogeneous in the angular step but more homogeneous in the atomic distances since proteins are not spherical and can have highly irregular shapes. Another possibility is to prune the rotations searched to reduce the number of orientations used. The rationale for this approach is that, for adequate results, smaller or more globular proteins should need a smaller number of orientations than larger or more irregularly-shaped proteins. The following subsections detail each of these approaches.

[1] Or 289 rotation axes, in some previous implementations, depending on exactly how the points are spread with respect to this cutoff for the hemisphere.

2.1 Redundancy in retained models

The standard approach to the filtering stage of the docking process is to retain the set of models with the highest scores for the scoring function used in this stage. Given the large search space, this function must be inexpensive to evaluate, and is generally based on estimates of the contact surface [4]. This is the current procedure in BiGGER. However, retaining the highest scoring models can result in redundancies because the step size in the translation search, which in BiGGER is 1Å, can be small enough to result in multiple nearly identical models. On the other hand, a larger translation step could result in missing the most favourable configurations. One way of solving this dilemma is to restrict the models retained so that, of two sufficiently similar models, only the highest scoring one is kept. This consideration, along with the asymmetry of the probe, lead us to the following constraints for reducing redundancy.

2.2 Defining the constraints

To prevent redundancy in rotation, we define the distance between two rotations as:

Definition 1. *Distance between rotations*

Let S be the set of N points with coordinates $c_1, c_2, ..., c_n$ and ρ_1 and ρ_2 two rotations. The distance between rotations ρ_1 and ρ_2 with respect to S is the largest distance between the pair of images for each point:

$$\mathrm{dist}_S(\rho_1, \rho_2) = \max_{c_i \in S} |\rho_1(c_i) - \rho_2(c_i)|$$

The rotation constraint that reduces the angular redundancy:

Constraint 1. *Rotation redundancy constraint*

Given a set of points S defining the shape of the probe, a distance parameter δ specifying the smallest non-redundant displacement, and a set R of rotations, then:

$$\forall \rho_i, \rho_{j \neq i} \in R, \mathrm{dist}_S(\rho_i, \rho_j) > \delta$$

Evidently, this constraint is trivial to fulfil with an empty set of rotations. However, we also want to cover all the possible orientations of the probe as well as possible. So the goal is to find the largest set of rotations that respect constraint 1.

To reduce redundancy in retained models, we also add the following constraint:

Constraint 2. *Model redundancy constraint*

Let M be the set of retained models, where each model $m_{\boldsymbol{t},\rho}$ is determined by the translation vector $\boldsymbol{t}$ and rotation ρ. Given a radius r specifying the redundancy neighbourhood, then:

$$\forall m_{\boldsymbol{t_i},\rho_i}, m_{\boldsymbol{t_j},\rho_j} \in M, \rho_i = \rho_j \implies |\boldsymbol{t_i} - \boldsymbol{t_j}| > r$$

2.3 Implementing the constraints

To speed up the calculation, we use a set of 20 atoms to represent the protein shape. These atoms are selected by first picking the atom farthest from the centre and then iteratively picking the atom with the largest distance to the closest atom in those previously picked. This is the set of points over which rotation distances will be calculated. We then create a list of uniformly distributed rotations with a 7.5° step for a total of $24 \times 48 \times 47 + 1 = 54145$ rotations. From this oversampling of the rotational search space we pick the original orientation as the first element of the set of the selected rotations. Then, iteratively, of all rotations for which the minimum distance to any in the selected set is larger than δ, we add the one with the smallest minimum distance to the selected set. We repeat this until no rotation is left that has a minimum distance to the selected set that is larger than δ. This method gives us a distribution of rotations that is more homogeneous in atomic displacement, adapted both to the shape and to the size of the protein, resulting in fewer orientations for smaller proteins and more orientations for larger proteins. For the work related here, the δ value for this rotation constraint was 6Å.

Since this is a greedy optimization, it does not guarantee the maximum possible number of rotations or the ideal distribution. However, it is fast. Building the set of rotations takes from a few seconds to a few minutes for each complex, less than 1% of the total docking time. Nevertheless, we are working on improving the set of rotations, both by improving the generation algorithm to optimize the distribution of rotations and to fine-tune the δ value.

The model redundancy constraint is implemented by storing a temporary list of the best candidate models during the translational search for a given orientation of the probe. Once the translational search for that orientation is complete, the sorted candidates are examined starting from the model with the largest surface contact and, whenever the program adds one model to the final list of selected models, it removes all models in the temporary list for which the translation vector is within 2.5Åof the inserted model in all three coordinates. This parameter also needs to be optimized by experimenting with different values, as well as the decision to use a cubic neighbourhood instead of a spherical one. These adjustments are still work in progress.

2.4 Benchmark tests

To test our method, we used the protein-protein Docking Benchmark Version 5 [16]. This is a benchmark of 230 cases of unbound docking of protein complexes. However, these 230 examples only span 225 protein complexes, as five of the 230 examples are additional binary complexes drawn from some large complexes in the pool of 225. Of these 225 complexes, in 217 the length of the probe was distributed with an approximately normal distribution with an average of 55Å and a standard deviation of 13Å, and ranging from 25Å through 86Å. The length of the probe was measured as the largest distance between any pair of atoms in the protein. This is an important measure because it affects both the number

of orientations necessary to adequately sample the rotational space and the size of the translational search space. In the other eight complexes the probe length was spread out from 98Å to 141Å. This sparse set of a few but extreme outliers caused difficulties in aggregating the results as a function of the length of the probe, which was necessary for judging the effects of the constraints in different conditions. For this reason, we ended up rejecting these eight complexes, leaving us with a benchmark set of 217 different complexes, each involving two proteins.

The protein-protein Docking Benchmark provides the unbound structures and the target complexes recreated by rigidly fitting the unbound partners to the known complex structure. These were the target complexes we used to evaluate the performance of the docking algorithm. In some cases, there are significant conformational changes between the bound and unbound proteins. Using these target complexes provided by the benchmark, we can evaluate the rmsd[2] of the docking predictions without adding the irreducible remaining error due to the conformational changes the proteins undergo when interacting, which can be as high as 8Å. Nevertheless, these conformational changes still add to the difficulty of the unbound docking, since we are trying to predict the correct fit of proteins that are not in the ideal conformation for fitting together, as would generally be the case in a real application. In addition to using the unbound conformations, we also rotated each probe protein randomly before docking. Other than the constraints described here, the docking predictions were run with the default parameters used by BiGGER and retaining a set of 5000 models.

To evaluate the results, we considered a model to be acceptable if the rmsd value computed for the probe was below 10Åand the rmsd value for the atoms at the interface was below 4Å. The interface is the set of atoms within 5Å of any atom of the other partner. These are criteria used in the CAPRI programme for assessing predictions of protein interactions [6].

3 Results and discussion

Figure 1 shows the relative effect of using both redundancy constraints compared to the base search algorithm of uniform rotation sampling and not discarding redundant models. These relative values are computed dividing the values for the docking runs with the redundancy constraints by the respective values for the base docking runs. The lines are smoothed using a Gaussian kernel with 5Å of standard deviation for each data point. The failure rate is the proportion of complexes for which no acceptable model was retained in the final set of 5000 candidates. The relative failure rate is always below one, meaning that the redundancy constraints result in a lower failure rate than the corresponding docking runs using the base algorithm. The relative number of non-redundant acceptable models is nearly always higher than one, which shows that the redundancy constraints increase the average number of acceptable models retained in the final set. The relative time is below one for the first three quartiles, meaning that, in

[2] The square root of the mean of the squared atomic deviations, in Ångstrom

75% of the cases, the number of orientations with the fixed displacement of 6Åis smaller than the default number of orientations. However, since they are better distributed, even with a smaller number of orientations to search the results are better than the results of the base method. For larger probe sizes, the number of rotations is larger than 6625, the default used by BiGGER. In this quartile, computation times can increase significantly. However, the average number of acceptable models retained also increases significantly, suggesting that the base set of 6625 orientations was inadequate for larger probes.

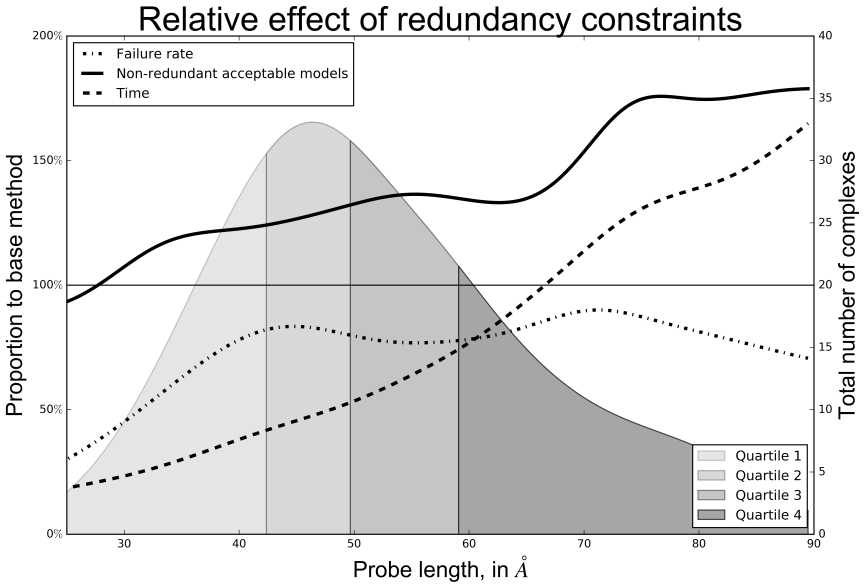


Fig. 1. The lines show the relative effects of the redundancy constraints on failure rate, average number of non-redundant acceptable models retained and computation time as a function of probe length. The values are in proportion to the values obtained with the base algorithm. The shaded curve shows the length distribution of the probes in the benchmark examples, divided into quartiles. All plots were smoothed using a Gaussian kernel with $\sigma = 5\mathring{A}$.

Table 1 shows the values comparing, for each quartile, four different cases. The *Base* case is the basic BiGGER algorithm without any redundancy constraints. The *Models* case uses Constraint 2 to prevent redundancy on the models retained. *Rotations* uses Constraint 1 to adapt the set of rotations to the shape and size of the probe, so that no two different orientations result in a maximum atomic displacement below 6Å. Finally, *Full* uses both constraints. The results are aggregated by each quartile of the distribution of probe lengths, with the last column showing the averages for all 217 test complexes. Each value is the

average value plus or minus the standard deviation for the average estimated by bootstrapping [3] with 10,000 replicas .

Table 1. Effect of redundancy constraints, by quartile

Average number of non-redundant models by complex					
Quartile	0-25%	25-50%	50-75%	75-100%	All
Base	13.0 ± 2.7	4.7 ± 0.7	3.8 ± 0.7	2.6 ± 0.4	6.1 ± 0.8
Model	18.9 ± 3.8	6.8 ± 1.2	5.1 ± 0.9	3.4 ± 0.5	8.6 ± 1.1
Rotation	9.3 ± 1.2	4.1 ± 0.5	3.6 ± 0.5	3.1 ± 0.6	5.1 ± 0.4
Full	14.6 ± 1.8	5.9 ± 0.8	5.3 ± 0.7	4.1 ± 0.7	7.5 ± 0.6

Percentage rate of failed predictions					
Quartile	0-25%	25-50%	50-75%	75-100%	All
Base	$9\% \pm 4\%$	$13\% \pm 5\%$	$28\% \pm 6\%$	$33\% \pm 6\%$	$21\% \pm 3\%$
Model	$9\% \pm 4\%$	$7\% \pm 4\%$	$15\% \pm 5\%$	$24\% \pm 6\%$	$14\% \pm 2\%$
Rotation	$9\% \pm 4\%$	$11\% \pm 4\%$	$22\% \pm 6\%$	$37\% \pm 7\%$	$20\% \pm 3\%$
Full	$5\% \pm 3\%$	$13\% \pm 5\%$	$20\% \pm 5\%$	$28\% \pm 6\%$	$17\% \pm 3\%$

Average computation time by complex (hours)					
Quartile	0-25%	25-50%	50-75%	75-100%	All
Base	0.7 ± 0.0	1.2 ± 0.1	1.4 ± 0.1	2.7 ± 0.2	1.5 ± 0.1
Model	0.7 ± 0.0	1.2 ± 0.1	1.5 ± 0.1	2.8 ± 0.2	1.5 ± 0.1
Rotation	0.2 ± 0.0	0.5 ± 0.0	0.8 ± 0.1	2.9 ± 0.3	1.1 ± 0.1
Full	0.2 ± 0.0	0.6 ± 0.0	0.9 ± 0.1	3.2 ± 0.4	1.2 ± 0.1

This table compares the basic BiGGER algorithm with docking imposing the constraint on model redundancy, the constraint on rotation redundancy and both. The results are aggregated by quartile of the distribution of probe lengths. The last column shows the aggregate values for all 217 test complexes.

The number of non-redundant complexes retained in the final set of 5000 candidates is relevant for estimating the difficulty of identifying the correct models within this set. Other factors being equal, after the second stage of evaluating this set with a more detailed scoring function, the greater the number of acceptable models present the easier it should be to pinpoint the correct complex structure. Looking at the four quartiles in the distribution of probe lengths, we can see that docking small probes is easier, resulting in around 10 to 20 non-redundant acceptable models. The failure rate, which is the percentage of complexes for which no acceptable model was retained in this final set of 5000, is also lowest for smaller probes. Thus, in this quartile, the most significant gain by combining the two constraints is in computation time, which is reduced to nearly a quarter of the time for the unconstrained docking. Using the rotation redundancy constraint alone can give us this performance improvement but results in a lower average number of acceptable models, although the failure rate is not significantly different.

On the second quantile, the results are similar, differing only in that the time decrease is less marked, at around 50%, and the increase in the average number of acceptable models seems to be greater, though still not significant. On the third quartile, however, the average number of non-redundant acceptable models retained is significantly higher for the dockings with both constraints than it is with the unconstrained dockings, and slightly higher than any of the constraints alone. The best explanation for this seems to be the antagonistic effects of increasing the sampling of the rotation space. While, on the one hand, such an increase increases the probability of not missing acceptable models during the search, on the other hand, increasing the sampling density also increases the number of incorrect models which can displace acceptable candidates from the final set of 5000 models to retain. This seems to be why the two constraints combined outperform either one alone, with the model redundancy constraint mitigating the negative effects of increasing the rotational sampling as the probe length increases, while the rotational redundancy constraint leads to a sample of orientations better suited to the shape and size of the probe.

On the last quartile this improvement in the average number of acceptable models retained is even more marked, being over 50%. This is particularly relevant because these are the hardest complexes to predict. The tendency for larger proteins to suffer greater conformational changes and the larger number of models to filter during the search result in significantly fewer acceptable models being retained in the final set and increases the chances that none will be retained. This is clear from the absolute values in the table. In these conditions, the redundancy constraints provide an important advantage in the slight decrease in failure rate and, in particular, in the significant increase in the number of acceptable models retained. In these more difficult complexes, however, the number of orientations sampled using the rotational redundancy constraint becomes larger than the default of 6625, and thus the computation time also increases.

4 Conclusions and future work

This paper presents an improvement on the search and filtering stage of protein docking using principles of constraint programming. By imposing constraints that prevent redundancies in the rotational search and the retention of candidate models, the average number of acceptable models increases, failure rates decrease slightly and computation times decrease in 75% of the cases. Of greater advantage, the average number of acceptable models retained increases significantly in the quartile corresponding to the most difficult complexes to model, where an improvement in the quality of the results is most important. Furthermore, since this is an algorithmic improvement that requires no additional data, it can be combined with other constraints that BiGGER can use, such as symmetry constraints [10] or constraints derived from predicted contacts [9].

There are still some open issues that we are currently exploring. The values of 6Å for the displacement in the rotation constraint and 2.5Åfor the redundancy of the models retained seem intuitively reasonable but must be systematically

compared to alternative values in order to optimize these constraints. Furthermore, it is quite possible that the optimal values depend on the size of the probe, especially for larger probes. We expect that the results presented here can be improved by fine-tuning these parameters and, possibly, adapting them to the size of the proteins involved. This work also focused on the effects of using these constraints on the search and filtering stage. We are currently working on testing these modifications on the full BiGGER docking pipeline currently under development, which begins with the prediction of likely contacts from from sequence data, proceeds with the constrained dockins and ends with the screening of the retained complexes using more detailed scoring function.

The source code for the implementation of the methods described here is available as part of the Open Chemera Library, at `https://github.com/lkrippahl/Open-Chemera`. The source code is published in the public domain and is free of any copyright restrictions.

Acknowledgements

This work was partially supported by funding from FCT MCTES and NOVA LINCS, UID/CEC/04516/2013.

References

1. Chen, R., Li, L., Weng, Z.: Zdock: an initial-stage protein-docking algorithm. Proteins: Structure, Function, and Bioinformatics 52(1), 80–87 (2003)
2. Dominguez, C., Boelens, R., Bonvin, A.M.J.J.: Haddock: a protein-protein docking approach based on biochemical or biophysical information. J Am Chem Soc 125(7), 1731–1737 (Feb 2003)
3. Efron, B.: Bootstrap methods: Another look at the jackknife. The Annals of Statistics 7(1), 1–26 (1979)
4. Halperin, I., Ma, B., Wolfson, H., Nussinov, R.: Principles of docking: An overview of search algorithms and a guide to scoring functions. Proteins: Structure, Function, and Bioinformatics 47(4), 409–443 (2002)
5. Hura, G.L., Menon, A.L., Hammel, M., Rambo, R.P., Poole Ii, F.L., Tsutakawa, S.E., Jenney Jr, F.E., Classen, S., Frankel, K.A., Hopkins, R.C., et al.: Robust, high-throughput solution structural analyses by small angle x-ray scattering (saxs). Nature methods 6(8), 606–612 (2009)
6. Janin, J.: Assessing predictions of protein-protein interaction: the capri experiment. Protein Sci 14(2), 278–283 (Feb 2005)
7. Katchalski-Katzir, E., Shariv, I., Eisenstein, M., Friesem, A.A., Aflalo, C., Vakser, I.A.: Molecular surface recognition: determination of geometric fit between proteins and their ligands by correlation techniques. Proceedings of the National Academy of Sciences 89(6), 2195–2199 (1992)
8. Krippahl, L., Barahona, P.: Applying constraint programming to rigid body protein docking. In: Beek, P. (ed.) Principles and Practice of Constraint Programming - CP 2005. Lecture Notes in Computer Science, vol. 3709, pp. 373–387. Springer, Berlin Heidelberg (2005)

9. Krippahl, L., Barahona, P.: Protein docking with predicted constraints. Algorithms for Molecular Biology 10(1), 9 (2015)
10. Krippahl, L., Barahona, P.: Symmetry constraints for modelling homo-oligomers. In: 11th Workshop on Constraint Based Methods for Bioinformatics (2015)
11. Palma, P.N., Krippahl, L., Wampler, J.E., Moura, J.J.: Bigger: a new (soft) docking algorithm for predicting protein interactions. Proteins 39(4), 372–384 (Jun 2000)
12. Roberts, V.A., Thompson, E.E., Pique, M.E., Perez, M.S., Ten Eyck, L.: Dot2: Macromolecular docking with improved biophysical models. Journal of computational chemistry 34(20), 1743–1758 (2013)
13. Saff, E.B., Kuijlaars, A.B.: Distributing many points on a sphere. The mathematical intelligencer 19(1), 5–11 (1997)
14. Schneidman-Duhovny, D., Inbar, Y., Polak, V., Shatsky, M., Halperin, I., Benyamini, H., Barzilai, A., Dror, O., Haspel, N., Nussinov, R., et al.: Taking geometry to its edge: fast unbound rigid (and hinge-bent) docking. Proteins: Structure, Function, and Bioinformatics 52(1), 107–112 (2003)
15. Taylor, J.S., Burnett, R.M.: Darwin: a program for docking flexible molecules. Proteins: Structure, Function, and Bioinformatics 41(2), 173–191 (2000)
16. Vreven, T., Moal, I.H., Vangone, A., Pierce, B.G., Kastritis, P.L., Torchala, M., Chaleil, R., Jiménez-García, B., Bates, P.A., Fernandez-Recio, J., et al.: Updates to the integrated protein–protein interaction benchmarks: Docking benchmark version 5 and affinity benchmark version 2. Journal of molecular biology 427(19), 3031–3041 (2015)

Global Optimization Methods for Genome Scaffolding

Sebastien François[1], Rumen Andonov[1], Hristo Djidjev[2], and Dominique Lavenier[1]

[1] IRISA/INRIA, Rennes, France
[2] Los Alamos National Laboratory, Los Alamos, NM 87545, USA

Abstract. We develop a method for solving genome scaffolding as a problem of finding a long simple path in a graph defined by the contigs that satisfies additional constraints encoding the insert-size information. Then we solve the resulting mixed integer linear program to optimality using the Gurobi solver. We test our algorithm on several chloroplast genomes and show that it is fast and outperforms other widely-used assembly algorithms by the accuracy of the results.

Keywords: genome assembly, scaffolding, contig, MILP, longest path problem, Gurobi

1 Introduction

High throughput sequencing (HTS) technologies provide millions of short genome fragments, called *reads*, which need to be assembled into a genome of interest. Today, most de-novo assemblers are based on de Bruijn graphs [12], whose vertices are all k-mers (k-length subwords of the reads) and whose edges connect all pairs of k-mers that share $k-1$ consecutive characters. Genomes can then be sought as maximal unambiguous paths in the de Bruijn graph. However, complex regions of the genome (i.e. regions with many repeats) generally fail to be assembled by this technique: if there are repeats (identical subregions of the genome) longer than the size of the reads, the entire genome cannot be built in a unique way. Various heuristics are used to bypass simple repeats, but they do not guarantee correct solutions. Hence, producing a full genome consists of several steps, namely assembly, scaffolding, and finishing, where the first step generates a list of *contigs* that represent the easy assembly regions of the genome.

In the second step, *scaffolding*, which is the focus of this paper, reads are linked together into *scaffolds*, which consist of sequence of contigs that overlap or are separated by gaps of given length. These gaps are generated during sequencing based on *paired-end* or *mate pair* reads [14,10]. These special reads can be represented as couples of fragments separated by a known distance (called *insert size*). They bring a long distance information that can be used for connecting contigs generated by de-Bruijn graphs. Scaffolding, which uses the set of contigs found during assembly and the insert-size information, is a very challenging problem, whose difficulties are rooted in technology imperfections including:

– Insert sizes are not precise. The technology provides approximate distance information only.
– For short contigs, which often represent short repeat regions, multiplicity cannot be precisely determined. This information is given by analyzing the coverage. Shorter the contig, worse the estimation.
– There may be erroneous contigs. Heuristics implemented for generating contigs may lead to chimeric sequences that wrongly connect two regions of the genome.

The difference between a contig and a scaffold is that a scaffold can contain regions that have not been completely solved. For example, for two contigs that have been unambiguously linked, the nucleotides sequence between them may have not been determined due to sequencing problem, or very high structure complexity. A last step (*finishing*) is generally required to enhance the scaffold. Additional information, such as sequences of close species, can be exploited.

This paper focuses on scaffolding only. Given a set of contigs and their relationships in terms of distances between them (insert sizes), we propose an optimization-based approach for finding the genome sequence as the longest sequence that is consistent with the given contig and linkage information. Specifically, we define a graph, which we call *contig graph*, whose vertices are the contigs and whose edges connect pairs of contigs that either overlap, or have a gap of size given by the insert-size information. Edges have weights that encode the corresponding distance information between the contigs and are negative in the case of overlaps and positive in the case of gaps. Vertices have weights equal to the lengths of the contigs they represent. Contigs with repeat factor s are represented as a set of s vertices with the same sets of neighbors. The length of a path in the resulting graph is defined as the sum of the weights of the vertices and edges in it. The scaffolding problem is reduced to finding a longest simple path such that as many as possible mate-pairs distances are satisfied (we call hereafter such path just a *longest path*). Since both conditions cannot generally be simultaneously satisfied, our objective function is a linear combination of them.

Unlike the shortest path problem, the longest path problem is NP-hard [3], which means that no polynomial time solution is likely to exist for it. We solve this problem by reformulating it as a mixed integer linear program (MILP) and developing a method that can solve the resulting optimization problem on genomes of up to 83 contigs and up to 900 binary variables. We analyze the performance of the algorithm on several chloroplast genomes and compare it to other scaffolding algorithms.

Most previous work on scaffolding is heuristics based, e.g., SSPACE [1], GRASS [4], and BESST [13]. Such algorithms may find in some cases good solutions, but their accuracies cannot be guaranteed or predicted. In contrast, our method always finds a longest path in the contig graph. There is no guarantee that the genome sequence corresponds to a longest path, but our experiments show that that is the case in many instances or, if not, there is a very small difference between the two. Exact algorithms for the scaffolding problem are

presented in [15], but the focus of that work is on finding structural properties of the contig graph that will make the optimization problem of polynomial complexity. In [11], integer linear programming is used to model the scaffolding problem, with an objective to maximize the number of links that are satisfied. In contrast, our objective is to maximize the length of the resulting scaffold. Moreover, we aim at producing a single path or cycle, rather than a set of paths and cycles. We believe that by requiring our solution to be a single path we avoid the risk of producing a set of paths for which the objective function is of high value, but which are inconsistent with a single path ordering. While our focus is on accuracy, [11] focuses on efficiency, and indeed their algorithm is faster than ours.

The contributions of this study are as follows:

- Our modeling of the scaffolding problem allows to solve *simultaneously* the set of subtasks specific for this problem like: contigs orientation and ordering, repeats, gap filling and scaffold extension.
- The scaffolding problem is reduced to finding a longest paths in a particular graph. In addition, these paths need to satisfy a set of distances between couples of vertices along these paths. We are not aware of previous approaches on scaffolding based on the longest path problem.
- We formulate the above problem as a mixed integer linear program (MILP) with several interesting properties like: cycles elimination constraints, using binary variables for the edges of the graph only. Vertices are modeled with real variables, but we prove that the integrality of these variables follows from other constraints.
- We tested this model on a set of chloroplast and bacteria genome data and showed that it allows to assemble the complete genome as a single scaffold. None of the publicly available scaffolding tools that we have tested targets single scaffolds (this is corroborated by the obtained numerical results).
- Our numerical experiments indicate that the relaxation of the mixed integer model is tight and produces upper bounds of excellent quality. This suggests a promising direction of research towards the scalability of our approach.

In the next Section 2 we describe our graph model and the formulation of the optimization problem and in Section 3 we present experimental results and comparison with other algorithms.

2 Modeling the scaffolding problem

2.1 Graph Modeling

We model the problem of scaffolding as path finding in a directed graph $G = (V, E)$ that we call a contig graph, where both vertices V and edges E are weighted. The set of vertices V is generated based on the set C of the contigs according the following rules: the contig i is represented by at least two vertices v_i and v_i' (forward/inverse orientation respectively). If the contig i is repeated k_i times, it generates $2k_i$ vertices. Denote $N = \sum_{i \in C} k_i$, therefore $|V| = 2N$.

The edges are generated following given patterns—a set of known overlaps/distances between the contigs. Any edge is given in the graph G in its forward/inverse orientation. We denote by e_{ij} the edge joining vertices v_i and v_j and the inverse of edge e_{ij} is $e_{j'i'}$. For any i, the weight w_i on a vertex v_i corresponds to the length of the contig i, while the weight l_{ij} on the edge e_{ij} corresponds to the value of the overlap/distance between contigs i and j. The problem then is to find a path in the graph G such that the total length (the sum over the traversed vertices and edges) is maximized, while a set of additional constraints are also satisfied:

- For any i, either vertex v_i or v_i' is visited (participates in the path).
- The orientations of the nodes does not contradict the constraints imposed by the mate-pairs. This is at least partially enforced by the construction of the graph.

To any edge $e \in E$ we associate a variable x_e. Its value is set to 1, if the corresponding edge participates in the assembled genome sequence (the associated path in our case), otherwise its value is set to 0. There are two kinds of edges: edges corresponding to overlaps between contigs, denote them by O (from overlaps), and edges associated with mate-pairs relationships, denote them by L (from links). We therefore have $E = L \cup O$. Let l_e be the length of the edge e. We have $l_e < 0 \ \ \forall e \in O$ and $l_e > 0 \ \forall e \in L$. Let w_v be the length of the contig corresponding to vertex v and denote $W = \sum_{v \in V} w_v$.

Let $A^+(v) \subset E$ (resp. $A^-(v) \subset E$) denote the subset of arcs in E leaving (resp. entering) node v.

2.2 Integer Linear Programming Formulation

We associate a binary variable for any edge of the graph, i.e.

$$\forall e \in O : x_e \in \{0,1\} \text{ and } \forall e \in L : g_e \in \{0,1\}. \tag{1}$$

Furthermore, to any vertex $v \in V$ we associate three variables, i_v, s_v, and t_v, which stand respectively for intermediate, source, and target for some path, and satisfy

$$0 \leq i_v \leq 1, \quad 0 \leq s_v \leq 1, \ 0 \leq t_v \leq 1. \tag{2}$$

All three variables are set to zero when the associated vertex v participates in none of the paths. Otherwise, it could be either a source/initial (noted by $s_v = 1, t_v = 0, i_v = 0$), or a target/final ($t_v = 1, s_v = 0, i_v = 0$), or an intermediate vertex, in which case the equalities $i_v = 1, t_v = 0$ and $s_v = 0$ hold. Moreover, each vertex (or its inverse) can be visited at most once, i.e.

$$\forall (v, v') : i_v + i_{v'} + s_v + s_{v'} + t_v + t_{v'} \leq 1. \tag{3}$$

The four possibles states for a vertex v (to belong to none of the paths, or otherwise, to be a source, a target, or an intermediate vertex in some path) are provided by the following two constraints

$$s_v + i_v = \sum_{e \in A^+(v)} x_e \leq 1 \tag{4}$$

and

$$t_v + i_v = \sum_{e \in A^-(v)} x_e \leq 1. \tag{5}$$

Finally, only one sequence (a single path) is searched for

$$\sum_{v \in V} s_v = 1 \text{ and } \sum_{v \in V} t_v = 1. \tag{6}$$

Theorem 1. *The real variables $i_v, s_v, t_v, \forall v \in V$ take binary values.*

Proof. Let us analyse the four possibles cases deduced from (4) and (5). Denote $S^+ = \sum_{e \in A^+(v)} x_e$ and $S^- = \sum_{e \in A^-(v)} x_e$.

i) $S^+ = 0$ and $S^- = 0$.
In this case it follows from (2) that $s_v = i_v = t_v = 0$.

ii) $S^+ = 1$ and $S^- = 1$.
The above is equivalent to $s_v + i_v = 1$ and $t_v + i_v = 1$, which leads to $s_v = t_v$. Moreover, from (3) we have $s_v = t_v = 0$ and $i_v = 1$.

iii) $S^+ = 1$ and $S^- = 0$.
The above is equivalent to $s_v + i_v = 1$ and $t_v + i_v = 0$, which leads to $s_v - t_v = 1$. Hence, from (2), we have $t_v = 0$ and therefore $s_v = 1$ and $i_v = 0$.

iv) $S^+ = 0$ and $S^- = 1$.
This case is analogous to iii) and we have $s_v = i_v = 0$ and $t_v = 1$. □

We introduce a continuous variable $f_e \in R^+$ to express the quantity of the flow circulating along the arc $e \in E$

$$\forall e \in E : 0 \leq f_e \leq W. \tag{7}$$

For $e \in O$, the value of x_e is set to 1, if the arc e carries some flow and 0, otherwise. In other words, no flow can use the arc e when $x_e = 0$ as ensured by constraint

$$f_e \leq W x_e \quad \forall e \in O. \tag{8}$$

We use the flows f_e in the following constraints

$$\forall v \in V : \sum_{e \in A^-(v)} f_e - \sum_{e \in A^+(v)} f_e \geq i_v(w_v + \sum_{e \in A^-(v)} l_e x_e) - W s_v \tag{9}$$

$$Ws_v \leq \sum_{e \in A^+(v)} f_e. \tag{10}$$

The purpose of the last two constraints is manifold. When a vertex v is a source ($s_v = 1$), (9) and (10) generate and output from it an initial flow of sufficiently big value (W is enough in our case). When v is an intermediate vertex ($i_v = 1$), constraint (9) forces the flow to decrease by at least $l_{(u,v)} + w_v$ units when it moves form vertex u to its adjacent vertex v. The value of the flow thus is decreasing and this feature forbids cycles in the context of (4) and (5). When v is a final vertex, (9) is simply a valid inequality since the initial flow value is big enough.

We furthermore observe that because of (5), the constraint (9) can be written as follows

$$\forall v \in V : \sum_{e \in A^-(v)} f_e - \sum_{e \in A^+(v)} f_e \geq i_v w_v + \sum_{e \in A^-(v)} l_e x_e - W s_v. \tag{11}$$

The constraint (11) is linear and we keep it in our model instead of (9).

Furthermore, binary variables g_e are associated with links. For $(s,t) \in L$, the value of $g_{(s,t)}$ is set to 1 only if both vertices s and t belong to the selected path and the length of the considered path between them is in the given interval $[\underline{L}_{(s,t)}, \overline{L}_{(s,t)}]$. Constraints related to links are :

$$g_{(s,t)} \leq s_s + i_s + t_s \text{ and } g_{(s,t)} \leq s_t + i_t + t_t \tag{12}$$

as well as

$$\forall (s,t) \in L : \sum_{e \in A^+(s)} f_e - \sum_{e \in A^-(t)} f_e \geq \underline{L}_{(s,t)} g_{(s,t)} - M(1 - g_{(s,t)}) \tag{13}$$

$$\forall (s,t) \in L : \sum_{e \in A^+(s)} f_e - \sum_{e \in A^-(t)} f_e \leq \overline{L}_{(s,t)} g_{(s,t)}) + M(1 - g_{(s,t)}), \tag{14}$$

where M is some big constant.

We search for a long path in the graph and such that as much as possible mate-paired distances are satisfied. The objective hence is :

$$\max(\sum_{e \in O} x_e l_e + \sum_{v \in V} w_v(i_v + s_v + t_v) + p \sum_{e \in L} g_e) \tag{15}$$

where p is a parameter to be chosen as appropriate (currently $p = 1$).

Remark: Note that omitting constraints (6) from the above model generates a set of paths that cover "optimally" the contig graph, rather than a single path. We have tested this variant of the model, but the obtained solutions were too much fragmented and of worse quality compared to the single-path model.

3 Computational results

Here we present the results obtained on a small set of chloroplast and bacteria genomes given in Table 1. Synthetic sequencing reads have been generated for these instances applying ART simulator [7]. For the read assembly step required to produce contigs we applied minia [2] with parameter `unitig` instead of `contig` (a unitig is a special kind of a high-confidence contig). Based on these unitigs, the overlaps between them, as well as the mate-pair distances, we generated a graph as explained in Section 2.1. The graph generated for the Atropa belladonna genome is given in Figure 1.

We compared our results with the results obtained by three of the most recent scaffolding tools– SSPACE [1], BESST [13], and Scaffmatch [9]. In order to evaluate the quality of the produced scaffolds, we applied the QUAST tool [5]. The results are shown on Table 2. We observe that our tool GST (from Genscale Scaffolding Tool) is the only one that consistently assembles the complete genome (an unique scaffold in #scaffolds column) with more than 98% (and in four cases at least 99.9%) correctly predicted genome fraction and zero misassembles.

Our results were obtained on a standard laptop (Intel(R) Core(TM) i3-4000M with 2 cores at 2.4 GHz and 8 GB of RAM), and using Gurobi [6] for solving the MILP models.

Datasets	Total length	#unitigs	#nodes	#edges	#mate-pairs
Acinetobacter	3 598 621	165	676	8344	4430
Wolbachia	1 080 084	100	452	7552	2972
Aethionema Cordifolium	154 167	83	166	898	600
Atropa belladonna	156 687	18	36	114	46
Angiopteris Evecta	153 901	16	32	144	74
Acorus Calamus	153 821	15	30	134	26

Table 1: Scaffolding datasets.

Our next computational experiments focussed on comparing various relaxations and other related formulations for the MILP model described in the previous section. Let us denote in the sequel by BR (Basic Real) the model defined by the linear constraints (1), (2), (3), (4), (5), (6), (7), (8), (11), (10), (12), (13), (14) and objective function (15). Let BB (from Basic Binary) denote the same model except that constraint (2) is substituted by its binary variant, i.e.

$$\forall v \in V : i_v \in \{0,1\} \text{ and } s_v \in \{0,1\} \text{ and } t_v \in \{0,1\}. \tag{16}$$

According to Theorem 1, the models BB and BR are equivalent in quality of the results. We are interested here in their running time behavior. We study

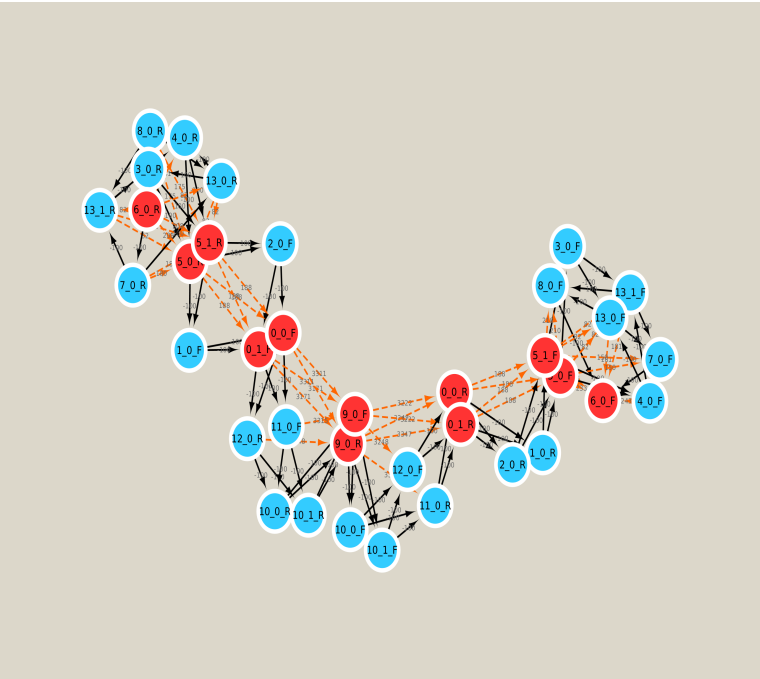


Fig. 1: The contig graph generated for the Atropa belladonna genome. Red/blue vertices correspond respectively to big/small contigs.

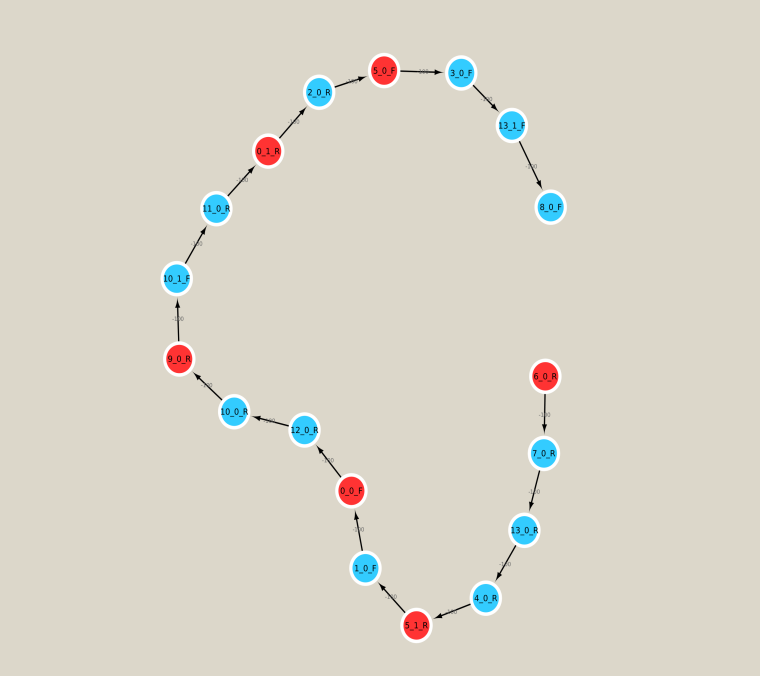


Fig. 2: The scaffold obtained for Atropa belladonna's genome shown on Figure 1.

Datasets	Scaffolder	Genome fraction	#scaffolds	# mis-assemblies	N's per 100 kbp
Acinetobacter	GST	98.536%	1	0	0
	SSPACE	98.563%	20	0	155.01
	BESST	98.539%	37	0	266.65
	Scaffmatch	98.675%	9	5	1579.12
Wolbachia	GST	98.943%	1	0	0
	SSPACE	97.700%	9	0	2036.75
	BESST	97.699%	49	0	642.90
	Scaffmatch	97.994%	2	2	3162.81
Aethionema Cordifolium	GST	100%	1	0	0
	SSPACE	95.550%	20	0	13603.00
	BESST	81.318%	30	0	1553.22
	Scaffmatch	82.608%	7	7	36892
Atropa belladonna	GST	99.987%	1	0	0
	SSPACE	83.389%	2	0	155.01
	BESST	83.353%	1	0	14.52
	Scaffmatch	83.516%	1	0	318.93
Angiopteris Evecta	GST	99.968%	1	0	0
	SSPACE	85.100%	4	0	0
	BESST	85.164%	2	0	1438.54
	Scaffmatch	85.684%	1	0	454.23
Acorus Calamus	GST	100%	1	0	0
	SSPACE	83.091%	4	0	126.39
	BESST	83.091%	4	0	127.95
	Scaffmatch	83.271%	1	1	3757.13

Table 2: Performance of different solvers on the scaffolding datasets from Table 1. GST is our tool.

as well the linear programming relaxation of BR (denoted by BR_{LP}) where the binary constraints (1) are substituted by

$$\forall e \in O : 0 \leq x_e \leq 1 \text{ and } \forall e \in L : 0 \leq g_e \leq 1. \tag{17}$$

Furthermore, let us relax in BR model all constraints related to mate-pairs distances (i.e. constraints (12, (13), (14)). We also omit from the objective function the last term that is associated to mate-pairs. Let us call this model LP (Longest Path) since it simply targets to find the longest path in the contig graph. Any solution of this model can be extended to a solution of BR model by a completion $g_e = 0\ \forall e \in L$. Its optimal value yields a lower bound for the main model BR.

The results obtained by each one of these models are presented in Table 3. From these results we first observe that, as expected, the model BR outperforms BB in running time. With respect to the quality of the obtained results, the results with the relaxed models are very encouraging. The upper bounds computed by the linear relaxation BR_{LP} are extremely close to the exact values computed by BR model. Furthermore, the quality of the lower bound found by the longest path approach LP is also very good. Interestingly, we observed for the given instances that this value is close to the genome's size. We presume that the LP model can be used for predicting the length of the genome when it is unknown.

4 Conclusion

We developed and tested algorithms for scaffolding based on a version of the longest path problem and MILP representation. Our algorithms significantly outperform three of the best known scaffolding algorithms with respect to the quality of the scaffolds. Regardless of that, we consider the current results as a work in progress. The biggest challenge is to extend the method to much bigger genomes. We plan to use some additional ideas and careful implementation to increase the scalability of the methods without sacrificing the accuracy of the results.

References

1. Boetzer, M., Henkel, C.V., Jansen, H.J., Butler, D., Pirovano, W.: Scaffolding preassembled contigs using SSPACE. Bioinformatics (Oxford, England) 27(4), 578–579 (Feb 2011)
2. Chikhi, R., Rizk, G.: Space-efficient and exact de Bruijn graph representation based on a Bloom filter. In: WABI. Lecture Notes in Computer Science, vol. 7534, pp. 236–248. Springer (2012)
3. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1990)
4. Gritsenko, A.A., Nijkamp, J.F., Reinders, M.J., Ridder, D.d.: GRASS: a generic algorithm for scaffolding next-generation sequencing assemblies. Bioinformatics 28(11), 1429–1437 (2012), `http://bioinformatics.oxfordjournals.org/content/28/11/1429.abstract`

Datasets	Models	Obj	1^{st} term (length)	2^{nd} term (# satisfied links)	Time
Acinetobacter	BB	N/A	N/A	N/A	15m00.000s*
	BR	3 598 689	3 598 499	190	6m27.440s
	BR_{LP}	3 598 977	3 597 826	1151	0m44.508s
	LP	3 598 518	3 598 518	N/A	1m16.318s
Wolbachia	BB	N/A	N/A	N/A	15m00.000s*
	BR	1 075 949	1 075 856	93	3m13.144s
	BR_{LP}	1 076 053	1 075 624	429	0m25.428s
	LP	1 075 857	1 075 857	N/A	0m18.694s
Atropa belladonna	BB	156 501	156 488	13	0m1.151s
	BR	156 501	156 488	13	0m0.780s
	BR_{LP}	156 507	156 468	39	0m0.720s
	LP	156 488	156 488	N/A	0m0.296s
Angiopteris Evecta	BB	145 542	145 534	8	0m28.728s
	BR	145 542	145 534	8	0m1.084s
	BR_{LP}	145 556	145 501	55	0m0.752s
	LP	145 535	145 535	N/A	0m0.308s
Acorus Calamus	BB	153 976	153 970	6	0m1.343s
	BR	153 976	153 970	6	0m1.060s
	BR_{LP}	153 981	153 959	22	0m0.768s
	LP	153 970	153 970	N/A	0m0.261s
Aethionema Cordifolium	BB	151 563	151 445	118	15m00.000s*
	BR	151 570	151 445	125	15m00.000s*
	BR_{LP}	151 610	151 200	410	0m1.992s
	LP	151 445	151 445	N/A	0m1.548s

Table 3: Performance of the basic MILP model and some of its relaxations and related formulations on the scaffolding datasets from Table 1. The symbol $*$ indicates that the corresponding execution has been stopped by time limit.

5. Gurevich, A., Saveliev, V., Vyahhi, N., Tesler, G.: QUAST: quality assessment tool for genome assemblies. Bioinformatics (Oxford, England) 29(8), 1072–1075 (Apr 2013)
6. Gurobi Optimization, I.: Gurobi optimizer reference manual version 3.0. (2010)
7. Huang, W., Li, L., Myers, J.R., Marth, G.T.: Art: a next-generation sequencing read simulator. Bioinformatics 28(4), 593–594 (2012), http://bioinformatics.oxfordjournals.org/content/28/4/593.abstract
8. Kolodner, R., Tewari, K.K.: Inverted repeats in chloroplast DNA from higher plants. Proceedings of the National Academy of Sciences of the United States of America 76(1), 41–45 (01 1979), http://www.ncbi.nlm.nih.gov/pmc/articles/PMC382872/
9. Mandric, I., Zelikovsky, A.: Scaffmatch: scaffolding algorithm based on maximum weight matching. Bioinformatics (2015), http://bioinformatics.

oxfordjournals.org/content/early/2015/05/14/bioinformatics.btv211.abstract
10. Medvedev, P., Pham, S., Chaisson, M., Tesler, G., Pevzner, P.: Paired de Bruijn graphs: A novel approach for incorporating mate pair information into genome assemblers. Journal of Computational Biology 18(11), 1625–1634 (11 2011), http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3216098/
11. Nicolas, B., Annie, C., Coletta, R., de Givry, S., Leleux, P., Thomas, S.: An integer linear programming approach for genome scaffolding. In: Workshop on Constraint based Methods for Bioinformatics (2015)
12. Pevzner, P.A., Tang, H., Waterman, M.S.: An Eulerian path approach to DNA fragment assembly. Proceedings of the National Academy of Sciences 98(17), 9748–9753 (2001), http://www.pnas.org/content/98/17/9748.abstract
13. Sahlin, K., Vezzi, F., Nystedt, B., Lundeberg, J., Arvestad, L.: BESST - efficient scaffolding of large fragmented assemblies. BMC Bioinformatics 15, 281 (2014)
14. Weber, J.L., Myers, E.W.: Human whole-genome shotgunsequencing. Genome Research 7(5), 401–409 (1997), http://genome.cshlp.org/content/7/5/401.short
15. Weller, M., Chateau, A., Giroudeau, R.: Exact approaches for scaffolding. BMC bioinformatics 16(Suppl 14), S2 (2015)

A Graph Constraints Formulation for Contigs Scaffolding

Éric Bourreau[1], Annie Chateau[1,2], Clément Dallard[1], Rodolphe Giroudeau[1]

[1] LIRMM - CNRS UMR 5506 - Montpellier, France
[2] IBC - Montpellier, France
{eric.bourreau,annie.chateau,clement.dallard,rodolphe.giroudeau}@lirmm.fr

Abstract. This paper presents a constraint-based approach for genome scaffolding, which is one important step in genome whole sequence production. We model it as an optimization problem on a graph built from a paired-end reads mapping on contigs. We describe our constraint model using a graph variable representation with classical graph constraints. We tested our approach together with several search strategies, on a benchmark of various genomes.

1 Introduction

Last decade was marked by the race to the production of new genomic sequences. Since new technologies of sequencing, known as *High Throughput Sequencing* (*HTS*) or *New Generation Sequencing* (*NGS*), are available, price of genome sequencing has consequently dropped. Technological advances in sequencing, but also in computer science, allow to conduct studies involving tens of thousands of genomes of the same species or related species. The projects "1000 genomes" bloom, and necessit a phenomenal processing power. However *HTS* is mostly based on a technology which splinters the genomic sequence, resulting in a large amount of paired-end reads even for quite small genomes. Most sequencing projects are experiencing bottlenecks in the production of complete sequences from the sequencing libraries, and produce genomes often in draft form. Hence, assembling and scaffolding steps have to be as optimized as possible to obtain a satisfying solution in reasonable time. One of the crucial steps involved in this process is genome *scaffolding*. Once sequencing and assembly of `DNA` molecules have been performed, we end up with a set of genomic sequences of various lengths, called *contigs*, representing pieces of genome. The main goal of scaffolding process is to find an order and an orientation on these contigs such that resulting collections of oriented contigs map as good as possible to the reference genome. Such collections are named *scaffolds* and would ideally represent the genome chromosomes, which could be either linear or circular.

The scaffolding process has been modeled as various combinatorial problems which are unfortunately computationally hard [1,2]. This observation naturally encourages to try different ways to solve the problem, from heuristic, approximation or exact resolution point of view. Most of current scaffolding solvers use heuristic methods to solve this problem. These solvers, unfortunately, do not

propose any confidence on the optimality of the solution they return. Some of them, like *Bambus* [3], *SSPACE* [4] and *SSAKE* [5], directly solve the graph problem using greedy algorithms. Their first obvious interest is their time complexity, since the corresponding algorithms are strictly polynomial. However, the solution may be very faulty since the graph is sometimes simplified and because it only guarantees a local maximum. Other solvers like *Bambus2* [6] uses structures recognition and $\mathcal{NP}$-hard problem's approximations to generate scaffolds. *Opera* [7] uses a graph contraction method before solving the scaffolding on the smaller contracted graph. However, the contraction step may alter the original graph such that the optimal solution on the contracted graph is not optimal on the original one. *SCARPA* [8] combines *Fixed Parameter Tractable* algorithm (to remove odd cycles on the original graph) and mixed integer programming (to join contigs in scaffolds). Once again the yielded solution is not necessarily the optimal solution, because of the odd cycle deletion. *GRASS* [9] and *MIP Scaffolder* [10] use mixed integer programs to solve the scaffolding problem, but always on a simplified graph and then can not be considered as exact methods either.
A previous work using an incremental strategy and Integer Linear Programming (ILP) was proposed in [11]. After several attempts to model the scaffolding problem with CSP, the authors finally chose a simple ILP formulation instead, in order to achieve scalability. However, this formulation was not totally satisfying, since it could not prevent from small circular chromosomes in the solution. Thus, it has to be cured with an iterative treatment to forbid those cycles. As one can observe, there is no solver offering exact resolution for the scaffolding problem, possibly resulting in different solutions from different solvers working on a same graph. In the present work, we choose the CSP approach, to attack this problem. Using a recent library, dedicated to graphs, namely Choco-graph[3] [12], we formulate the contig scaffolding problem in a simple manner, given in Section 3. We discuss how search strategies could have an effect on the efficiency of the method, and run some experiments on a dataset of small instances, in Section 4.

2 Notation and description of the problem

In what follows, we consider $G = (V, E)$ an undirected graph with an even number $2n$ of vertices and without self loops. We suppose that there exists a perfect matching in G, denoted by M^*. Let $w : E \to \mathbb{N}$ be a weight function on the edges. In the bioinformatic context, edges in M^* represent contigs, and the other edges figure the ways to link the contigs together, their weight representing the support of each of these hypotheses (e.g. the number of pairs of reads matching on both contigs). Figure 1 shows an example of a scaffold graph.

[3] https://github.com/chocoteam/choco-graph

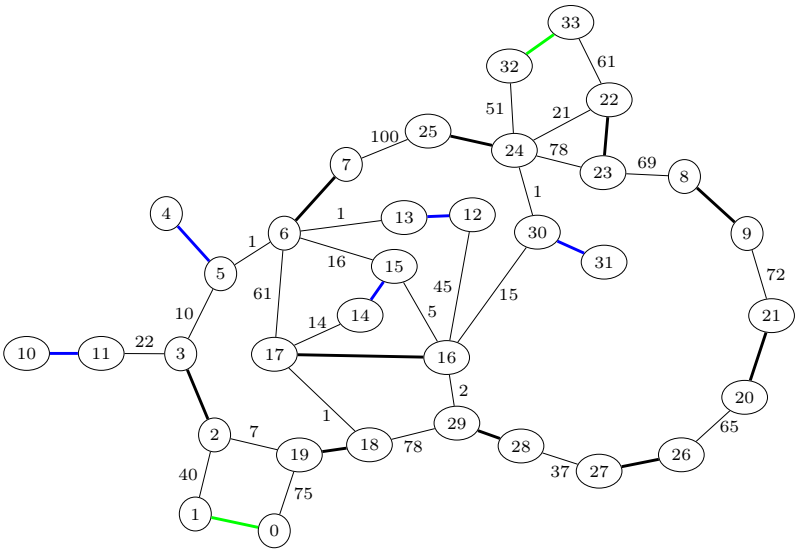


Fig. 1. A scaffold graph with 17 contigs (bold edges) and 26 (weighted) links between them, corresponding to the `ebola` data set (see Section 4). Contig-edges in green and blue correspond to contig of size small enough to be inserted between two other contigs, leading to possible misplacements in the final scaffolding. Green contigs are still well placed in the solution, but blue one are ambiguous.

In order to model the genomic structure by fixed numbers of linear (paths) and circular (cycles) chromosomes, the class of considered problems are parameterized by two integers, respectively denoted by σ_{p} and σ_{c}. These parameters correspond to what is *desired* as genomic structure, however it is not always possible to exactly obtain this structure. Thus we consider a relaxed version of the problem, called Scaffolding, together with the original one, denoted as Strict Scaffolding.

We may use the notation $\mathrm{Gr}(\cdot)$ to denote the induced graph of an edge set. For instance, let $G = (V, E)$ be a graph, then $\mathrm{Gr}(E) = G$.

In the following, we call *alternating cycle* (resp. *alternating path*) in G, relatively to a perfect matching M^* of G, a cycle (resp. a path) with edges alternatively belonging to M^* or not (resp. and where extremal edges belong to M^*). Notice that an alternating-cycle (resp. alternating-path) has necessarily an even number of vertices, at least four (resp. two). The class of Scaffolding problems are defined as follows:

Scaffolding (SCA)
Input: $G = (V, E)$, $\omega : E \to \mathbb{N}$, perfect matching M^* in G, $\sigma_{\mathrm{p}}, \sigma_{\mathrm{c}}, k \in \mathbb{N}$
Question: Is there an $S \subseteq E \backslash M^*$ such that $\mathrm{Gr}(S \cup M^*)$ is a collection of $\geq \sigma_{\mathrm{p}}$ alternating paths and $\leq \sigma_{\mathrm{c}}$ alternating cycles and $\omega(S) \geq k$?

The variant of the problem that asks for *exactly* σ_p paths and *exactly* σ_c cycles is called Strict Scaffolding (SSCA). When we want to precise particular values for σ_p and σ_c, we refer to the problem as $(\sigma_p, \sigma_c)-$Scaffolding. We refer to the optimization variants of Scaffolding that ask to minimize or maximize $\omega(S)$ as Min Scaffolding and Max Scaffolding, respectively. In what follows, we mainly focus on both problems Max Scaffolding and Max Strict Scaffolding, which correspond to the bioinformatic problem.

Problems Strict Scaffolding and Scaffolding received much attention in the framework of complexity and approximation [2,13]. In these articles, the authors showed the hardness of Scaffolding even in presence of restricted parameters or graph classes (cycle length, bipartite planar graph, tree, ...). Some lowers bounds according to complexity hypothesis ($\mathcal{P} \neq \mathcal{NP}$, $\mathcal{ETH}$) are proposed using the Gap-reduction, and reduction preserving lower bound in the field of exact exponential time algorithms. On positive side, some upper bounds with efficient polynomial-time approximation algorithms are designed. Theoretical aspects of Scaffolding are completed by a parameterized complexity approach in [14] and [13]. In such context, the authors proposed some negative results about the quest of a polynomial kernel, or a $\mathcal{FPT}$-algorithm.

3 Model and algorithms

In this section, we propose to solve, sometimes to optimality, the Strict Scaffolding problem with Constraint Programming. First, we remind some definitions, especially on not so classical graph variables. Then we describe variables and constraints used to model Strict Scaffolding. Finally we describe search strategies used to solve the optimization problem.

a - Constraint Programming Definitions

Definition 1. *A **domain** of a variable x, denoted $D(x)$, is a bounded set of values which can be affected to x. We note $\underline{x}$ (resp. $\overline{x}$) the lower bound (resp. upper bound) of domain $D(x)$.*

Definition 2. *A **constraint** $C_i \in \mathcal{C}$ on the subset of m variables $\mathcal{X}(C_i) = \{x_{i_1}, x_{i_2}, \ldots, x_{i_m}\}$ is a subset of $D(x_{i_1}) \times D(x_{i_2}) \times \ldots \times D(x_{i_m})$. It determines the m-tuples of values allowed to be assigned to variables $x_{i_1}, x_{i_2}, \ldots, x_{i_m}$.*

Definition 3. *A **CSP** is a triplet ($\mathcal{X}$,$\mathcal{D}$,$\mathcal{C}$), where $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ is a set of variables, $\mathcal{D} = \{D(x_1), D(x_2), \ldots, D(x_n)\}$ is a set of domains describing the different values which could be assigned to the variables in $\mathcal{X}$ and $\mathcal{C}$ is a set of constraints between the variables in $\mathcal{X}$.*

Definition 4. *A **solution** of a CSP is a set of assignments of values to variables, $\{(x_1, a_1), (x_2, a_2), \ldots, (x_n, a_n)\}$, with $\forall i \in [1, n], a_i \in D(x_i)$, satisfying all constraints in $\mathcal{C}$.*

Let us remark these general definitions related to CSP do not fix the possible types of variables yet. Classically, we manipulate integer values in integer domains for integer domain variables, but more recently, set variables were introduced and even graph variables [15,16], yielding easier modeling. We exploit here those improvements to expressivity.

Concerning a set variable x, there exists a compact representation of the domain $D(x)$, where we specify two sets of elements [17]: the set of elements that *must* belong to the set assigned to x (which we still call the lower bound $\underline{x}$) and the set of elements that *may* belong to this set (the upper bound $\overline{x}$). The domain itself can be represented as a lattice structure corresponding to the partial order defined by sets inclusion. When defined in Choco, a set variables are encoded with two classical bounds: the union of the all set of possible values called the *envelope* and the intersection of all set of possible values called the *kernel*.

Generalizing this point of view, a graph can be seen as two sets V and E with an inherent constraint specifying that $E \subseteq V \times V$. The domain $D(G)$ of a graph variables G is specified by two graphs: a lower bound graph $\underline{G}$ and an upper bound graph $\overline{G}$, such that the domain is the set of all subgraphs of the upper bound which are supergraphs of the lower bound. For a better understanding of graph variables, we refer to Section "3.10. Graph based Constraints" of the Global Constraint review [18],

b - Constraint Model

Variables As previously described, we introduce an undirected graph variable $Gv(Vv, Ev)$ (Gv for Graph variable) whose value will represent the underlying solution. The variable $Gv(Vv, Ev)$ is derived from the graph $G(V, E)$ previously defined in Section 2, with allowed self loops on vertices. Although in the original formulation of Strict Scaffolding problem, there are no self loops allowed, here they will help us to differentiate cycles, which do not contain any, and paths with a final loop on each extremity. A self loop incident to a vertex u counts for one in its degree. All vertices are mandatory and simply belong to $\underline{\text{Vv}}$. Edges in M^* are also mandatory and belong to $\underline{\text{Ev}}$. We will look for a solution by adding, or not, remaining edges to the kernel.

For cost optimisation, we decide to manipulate only $|Vv|$ integer variables to handle the positive weights on a solution. Let's remark this is a more compact model than having $|Ev|$ variables to represent null of positive weights on edges. We denote by $wmax$ the maximum weight of an edge that could be met in the graph, and $E = (E_{ij})$ a boolean variable matrix channeling the graph Gv (*i.e.* if edge between i and j is in the kernel, $E_{ij} = 1$; if the edge is out of the envelope, $E_{ij} = 0$; else domain is $\{0, 1\}$) The integer variables are:

- a vector $Weights$ of size $|Vv|$. The variable $Weights[u]$ represents the sum of weights of edges incident to node u in the solution. Its domain $D(Weigths[u])$ is the set $\{0, \ldots, wmax + 1\}$. In order to count each weight only once in the solution, we derive from the weight function w defined in Section 2 an upper triangle weight matrix denoted by w^*.

– a variable $TotalWeight$ to sum up all the $Weights$. We can restrict initial domain of $TotalWeight$ to lower and upper bound computed by any heuristic described in previous sections. Here it belongs to $\{0, \ldots, |Vv| * wmax\}$.

We define the following constraint model, very simply defining the problem:

CSP Model : Scaffold Problem with Graph Var and Graph Constraints

$$connected\_components(Gv, \sigma_{\mathrm{p}} + \sigma_{\mathrm{c}}) \tag{1}$$

$$minDegree(Gv, 2) \tag{2}$$

$$maxDegree(Gv, 2) \tag{3}$$

$$nbLoops(Gv, 2\sigma_{\mathrm{p}}) \tag{4}$$

$$Weights[i] = \sum_{j \in Vv} (w_{ij} E_{ij}) \qquad \forall\, i \in Vv \tag{5}$$

$$TotalWeight = \sum_{i \in Vv} Weights[i] \tag{6}$$

In Equation 1, Gv is constrained to have a specific number of connected components (σ_{p}+σ_{c}). By default choco solver use fast filtering rules, first computing all connected components of G by performing one Depth First Search (using Tarjan algorithm [19]) where time complexity is $\mathcal{O}(|Vv| + |Ev|)$ and check their number according to the parameters. If necessary, better propagation can be performed by looking for articulation points in time complexity $\mathcal{O}(|Vv|.|Ev|)$, or even better by managing dominator [20]. This was not necessary in choco to get good performances.

In Equation 2 and Equation 3, we linearly maintain degree for each node u in Gv by checking size of upper bound and lower bound of variable-set E_u, designing edges incident to vertex u, and if necessary by applying complete filtering, removing (or forcing) associated edges. Moreover, since $M^* \subseteq Ev$ we necessarily construct alternating paths and cycles when adding consistent edges to the kernel.

In Equation 4, number of nodes with self loop is linearly maintained to adjust the parameter with complete filtering when bounds are reached. As explained, a cycle will not have self loops but paths will have it at extremities (only one due to upper triangle matrix). Then, without the need to distinguish paths and cycles, this constraint guarantees the solution to have exactly σ_{p} paths. Since Equation 1 fixes the number of connected components, we also have exactly σ_{c} cycles.

The remaining Equation 5 and Equation 6 are simple scalar constraints.

c - Search Strategies

As search implementation, we use different variables families (branching on edges or directly branching on cost) and we focus on ordering the variables to be assigned. We test :

1. a static lexicographic strategy assignment on edges (E_{ij}), meaning that variables are simply not sorted;
2. a random strategy on edges (E_{ij}) with max value first, which help us to have a median behavior on edges branching;
3. a dynamic variable ordering heuristic, called *dom over wdeg*, applied to weight variables (default strategy in our solver). This strategy consists in ordering the involved variables by increasing size of domain, combined with a weighted sum on the constraints they belong to. This strategy is supposed to well behave for a majority of problems [21];
4. a maximum value strategy on cost: as we have to maximize the $TotalWeight$ variable, we use a standard max domain value strategy first on $Weights$ variables: by propagation, assigning first edges with biggest weights leads to connect edges with maximum numbers of pairs of reads mapping on both contigs;
5. a max-regret strategy: assigning first edge with biggest weight for variable with biggest difference between maximum and previous value in domain (aka regret, if not chosen). This last strategy yields usually good results on such "max $\sum$" optimization problem.

4 Experiments

Scaffold graphs used to run our experiments are coming from two sources. A first dataset, called *real instances*, has been build using the following pipeline:

1. We choose a reference genome, on a public database, for instance on the Nucleotide NCBI database[4]. Table 1 shows the selected genomes used to perform our experiments. They were chosen because of their various origins: a virus, an insect, a plant and a yeast; and their small size: two of them comes from organelles, a mitochondrion and a chloroplast, which have small genomes.
2. We simulate paired-end reads, using a tool like `wgsim` [22]. The chosen parameters are an insert size of 500bp and a read length L of 100bp.
3. We assemble those simulated reads using a *de novo* assembly tool, based on a de Bruijn graph efficient representation. This tool is `minia` [23], and was used with a size $k = 30$ for the k-mers stored in the de Bruijn graph.
4. We map the reads on the contigs, using `bwa` [24]. This mapping tool was chosen according to results obtained in [25], a survey on scaffolding tools.
5. We generate the scaffold graph from the mapping file. Statistics on number of vertices and edges in produced scaffold graphs can be viewed on Table 2.

[4] `http://www.ncbi.nlm.nih.gov/`

Table 1. Dataset of real instances.

Species (Alias)	Size (bp)	Type	Accession number
Zaire ebolavirus (`ebola`)	18959	Complete genome	NC_002549.1
Danaus plexippus (`monarch`)	15314	Mitochondrion	NC_021452.1
Oryza sativa Japonica (`rice`)	134525	Chloroplast	X15901.1
Saccharomyces cerevisiae (`sacchr3`)	316613	Chromosome 3	X59720.2
Saccharomyces cerevisiae (`sacchr12`)	1078177	Chromosome 12	NC_001144.5

Since it is quite complicated to find real instances through fully meeting needed parameters, especially size of the scaffold graphs, and to perform average analysis on classes with only one element, a second dataset of generated scaffold graphs was used to complete the size gap between our real instances: the `rice` scaffold graph counts 84 contigs, but `sacchr3` counts 296, and `sacchr12` counts 889. Instances were generated by the tool `Grimm` [26]. The parameters used to generate this dataset were chosen to be similar to the ones observed on real instances. However we are conscious that they do not exactly meet the reality. A set of thirty instances were generated by pair of parameters $(\#vertices, \#edges)$: these pairs come from real instances parameters, completed by $\{(200, 300), (300, 450), (400, 600), (500, 750)\}$.

We run experiments on a MacBook Pro with Intel i7 2.8Ghz processor and 4 Go RAM. First we evaluate each of the previously described strategies. Then, using the best strategy, we solve the real instances and discuss the convenience of such modeling. Finally, we compare obtained results to previous ILP approach.

5 Results

Testing search strategies.

Figure 2 shows the comparison between the different tested search strategies on a generated instance with 200 vertices. Scores shown on this figure is simply the total weight of the current solution. As expected, lexicographic and random strategies on edges do not perform well. This is due to the fact that scaffold graph have a very peculiar structure. They are sparse, and as one can see on Figure 1, the degree of vertices is quite small. For cost based strategies, surprisingly the standard max value performs very well contrary to max regret. We can explain this by the correlation between weight value (based on the number of bridging reads) and the probability that this will be a good link. Weight value are not randomly distributed but are extracted from partial information (reads) coming from a precise structure (the underlying connected structuration of the chromosome).

Let's remark that default dom/wdeg shows its classical robust good behavior without any knowledge !

In what follows, all experiments were performed using the max value strategy.

Results on real instances.

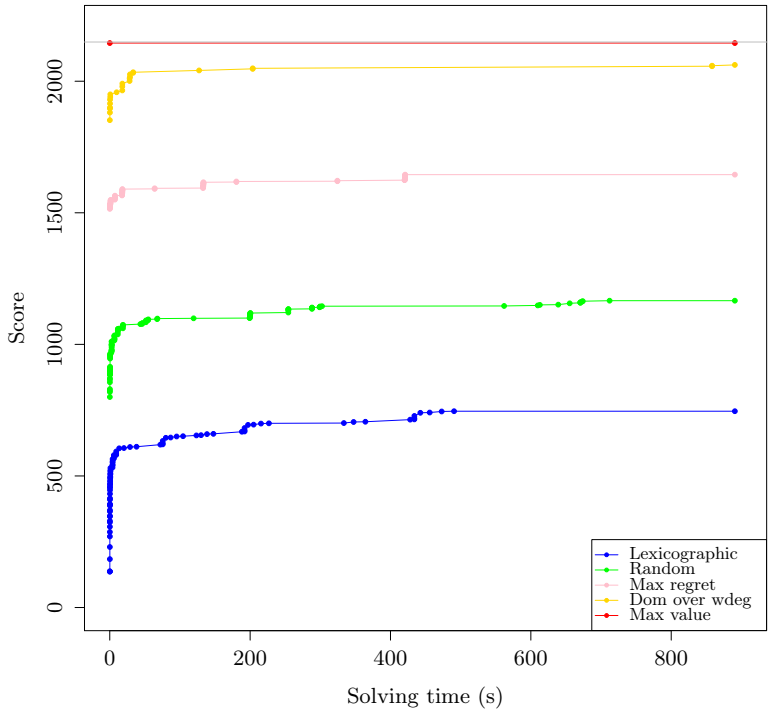


Fig. 2. Comparison of search strategies.

Table 2. Search for optimal solutions on real instances.

Instances	parameters				first sol			optimality		
name	nbContigs	nbEdges	σ_p	σ_c	value	searchNodes	time (s)	value	searchNodes	time (s)
monarch	14	19	5	0	520	16	0.013	520	16	0.013
ebola	17	26	3	1	793	20	0.026	793	20	0.026
ebola	17	26	4	0	707	42	0.040	707	56	0.123
rice	84	139	12	0	4377	106	0.091	4382	147k	177
sacch3	296	527	36	0	14845	406	0.48		> 1M	>3600
sacch12	889	1522	118	1	Out of Memory					

As one can observe in Table 2, we tested our program on several scaffold graphs produced from biological data. It is interesting to note that the first solution has already a high score and is found very quickly (less than one second with very few backtracks). Having a closer look to `ebola`, we noticed that scaffold graphs may contain what we call bad *contig jumps*, meaning there exists a contig a which should appear between two contigs b and c in the genome but such that the optimal solution does not contain $\{a, b, c\}$ in any scaffold. When the contig a is not included in the solution at its place in a path, we say that it is a *forgotten* contig. Necessarily, the length of a is small enough to be inserted in the gap between b and c, the latter being smaller than the insert size of the library. For instance, the ebola graph should ideally contain one unique linear scaffold representing the linear chromosome of ebola. Nevertheless, the optimal solution contains at least four scaffolds because it exists forgotten contigs which can be considered as scaffolds on their owns. On Figure 1, two small contigs appear without any bad consequence, namely 0-1 and 32-33. Indeed, the optimal path includes them. However, other cases are not so easy to solve: for instance the ambiguity between paths 31-30-16 and 13-12-16 leads to choose the latter one, only considering weights. But it is a hidden contig jump, and further examination of inter-contig distances should be included to disambiguate and include both contigs 12-13 and 30-31 in a same path of the solution. Same situation occurs for paths 4-5-3 and 10-11-4. Case of contig jump 14-15 is quite different: the sums of weights on side edges 6-15 and 14-17 is less that the weight of the "by-pass" edge 6-17. In such case, contig 14-15 is not included in the solution. In a nutshell, contigs 30-31, 4-5 and 14-15 are forgotten contigs, explaining the four scaffolds instead of only one expected. Here, we express the necessity to pre-process the graph to treat such contigs and we will consider it as a priority in our future works.

Comparison to previous ILP approach In [11], we proposed an ILP based incremental approach, which was able to heuristically handle large instances. The underlying idea is very simple and consists in optimizing the score, under constraints on degrees, and use an external treatment to forbid detected cycles. We ran this tool on our real dataset. The main difference with actual model is that we considered only cases with paths, and systematically forbade cycles. Thus, the actual model is more expressive, since it allows a given number of cycles.

Table 3. Comparison with previous ILP approach.

Instances	**first sol**		**optimality**		**ILP [11]**	
name	**value**	**time (s)**	**value**	**time (s)**	**value**	**time (s)**
monarch	520	0.013	520	0.013	507	0.00025
ebola	793	0.026	793	0.026	776	0.00028
rice	4377	0.091	4382	177	4320	0.00036
sacch3	14845	0.48		>3600	14616	0.0071

Table 3 shows comparison of solving time and scores between Choco-graph and ILP heuristic. What is noticeable is that, as expected, computation time stays very low when the size of instances increases, and that ILP does not provide an optimal score on these instances. More surprisingly, the score given by first solution is better than ILP score, meaning that efforts made on modelization are somehow rewarded.

6 Conclusion and future works

Classically, modeling real problems brings a gap between customers and modelers. Here, a first gap exist between biologist researchers and bioinformatic researchers to express biological problem into graph modeling problem. Constraint Programming provides a natural declarative way to express constraints on a given problem without worsen the modeling gap between combinatorial problem description and combinatorial solvers resolution, contrary to what happened with previous attempts using SMT, SAT or ILP models. Moreover, the Graph variable development are absolutely convenient to the modeling of combinatorial problems on graphs and we present here a typical example where its usefulness is demonstrated. Although the underlying problem is $\mathcal{NP}$-hard, and there is no hope to quickly solve very large instances in a reasonable time, we could improve solving time by introducing more constraints to help propagation efficiency. For instance, by considering contig lengths and expected lengths of chromosome, if it is known, we could set the minimum length of a cycle or a path. Or, considering that we could not guarantee that all linking information are provided by the original scaffold graph, we could consider only a maximum number of cycles, but allows the number of paths to be itself a variable. Finally, it would be interesting to embed the solver in an interactive tool which allows an expert user to solve and visualize on a given instance.

Acknowledgements

We want to thanks Valentin Pollet for an early version of the Choco graph code.

References

1. Daniel H. Huson, Knut Reinert, and Eugene W. Myers. The greedy path-merging algorithm for contig scaffolding. *Journal of the ACM (JACM)*, 49(5):603–615, 2002.
2. Annie Chateau and Rodolphe Giroudeau. A complexity and approximation framework for the maximization scaffolding problem. *Theor. Comput. Sci.*, 595:92–106, 2015.
3. Mihai Pop, Daniel S Kosack, and Steven L. Salzberg. Hierarchical scaffolding with bambus. *Genome research*, 14(1):149–159, 2004.
4. Marten Boetzer, Christiaan V. Henkel, Hans J. Jansen, Derek Butler, and Walter Pirovano. Scaffolding pre-assembled contigs using sspace. *Bioinformatics*, 27(4):578–579, 2011.

5. René L Warren, Granger G Sutton, Steven JM Jones, and Robert A Holt. Assembling millions of short dna sequences using ssake. *Bioinformatics*, 23(4):500–501, 2007.
6. Sergey Koren, Todd J. Treangen, and Mihai Pop. Bambus 2: scaffolding metagenomes. *Bioinformatics*, 27(21):2964–2971, 2011.
7. Song Gao, Wing-Kin Sung, and Niranjan Nagarajan. Opera: reconstructing optimal genomic scaffolds with high-throughput paired-end sequences. *Journal of Computational Biology*, 18(11):1681–1691, 2011.
8. Nilgun Donmez and Michael Brudno. Scarpa: scaffolding reads with practical algorithms. *Bioinformatics*, 29(4):428–434, 2013.
9. Alexey A Gritsenko, Jurgen F. Nijkamp, Marcel J. T. Reinders, and Dick de Ridder. Grass: a generic algorithm for scaffolding next-generation sequencing assemblies. *Bioinformatics*, 28(11):1429–1437, 2012.
10. Leena Salmela, Veli Mäkinen, Niko Välimäki, Johannes Ylinen, and Esko Ukkonen. Fast scaffolding with small independent mixed integer programs. *Bioinformatics*, 27(23):3259–3265, 2011.
11. Nicolas Briot, Annie Chateau, Rémi Coletta, Simon De Givry, Philippe Leleux, and Thomas Schiex. An Integer Linear Programming Approach for Genome Scaffolding. In *Workshop Constraints in Bioinformatics*, 2014.
12. Jean-Guillaume Fages, Narendra Jussien, and Xavier Lorca a nd Charles Prud'homme. Choco3: an open source java constraint programming library, 2013.
13. Mathias Weller, Annie Chateau, Clément Dallard, and Rodolphe Giroudeau. Scaffolding problems revisited: Complexity, approximation and fixed parameter tractable algorithms, and some specials cases. *submited to Algorithmica*, 2016.
14. Mathias Weller, Annie Chateau, and Rodolphe Giroudeau. Exact approaches for scaffolding. *BMC Bioinformatics*, 16(Suppl 14):S2, 2015.
15. Jean-Charles Régin. Modeling problems in constraint programming. *Tutorial CP*, 4, 2004.
16. Grégoire Dooms, Yves Deville, and Pierre Dupont. CP (graph): Introducing a graph computation domain in constraint programming. In *Principles and Practice of Constraint Programming-CP 2005*, pages 211–225. Springer, 2005.
17. Carmen Gervet. Interval propagation to reason about sets: definition and implementation of a practical language. *Constraints*, pages 191–244, 1997.
18. Jean-Charles Regin. Global constraints: A survey. In Pascal van Hentenryck and Michela Milano, editors, *Hybrid Optimization*, volume 45 of *Springer Optimization and Its Applications*, pages 63–134. Springer New York, 2011.
19. Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.
20. Luis Quesada, Peter Van Roy, Yves Deville, and Raphaël Collet. Using dominators for solving constrained path problems. In *Practical Aspects of Declarative Languages, 8th International Symposium, PADL 2006, Charleston, SC, USA, January 9-10, 2006, Proceedings*, pages 73–87, 2006.
21. Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *In Proceedings of ECAI'04*. Citeseer, 2004.
22. Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor T. Marth, Gonçalo R. Abecasis, and Richard Durbin. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079, 2009.

23. Rayan Chikhi and Guillaume Rizk. Space-efficient and exact de bruijn graph representation based on a bloom filter. In *Algorithms in Bioinformatics - 12th International Workshop, WABI 2012, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 236–248, 2012.
24. Heng Li and Richard Durbin. Fast and accurate long-read alignment with burrows-wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.
25. Martin Hunt, Chris Newbold, Matthew Berriman., and Thomas D. Otto. A comprehensive evaluation of assembly scaffolding tools. *Genome Biology*, 15(3):R42, 2014.
26. Adel Ferdjoukh, Eric Bourreau, Annie Chateau, and Clémentine Nebut. A Model-Driven Approach for the Generation of Relevant and Realistic Test Data. In *SEKE 2016*, page to appear, 2016.