Proceedings of WCB06
Workshop on
Constraint Based Methods for Bioinformatics

Alessandro Dal Palù, Agostino Dovier, and Sebastian Will

September 25, 2006, Nantes (France)

Program Committee

Rolf Backofen Freiburg Univ., Germany
Pedro Barahona Univ. Nova de Lisbona, Portugal
Mats Carlsson SICS, Uppsala Sweden
Alessandro Dal Palù (co-chair) Parma Univ., Italy
Simon De Givry INRA, Toulouse, France
Agostino Dovier (co-chair) Udine Univ., Italy
Francois Fages INRIA Rocquencourt, France
Tony Kusalik Saskatchevan Univ., Canada.
Enrico Pontelli NMSU (USA)
Sebastian Will (co-chair) Freiburg Univ., Germany

External referees

Luca Bortolussi
Elisabetta De Maria
Carla Piazza
Simone Scalabrin
Nicola Vitacolonna
Marco Zantoni

Preface

Modern molecular biology increasingly relies upon computational methods for analyzing and dealing with its vast amounts of biological data. Due to these demands of life sciences, bioinformatics (aka. computational biology) itself is a challenging and fast growing area of research. By promoting and more and more enabling biological and medical research, it is of utmost importance for our understanding of life. Major contributions to this discipline can have thousands of positive effects in medicine, agriculture, or industry. To pick out only a few examples, bioinformatics tackles problems related to:

- Recognition, analysis, and organization of DNA sequences
- Biological systems simulations (for metabolic or regulatory networks)
- Prediction of the spatial conformation of a biological polymer, given its sequence of monomers (in particular for proteins and RNA)

All these problems can be naturally formalized using constraints over finite domains or intervals of reals. Biology is a source of extremely interesting and challenging problems that can be encoded exploiting the application of recent and more general techniques of constraint programming. In this framework, some problems that have been successfully tackled are:

- The fundamental bioinformatics problem of sequence alignment can be solved by recent inference based constraint methods
- Biological systems simulations can be easily designed using concurrent constraint programming, and
- the constrained-based prediction of protein conformations promoted the development of new search strategies, new constraint solvers, and general symmetry breaking.

The workshop WCB06 ties in with the successful CP workshops on Constraints and Bioinformatics/Biocomputing held in CP'97 and CP'98 and it is an immediate follow up to the workshop WCB'05 (ICLP'05).

The main aim of this workshop is to share recent results in this area (new constraint solvers, new prediction programs) and to present new challenging problems that can be addressed using constraint-based methods. Among the papers submitted, seven of them have been judged deserved to be presented. Moreover, the workshop experiences the invited talk by François Fages (INRIA Rocquencourt, France) who describes how temporal logics with constraints can be used to express the biological properties of cellular processes, to model-check them, and to search for reaction rules and parameters satisfying them.

Alessandro Dal Palù
Agostino Dovier
Sebastian Will

Contents

On Using Temporal Logic with Constraints to Express Biological Properties of Cell Processes

François Fages

INRIA Rocquencourt, France
Francois.Fages@inria.fr

Abstract

One promise of systems biology is to model biochemical processes at a sufficiently large scale so that the behavior of a complex system can be predicted under various conditions in *in silico* experiments. The language approach to systems biology aims at designing formal languages for describing biochemical mechanisms, processes and systems at different levels of abstraction, and for providing automated reasoning tools to assist the biologists [1].

The pioneering use of the π-calculus process algebra for modeling cell signalling pathways in [2], has been the source of inspiration of numerous works in the line of process calculi [3] and of their stochastic extensions [4]. The biochemical abstract machine BIOCHAM[1] [5] has been designed as a simplification of the process calculi approach to model biological processes, using a language of reaction rules that is both more natural to the biologists, and well suited to consider different dynamics and use model-checking techniques [6].

In BIOCHAM, the rule-based language is used for modeling biochemical networks at three abstraction levels:

1. the boolean semantics, where one associates to each object (protein, gene, etc.) a boolean variable representing its presence or absence in the system, and the reaction rules are interpreted by a highly non-deterministic *asynchronous transition system* representing competition between reactions;
2. the concentration semantics, where one associates to each object a real number representing its concentration, and the reaction rules are interpreted with their kinetic expressions by a set of non-linear ordinary differential equations (ODE);
3. the stochastic semantics. where one associates to each BIOCHAM object an integer representing the number of molecules in the system, and the rules are interpreted as a continuous time Markov chain.

One striking feature of this multi-level approach is that in the three cases, temporal logic can be used to formalize the biological properties of the system, and verify them by different model-checking techniques [7]. The thesis is that, to a large extend, one can make the following identifications:

[1] BIOCHAM is a free software implemented in Prolog and distributed under the GPL license. It is downloadable on the web at `http://contraintes.inria.fr/BIOCHAM`

biological model = transition system,
biological property = temporal logic formula,
biological validation = model-checking.

At the boolean level, the *Computation Tree Logic* CTL [8] allows us to express *qualitative properties* about the production of some protein (reachability), the checkpoints for its production, the stability or oscillations for its presence, etc. These properties are known from biological experiments in wild-life or mutated organisms. Some of the most used CTL formulae are abbreviated in BIOCHAM as follows:

- `reachable(P)` stands for $EF(P)$;
- `steady(P)` stands for $EG(P)$;
- `stable(P)` stands for $AG(P)$;
- `checkpoint(Q,P)` stands for $!E(!Q\ U\ P)$;
- `oscil(P)` stands for $AG((P \Rightarrow EF\ !P) \wedge (!P \Rightarrow EF\ P))$.

In this setting, such properties can be checked with state-of-the-art symbolic model checkers such as NuSMV using binary decision diagrams [9]. The performances obtained on a large model of the mammalian cell cycle control after Kohn's map [10], involving 800 rules and 500 variables, have been shown to be of the order of a few tenths of seconds to compile the model, and check simple CTL formulae [11].

At the concentration level, we use a first-order fragment of Linear Time Logic (LTL) with *arithmetic constraints* containing equality, inequality and arithmetic operators ranging over the real values of concentrations and of their derivatives. For instance `F([A]>10)` expresses that the concentration of `A` eventually gets above the threshold value 10. `G([A]+[B]<[C])` expresses that the concentration of C is always greater than the sum of the concentrations of A and B. Oscillation properties, abbreviated as `oscil(M,K)`, are defined as a change of sign of the derivative of M at least K times in the time horizon:
`F((d[M]/dt > 0) & F((d[M]/dt < 0) & F((d[M]/dt > 0)...)))`. The abbreviated formula `oscil(M,K,V)` adds the constraint that the maximum concentration of M must be above the threshold V in at least K oscillations.

Under the hypothesis that the initial state is completely defined, numerical integration methods (such as Runge-Kutta or Rosenbrock methods) provide a discrete simulation trace. This trace constitutes a linear Kripke structure in which LTL formulae with constraints can be interpreted and model-checked [7]. Since constraints refer not only to concentrations, but also to their derivatives, we consider traces of the form $(< t_0, x_0, dx_0/dt >, < t_1, x_1, dx_1/dt >, ...)$ where at each time point, t_i, the trace associates the concentration values of the x_i's and the values of their derivatives dx_i/dt.

Beyond making simulations, and checking properties of the models, the temporal properties can also be turned into specifications and temporal logic constraints for automatically searching and learning modifications or refinements of the model when incorporating new biological knowledge. This is implemented in BIOCHAM by a combination of model-checking, search and machine learning techniques in the three abstraction levels.

For instance, in a simple continuous model of the cell cycle after Tyson [12], the search of parameter values for kinetic parameters k_3 and k_4, so that the concentration

of the cyclin `Cdc2-Cyclin p1` oscillates three times in the time horizon 150, can be formalized as follows :

```
biocham: learn_parameters([k3, k4], [(0, 200), (0, 200)], 20,
                          oscil(Cdc2-Cyclin~{p1},3),150).
First values found that make oscil(Cdc2-Cyclin~{p1},3) true:
parameter(k3,).
parameter(k4,).
```

The system finds the parameter values $k_3 = 10$ and $k_4 = 70$ satisfying the specification. However, the corresponding curve depicted in Fig. 1 exhibits damped oscillations. The addition of the constraint of reaching a concentration value greater than 0.1 for this complex produces the values $k_3 = 10$, $k_4 = 120$ and the curve depicted in Fig. 2. The specification can be further refined by imposing a constraint of period equal to 35 time units, `period(Cdc2-Cyclin~{p1},35)`. This produces the curve depicted in Fig. 3 which is close to the original model.

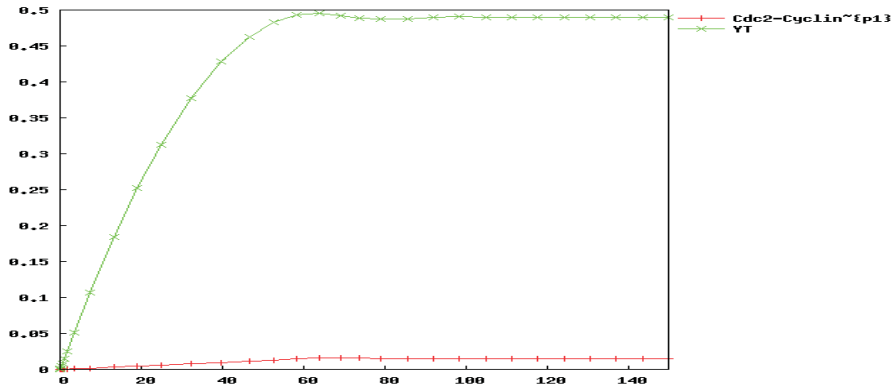


Fig. 1. Cyclin concentration curve for the parameter values $k_3 = 10$ and $k_4 = 70$ found in response to the query `oscil(Cdc2-Cyclin~{p1},3)`.

These first results implemented in BIOCHAM are quite encouraging and motivate us to go further in the direction of the formal specification of biological systems and in the improvement of the search algorithms. A coupled model of the cell cycle and the circadian cycle is under development along these lines in BIOCHAM with applications to cancer chronotherapies.

Acknowledgements. The BIOCHAM project is a joint work with Nathalie Chabrier-Rivier, Sylvain Soliman and Laurence Calzone, with contributions from Sakina Ayata, Loïc Fosse, Lucie Gentils, Shrivaths Rajagopalan and Nathalie Sznajder. In addition, support from the EU STREP project April-II and the EU Network of Excellence REWERSE are warmly acknowledged.

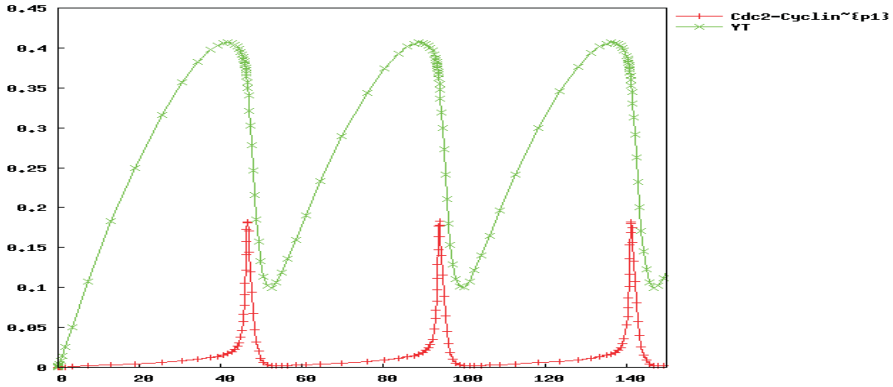


Fig. 2. Concentration curve for parameter values $k_3 = 10$ and $k_4 = 120$ found in response to the query `oscil(Cdc2-Cyclin~{p1},3,0.1)`.

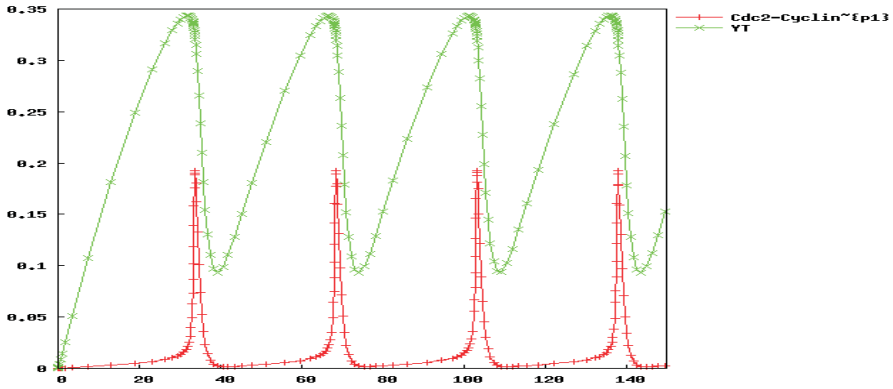


Fig. 3. Concentration curve for $k_3 = 10$ et $k_4 = 190$ obtained in response to the query `period(Cdc2-Cyclin~{p1},35)`.

References

1. Fages, F.: From syntax to semantics in systems biology - towards automated reasoning tools. In: Converging Sciences. Volume 3939 of Lecture Notes in Bioinformatics., Trento, Italy, Springer-Verlag (2004)
2. Regev, A., Silverman, W., Shapiro, E.Y.: Representation and simulation of biochemical processes using the pi-calculus process algebra. In: Proceedings of the sixth Pacific Symposium of Biocomputing. (2001) 459–470
3. Cardelli, L.: Brane calculi - interactions of biological membranes. In Danos, V., Schächter, V., eds.: CMSB'04: Proceedings of the second international workshop on Computational Methods in Systems Biology. Volume 3082 of Lecture Notes in BioInformatics., Springer-Verlag (2004) 257–280
4. Phillips, A., Cardelli, L.: A correct abstract machine for the stochastic pi-calculus. Transactions on Computational Systems Biology (to appear) Special issue of BioConcur 2004.
5. Fages, F., Soliman, S., Chabrier-Rivier, N.: Modelling and querying interaction networks in the biochemical abstract machine BIOCHAM. Journal of Biological Physics and Chemistry **4** (2004) 64–73
6. Chabrier, N., Fages, F.: Symbolic model cheking of biochemical networks. In Priami, C., ed.: CMSB'03: Proceedings of the first workshop on Computational Methods in Systems Biology. Volume 2602 of Lecture Notes in Computer Science., Rovereto, Italy, Springer-Verlag (2003) 149–162
7. Calzone, L., Chabrier-Rivier, N., Fages, F., Soliman, S.: Machine learning biochemical networks from temporal logic properties. Transactions on Computational Systems Biology (2006) CMSB'05 Special Issue (to appear).
8. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
9. Cimatti, A., Clarke, E., Enrico Giunchiglia, F.G., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: Nusmv 2: An opensource tool for symbolic model checking. In: Proceedings of the International Conference on Computer-Aided Verification, CAV'02, Copenhagen, Danmark (2002)
10. Kohn, K.W.: Molecular interaction map of the mammalian cell cycle control and DNA repair systems. Molecular Biology of the Cell **10** (1999) 2703–2734
11. Chabrier-Rivier, N., Chiaverini, M., Danos, V., Fages, F., Schächter, V.: Modeling and querying biochemical interaction networks. Theoretical Computer Science **325** (2004) 25–44
12. Tyson, J.J.: Modeling the cell division cycle: cdc2 and cyclin interactions. Proceedings of the National Academy of Sciences **88** (1991) 7328–7332

Modeling Biological Systems in Stochastic Concurrent Constraint Programming*

Luca Bortolussi and Alberto Policriti

Dept. of Mathematics and Informatics
University of Udine, Udine, Italy
bortolussi|policriti@dimi.uniud.it

Abstract. We present an application of stochastic Concurrent Constraint Programming (sCCP) for modeling biological systems. We provide a library of sCCP processes that can be used to describe straightforwardly biological networks. In the meanwhile, we show that sCCP proves to be a general and extensible framework, allowing to describe a wide class of dynamical behaviours and kinetic laws.

1 Introduction

Computational Systems Biology is a extremely fertile field, where many different modeling techniques are used [7] in order to capture the intrinsic dynamics of biological systems. These techniques are very different both in spirit and in the mathematics they use. Some of them are based on the well known instrument of *Differential Equations*, mostly ordinary, and therefore they represent phenomena as *continuous and deterministic*, cf. [8] for a survey. On the other side we find *stochastic and discrete* models, that are usually simulated with *Gillespie's algorithm* [14], tailored for simulating (exactly) chemical reactions. In the middle, we find hybrid approaches like the *Chemical Langevin Equation* [11], a stochastic differential equation that bridges partially these two opposite formalisms.

In the last few years a compositional modeling approach based on *stochastic process algebras* (SPA) emerged [22], based on the inspiring parallel between molecules and reactions on one side and processes and communications on the other side. Stochastic process algebras, like stochastic π-calculus [20], have a simple and powerful syntax and a stochastic semantics expressed in terms of Continuous Time Markov Chains [19], that can be simulated with an algorithm equivalent to Gillespie's one. Since their introduction, SPA have been used to model, within the same framework, biological systems described at different level of abstractions, like biochemical reactions [21] and genetic regulatory networks [1].

Stochastic modeling of biological systems works by associating a rate to each active reaction (or, in general, interaction); rates are real numbers representing the frequency or propensity of interactions. All active reactions then undergo a (stochastic) race condition, and the fastest one is executed. Physical justification of this approach can be

* This work has been partially supported by PRIN 2005 project 2005015491 and by FIRB 2003 project RBNE03B8KK.

found in [13]. These rates encode all the quantitative information of the system, and simulations produce discrete temporal traces with variable delay between events.

In this work we show how stochastic Concurrent Constraint Programming [2] (sCCP), another SPA recently developed, can be used for modeling biological systems. sCCP is based on Concurrent Constraint Programming [23] (CCP), a process algebra where agents interact by posting constraints on the variables of the system in the constraint store, cf. Section 2.

In order to underline the rationale behind the usage of sCCP, we take an high level point of view, providing a general framework connecting elements of biological systems with elements of the process algebra. Subsequently, we show how this general framework gets instantiated when focused on particular classes of biological system, like networks of biochemical reactions and gene regulatory networks.

In our opinion, the advantages of using sCCP are twofold: the presence of both quantitative information and computational capabilities at the level of the constraint systems and the presence of functional rates. This second feature, in particular, allows to encode in the system different forms of dynamical behaviours, in a very flexible way. Quantitative information, on the other hand, allows a more compact representation of models, as part of the details can be described in relations at the level of the store.

The paper is organized as follows: in Section 2 we review briefly sCCP, in Section 3 we describe a high level mapping between biological systems and sCCP, then we instantiate the framework for biochemical reactions (Section 3.1) and gene regulatory networks (Section 3.2). Finally, in Section 4, we draw final conclusions and suggest further directions of investigation.

2 Stochastic Concurrent Constraint Programming

In this section we present a stochastic version [2] of Concurrent Constraint Programming [23], which will be used in the following as a modeling language for biological systems.

2.1 Concurrent Constraint Programming

Concurrent Constraint Programming (CCP [23]) is a process algebra having two distinct entities: agents and constraints. Constraints are interpreted first-order logical formulae, stating relationships among variables (e.g. $X = 10$ or $X + Y < 7$). CCP-Agents compute by adding constraints (`tell`) into a "container" (the *constraint store*) and checking if certain relations are entailed by the current configuration of the constraint store (`ask`). The communication mechanism among agents is therefore asynchronous, as information is exchanged through global variables. In addition to `ask` and `tell`, the language has all the basic constructs of process algebras: non-deterministic choice, parallel composition, procedure call, plus the declaration of local variables. This dichotomy between agents and the constraint store can be seen as a form of separation between computing capabilities (pertaining to the constraint store) and the logic of interactions (pertaining to the agents). From a general point of view, the main difference between CCP and π-calculus resides really in the computational power of the former. π-calculus, in fact, has

$$Program = D.A$$

$$D = \varepsilon \mid D.D \mid p(\mathbf{x}) :- A$$

$$\pi = \text{tell}_\lambda(c) \mid \text{ask}_\lambda(c)$$
$$M = \pi.A \mid \pi.A.p(\mathbf{y}) \mid M + M$$
$$A = \mathbf{0} \mid \text{tell}_\infty(c).A \mid \exists_x A \mid M \mid (A \parallel A)$$

Table 1. Syntax of of sCCP.

to describe everything in terms of communications only, a fact that may result in cumbersome programs in all those situations in which "classical" computations are directly or indirectly involved.

The *constraint store* is defined as an algebraic lattice structure, using the theory of cylindric algebras [15]. Essentially, we first choose a first-order language together with an interpretation, which defines a semantical entailment relation (required to be decidable). Then we fix a set of formulae, closed under finite conjunction, as the primitive constraints that the agents can add to the store. The algebraic lattice is obtained by considering subsets of these primitive constraints, closed by entailment and ordered by inclusion. The least upper bound operation in the lattice is denoted by $\sqcup$ and it basically represents the conjunction of constraints. In order to model local variables and parameter passing, the structure is enriched with cylindrification and diagonalization operators, typical of cylindric algebras [15]. These operators allow to define a sound notion of substitution of variables within constraints. In the following we denote the entailment relation by $\vdash$ and a generic constraint store by $\mathcal{C}$. We refer to [6,24,23] for a detailed explanation of the constraint store.

2.2 Syntax of sCCP

The stochastic version of CCP (sCCP [2]) is obtained by adding a stochastic duration to the instructions interacting with the constraint store $\mathcal{C}$, i.e. `ask` and `tell`. More precisely, each instruction is associated with a continuous random variable T, representing the time needed to perform the corrisponding operations in the store (i.e. adding or checking the entailment of a constraint). This random variable is exponentially distributed (cf. [19]), i.e. its probability function is

$$f(\tau) = \lambda e^{-\lambda\tau}, \tag{2.1}$$

where λ is a positive real number, called the rate of the exponential random variable, which can be intuitively seen as the expected frequency per unit of time.

In our framework, the rates associated to `ask` and `tell` are functions

$$\lambda : \mathcal{C} \to \mathbb{R}^+,$$

depending on the current configuration of the constraint store. This means that the speed of communications can vary according to the particular state of the system, though in every state of the store the random variables are perfectly defined (their rate is evaluated to a real number). This fact gives to the language a remarkable flexibility in modeling biological systems, see Section 3 for further material on this point.

The syntax of sCCP can be found in Table 1. An sCCP program consists in a list of procedures and in the starting configuration. Procedures are declared by specifying their name and their free variables, treated as formal parameters. Agents, on the other hand, are defined by the grammar in the last three lines of Table 1. There are two different actions with temporal duration, i.e. `ask` and `tell`, identified by π. Their rate λ is a function as specified above. These actions can be combined together into a guarded choice M (actually, a mixed choice, as we allow both ask and tell to be combined with summation). In the definition of such choice, we force procedure calls to be always guarded. In fact, they are instantaneous operations, thus guarding them by a timed action allows to avoid instantaneous infinite recursive loops, like those possible in $p : -A \parallel p$. In summary, an agent A can choose between different actions (M), it can perform an instantaneous `tell`, it can declare a variable local ($\exists_x A$) or it can be combined in parallel with other agents.

The syntax presented here is slightly different from that of [2]. In fact, the class of instantaneous actions is expanded: in [2] it contained only the declaration of local variables, while here it contains also procedure call and a version of `tell`. Nevertheless, the congruence relation defined in [2], ascribing the usual properties to the operators of the language (e.g. associativity and commutativity to $+$ and $\parallel$), remains the same. The configurations of sCCP programs will vary in the quotient space modulo this congruence relation, denoted by $\mathcal{P}$.

2.3 Operational Semantics of sCCP

The definition of the operational semantics is given specifying two different kinds of transitions: one dealing with instantaneous actions and the other with stochastically timed ones. This is also a novelty w.r.t. [2], though in the previous version an instantaneous transition was implicitly defined in order to deal with local variables. The basic idea of this operational semantics is to apply the two transitions in an interleaved way: first we apply the transitive closure of the instantaneous transition, then we do one step of the timed stochastic transition. To identify a state of the system, we need to take into account both the agents that are to be executed and the current configuration of the store. Therefore, a configuration will be a point in the space $\mathcal{P} \times \mathcal{C}$.

The recursive definition of the instantaneous transition $\longrightarrow \subseteq (\mathcal{P} \times \mathcal{C}) \times (\mathcal{P} \times \mathcal{C})$ is shown in Table 2. Rule $(IR1)$ models the addition of a constraint in the store through the least upper bound operation of the lattice. Recursion corresponds to rule $(IR2)$, which consists in substituting the actual variables to the formal parameters in the definition of the procedure called. In rule $(IR3)$, local variables are replaced by fresh global variables, while in $(IR4)$ the other rules are extended compositionally. Observe that we do not need to deal with summation operator at the level of instantaneous transition, as all the choices are guarded by (stochastically) timed actions. The syntactic restrictions imposed to instantaneous actions guarantee that $\longrightarrow$ can be applied only for a finite

$$(IR1)\quad \langle \mathrm{tell}_\infty(c).A, d\rangle \longrightarrow \langle A, d \sqcup c\rangle$$

$$(IR2)\quad \langle p(\mathbf{x}), d\rangle \longrightarrow \langle A[\mathbf{x}/\mathbf{y}], d\rangle \qquad \text{if } p(\mathbf{y}) :- A$$

$$(IR3)\quad \langle \exists_x A, d\rangle \longrightarrow \langle A[y/x], d\rangle \qquad \text{with } y \text{ fresh}$$

$$(IR4)\quad \frac{\langle A_1, d\rangle \longrightarrow \langle A_1', d'\rangle}{\langle A_1 \parallel A_2, d\rangle \longrightarrow \langle A_1' \parallel A_2, d'\rangle}$$

Table 2. Instantaneous transition for stochastic CCP

$$(SR1)\quad \langle \mathrm{tell}_\lambda(c).A, d\rangle \Longrightarrow_{(1,\lambda(d))} \overrightarrow{\langle A, d \sqcup c\rangle}$$

$$(SR2)\quad \langle \mathrm{ask}_\lambda(c).A, d\rangle \Longrightarrow_{(1,\lambda(d))} \overrightarrow{\langle A, d\rangle} \qquad \text{if } d \vdash c$$

$$(SR3)\quad \frac{\langle M_1, d\rangle \Longrightarrow_{(p,\lambda)} \overrightarrow{\langle A_1', d'\rangle}}{\langle M_1 + M_2, d\rangle \Longrightarrow_{(p',\lambda')} \overrightarrow{\langle A_1', d'\rangle}}$$
$$\text{with } p' = \tfrac{p\lambda}{\lambda + \mathrm{rate}(M_2, d)} \text{ and } \lambda' = \lambda + \mathrm{rate}(M_2, d)$$

$$(SR4)\quad \frac{\langle A_1, d\rangle \Longrightarrow_{(p,\lambda)} \overrightarrow{\langle A_1', d'\rangle}}{\langle A_1 \parallel A_2, d\rangle \Longrightarrow_{(p',\lambda')} \overrightarrow{\langle A_1' \parallel A_2, d'\rangle}}$$
$$\text{with } p' = \tfrac{p\lambda}{\lambda + \mathrm{rate}(A_2, d)} \text{ and } \lambda' = \lambda + \mathrm{rate}(A_2, d)$$

Table 3. Stochastic transition relation for stochastic CCP

number of steps. Moreover it can be proven that it is confluent. Given a configuration $\langle A, d\rangle$ of the system, we denote by $\overrightarrow{\langle A, d\rangle}$ the configuration obtained by applying the transitions $\longrightarrow$ as long as it is possible (i.e., by applying the transitive closure of $\longrightarrow$). The confluence property of $\longrightarrow$ implies that $\overrightarrow{\langle A, d\rangle}$ is well defined.

The stochastic transition $\Longrightarrow \subseteq (\mathcal{P} \times \mathcal{C}) \times [0,1] \times \mathbb{R}^+ \times (\mathcal{P} \times \mathcal{C})$ is defined in Table 3. This transition is labeled by two numbers: intuitively, the first one is the probability of the transition, while the second one is its global rate, see Section 2.4 for further details. Rule $(SR1)$ deals with timed tell action, and works similarly to rule $(IR1)$. Rule $(SR2)$, instead, defines the behaviour of the ask instruction: it is active only if the asked constraint is entailed by the current configuration of the constraint store. Rules (SR3) and (SR4), finally, deal with the choice and the parallel construct. Note that, after performing one step of the transition $\Longrightarrow$, we apply the transitive closure of $\longrightarrow$. This guarantees that all actions enabled after one $\Longrightarrow$ step are timed. In Table 3 we use

the function rate : $\mathcal{P} \times \mathcal{C} \to \mathbb{R}$, assigning to each agent its global rate. It is defined as follows:

Definition 1. *The function* rate : $\mathcal{P} \times \mathcal{C} \to \mathbb{R}$ *is defined by*

1. $\text{rate}(\mathbf{0}, d) = 0$;
2. $\text{rate}(\text{tell}_\lambda(c).A, d) = \lambda(d)$;
3. $\text{rate}(\text{ask}_\lambda(c).A, d) = \lambda(d)$ *if* $d \vdash c$;
4. $\text{rate}(\text{ask}_\lambda(c).A, d) = 0$ *if* $d \nvdash c$;
5. $\text{rate}(M_1 + M_2, d) = \text{rate}(M_1, d) + \text{rate}(M_2, d)$.
6. $\text{rate}(A_1 \parallel A_2, d) = \text{rate}(A_1, d) + \text{rate}(A_2, d)$;

Using relation $\Longrightarrow$, we can build a labeled transition system, whose nodes are configurations of the system and whose labeled edges correspond to derivable steps of $\Longrightarrow$. As a matter of fact, this is a multi-graph, as we can derive more than one transition connecting two nodes (consider the case of $\text{tell}_\lambda(c) + \text{tell}_\lambda(c)$). Starting from this labeled graph, we can build a Continuous Time Markov Chain (cf. [19] and next section) as follows: substitute each label (p, λ) with the real number $p\lambda$ and add up the numbers labeling edges connecting the same nodes. More details about the operational semantics can be found in [2].

2.4 Continuous Time Markov Chains and Gillespie's Algorithm

A Continuous Time Markov Chain (CTMC for short) is a continuous-time stochastic process $(X_t)_{t \geq 0}$ taking values in a discrete set of states S and satisfying the memoryless property, $\forall n, t_1, \ldots, t_n, s_1, \ldots, s_n$:

$$P\{X_{t_n} = s_n \mid X_{t_{n-1}} = s_{n-1}, \ldots, X_{t_1} = s_1\} = P\{X_{t_n} = s_n \mid X_{t_{n-1}} = s_{n-1}\}. \tag{2.2}$$

A CTMC can be represented as a directed graph whose nodes correspond to the states of S and whose edges are labeled by real numbers, which are the rates of exponentially distributed random variables (defined by the probability density (2.1)). In each state there are usually several exiting edges, competing in a race condition in such a way that the fastest one is executed. The time employed by each transition is drawn from the random variable associated to it. When the system changes state, it forgets its past activity and starts a new race condition (this is the memoryless property). Therefore, the traces of a CTMC are made by a sequence of states interleaved by variable time delays, needed to move from one state to another.

The time evolution of a CTMC can be characterized equivalently by computing, in each state, the normalized rates of the exit transitions and their sum (called the exit rate). The next state is chosen according to the probability distribution defined by the normalized rates, while the time spent for the transition is drawn from an exponentially distributed random variable with parameter equal to the exit rate.

This second characterization can be used in a Monte-Carlo simulation algorithm. Suppose to be in state s; then draw two random numbers, one according to the probability given by the normalized rates, and the second according to an exponential probability distribution with parameter equal to the exit rate. Then choose the next state according to the first random number, and increase the time according to the second. The

procedure sketched here is essentially the content of the Gillespie's algorithm [13,14], originally derived in the context of stochastic simulation of chemical reactions. Indeed, the stochastic description of chemical reactions is exactly a Continuous Time Markov Chain [12].

2.5 Stream Variables

In the use of sCCP as a modeling language for biological systems, many variables will represent quantities that vary over time, like the number of molecules of certain chemical species. In addition, the functions returning the stochastic rate of communications will depend only on those variables. Unfortunately, the variables we have at our disposal in CCP are rigid, in the sense that, whenever they are instantiated, they keep that value forever. However, time-varying variables can be easily modeled as growing lists with an unbounded tail: $X = [a_1, \ldots, a_n|T]$. When the quantity changes, we simply need to add the new value, say b, at the end of the list by replacing the old tail variable with a list containing b and a new tail variable: $T = [b|T']$. When we need to compute a function depending on the current value of the variable X, we need to extract from the list the value immediately preceding the unbounded tail. This can be done by defining the appropriate predicates in the first-order language over which the constraint store is built. As these variables have a special status in the presentation hereafter, we will refer to them as *stream variables*. In addition, we will use a simplified notation that hides all the details related to the list update. For instance, if we want to add 1 to the current value of the stream variable X, we will simply write $X = X + 1$. The intended meaning of this notation is clearly: "extract the last ground element n in the list X, consider its successor $n+1$ and add it to the list (instantiating the old tail variable as a list containing the new ground element and a new tail variable)".

2.6 Implementation

We have developed an interpreter for the language that can be used for running simulations. The simulation engine is based on the Gillespie's Algorithm, therefore it performs a Monte-Carlo simulation of the underlying CTMC. The memoryless property of the CTMC guarantees that we do need to generate all its nodes to perform a simulation, but we need to store only the current state. By syntactic analysis of the current set of agents in execution, we can construct all the exit transitions and compute their rates, evaluating rate functions w.r.t. the current configuration of the store (actually, those functions depend only on stream variables, thus their computation has two steps: extract the current value of the variables and evaluate the function). Then we apply the Gillespie's procedure to determine the next state and the elapsed time, updating the system by modifying the current set of agents and the constraint store according to the chosen transition.

The interpreter is written in SICStus Prolog [10]. It is composed by a parser, accepting a program written in sCCP and converting it into an internal list-based representation. The main engine operates therefore by inspecting and manipulating the lists representing the program. The constraint store is managed using the constraint solver on finite domains of SICStus. Stream variables are not represented as lists, but rather as global variables using the meta-predicates `assert` and `retract` of Prolog. The choice

Measurable Entities $\leftrightarrow$	Stream Variables
Logical Entities $\leftrightarrow$	Processes (Control Variables)
Interactions $\leftrightarrow$	Processes

Table 4. Schema of the mapping between elements of biological systems (left) and sCCP (right).

of working with finite domains is mainly related to the fact that the biological systems analyzed can be described using only integer values[1].

In every execution cycle we need to inspect all terms in order to check if they enable a transition. Therefore, the complexity of each step is linear in the size of the (representation) of the program. This can be easily improved by observing that an enabled transition that is not executed remains enabled also in the future.

The correctness of the virtual machine can be proven by showing that it simulates exactly the same CTMC defined by the sCCP program. This can be done by showing that the exit rate and the probability distribution on exiting transitions are computed correctly, according to the operational semantics of sCCP.

3 Modeling Biological Systems

Taking an high level point of view, biological systems can be seen as composed essentially by two ingredients: (biological) entities and interactions among those entities. For instance, in biochemical reaction networks, the molecules are the entities and the chemical reactions are the possible interactions, see [22] and Section 3.1. In gene regulatory networks, instead, the entities into play are genes and regulatory proteins, while the interactions are production and degradation of proteins, and repression and enhancement of gene's expression, cf. [1] and Section 3.2. In addition, entities fall into two separate classes: measurable and logical. Measurable entities are those present in a certain quantity in the system, like proteins or other molecules. Logical entities, instead, have a control function (like gene gates in [1]), hence they are neither produced nor degraded. Note that logical entities are not real world entities, but rather they are part of the models.

The translation scheme between the previously described elements and sCCP objects is summarized in Table 4. Measurable entities are associated exactly to stream variables introduced at the end of Section 2. Logical entities, instead, are represented as processes actively performing control activities. In addition, they can use variables of the constraint store either as control variables or to exchange information. Finally, each

[1] The real valued rates and the stochastic evolution are tight with the definition of the semantics and not with the syntax of the language, thus we do not need to represent them in the store.

interaction is associated to a process modifying the value of certain measurable stream variables of the system.

Associating variables to measurable entities means that we are representing them as part of the environment, while the active agents are associated to the different actions capabilities of the system. These actions have a certain duration and a certain propensity to happen: a fact represented here in the standard way, i.e. associating to each action a stochastic rate. Actually, the speed of most of these actions depends on the quantity of the basic entities they act on. This fact shows clearly the need for having functional rates, which can be used to describe these dependencies explicitly.

In the next subsections we instantiate this general scheme, in order to deal with two classes of biological systems: networks of biochemical reactions and genetic regulatory networks.

3.1 Modeling Biochemical Reactions

Network of biochemical reactions are usually modeled through chemical equations of the form $R_1 + \ldots + R_n \rightarrow_k P_1 + \ldots + P_m$, where the n reactants R_i's (possibly in multiple copies) are transformed into the m products P_j's. In the equation above, either n or m can be equal to zero; the case $m = 0$ represents a degradation reaction, while the case $n = 0$ represents an external feeding of the products, performed by an experimenter. Actually, the latter is not a proper chemical reaction but rather a feature of the environmental setting, though it is convenient to represent it within the same scheme. Each reaction has an associated rate k, representing essentially its basic speed. The actual rate of the reaction is $k \cdot [R_1] \cdots [R_n]$, where $[R_i]$ denotes the number of molecules R_i present in the system. There are cases when a more complex expression for the rate of the reaction is needed, see [25] for further details. For instance, one may wish to describe an enzymatic reaction using a Michaelis-Menten kinetic law [8], rather than modeling explicitly the enzyme-substrate complex formation (as simple interaction/communication among molecules, cf. example below). A set of different *biochemical arrows* (corresponding to different biochemical laws) is shown in Table 5; this list is not exhaustive, but rather a subset of the one presented in [25]. Adding further arrows is almost always straightforward.

In Table 5, we also show how to translate biochemical reactions into sCCP processes. The basic reaction $R_1 + \ldots + R_n \rightarrow_k P_1 + \ldots P_m$ is associated to a process that first checks if all the reactants needed are present in the system (asking if all $[R_i]$ are greater than zero), then it modifies the variables associated to reactants and products, and finally it calls itself recursively. Note that all the `tell` instructions have infinite rate, hence they are instantaneous transitions. The rate governing the speed of the reaction is the one associated to `ask` instruction. This rate is nothing but the function $r_{MA}(k, X_1, \ldots, X_n) = k \cdot X_1 \cdots X_n$ representing mass action dynamics. Note that $\rightleftharpoons$ is a shorthand for the forward and the backward reactions. The arrow $\mapsto^{E}_{K,V_0}$ has a different dynamics, namely Michaelis-Menten kinetics: $r_{MM}(K, V_0, S) = \frac{V_0 S}{S+K}$. This reaction approximates the conversion of a substrate into a product due to the catalytic action of enzyme E when the substrate is much more abundant than the enzyme (quasi-steady state assumption, cf. [8]). The last arrow, instead, is associated to Hill's kinetics. The

$R_1 + \ldots + R_n \to_k P_1 + \ldots + P_m$	$\begin{array}{l}\text{reaction}(k,[R_1,\ldots,R_n],[P_1,\ldots,P_m]) :- \\ \quad \text{ask}_{r_{MA}(k,R_1,\ldots,R_n)}\,(\bigwedge_{i=1}^{n}(R_i>0))\,. \\ \quad \big(\,\|_{i=i}^{n}\,\text{tell}_\infty(R_i = R_i - 1)\,\| \\ \quad \|_{j=1}^{m}\,\text{tell}_\infty(P_j = P_j + 1)\big)\,. \\ \quad \text{reaction}(k,[R_1,\ldots,R_n],[P_1,\ldots,P_m])\end{array}$
$R_1 + \ldots + R_n \rightleftharpoons_{k_2}^{k_1} P_1 + \ldots + P_m$	$\begin{array}{l}\text{reaction}(k_1,[R_1,\ldots,R_n],[P_1,\ldots,P_m])\,\| \\ \text{reaction}(k_2,[P_1,\ldots,P_m],[R_1,\ldots,R_n])\end{array}$
$S \mapsto^{E}_{K,V_0} P$	$\begin{array}{l}\text{mm\_reaction}(K,V_0,S,P) \qquad \text{ask}_{r_{MM}(K,V_0,S)}\,(S>0). \\ \quad (\text{tell}_\infty(S = S-1)\,\|\,tell_\infty(P = P+1))\,. \\ \quad \text{mm\_reaction}(K,V_0,S,P)\end{array}$
$S \mapsto^{E}_{K,V_0,h} P$	$\begin{array}{l}\text{hill\_reaction}(K,V_0,h,S,P) \qquad \text{ask}_{r_{Hill}(K,V_0,h,S)}\,(S>0). \\ \quad (\text{tell}_\infty(S = S-h)\,\|\,tell_\infty(P = P+h))\,. \\ \quad \text{Hill\_reaction}(K,V_0,h,S,P)\end{array}$

where

$$r_{MA}(k,X_1,\ldots,X_n) = k \cdot X_1 \cdots X_n; \quad r_{MM}(K,V_0,S) = \tfrac{V_0 S}{S+K}; \quad r_{Hill}(k,V_0,h,S) = \tfrac{V_0 S^h}{S^h+K^h}$$

Table 5. Translation into sCCP of different biochemical reaction types, taken from the list of [25]. The reaction process models a mass-action-like reaction. It takes in input the basic rate of the reaction, the list of reactants, and the list of products. These list can be empty, corresponding to degradation and external feeding. The process has a blocking guard that checks if all the reactants are present in the system. The rate of the ask is exactly the global rate of the reaction. If the process overcomes the guard, it modifies the quantity of reactants and products and then it calls itself recursively. The reversible reaction is modeled as the combination of binding and unbinding. The third arrow corresponds to a reaction with Michaelis-Menten kinetics. The corresponding process works similarly to the reaction one, but the rate function is different. Here, in fact, the rate function is the one expressing Michaelis-Menten kinetics. See Section 3.1 for further details. The last arrow replaces Michaelis-Menten kinetics with Hill's one (see end of Section 3.1).

dynamics represented here is an improvement on the Michaelis-Menten law, where the exponent h encodes some information about the spatial behaviour of the reaction.

Comparing the encoding of biochemical reaction into sCCP with the encoding into other process algebras like π-calculus [22], we note that the presence of functional rates gives much more flexibility in the modeling phase. In fact, this form of rates allows to describe dynamics that are different from Mass Action. Notable examples are exactly Michaelis-Menten's and Hill's cases, represented by the last two arrows. This is not possible wherever only constant rates are present, as the definition of the operational semantics constrain the dynamics to be Mass-Action like. More comments about this fact can be found in [3].

$$\text{enz\_reaction}(k_1,k_{-1},k_2,S,E,ES,P) :-$$
$$\text{reaction}(k_1,[S,E],[ES]) \parallel \text{reaction}(k_{-1},[ES],[E,S]) \parallel \text{reaction}(k_2,[ES],[E,P]).$$
$$\text{enz\_reaction}(k_1,k_{-1},k_2,S,E,ES,P) \parallel \text{reaction}(k_{prod},[],[S]) \parallel \text{reaction}(k_{deg},[P],[])$$

Table 6. sCCP program for an enzymatic reaction with mass action kinetics. The first block defines the predicate enz_reaction(k_1,k_{-1},k_2,S,E,ES,P), while the second block is the definition of the entire program. The predicate reaction has been defined in Table 5.

Example: Enzymatic Reaction As a first and simple example, we show the model of an enzymatic reaction. We provide two different descriptions, one using a mass action kinetics, the other using a Michaelis-Menten one, see Table 5.

In the first case, we have the following set of reactions:

$$S+E \rightleftharpoons^{k_1}_{k_{-1}} ES \rightarrow_{k_2} P+E; \quad P \rightarrow_{k_{deg}}; \quad \rightarrow_{k_{prod}} S, \tag{3.1}$$

corresponding to a description of an enzymatic reaction that takes into account also the enzyme-substrate complex formation. Specifically, substrate S and enzyme E can bind and form the complex ES. This complex can either dissociate back into E and S, or be converted into the product P and again enzyme E. Moreover, in this particular system we added degradation of P and external feeding of S, in order to have continuous production of P. The sCCP model of this reaction can be found in Table 6. It is simply composed by 5 reaction agents, one for each arrow of the equations (3.2). The three reactions involving the enzyme are grouped together under the predicate `enz_reaction{k1,k-1,k2,S,E,ES,P}`, that will be used in following subsections.

Simulations were performed with the simulator described in Section 2.6, and the trend of product P is plotted in Figure 1 (left). Parameters of the system were chosen in order to have, at regime, almost all the enzyme molecules in the complexed state, see caption of Figure 1 (top) for details.

For this simple enzymatic reaction, the quasi-steady state assumption holds [8], therefore replacing the substrate-enzyme complex formation with a Michaelis-Menten kinetics should leave the system behaviour unaltered. This intuition is confirmed by Figure 1 (bottom), showing the plot of the evolution over time of product P for the following system of reactions:

$$S \mapsto^{E}_{K,V_0} P; \quad P \rightarrow_{k_{deg}}; \quad \rightarrow_{k_{prod}} S,$$

whose sCCP can be derived easily from Table 5.

A slightly more complicated version of the above example is the case in which some level of cooperativity of the enzyme is to be modeled (Hill's case). The set of reactions in this case is an extension of the above one and can be written as:

$$n \times S+E \rightleftharpoons^{k_1}_{k_{-1}} ES_n \rightarrow_{k_2} n \times P+E; \quad P \rightarrow_{k_{deg}}; \quad \rightarrow_{k_{prod}} S. \tag{3.2}$$

In this case the sCCP program is a straightforward extension of the previous one:
while the rest of the coding is entirely similar to the previous case.

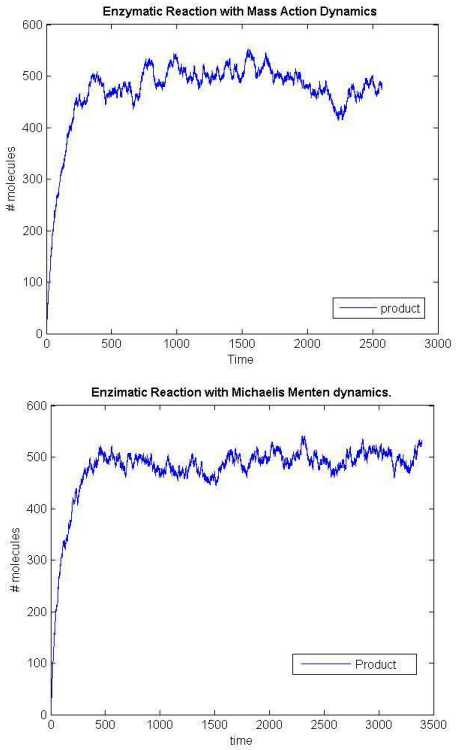


Fig. 1. (**top**) Mass Action dynamics for an enzymatic reaction. The graph shows the time evolution of the product P. Rates used in the simulation are $k_1 = 0.1$, $k_{-1} = 0.001$, $k_2 = 0.5$, $k_{deg} = 0.01$, $k_{prod} = 5$. Enzyme molecules E are never degraded (though they can be in the complex status), and initial value is set to $E = 10$. Starting value for S is 100, while for P is zero. Notice that the rate of complexation of E and S into ES and the dissociation rate of ES into E and P are much bigger than the dissociation rate of ES into E and S. This implies that almost all the molecules of E will be found in the complexed form. (**bottom**) Michaelis-Menten dynamics for an enzymatic reaction. The graph shows the time evolution of the product P. Rates k_{deg} and k_{prod} are the same as above, whilst $K = 5.01$ and $V_0 = 5$. These last values are derived from mass action rates in the standard way, i.e. $K = \frac{K_2+k_{-1}}{k_1}$ and $V_0 = k_2 E_0$, where E_0 is the starting quantity of enzyme E, cf. [8] for a derivation of these expressions. Notice that the time spawn by this second temporal series is longer than the first one, despite the fact that simulations lasted the same number of elementary steps (of the labeled transition system of sCCP). This is because the product formation in the Michaelis-Menten dynamics model is a one step reaction, while in the other system it is a two step reaction (with a possible loop because of the dissociation of ES into E and S).

Also in this case a comparison with the reaction obtained with the computed Hill coefficient

$$S \mapsto^{E}_{K,V_0,n} P; \quad P \rightarrow_{k_{deg}}; \quad \rightarrow_{k_{prod}} S,$$

$$\text{n\_enz\_reaction}(k_1, k_{-1}, k_2, S, E, ES, P) :-$$
$$\text{reaction}(k_1, [n \times S, E], [ES_n]) \parallel \text{reaction}(k_{-1}, [ES_n], [E, n \times S]) \parallel \text{reaction}(k_2, [ES_n], [E, n \times P]).$$

$$\begin{array}{l}
\text{enz_reaction}(k_a, k_d, k_r, KKK, E1, KKKE1, KKKS) \parallel \\
\text{enz_reaction}(k_a, k_d, k_r, KKKS, E2, KKKSE2, KKK) \parallel \\
\text{enz_reaction}(k_a, k_d, k_r, KK, KKKS, KKKKKS, KKP) \parallel \\
\text{enz_reaction}(k_a, k_d, k_r, KKP, KKP1, KKPKKP1, KK) \parallel \\
\text{enz_reaction}(k_a, k_d, k_r, KKP, KKKS, KKPKKKS, KKPP) \parallel \\
\text{enz_reaction}(k_a, k_d, k_r, KKPP, KKP1, KKPPKKP1, KKP) \parallel \\
\text{enz_reaction}(k_a, k_d, k_r, K, KKPP, KKKPP, KP) \parallel \\
\text{enz_reaction}(k_a, k_d, k_r, KP, KP1, KPKP1, K) \parallel \\
\text{enz_reaction}(k_a, k_d, k_r, KP, KKPP, KPKKPP, KPP) \parallel \\
\text{enz_reaction}(k_a, k_d, k_r, KPP, KP1, KPPKP1, KP)
\end{array}$$

Table 7. sCCP code for the MAP-Kinase signaling cascade. The enz_reaction predicate has been defined in Section 3.1. For this example, we set the complexation rates (k_a), the dissociation rates (k_d) and the product formation reaction rates (k_r) equal for all the reactions involved. For the actual values used in the simulation, refer to Figures 3 and 4.

can be easily carried out. Notice that the Hill's exponent corresponds exactly to the degree of cooperativity of the enzyme.

Also a more refined approach to the case of Hill's kinetics is possible, decomposing the n-fold reaction in a series of n separated by Mass Action equation simulations.

Example: MAP-Kinase Cascade A cell is not an isolated system, but it communicates with the external environment using complex mechanisms. In particular, a cell is able to react to external signals, i.e. to signaling proteins (like hormones) present in the proximity of the external membrane. Roughly speaking, this membrane is filled with receptor proteins, that have a part exposed toward the external environment capable of binding with the signaling protein. This binding modifies the structure of the receptor protein, that can now trigger a chain of reactions inside the cell, transmitting the signal straight to the nucleus. In this signaling cascade a predominant part is performed by a family of proteins, called Kinase, that have the capability of phosphorylating other proteins. Phosphorylation is a modification of the protein fold by attaching a phosphorus molecule to a particular amino acid of the protein. One interesting feature of these cascades of reactions is that they are activated only if the external stimulus is strong enough. In addition, the activation of the protein at the end of the chain of reactions (usually an enzyme involved in other regulation activities) is very quick. This behaviour of the final enzyme goes under the name of ultra-sensitivity [17].

In Figure 2 a particular signaling cascade is shown, involving MAP-Kinase proteins. This cascade has been analyzed using differential equations in [17] and then modeled and simulated in stochastic Pi-Calculus in [4]. We can see that the external stimulus, here generically represented by the enzyme E_1, triggers a chain of enzymatic reactions. MAPKKK is converted into an active form, called MAPKKK*, that is capable of phosphorylating the protein MAPKK in two different sites. The diphosphorylated version

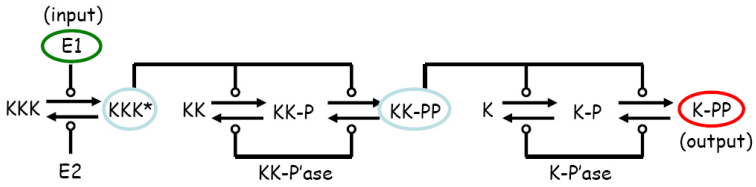


Fig. 2. Diagram of the MAP-Kinase cascade. The round-headed arrow schematically represents an enzymatic reaction, see Section 3.1 for further details. This diagram has been stolen from a presentation of Luca Cardelli, held in Dobbiaco, September 2005.

MAPKK-PP of MAPKK is the enzyme stimulating the phosphorylation of another Kinase, i.e. MAPK. Finally, the diphosphorylated version MAPK-PP of MAPK is the output of the cascade.

The sCCP program describing MAP-Kinase cascade is shown in Table 7. The program itself is very simple, and it uses the mass action description of an enzymatic reaction (cf. Table 5). It basically consists in a list of the reactions involved, put in parallel. The real problem in studying such a system is in the determination of its 30 parameters, corresponding to the basic rates of the reactions involved. In addition, we need to fix a set of initial values for the proteins that respects their usual concentrations in the cell. Following [4], in Figure 3 we skip this problem and assign a value of 1.0 to all basic rates, while putting 100 copies of MAPKKK, MAPKK and MAPK, 5 copies of E2, MAPKK-P'ase, and MAPK-P'ase and just 1 copy of the input E1. This simple choice, however, is enough to predict correctly all the expected properties: the MAPK-PP time evolution, in fact, follows a sharp trend, jumping from zero to 100 in a short time. Remarkably, this property is not possessed by MAPKK-PP, the enzyme in the middle of the cascade. Therefore, this switching behaviour exhibited by MAPK-PP is intrinsically connected with the double chain of phosphorylations, and cannot be obtained by a simpler mechanism. Notice that the fact that the network works as expected using an arbitrary set of rates is a good argument in favor of its robustness and resistance to perturbations.

In Figure 4, instead, we choose a different set of parameters, as suggested in [17] (cf. its caption). We also let the input strength vary, in order to see if the activation effect is sensitive to its concentration. As we can see, this is the case: for a low value of the input, no relevant quantity of MAPK-PP is present in the system.

3.2 Modeling Gene Regulatory Networks

In a cell, only a subset of genes are expressed at a certain time. Therefore, an important mechanism of the cell is the regulation of gene expression. This is obtained by specific proteins, called *transcription factors*, that bind to the promoter region of genes (the portion of DNA preceding the coding region) in order to enhance or repress their transcription activity. These transcription factors are themselves produced by genes,

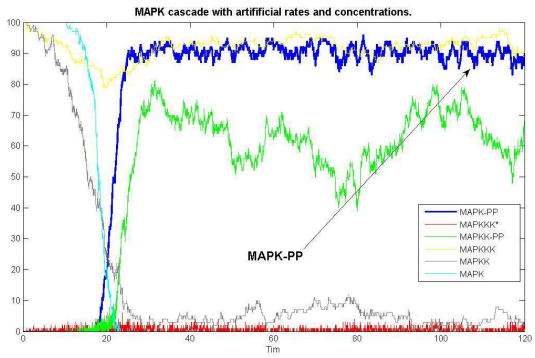


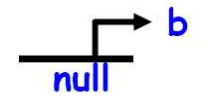
Fig. 3. Temporal trace for some proteins involved in the MAP-Kinase cascade. Traces were generated simulating the sCCP program of Table 7. In this simulation, the rates k_a, k_d, k_r were all set to one. We can notice the sharp increase in the concentration of the output enzyme, MAPK-PP, and its stability in the high expression level. The enzyme MAPKK-PP, the activator of MAPK phosphorylations, instead has a more unstable trend of expression.

thus the overall machinery is a networks of genes producing proteins that regulate other genes. The resulting system is highly complex, containing several positive and negative feedback loops, and usually very robust. This intrinsic complexity is a strong argument in favor of the use of a mathematical formalism to describe and analyze them. In the literature, different modeling techniques are used, see [7] for a Survey. However, we focus on a modeling formalism based on stochastic π-calculus [1].

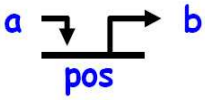
In [1], the authors propose to model gene networks using a small set of "logical" gates, called *gene gates*, encoding the possible regulatory activities that can be performed on a gene. Specifically, there are three types of gene gates: *nullary gates*, *positive gates* and *negative gates*. Nullary gates represent genes with transcriptional activity, but with no regulation. Positive gates are genes whose transcription rate can be increased by a transcription factor. Finally, negative gates represent genes whose transcription can be inhibited by the binding of a specific protein. At the level of abstraction of [1], the product of a gene gate is not a mRNA molecule, but directly the coded protein. These product proteins are then involved in the regulation activity of the same or of other genes and can also be degraded.

We propose now an encoding of gene gates within sCCP framework, in the spirit of Table 4. Proteins are measurable entities, thus they are encoded as stream variables; gene gates, instead, are logical control entities and they are encoded as agents. The degradation of proteins is modeled by the reaction agent of Table 5. In Table 8 we present the sCCP agents associated to gene gates. A nullary gate simply increases the quantity of the protein it produces at a certain specified rate. Positive gates, instead, can produce their coded protein at the basic rate or they can enter in an enhanced state where production happens at an higher rate. Entrance in this excited state happens at a rate proportional to the quantity of transcription factors present in the system. Negative

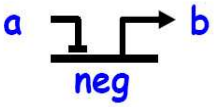
gates behave similarly to positive ones, with the only difference that they can enter an inhibited state instead of an enhanced one. After some time, the inhibited gate returns to its normal status. A specific gene, generally, can be regulated by more than transcription factor. This can be obtained by composing in parallel the different gene gates.



$$\text{null\_gate}(k_p, X) :- \\ \text{tell}_{k_p}(X = X + 1).\text{null\_gate}(k_p, X)$$



$$\text{pos\_gate}(k_p, k_e, k_f, X, Y)\ \text{tell}_{k_p}(X = X + 1).\text{pos\_gate}(k_p, k_e, k_f, X, Y) \\ + \text{ask}_{r(k_e,Y)}(true).\text{tell}_{k_e}(X = X + 1).\text{pos\_gate}(k_p, k_e, k_f, X, Y)$$



$$\text{neg\_gate}(k_p, k_i, k_d, X, Y)\ \text{tell}_{k_p}(X = X + 1).\text{neg\_gate}(k_p, k_i, k_d, X, Y) \\ + \text{ask}_{r(k_i,Y)}(true).\text{ask}_{k_d}(true).\text{neg\_gate}(k_p, k_i, k_d, X, Y)$$

$$\text{where } r(k, Y) = k \cdot Y.$$

Table 8. Scheme of the translation of gene gates into sCCP programs. The null gate is modeled as a process continuously producing new copies of the associated protein, at a fixed rate k_p. The negative gate is modeled as a process that can either produce a new protein or enter in an repressed state due to the binding of the repressor. This binding can happen at a rate proportional to the concentration of the repressor. After some time, the repressor unbinds and the gate return in the normal state. The enhancing of activators in the pos gate, instead, is modeled here in an "hit and go" fashion. The enhancer can hit the gate and make it produce a protein at an higher rate than usual. The hitting rate is proportional to the number of molecules of the stimulating protein.

Example: Bistable Circuit The first example, taken from [1], is a gene network composed by two negative gates repressing each other, see Figure 5. The sCCP model for this simple network comprehends two negative gates: the first producing protein *A* and repressed by protein *B*, the second producing protein *B* and repressed by protein *A*. In addition, there are the degradation reactions for proteins *A* and *B*. This network is bistable: only one of the two proteins is expressed. If the initial concentrations of *A* and *B* are zero, then the stochastic fluctuations happening at the beginning of the simulations decide which of the two fix points will be chosen. In Figure 5 we show one

possible outcome of the system, starting with zero molecules of *A* and *B*. In this case, protein *A* wins the competition. Notice that the high sensitivity of this system makes it unsuitable for biological system.

Example: Repressilator The repressilator [9] is a synthetic biochemical clock composed of three genes expressing three different proteins, **tetR**, **λcI**, **LacI**, that have a regulatory function in each other's gene expression. In particular, protein **tetR** inhibits the expression of protein **λcI**, while protein **λcI** represses the gene producing protein **LacI** and, finally, protein **LacI** is a repressor for protein **tetR**. The expected behavior is an oscillation of the concentrations of the tree proteins with a constant frequency.

The model we present here is extracted from [1], and it is constituted by three negative gene gates repressing each other in cycle (see Figure 6). The result of a simulation of the sCCP program is shown in Figure 6, where the oscillatory behaviour is manifest. In [1] it is shown that the oscillatory behaviour is stable w.r.t. changes in parameters. Interestingly, some models of the repressilator using differential equations do not show this form of stability. More comments on the differences between continuous and discrete models of repressilator can be found in [3].

3.3 Modeling the Circadian Clock

In this section we provide as a final example the model of a system containing regulatory mechanism both at the level of genes and at the level of proteins. The system is schematically shown in Figure 7. It is a simplified model of the machinery involved in the circadian rhythm of living beings. In fact, this simple network is present in a wide range of species, from bacteria to humans. The circadian rhythm is a typical mechanism responding to environmental stimuli, in this case the periodic change between light and dark during a day. Basically, it is a clock, expressing a protein periodically with a stable period. This periodic behaviour, to be of some use, must be stable and resistant to both external and internal noise. Here with internal noise we refer to the stochastic fluctuations observable in the concentrations of proteins. The model presented here is taken from [26], a paper focused on the study of the resistance to noise of this system. Interestingly, they showed that the stochastic fluctuations make the oscillatory behaviour even more resistant. Our aim, instead, is that of showing how a system like this can be modeled in an extremely compact way, once we have at disposal the libraries of Sections 3.1 and 3.2.

The system is composed by two genes, one expressing an activator protein A, the other producing a repressor protein R. The generation of a protein is depicted here in more detail than in Section 3.2, as the transcription phase of DNA into mRNA and the traduction phase of mRNA into the protein are both modeled explicitly. Protein A is an enhancer for both genes, meaning that it regulates positively their expression. Repressor R, instead, can capture protein A, forming the complex AR and making A inactive. Proteins A and R are degraded at a specific rate (see the caption of Figure 7 for more details about the numerical values), but R can be degraded only if it is not in the complexed form, while A can be degraded in any form. Notice that the regulation activity of A is modeled by an explicit binding to the gene, which remains stimulated

$$
\begin{array}{c}
\text{pos\_gate}(\alpha_A, \alpha'_A, \gamma_A, \theta_A, M_A, A) \parallel \\
\text{pos\_gate}(\alpha_R, \alpha'_R, \gamma_R, \theta_R, M_R, A) \parallel \\
\text{reaction}(\beta_A, [M_A], [A]) \parallel \\
\text{reaction}(\delta_{MA}, [M_A], []) \parallel \\
\text{reaction}(\beta_R, [M_R], [R]) \parallel \\
\text{reaction}(\delta_{MR}, [M_R], []) \parallel \\
\text{reaction}(\gamma_C, [A, R], [AR]) \parallel \\
\text{reaction}(\delta_A, [AR], [R]) \parallel \\
\text{reaction}(\delta_A, [A], []) \parallel \\
\text{reaction}(\delta_R, [R], [])
\end{array}
$$

Table 9. sCCP program for the circadian rhythm regulation system of Figure 7. The agents used have been defined in the previous sections. The first four reaction agents model the translation of mRNA into the coded protein and its degradation. Then we have complex formation, and the degradation of R and A. The pos_gate agent has been redefined as follows, in order to take into account the binding/unbinding of the enhancer: pos_gate$(K_p, K_e, K_b, K_u, P, E)$:- pos_gate_off$(K_p, K_e, K_b, K_u, P, E)$; pos_gate_off$(K_p, K_e, K_b, K_u, P, E)$:- tell$_{K_p}(P = P + 1)$.pos_gate_off$(K_p, K_e, K_b, K_u, P, E)$ + ask$_{r_{ma}(K_b,E)}(E > 0)$.pos_gate_on$(K_p, K_e, K_b, K_u, P, E)$; pos_gate_on$(K_p, K_e, K_b, K_u, P, E)$:- tell$_{K_e}(P = P + 1)$.pos_gate_on$(K_p, K_e, K_b, K_u, P, E)$ + ask$_{K_u}$(true).pos_gate_off$(K_p, K_e, K_b, K_u, P, E)$.

until A unbinds. This mechanism is slightly different from the positive gate described in Section 3.2, but the code can be adapted in a straightforward manner (we simply need to define two states for the gene: bound and free, see caption of Table 9).

The code of the sCCP program modeling the system is shown in Table 9. It makes use of the basic agents defined previously, and it is very compact and very easy and quick to write. In Figure 8 (top) we show the evolution of proteins A and R in a numerical simulation performed with the interpreter of the language. As we can see, they oscillate periodically and the length of the period is remarkably stable. Figure 8 (bottom), instead, shows what happens if we replace the bind/unbind model of the gene gate with the "hit and go" code of Section 3.2 (where the enhancer do not bind to the gene, but rather puts it into a stimulated state that makes the gene produce *only the next protein* quicker). The result is dramatic, the periodic behaviour is lost and the system behaves in a chaotic way.

4 Conclusion and future work

In this paper we presented an application of stochastic concurrent constraint programming for modeling of biological systems. We dealt with two main classes of biological networks: biochemical reactions and gene regulation. The main theme is the use of constraints in order to store information about the biological entities into play; this lead straightforwardly to the definition of a general purpose library of processes that can be used in the modeling phase (see Sections 3.1 and 3.2). However, this is only a part of

the general picture, as there are more complex classes of biological systems that need to be modeled, like transport networks and membranes. In addition, all these systems are strongly interconnected, and they must be modeled altogether in order to extract deep information about living beings. We believe that the flexibility of constraints makes sCCP a powerful general purpose language that can be simply programmed, extended with libraries, and used to model all these different classes of systems in a compact way. For instance, different kinds of spatial information, like exact position of molecules or the compartment they are in, can be easily represented using suitable constraints.

Biochemical reactions can be challenging to model, because proteins can form very big complexes that are built incrementally. Therefore, we can find in the cell a huge number of sub-complexes. Usually, these networks are described by biologists with diagrams, like Kohn maps [18], that are very compact, because they represent complexes and sub-complexes implicitly. Modeling these networks explicitly, instead, can be extremely difficult, due to the blow up of the number of different molecules of the system. A calculus having complexation as a primitive operation, the κ-calculus, has been developed in [5]. It offers a compact way to represent formally these diagrams. Constraints can be used to encode this calculus elegantly, by representing complexes implicitly, i.e. as lists of basic constituents.

Another interesting feature that sCCP offers are functional rates. As shown in Section 3.1, they can be used to represent more complex kinetic dynamics, allowing a more compact description of the networks. In this direction, we need to make deeper analysis of the relation between these different kinetics in the context of stochastic simulation, in order to characterize the cases where these different kinetics can be used equivalently. Notice that the use of complex rates can be seen as an operation on the Markov Chain, replacing a subgraph with a smaller one, hiding part of its complexity in the expression of rates. This seems to be a sort of non-trivial lumpability relation [19], though further studies are necessary.

In [3], the authors investigate the expressivity gained by the addition of functional rates to the language. They suggest that there is an increase of power in terms of dynamical behaviours that can be reproduced, after encoding in sCCP a wide class of differential equations. This problem, together with the inverse one of describing sCCP programs by differential equations, is an interesting direction of research, which may lead to an integration of these different techniques, see [16,3] for further comments.

Finally, we plan to implement a more powerful and fast interpreter for the language, using also all available tricks to increase the speed of stochastic simulations [12]. Moreover, we plan to tackle also the problem of distributing efficiently the stochastic simulations of programs written in sCCP.

References

1. R. Blossey, L. Cardelli, and A. Phillips. A compositional approach to the stochastic dynamics of gene networks. *T. Comp. Sys. Biology*, pages 99–122, 2006.
2. L. Bortolussi. Stochastic concurrent constraint programming. In *Proceedings of 4th International Workshop on Quantitative Aspects of Programming Languages, QAPL 2006*, 2006.
3. L. Bortolussi and A. Policriti. Relating stochastic process algebras and differential equations for biological modeling. *Proceedings of PASTA 2006*, 2006.

4. L. Cardelli and A. Phillips. A correct abstract machine for the stochastic pi-calculus. In *Proceeding of Bioconcur 2004*, 2004.
5. V. Danos and C. Laneve. Formal molecular biology. *Theor. Comput. Sci.*, 325(1):69–110, 2004.
6. F.S. de Boer, A. Di Pierro, and C. Palamidessi. Nondeterminism and infinite computations in constraint programming. *Theoretical Computer Science*, 151(1), 1995.
7. H. De Jong. Modeling and simulation of genetic regulatory systems: A literature review. *Journal of Computational Biology*, 9(1):67–103, 2002.
8. L. Edelstein-Keshet. *Mathematical Models in Biology*. SIAM, 2005.
9. M.B. Elowitz and S. Leibler. A syntetic oscillatory network of transcriptional regulators. *Nature*, 403:335–338, 2000.
10. Swedish Institute for Computer Science. Sicstus prolog home page.
11. D. Gillespie. The chemical langevin equation. *Journal of Chemical Physics*, 113(1):297–306, 2000.
12. D. Gillespie and L. Petzold. *System Modelling in Cellular Biology*, chapter Numerical Simulation for Biochemical Kinetics. MIT Press, 2006.
13. D.T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J. of Computational Physics*, 22, 1976.
14. D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. of Physical Chemistry*, 81(25), 1977.
15. L. Henkin, J.D. Monk, and A. Tarski. *Cylindric Algebras, Part I*. North-Holland, Amsterdam, 1971.
16. J. Hillston. Fluid flow approximation of pepa models. In *Proceedings of the Second International Conference on the Quantitative Evaluation of Systems (QEST'05)*, 2005.
17. C.F. Huang and J.T. Ferrell. Ultrasensitivity in the mitogen-activated protein kinase cascade. *PNAS, Biochemistry*, 151:10078–10083, 1996.
18. K. W. Kohn. Molecular interaction map of the mammalian cell cycle control and dna repair systems. *Molecular Biology of the Cell*, 10:2703–2734, August 1999.
19. J. R. Norris. *Markov Chains*. Cambridge University Press, 1997.
20. C. Priami. Stochastic π-calculus. *The Computer Journal*, 38(6):578–589, 1995.
21. C. Priami and P. Quaglia. Stochastic π-calculus. *Briefings in Bioinformatics*, 5(3):259–269, 2004.
22. C. Priami, A. Regev, E. Y. Shapiro, and W. Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Inf. Process. Lett.*, 80(1):25–31, 2001.
23. V. A. Saraswat. *Concurrent Constraint Programming*. MIT press, 1993.
24. V. A. Saraswat, M. Rinard, and P. Panangaden. Semantics foundations of concurrent constraint programming. In *Proceedings of POPL*, 1991.
25. B. E. Shapiro, A. Levchenko, E. M. Meyerowitz, Wold B. J., and E. D. Mjolsness. Cellerator: extending a computer algebra system to include biochemical arrows for signal transduction simulations. *Bioinformatics*, 19(5):677–678, 2003.
26. J. M. G. Vilar, H. Yuan Kueh, N. Barkai, and S. Leibler. Mechanisms of noise resistance in genetic oscillators. *PNAS*, 99(9):5991, 2002.

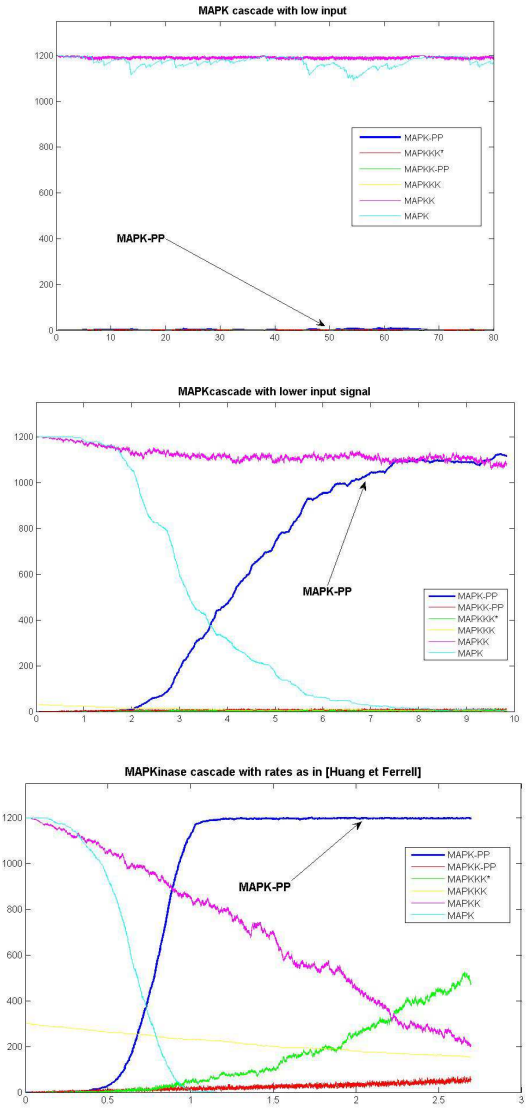


Fig. 4. Comparison of the temporal evolution of the MAP-Kinase cascade for different concentrations of the enzyme MAPKKK. As argued in [17], this is equivalent to the variation of the input signal E1. Rates are equal for all reactions, and have the following values: $k_a = 1$, $k_d = 150$, $k_r = 150$. This corresponds to a Michaelis-Menten rate of 300 for all the enzymatic reactions. The initial quantity of MAPKK and MAPK is set to 1200, the initial quantity of phosphatase MAPK-P'ase is set to 120, the initial quantity of other phosphatase and the enzyme E2 is set to 5, and the initial quantity of E1 is 1. (**top**) The initial quantity of MAPKKK is 3. We can see that there is no sensible production of MAPK-PP. (**middle**) The initial quantity of MAPKKK is 30. Enzyme MAPK-PP is produced but its trend is not sharp, as expected. (**bottom**) The initial quantity of MAPKKK is 300. The system behaves as expected. We can see that the increase in the concentration of MAPK-PP is very sharp, while MAPKK-PP grows very slowly in comparison.

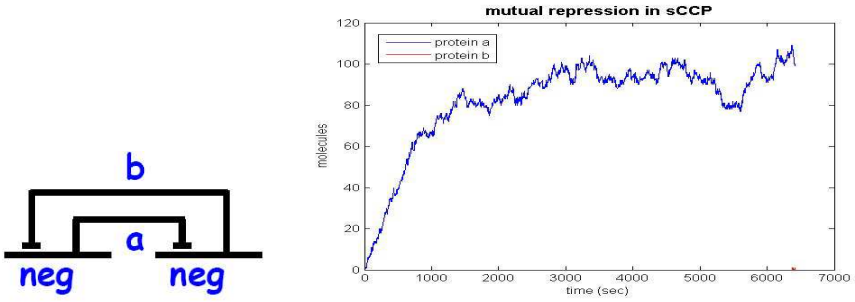


Fig. 5. Bistable circuit. (**right**) Diagram of gene gates involved. (**left**) Time evolution of the circuit. The negative gates have the same rates, set as follows: basic production rate is 0.1 (k_p in Table 8), degradation rate of proteins is 0.0001, inhibition rate (k_i) is 1 and inhibition delay rate (k_d) is 0.0001. Both proteins have an initial value of zero. This graph is one of the two possible outcomes of this bistable network. In the other the roles of the two proteins are inverted.

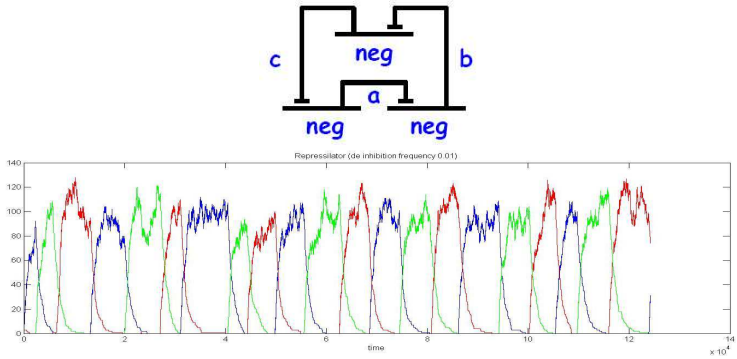


Fig. 6. Repressilator. (**right**) Diagram of gene gates involved. (**left**) Time evolution of the circuit. The negative gates have the same rates, set as follows: basic production rate is 0.1 (k_p in Table 8), degradation rate of proteins is 0.0001, inhibition rate (k_i) is 1 and inhibition delay rate (k_d) is 0.0001. All proteins have an initial value of zero. The time evolution of the repressilator is stable: all simulation traces show this oscillatory behaviour. However, the oscillations among different traces usually are out of phase, and the frequency of the oscillatory pattern varies within the same trace. Remarkably, the average trend of the three proteins shows no oscillation at all, see [3] for further details.

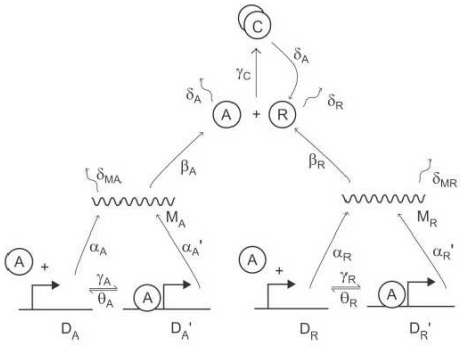


Fig. 7. Biochemical network for the circadian rhythm regulatory system. The figure is taken from [26], like numerical values of rates. Rates are set as follows: $\alpha_A = 50$, $\alpha'_A = 500$, $\alpha_R = 0.01$, $\alpha'_R = 50$, $\beta_A = 50$, $\beta_R = 5$, $\delta_{MA} = 10$, $\delta_{MR} = 0.5$, $\delta_A = 1$, $\delta_R = 0.2$, $\gamma_A = 1$, $\gamma_R = 1$, $\gamma_C = 2$, $\theta_A = 50$, $\theta_R = 100$.

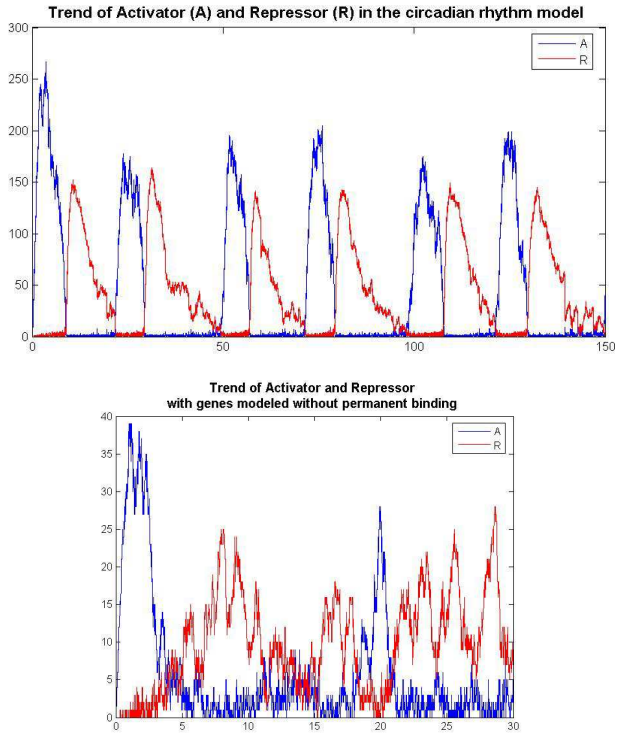


Fig. 8. Time evolution for circadian rhythm model in sCCP. (top) The figure refers to the system described in Figure 7, with parameters described in the caption of the figure. We can see the regularity of the period of the oscillations. (bottom) The graph shows the time evolution for the model where the process governing the gene is the pos_gate described in Table 8. The periodic behaviour, with this simple modification, is irremediably lost.

Chemera: Constraints in Protein Structural Problems

Ludwig Krippahl, Pedro Barahona

Departamento de Informática,FCT/UNL. 2829-516 Monte de Caparica, Portugal
ludi@di.fct.unl.pt

Abstract. Chemera is a molecular modelling software package that includes the algorithms BiGGER (Bimolecular complex Generation with Global Evaluation and Ranking), for modelling protein interactions and protein complex structures [1,2,3], and PSICO (Processing Structural Information with Constraint programming and Optimisation), to integrate experimental and theoretical data to solve protein structures [4, 5]. This paper focuses on the constraint programming aspects of Chemera, namely constrained docking, which allows the user to restrict the search for protein-protein complex models in a manner consistent with the ambiguity of some experimental data, and the processing of structural constraints to generate approximate models of protein structures from heterogeneous data. This allows the user to take advantage of experimental data obtained by a wide range of techniques, from spectroscopy to site-directed mutagenesis, and to integrate these data with theoretical considerations such as homology models, secondary structure prediction, or reaction mechanisms. For modelling protein complexes, this is done by specifying sets of potential contacts between the two proteins and how many of those contacts must be enforced, without having to specify exactly which contacts to enforce, which models the possibility of some of experimental results being due to effects other than proximity to the docking partner. For modelling the structure of a single protein, the information can be encoded as distances between atoms, as fixed relative positions for groups of atoms, or of constraints on angles of rotation around atomic bonds connecting rigid groups.

1. Introduction

Protein-protein interactions play a central role in biochemical reactions, and understanding these interactions is an important step in several fields of biochemical research. Modelling software provides useful tools to help researchers elucidate protein interaction mechanisms, and two decades since the pioneering work of Katzir and others [6] have seen significant developments in algorithms to generate models and scoring functions to select the most likely candidates. Examples from the CAPRI (Critical Assessment of Protein Interactions) experiment [7] illustrate the diversity of protein interaction modelling (protein docking) packages currently available [8, 9, 10, 11].

A common trend in these approaches is to try to model interactions using only knowledge derived from the structure and physicochemical properties of the proteins involved. Some algorithms have been developed [1, 12] or adapted [13] to use data on

the interaction mechanisms, but this approach is still the exception rather than the norm. BiGGER is one of these exceptions, and the Chemera modelling package has been developed from the start to help the researcher bring into the modelling process as much data as available. Previous results show that BiGGER can be a powerful modelling tool when used in this manner [2, 14, 15, 16, 17, 18].

Important as it is to have a good model of the structure formed by the interaction of different proteins (a protein complex), it is even more important to know the structure of each partner, a prerequisite for modelling the complex. In this field the common approaches have been either theoretical, to try to predict the structure from the physical properties of the amino acid sequence in the protein, or homologies with other known structures, or experimental, specializing on the processing of data from specific techniques like Nuclear Magnetic Resonance (NMR) spectroscopy. PSICO aims at bringing the two approaches together by providing a flexible framework for processing geometrical constraints and thus integrate information from all relevant sources in the modelling of a protein structure. NMR data can be modelled as distance constraints [5, 4] or as torsion-angle constraints [3], homology or secondary structure prediction data can be modelled as rigid-group constraints [3], energy functions can be included in the local-search optimization stage, and amino acid properties relevant for protein folding, such as hydrophobicity, can be part of the enumeration heuristics during constraint processing.

2. BiGGER: the docking algorithm.

At the core of our protein docking algorithm is the representation of the protein shapes and the measure of surface contact. The former is a straightforward representation using a regular cubic lattice of cells, similar to that commonly used in the Fast Fourier Transform (FFT) methods derived from [6]. In BiGGER the cells do not correspond to numerical values, but each cell can be either an empty cell, a surface cell, or a core cell. The surface cells define the surface of the structure, and the overlap of surface cells measures the surface of contact. Figure 1 illustrates these concepts, showing on the first two panels a cutaway diagram of the grid representing a protein structure, and on the third panel a cutaway diagram of two grids in contact, showing the contact region corresponding to a set of overlapping surface cells.

This representation has several advantages over the FFT approach, requiring about a thousand times less memory (approximately 15Mb in BiGGER vs 8Gb for FFT in large proteins) and being up to ten times faster than FFT [19]. BiGGER also models side-chain flexibility implicitly by adjusting the core grid representation [1] and allows for hard or soft docking simulations depending on the nature of the interaction to model. Furthermore, this representation and the search algorithm can take advantage of information about the interaction to simultaneously improve the results and speed up the calculations.

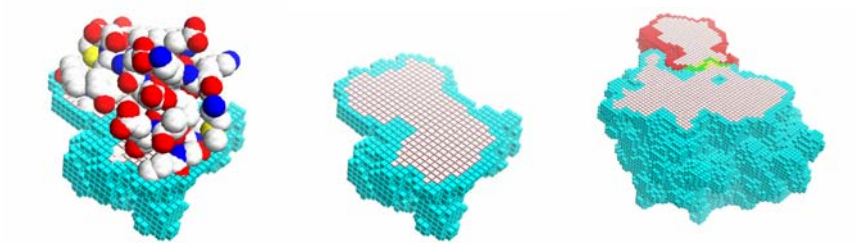


Fig. 1. The image on the left shows a protein structure overlaid on a cutaway of the respective grid, with spheres representing the atoms of the protein. The centre figure shows only the grid generated for this protein, cut to show the surface in light blue and the core region in grey. The rightmost image shows two grids (red and blue) in contact.

2.1 Restricting the search to surface overlapping regions.

A significant proportion of all possible configurations for the two grids results in no surface overlap. Much can be gained by restricting the search to those configurations where surface cells of one grid overlap surface cells of the other. This is achieved by encoding the grids in a convenient way: instead of individual cells, grids are composed of lists of intervals specifying the segments of similar cells along the X coordinate. These lists are arranged in a two-dimensional array on the Y-Z plane.

This encoding not only reduces the memory requirements for storing the grids, but also leads naturally to searching along the X axis by comparing segments instead of by running through all the possible displacements along this coordinate. Given two surface segments, one from each structure and aligned in the same Y and Z coordinates, we can calculate the displacements where overlap will occur simply from the X coordinates of the extremities of the segments.

Representing by a variable the displacement of one structure relative to the other along the X direction, this approach of comparing segments efficiently enforces the constraint requiring surface overlaps by reducing the domain of this variable to only those values where the constraint is verified, as we explain in the next section.

2.2 Eliminating regions of core overlap

Another important constraint in this problem is that the core regions of the grids cannot overlap, for that indicates the structures are occupying the same space instead of being in contact. By identifying the configurations where such overlaps occur, it is possible to eliminate from consideration those surface segments on each structure that cannot overlap surface segments on the other structure without violating the core overlap constraint. Some surface segments can thus be discarded from each search along the X axis. Figure 2 illustrates this procedure.

One structure, labelled A, is shown in the centre of the image. The other structure, labelled B, will be moved along the horizontal direction to scan all possible configurations but, from the overlap of core segments, a set of positions along the

horizontal direction can be eliminated. Structure B is shown in position 1 to the right of A and in position 39 to the left of A, but, in this case, it cannot occupy positions in the centre.

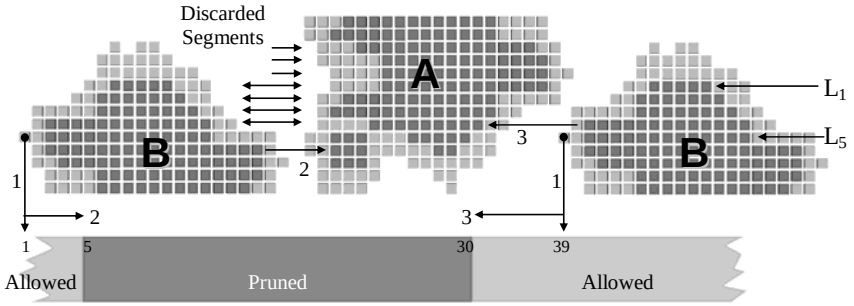


Fig. 2. Grid B is translated along the horizontal direction relative to grid A. The vertical arrows marked 1 indicate the position of B on the lower horizontal bar, which shows the allowed and forbidden values for the position of B. The arrows marked 2 and 3 show the allowed displacement of B. The group of horizontal arrows indicates segments to be discarded.

The domain of variable x, representing the displacement of one structure relative to the other along the X direction, can be pruned from the values 5 to 30. This is a contiguous interval in this example, but the domain of x can be an arbitrary set of intervals in the general case. This domain reduction due to the core overlap constraint propagates to the surface overlap, since some surface segments of A and B will not overlap in valid configurations. Some of these are shown in Figure 3 by the group of arrows to the left of structure A (Discarded Segments, Figure 3). For the last double arrow, for example, the surface cells of structures A and B would only overlap for x=7, a value pruned from the domain of x. In contrast, in the line below such overlap occurs for x = 3, a value kept in the domain. The top three arrows point to surface segments on structure A which can be ignored in this case. The top three surface segments on structure B cannot be ignored because they may overlap with the surface segments of A on the other side, once B is moved to the right of A, but the following four arrows indicate that both the segments to the left of A and those to the right of B can be ignored. Thus the core overlap constraint allows us to reduce the number of surface segments to consider when counting surface overlaps.

This approach can be generalized for the translational search. Three variables, z, y, and x, and their respective domains, D_z, D_y, and D_x, represent the translation of B with respect to A. The domains are initialised to include all translations that may result in contacts by a bounds consistency check: if MaxA/MaxB and MinA/MinB are the maximum/minimum coordinate values along the Z axis for the surface grid cells of the two structures, D_z is initialised to [(MinA-MaxB; MaxB-MinA)]. The same procedure applies to D_y and D_x (lines 3 and 5), but only considering the parts of the structure that can overlap (D_y depends on the value of z, D_x depends on the values of z and y). We shall see in the next sections that these domains can be further pruned by other constraints on the minimum overlap score (section 2.3) and distances between points in the two structures (Section 2.4), so D_z, D_y, and D_x are not necessarily single

intervals but sets of intervals. This pruning (Sections 2.3 and 2.4) occurs at the initialisation of the domains.

For each z and y translation value, D_x is initialized (line 5), and a list generated for the matching sets of core grid segments for the two structures. Grid segment sets are matching when they are aligned by the z, y translation of the B structure, so each entry on this list corresponds to a location in the Z,Y plane and contains the core segments of both structures that are aligned at that location by translating the B structure by the z, y values. Figure 2 shows two such sets, marked L_1 and L_5, which would be respectively the first and fifth entries of the list of matching core grid segments. L_1 contains one core grid segment from A and one from B, L_5 contains two core grid segments from A and one from B.

The BiGGER algorithm then (line 7) imposes bounds consistency on these sets of core grids segments, which requires $O(k^2)$ operations, where k is the number of intervals defined by the core grid segments for each line and for each structure. This reduces the possible translation values, D_x, and affects the generation the surface segments lists to take into account D_x, including only those segments that could overlap given this domain (again, by imposing bounds consistency on the intervals). Finally, the overlap of surface cells is determined for each allowed translation value in D_x. This requires testing the bounds of the matching surface segments in a way similar to imposing bounds consistency, which is of $O(k^2)$ for each line, and then counting the contacts along X, which is of O(N).

The algorithm performs $O(N^2)$ steps by looping through the D_z and D_y (lines 2 and 4), and in each of these steps it loops through the Z,Y plane twice to find the matching core and surface segments (lines 6 and 8) and compare the segment bounds. So each step in the z, y loop is $O(N^2k^2)$, where k is the number of segments per line. Except for fractal structures, k is a small constant. For convex shapes, for example, k is always two or less, and even for complex shapes like proteins k is seldom larger than two. Thus the time complexity of the search algorithm when imposing bounds constraints on the overlap of surface and core grid cells is $O(N^4)$, very close to the $O(N^3Log(N))$ of the FFT method. Furthermore, the comparisons done in the BiGGER algorithm are much faster and this constant factor makes BiGGER more efficient for values of N up to several hundred [5]. Finally, the space complexity of BiGGER is $O(N^2)$, significantly better and with a lower constant factor than the FFT space complexity of $O(N^3)$.

2.3 Restricting the lower bounds on surface contact

Branch and Bound is a common technique that Constraint Programming often uses in optimisation problems, to restrict the domains of the variables to where it is still possible to obtain a better value for the function to optimise. In this case, we wish to optimise the overlap of surface cells, and restrict the search to those regions where this overlap can be higher than that of the lowest ranking model to be kept.

This constraint is applied to the Z and Y coordinate search loops, by counting the total surface cells for each grid as a function of the Z coordinate (that is, the sum over each X, Y plane) and as a function of each Y, Z pair (that is, the sum of each line in the X axis). The determination of the Z translation domain considers the list of total

surface cells for each X,Y plane along the Z axis. For each Z translation value these two lists will align in a different way, as the one structure is displaced in the Z direction relative to the other. The minimum of each pair of aligned values gives the maximum possible surface overlap for that X,Y plane at this Z translation, and the sum of these minima gives the maximum possible surface overlap for this Z translation. Since there are O(N) possible Z translations to test and, for each, O(N) values to compare and add, this step requires $O(N^2)$ operations.

The same applies to restricting the Y translation domain, but taking into account the current value of variable z. This is also an $O(N^2)$ operation identical to the pruning of the Z domain, but must be repeated for each value of the z translation variable, adding a total time complexity of $O(N^3)$ to the algorithm. Since the BiGGER algorithm has a time complexity of $O(N^4)$, these operations do not result in a significant efficiency loss.

By setting a minimum value for the surface contact count, or by setting a fixed number of best models to retain, this constraint allows the algorithm to prune the search space so as to consider only regions where it is possible to find matches good enough to include in the set of models to retain. In general, this pruning results in a modest efficiency gain of up to 30% in medium-sized grids, but with decreasing returns as higher grid sizes lead to thinner surface regions and shift the balance between the total surface counts and the size of the grid [5]. However, this can benefit some applications like soft docking [1], where the surface and core grids are manipulated to model flexibility in the structures to dock, or if the minimum acceptable surface contact is high.

2.4 Constraining the Search Space

In some cases there is information about distances between points in the structures, information that can be used to restrict the search region. If this information is a conjunction of distance limits, then it is trivial to restrict the search to the volumes allowed by all the distances. However, real applications may be more complex.

For modelling protein interactions, it is often the case that one can obtain data on important residues or atoms from such techniques as site directed mutagenesis or NMR titrations, or even from theoretical considerations, but it is rare to be absolutely certain of these data. The most common situation is to have a set of likely distance constraints of which not all necessarily hold. Typically, we would like to impose a constraint of the form:

At least K atoms of set A must be within R of at least one atom of set B (**1**)

where set A is on one protein and set B on the other, and R a distance value. This constraint results in combinatorial problem with a large number of disjunctions, since the distances need only hold for at least one of any combination of K elements of A.

Since the real-space (geometrical) search of BiGGER can be seen as three nested cycles spanning the Z, Y, and X coordinates, from the outer to the inner cycle, we can decompose the enforcement of constraint (1) by projecting it in each of the three directions:

At least K atoms of set A must be within R_ω of at least one atom of set B (**2**)

where R_ω replaces the Euclidean distance R and represents the modulus of coordinate differences on one axis Z, Y or X. R_ω has the same value of R; the different notation is to remind us that this is not a Euclidean distance value, but its projection on one coordinate axis. This makes the constraint slightly less stringent, by considering the distance to be a cube of side 2R instead of a sphere of diameter 2R, but this can be easily corrected by testing each candidate configuration to see if it also respects Euclidean distance.

The propagation algorithm is the same for each axis and consists of two steps. The first step is to determine the neighbourhood of radius R of atoms in group B, projected on the coordinate axis being considered. The next step is to generate a list of segments representing the displacements for which at least K atoms of group A are inside the segments defining the neighbourhood R of the atoms in group B.

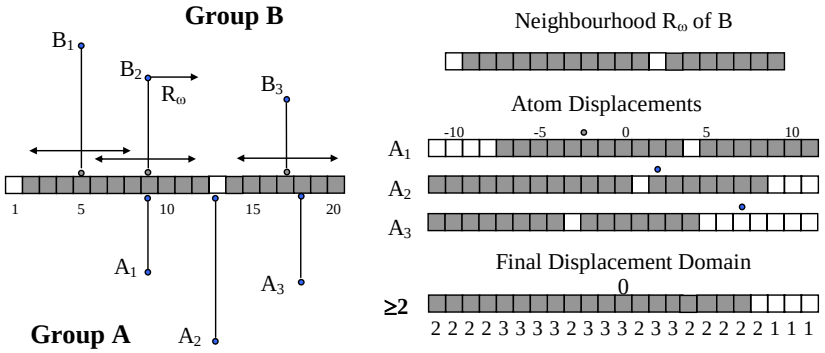


Fig. 3. Generating the displacement domain in one dimension. The left panel shows the generation of the neighbourhood of radius R of group B. The panel on the right shows the allowed displacements for each atom, and the final displacement domain for a K value of 2.

The calculation of the neighbourhood of B in some coordinate (either X, Y or Z) is illustrated in Figure 3. The positions of atoms B_1, B_2 and B_3 in this coordinate are respectively 5, 9 and 17. Their neighbourhoods within a distance 3 are (2;8), (6;12) and (14;20). Merging the two first intervals, the neighbourhood 3 of the atom set B is thus (2;12) and (14;20).

To calculate the displacement values that place an atom of group A inside the neighbourhood of group B we only have to shift the segments defining the neighbourhood of B by the coordinate value of the atom. For example, atom A_1, with coordinate 9, lies inside the neighbourhood 3 of B if its displacement lies in the range (-7;3) or (5;11). Similarly, atoms A_2 and A_3, with coordinate values 13 and 18, respectively may be displaced by (-11;-1) or (1;7) and (-16;-6) or (-4;2).

Once we have the displacement segments for all atoms, we must generate the segments describing the region at least K atoms are in the neighbourhood of B, which is a simple counting procedure (hence, the constraint (2) need not be limited to

specifying a lower bound for the distances to respect. The value of K can also be an upper bound, or a specific value, or even any number of values).
In this case, there are at least two atoms of set A within neighbourhood 3 of atom set B if the displacement lies in ranges (-11;3) and (5;7). In ranges (-7;-6) and (-4;-1) all 3 A atoms are in the neighbourhood 3 of B.

The propagation of constraints of the type (2) produces translation domains that are used to initialise domains D_x, D_y and D_z in the translation search (see section 2.2). Thus the propagation of constraint (2) prunes the domain of the allowed displacements in all 3 axes, in a nested sequence. First the domain along the Z axis is determined and pruned adequately; then, for each remaining value of the displacement along Z, the domain of the displacements along Y is pruned; finally, for each remaining (Z, Y) pair, the constraint is enforced on the displacement along X.

The time complexity of enforcing constraint (2) in one axis is O(a+b+N), where a is the number of atoms in group A and b the number of atoms in group B, and N is the grid size. Since this must be done for the translation dimensions the overall complexity contribution is $O(N^3)$, which does not change the $O(N^4)$ complexity of the geometric search algorithm, and pruning the search space speeds up the search considerably [5].

3. PSICO: modelling protein structure.

There are several sources of information that can help model the structure of a protein. First of all, the amino acid sequences of the protein chains determines most chemical bonds, restricting interatomic distances in many atom pairs, angles formed by atom triplets, of even larger groups of atoms that are effectively rigidily bound together by the chemical bonds. NMR data provides distance constraints by showing that two atoms must be close enough for the Nuclear Overhauser Effect to be felt, limits the angles of rotation around some chemical bonds, or can even suggest limits for relative special orientations of groups of atoms with Residual Dipolar Coupling data. Furthermore, homology with known structures or modelling secondary structure can provide detailed information of the structure of parts of the protein being modelled. We can divide these into three types of constraints: distance constraints between two atoms, group constraints that fix the relative positions of a group of atoms in a rigid configuration, and torsion angle constraints that restrict the relative orientation of two groups joined together by a chemical bond.

3.1 Distance Constraints and Enumeration

The chemical information that is known from the protein sequence provides bond length and bond angle constraints. Bond length constraints are also distance constraints, and the bond angles can be modelled by sets of distance constraints. In fact, the structure and flexibility of an amino acid can be modelled by a conjunction of pairwise distance constraints between all the atoms. To model this information we consider two types of constraints: *In* constraints (eq. 3) and *Out* constraints (eq. 4).

In constraint $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \leq k$ **(3)**

Out constraint $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \geq k$ **(4)**

These two constraint types are used to model all the chemical structural information, whether it is known beforehand or from the NMR spectroscopy experiments.

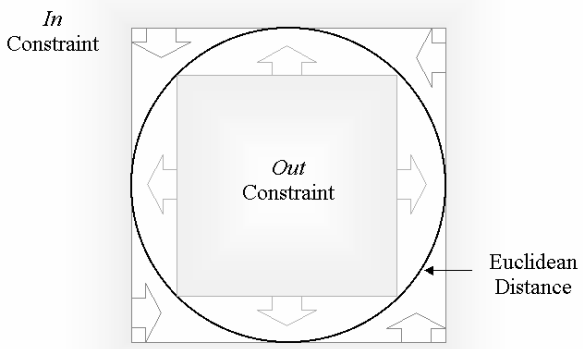


Fig. 4. This figure shows the difference between an Euclidean distance constraint and the simplified constraints used in our model. Note that an *In* constraint region contains the Euclidean distance region, whereas an *Out* constraint must be contained within the Euclidean distance region.

In practice, these Euclidean distance constraints are expensive to propagate, so we use an approximation (eqs. 5 and 5).

In constraint $\max(|x_1 - x_2|, |y_1 - y_2|, |z_1 - z_2|) \leq k$ **(5)**

Out constraint $|x_1 - x_2| \geq \alpha k \vee |y_1 - y_2| \geq \alpha k \vee |z_1 - z_2| \geq \alpha k \quad \alpha = 1/\sqrt{3}$ **(6)**

The parameter α is needed to insure that the simplified *Out* constraint does not exclude regions allowed by the Euclidean distance constraint. This simplification is illustrated in Figure 2.

The variables we wish to determine are the positions of the geometric centres of the atoms, that is, the (x, y, z) coordinates in a single variable with a three dimensional domain, and this domain is represented as a set of cuboid regions. One cuboid defines the *Good* region, which is the volume that contains the possible

positions for the atom. A set of non-overlapping cuboids contained in the *Good* region defines the *NoGoods* region, which contains the positions from which the atom must be excluded (see Figure 5).

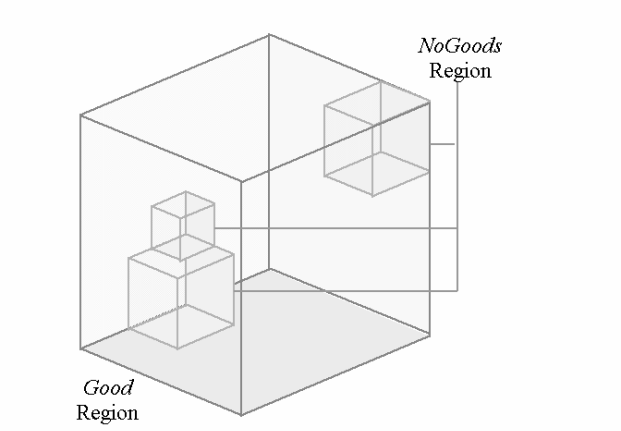


Fig. 5. The domain for the position of an atom is composed of two regions. The *Good* region is a cuboid that defines the positions for the atom that comply with the set of *In* constraints. The *NoGoods* region is a set of non-overlapping cuboids that define the volumes within the *Good* region from which the atom is excluded by the *Out* constraints.

We distinguished between the two types of distance constraints (*In* and *Out*) because of the way in which they are propagated (also see Figure 5).

- The *In* constraints are propagated simply by intersection operations. The *Good* region of atom A will be the intersection of the current *Good* region of A with the neighbourhood of the *Good* region of atom B. This neighbourhood is defined as the *Good* region of B augmented by the distance value of the *In* constraint between A and B. After this operation the *NoGoods* region of A is intersected with the *Good* region to guarantee that it is always contained in the latter. The intersection of two cuboid blocks is very simple to calculate, requiring only Max and Min operations on the extremity coordinates, so propagation of *In* constraints is very efficient.
- For an *Out* constraint the propagation involves adding the exclusion region defined by the constraint to the *NoGoods* region of the affected atom. The most complex operation in this process is insuring that the *NoGoods* region consists of non-overlapping cuboids. This reduces propagation efficiency, but simplifies the task of determining the cases of failure when the *NoGoods* region becomes identical to the *Good* region.

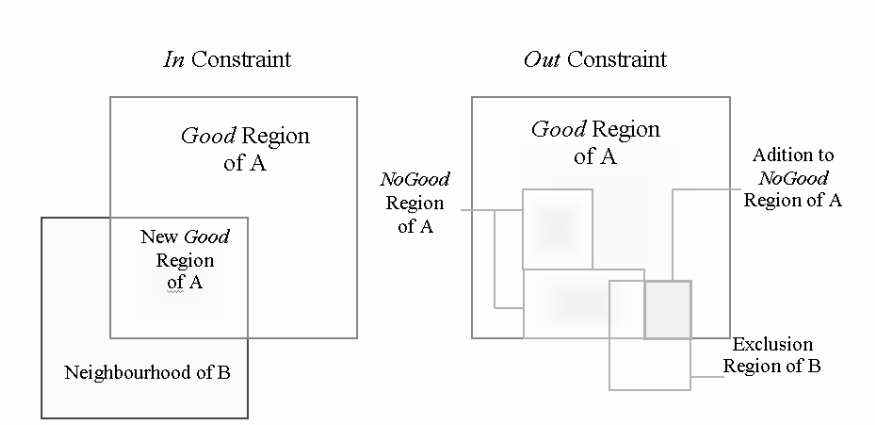


Fig. 6. This figure shows the propagation of both types of constraints. For *In* constraint propagation, the domain of atom A is reduced by intersecting the *Good* region of A with the neighbourhood of B. For *Out* constraint propagation a *NoGood* cuboid region is added. This *NoGood* region is obtained by intersecting the *Good* region of A with the exclusion region of B. Note that part of the new *NoGood* block in A (thick line rectangle) overlaps the original *NoGoods* region, so only the non-overlapping part is added (darker grey shaded area).

Arc-consistency is guaranteed by propagating the constraints on each atom that suffered a domain restriction until no domain changes. After complete propagation, one atom is selected for enumeration, and the propagation step is repeated.

Enumeration intercalates the arc-consistency enforcement following a first fail approach on a round robin system. First, the atom with the smallest domain that was not selected in the current enumeration round is selected for enumeration. Exception is made if the coordinate domain is smaller than 2.0Å for all three coordinates, in which case the atom is considered sufficiently determined and no domain reduction is necessary. The domain of this atom is then split into two similarly sized domains by 'cutting' across the longest coordinate axis (x, y or z) of the domain. The domain of the atom will be one of these two 'halves'.

Enumeration heuristics now come into play. One simple heuristic that was shown to be successful [3,4] was to chose for the new domain that half less occupied by the domains of all other atoms, but additional considerations such as the chemical nature of the amino acid or the prediction of local structures can play a role at this stage to inform the choice of which regions of the domain to eliminate.

Since the domain for the enumerated atom is reduced, constraints are then propagated (as discussed above), and then another atom is selected for enumeration (the atom with the smallest domain not selected yet). This process of selection and domain reduction is repeated until all atoms were selected once, after which a new round of enumeration starts. In case of failure it is possible to backtrack and try different domain reductions, but backtracking is limited both for practical reasons and because it is often the case that the set of constraints is inconsistent due to experimental noise, and in these cases the user needs some structure, even if only partially correct, to help correct the inconsistencies by reassigning the constraints.

3.2 Rigid Group Constraints

The last section outlined the basic framework for PSICO: the domain representations, arc-consistency intercalated with a round-robin enumeration, and limited backtracking. This provides an approximate solution to the problem that can then be refined by local search optimisation. The propagation of rigid group constraints extends this framework to include the information on the configuration of groups of atoms. These can be a prosthetic group or a domain of known structure for which we can know the relative positions of all atoms but which fits within the structure of the protein in an unknown position and orientation. This section explains how these constraints can be propagated in order to reduce the domains of the atoms involved.

Given a fixed orientation, it is trivial to reduce the domains of the atoms in a rigid group. This requires simply that we determine the limits for the translations of the group that do not place any atom outside its domain. Denoting by w_c one of the coordinates of the center of the group (x, y or z), by w_j the same coordinate for atom j, and by w_{max} and w_{min} the upper and lower limits, respectively, for that coordinate of a domain (of atom j or of the center c), such limits are related by the following equations:

$$w_{\max c} = Min_{j=1}^{n}(w_{\max j} + (w_c - w_j)) \quad \textbf{(7a)}$$

$$w_{\min c} = Max_{j=1}^{n}(w_{\min j} + (w_c - w_j)) \quad \textbf{(7b)}$$

Note that the absolute values of w_c and w_j are irrelevant; only the coordinate difference w_c-w_j is important, and is independent of translation.

Equations 7 assume a fixed orientation of the group, but we cannot make that assumption, since the group is free to rotate. Without loss of generality, we shall consider the case of the limits in the x and y coordinates as a function of a rotation around the z axis, centered on the centre point of the group. Hence, the term (w_j-w_c) in equation 1 may actually stand for the x-or y-components of the vector from atom j to the centre of the group, or, in other words, the position of the centre relative to atom j.

This vector is a function of the orientation of the group. Denoting by ψ the rotation around the z axis, by *A* the amplitude of the projection of the vector onto the xy plane (orthogonal to the rotation axis) and by α_j the angle of the vector y_c-y_j at ψ=0, then the terms $w_c - w_j$ for the x and y coordinates are given by

$$x_c - x_j = A_j \cos(\psi + \alpha) = A_j \sin(\psi + \alpha_j + \pi/2) \quad \textbf{(8a)}$$

$$y_c - y_j = A_j \sin(\psi + \alpha) \quad \textbf{(8b)}$$

Figure 7 shows the case for the y-coordinate (the x-coordinate is shifted by 90º).

First, the orientation of the group around the x axis is fixed in an angle we designate χ. Next, the rotation around the y axis is fixed at angle φ. For each (χ;φ) pair, equations 2 can be used to describe the x and y coordinates for each atom as a function of the angle ψ, corresponding to the rotation around the z axis. An equation

similar to equation 2 can also be used to describe the z coordinates of atom j (related to the centre of the group) as a function of the second rotation, φ, around the y axis.

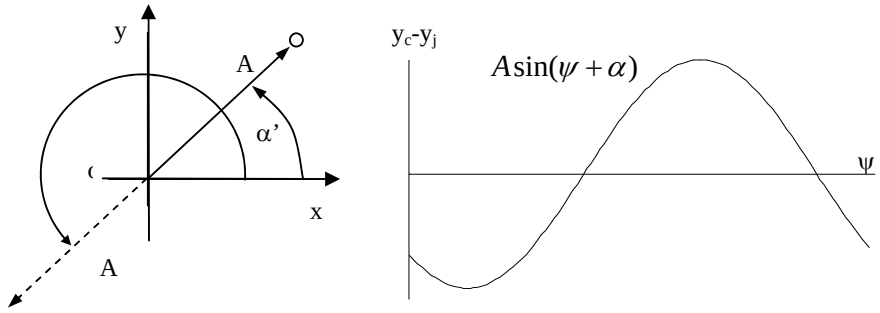


Fig. 7. The position of an atom relative to the centre of the group as a function of the rotation angle ψ can be expressed as a sine function with amplitude A and phase α'. The position of the center relative to the atom is a similar curve, but with the phase shifted by 180º (α), giving the sine wave curve shown on the right.

With no loss of generality, we may replace y_c and y_j in equations 8 with L_c and L_j to denote, respectively, the domain limits of the centre and of atom j in an arbitrary x, y, or z coordinate and by θ an arbitrary rotation angle (φ or ψ), and compute the contribution of each atom to the limits on the translation of the centre of the group by means of Equation 9 below:

$$L_c = A_j \sin(\theta + \alpha_j) + L_j \quad \textbf{(9)}$$

To determine the limits for the placement of the group as a function of the rotation around one axis, considering the rotation around the other axes fixed, we need but intersect the contributions of all atoms to these limits. Now we need to extend this to rotations around all three axes.

Dividing the rotations into finite intervals, each orientation corresponds to an interval of angles, instead of just a single angle, and each coordinate to an interval of values. This way each rotation can be divided into a manageable number of orientations. However, whereas rotating coordinates around an angular value gives a single values for the coordinates, rotating around an interval of angles results in intervals of coordinates. However, as long as we guarantee that the intervals for the angles partition the rotation with a step size that is a sub-multiple of 90º, the sine functions will be monotonous in each interval and the intervals of the corresponding coordinates are trivial to calculate (see reference 3 for more details).

Because of this approach, the previous equations apply not to single coordinate values, but to intervals. However, this extension raises no problems, neither conceptually nor in the implementation.

3.3 Torsion Angle Constraints

In some cases, it is possible for a molecule to change configuration by groups of atoms rotating around a chemical bond. It is this process that allows proteins to fold into their shapes, and the angle of such a rotation is called the torsion angle. Some experimental techniques may provide constraints on torsion angles, and this is useful information when modelling a protein structure.

The propagation of these constraints is an extension to the rigid group constraint propagation discussed in the previous section. We can consider that two rigid groups connected by a bond allowing rotation is a single rigid group if the torsion angle is fixed. If the torsion angle is an interval, we can account for the relative coordinates of all atoms in the two groups by using the corresponding intervals, in a way similar to that discussed in the previous section (also see reference 3).

The extension is thus to add another rotation for every torsion angle to account for, and this procedure allows us to extend the rigid group constraint propagation to any number of rigid groups connected by torsion angles.

While the addition of one torsion angle multiplies the computational cost of propagation by the number of angle intervals to consider, the increase in the size of the rigid group by considering the two connected groups as a single group in each angular step generally decreases the computation cost by restricting the orientations in which the atoms can be placed without violating the limits of their domains, because with larger groups it is harder to respect all domains in any given orientation than it is to do so with smaller groups.

There is a trade off between total group size and number of torsion angles to use, and the right trade off is also a function of the constraints on the torsion angles and the size of the atom domains at the time of propagation, so currently we are researching the best ways to optimize torsion angle constraint propagation taking into account all these factors.

4. Chemera: current applications and future work

Chemera is the interface to all BiGGER and PSICO calculations and, in addition, a set of tools that can be used to visualise the properties of the interaction partners and the models generated for protein complexes and single protein structures. These tools are all integrated in a single visual environment, and include:

Electrostatics. Chemera calculates and displays electrostatic fields using the Poisson-Boltzmann equations, which take into account the ionic strength of the medium (Figure 8-A).

Clustering and Scoring. Chemera can group similar models together according to user-defined thresholds of similarity; calculate new scores based on contacts, average, or minimum distances between groups of atoms; export and import scores and lists of models to and from spreadsheet applications, display groups of models selected according to different scores and display groups of models using simplified representations that show the distribution of many models simultaneously. (Figure 8-B)

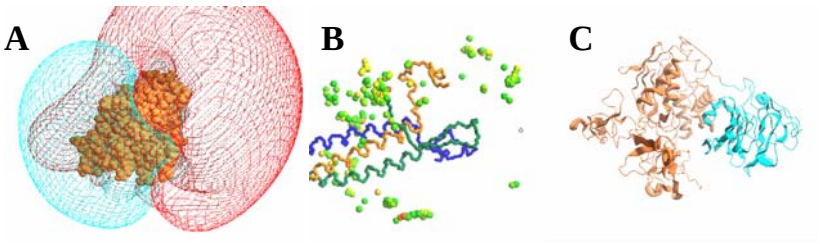


Fig. 8. Visualisation possibilities in Chemera. See text for details.

Web Services. Chemera can also interface with several web services, to assign secondary structure elements, identify domains, display sequence conservation along the protein structure (Figure 8-C). Integration of Web services in Chemera is being developed as part of the European Network of Excellence project REWERSE [20], for the development of Web reasoning and semantics.

Thus constraint programming techniques in Chemera are seamlessly integrated into a general molecular modelling package. This is an important aspect because research and development in this area is very dependent on a close interaction with the end users in the biochemistry community. Past experience [2, 14, 15, 16, 17, 18] and work currently in progress on several protein interactions (*e.g.* Aldehyde Oxidoreductase and Flavodoxin, Ferredoxin NADP Reductase and Ferredoxin, Fibrinogen and Gelatinase A) demonstrate this for the BiGGER docking algorithm, which is currently available in Chemera 3.0 [21]. Our efforts at present involve forming a similar partnership for the testing and application of PSICO to real problems with the aim of future inclusion of this algorithm in a publicly available release of Chemera.

Acknowledgements: We thank CENTRIA and the financial support of the Fundação para a Ciência e Tecnologia (BD 19628/99; BPD 12328/03, PROTEIN project).

References

1. Palma PN, Krippahl L, Wampler JE, Moura, JJG. 2000. BiGGER: A new (soft) docking algorithm for predicting protein interactions. Proteins: Structure, Function, and Genetics 39:372-84.
2. Krippahl L, Moura JJ, Palma PN. 2003. Modeling protein complexes with BiGGER. Proteins: Structure, Function, and Genetics. V. 52(1):19-23.
3. Krippahl, L, Barahona P. Applying Constraint Programming to Rigid Body Protein Docking (2005), Lecture Notes in Computer Science CP 2005: 373-387.
4. Krippahl, L., Barahona, P., PSICO: Solving Protein Structures with Constraint Programming and Optimisation, Constraints 2002, 7, 317-331
5. Krippahl, L., Barahona, P., Applying Constraint Programming to Protein Structure Determination, Principles and Practice of Constraint Programming, Springer Verlag, 1999 289-302

6. Katchalski-Katzir E, Shariv I, Eisenstein M, Friesem AA, Aflalo C, Vakser IA. 1992 Molecular surface recognition: determination of geometric fit between proteins and their ligands by correlation techniques. Proc Natl Acad Sci U S A. 1992 Mar 15;89(6):2195-9.
7. Janin J. 2002. Welcome to CAPRI: A Critical Assessment of PRedicted Interactions. Proteins: Structure, Function, and Genetics Volume 47, Issue 3, 2002. Pages: 257
8. Fernandez-Recio J, Totrov M, Abagyan R 2002. Soft protein-protein docking in internal coordinates. Protein Sci. 2002 Feb;11(2):280-91.
9. Schneidman-Duhovny D, et al. 2003. Taking geometry to its edge: fast unbound rigid (and hinge-bent) docking. Proteins. 2003 Jul 1;52(1):107-12.
10. Gardiner EJ, Willett P, Artymiuk PJ. 2001 Protein docking using a genetic algorithm. Proteins. 2001 Jul 1;44(1):44-56.
11. Ritchie DW, Kemp GJ. 2000 Protein docking using spherical polar Fourier correlations. Proteins. 2000 May 1;39(2):178-94.
12. Dominguez C, Boelens R, Bonvin AM. HADDOCK: a protein-protein docking approach based on biochemical or biophysical information. J Am Chem Soc. 2003 Feb 19;125(7):1731-7.
13. Gabb HA, Jackson RM, Sternberg MJ. Modelling protein docking using shape complementarity, electrostatics and biochemical information. J Mol Biol. 1997 Sep 12;272(1):106-20.
14. Pettigrew GW, Goodhew CF, Cooper A, Nutley M, Jumel K, Harding SE. 2003, The electron transfer complexes of cytochrome c peroxidase from *Paracoccus denitrificans*. Biochemistry. 2003 Feb 25;42(7):2046-55.
15. Pettigrew GW, Prazeres S, Costa C, Palma N, Krippahl L, Moura I, Moura JJ. 1999. The structure of an electron transfer complex containing a cytochrome c and a peroxidase. J Biol Chem. 1999 Apr 16;274(16):11383-9.
16. Pettigrew GW, Pauleta SR, Goodhew CF, Cooper A, Nutley M, Jumel K, Harding SE, Costa C, Krippahl L, Moura I, Moura J. 2003. Electron Transfer Complexes of Cytochrome c Peroxidase from Paracoccus denitrificans Containing More than One Cytochrome. Biochemistry 2003, 42, 11968-81
17. Morelli X, Dolla A., Czjzek M, Palma PN, Blasco, F, Krippahl L, Moura JJ, Guerlesquin F. 2000. Heteronuclear NMR and soft docking: an experimental approach for a structural model of the cytochrome c553-ferredoxin complex. Biochemistry 39:2530-2537.
18. Morelli X, Palma PN, Guerlesquin F, Rigby AC. 2001. A novel approach for assessing macromolecular complexes combining soft-docking calculations with NMR data. Protein Sci. 10:2131-2137.
19. Krippahl L, Brarahona P. Applying Constraint Programming to Rigid Body Protein Docking. Submitted to Principles and Practice of Constraint Programming, Springer Verlag, 2005
20. http://rewerse.net/
21. http://www.cqfb.fct.unl.pt/bioin/chemera/

Exploiting Model Checking in Constraint-based Approaches to the Protein Folding Problem ⋆

Elisabetta De Maria, Agostino Dovier, Angelo Montanari, and Carla Piazza

Dipartimento di Matematica e Informatica, Università di Udine
via delle Scienze 206, 33100 Udine, Italy
{demaria,dovier,montana,piazza}@dimi.uniud.it

Abstract. In this paper we show how model checking can be used to drive the solution search in the protein folding problem encoded as a constraint optimization problem. The application of the model checking technique allows us to distinguish between meaningful protein conformations and bad ones. This classification of conformations can then be exploited by constraint solvers to significatively prune the search space of the protein folding problem. Furthermore, our approach seems promising in the study of folding/energy landscapes of proteins.

1 Introduction

In this paper we show how model checking can be used to drive the solution search in the protein folding problem encoded as a constraint optimization problem. Given the molecular composition of a protein, i.e., a list of amino acids, known as its *primary structure*, the *protein structure prediction* (or *protein folding*) problem consists in determining the 3D shape (*tertiary structure* or *conformation*) that the protein assumes in normal conditions in biological environments [5].

To solve the protein folding problem it is crucial to determine the conformations of the amino acid sequences in the 3D space with minimum energy. It is indeed widely accepted that a state with minimum energy represents the protein's natural shape (a.k.a. the *native conformation*). The energy of a conformation can be modeled by means of suitable *energy functions*, which express the energy level in terms of the interactions between pairs of amino acids [3]. Since the protein folding problem is extremely complex, it is often simplified in several respects. A common simplification consists in using *lattice space models* to restrict the admissible positions of the amino acids in the space [11]. The energy function can be simplified as well, e.g., by adopting the 20×20 potential matrix proposed by [7,8] or the simpler HP model [1,2]. For the sake of simplicity, in the following we assume a 2D finite lattice included in $\mathbb{N}^2$ and the HP energy model. However, our approach can be easily extended to 3D lattices. Furthermore, it is not difficult to replace the HP model with a more refined energy model that keeps track of the variety of interactions among the 20 kinds of amino acids.

In this work we show how model checking techniques can be exploited to investigate the relationships among the different possible conformations of proteins. We model

⋆ This work has been partially supported by PRIN 2005 project 2005015491 and by FIRB 2003 project RBNE03B8KK.

the solution space of the protein folding problem as a finite transition system whose states are all the possible conformations of a protein and whose transitions represent admissible transformations of conformations. Then, we take advantage of temporal logic to specify and check relevant properties of such a system. As an example, we show how to check whether there exists a path from a given conformation to a conformation with an energy level below a certain threshold whose length is less than or equal to a given value. In particular, we are interested in identifying patterns common to different proteins. These patterns can be used to improve the solution search in existing constraint-based protein folding algorithms as well as to understand protein functions. In general, constraints allow one to easily model minimization problems. Once the constraint model is defined, a constraint solver can indeed be used to search for solutions. This search exploits the constraints to prune the solution space. In the following, we show how model checking can be used to identify meaningful properties of protein conformations that can be encoded as additional constraints to be used to further reduce the solution space.

The paper is organized as follows. In Section 2 we introduce the HP model. In Section 3 we describe how to generate, for any given protein, the corresponding finite transition system. In Section 4 we show how to express relevant properties of protein conformations in temporal logic. In Section 5 we report preliminary experimental results and we outline some ongoing developments of the work.

2 The HP model of proteins

The HP model on a 2D discrete lattice, where every conformation of a protein is a self-avoiding walk in $\mathbb{Z}^2$, is commonly used to represent the conformations and the energy function of proteins [12]. Such a model reduces the 20-letter alphabet of amino acids to a two-letter alphabet $\{H,P\}$, where H (resp., P) represents a hydrophobic (resp., polar) amino acid. The energy function states that the energy contribution of a *contact* between two amino acids is -1 if both of them are H amino acids, 0 otherwise.

Hereafter, we represent an HP sequence as an element in $\{0,1\}^*$, where 1 (resp., 0) stands for an H (resp., P) amino acid. Furthermore, for $i = 0,1,\ldots,n$, we denote by s_i the i-th element of a sequence s of $n+1$ elements. The subset of admissible protein conformations is defined as follows.

Definition 1 (Folding). *A* folding ω *of a sequence* $s = s_0 \ldots s_n$ *is a function* $\omega : [0 \ldots n] \to \mathbb{Z}^2$ *such that*

(i) $\forall 0 \leq i < n\,(|\omega(i) - \omega(i+1)| = 1)$, *that is, if* $\omega(i) = (X_i, Y_i)$ *and* $\omega(i+1) = (X_{i+1}, Y_{i+1})$, *then* $|X_i - X_{i+1}| + |Y_i - Y_{i+1}| = 1$;
(ii) $\forall i \neq j(\omega(i) \neq \omega(j))$ *(*ω *is self avoiding).*

We say that two amino acids s_i and s_j of a given folding ω are *connected neighbors* if $j = i \pm 1$ and that they are *topological neighbors* if they are not connected and $|\omega(i) - \omega(j)| = 1$.
In the HP model, the energy of a folding is given by the opposite of the number of topological HH neighbors, e.g., if there exist k topological HH neighbors in ω, then the energy of ω is $-k$.

Definition 2 (Folding Energy). *Given a sequence $s = s_0 \dots s_n$, the energy of a folding of s is:*

$$E = \sum_{1 \le i+1 < j \le n} B_{i,j} \cdot \delta(s_i, s_j)$$

where $B_{i,j}$ is equal to -1 whenever both s_i and s_j are H amino acids, 0 otherwise, and $\delta(s_i, s_j)$ is 1 if s_i and s_j are topological neighbors, 0 otherwise.

Hence, a folding has minimum energy if it maximizes the number of HH contacts.
Given a sequence $s = s_0 \dots s_n$, we assume its length to be n, i.e., it is equal to the number of "segments" it is made of. To represent the conformations of a sequence of length n, we use the subset $\angle = \{(i, j) : i \in [0, 2n], j \in [0, 2n]\}$ of $\mathbb{N}^2$.
Without loss of generality, we assume $\omega(0) = (n, n)$ and, in order to avoid simple symmetries, we fix $\omega(1) = (n, n+1)$.
Notice that, once the coordinates of a segment have been fixed, the next segment in the sequence can only assume three possible directions with respect to the preceding one: left (l), forward (f), and right (r). As a result, a folding of a sequence of length n can be represented as a string of length $n-1$ on the alphabet $\{l, f, r\}$ [1]. As an example, the sequence of Figure 1 is represented by the string *rllf*.

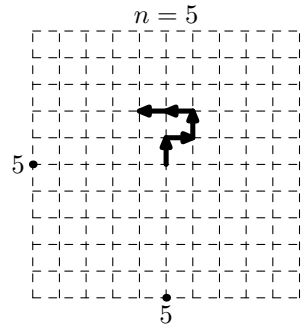


Fig. 1. String *rllf* on 10×10 lattice.

The number of all possible foldings of a sequence of length n, where the orientation of the first segment is fixed as above, is bounded by 3^{n-1}. It is commonly accepted that the number C_n of self-avoiding walks of length n grows according to the following formula $C_n = B \cdot \mu^n \cdot n^{\gamma-1}$, where $B \sim 1.93$, $\mu \sim 2.63$, and $\gamma = 43/32$ [13], and thus the number of self-avoiding walks of length n, where the orientation of the first segment is fixed, is $D_n = C_n/4$. In [15] Ngo and Marks have shown that protein folding problem on 2D-lattices is NP-complete.

Now we formally define the set of valid transformations among foldings. Roughly speaking, a valid transformation of a given folding f consists in selecting at random a

[1] To avoid symmetries it is possible to consider only strings with prefixes of the form f^*r.

position in f and performing a rotation of the part of f between this position and the ending position (*pivot move*).

Definition 3 (Pivot move). *Let $f = f_2 \dots f_n$, with $f_i \in \{l, f, r\}$ for all $2 \leq i \leq n$, be a folding of a sequence s of length n. A folding f' of s is obtained from f through a* pivot move *with pivot $k-1$, with $2 \leq k \leq n$, if $f'_i = f_i$ for all $i \neq k$ and $f'_k \neq f_k$.*

Given a folding of a sequence of length n, since the number of possible pivots is $n-1$ and each one may give rise to two moves, i.e., rotations, the number of successor foldings is at most $2(n-1)$ (some of these conformations could violate the self avoiding condition). As an example, consider the sequence of length 4 whose folding is represented by the string *ffl*. The foldings obtained by pivot moves are the 6 foldings *lfl*, *rfl*, *fll*, *frl*, *fff*, *ffr*. They are graphically depicted in Figure 2. It is possible to show that pivot moves are ergodic, namely, they cover the entire folding space [5].

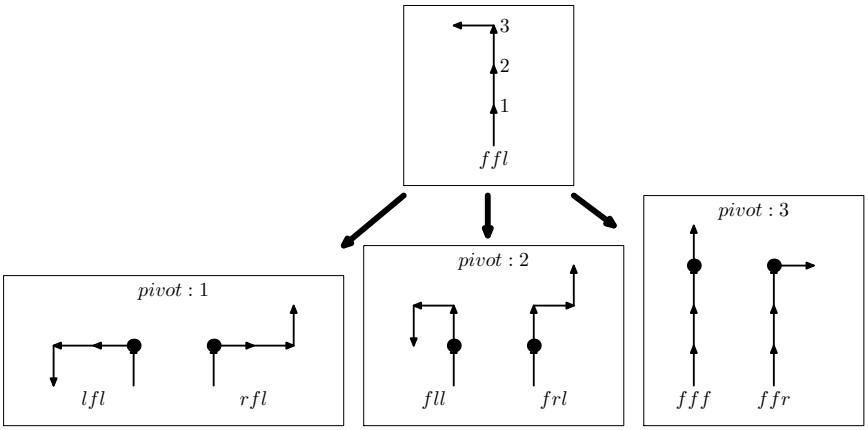


Fig. 2. Pivot moves from string *ffl*.

3 Protein transition systems

In this section we propose an approach to the formal verification of interesting protein conformation properties based on *model checking* [4]. Model checking allows one to verify desirable properties of a system by an exhaustive enumeration of all the states reachable by the system. We model the set of protein foldings and their relationships as a finite transition system and we use (linear or branching) propositional temporal logic to specify relevant system properties [9,17].

Definition 4 (Transition System). *Let AP be a set of atomic propositions. A* transition system *over AP is a tuple $M = (Q, T, L)$, where*

- *Q is a finite set of states;*

– *$T \subseteq Q \times Q$ is a total transition relation, that is, for every state $q \in Q$ there is a state $q' \in Q$ such that $T(q,q')$;*
– *$L : Q \to 2^{AP}$ is a labeling function that maps every state into the set of atomic propositions that hold at it.*

The 2D Protein Transition System is defined as follows:

Definition 5 (2D Protein Transition System). *The 2D Protein Transition System of a string P of length n over $\{0,1\}$ is a tuple $M_P = (Q,T,L)$, where*

– *Q is the set of all foldings of length n on the $2n \times 2n$ 2D lattice;*
– *$T \subseteq Q \times Q$ contains the pairs of states (q_1,q_2) such that q_2 can be obtained from q_1 by a pivot move;*
– *$L : Q \to 2^{AP}$ is a labeling function over the set AP of atomic propositions which consists of the following 3(n-1) predicates*
$$2nd\_l, \ldots, nth\_l, \quad 2nd\_f, \ldots, nth\_f, \quad 2nd\_r, \ldots, nth\_r,$$
plus the following three predicates
$$min\_en, \quad inter\_en, \quad max\_en,$$
where for all $2 \leq i \leq n$, the predicate ith_l (resp., ith_f, ith_r) holds at a state q if the i-th segment of q has a left *(resp.,* forward, right*) orientation and min_en (resp., inter_en, max_en) holds at a state q if the energy of q is minimum (resp., intermediate, 0).*

It is possible to prove that the 2D Protein Transition System corresponding to a given protein has the following properties.

Proposition 1 (Properties of the 2D Protein Transition System).

1. *It is* strongly connected*, i.e., for each pair of states q_1 and q_2, there is a path from q_1 to q_2.*
2. *It is* symmetric*, i.e., for each pair of states q_1 and q_2, if (q_1,q_2) belongs to T, then (q_2,q_1) belongs to T.*
3. *The* maximum incidence degree *D*=$\max_{q \in Q} |\{(q,q') : (q,q') \in T\}|$ *is 2(n-1).*

Item 1 of Proposition 1 holds since pivot moves are ergodic [5]. Item 2 of Proposition 1 holds because, if state q_2 can be obtained from state q_1 performing a pivot move, then q_1 can be obtained from q_2 performing the opposite move. Item 3 immediately follows from Definition 3.

As far as the energy of a protein is concerned, from our experimental results it turns out that the majority of states has a high energy and that only a few states have minimum energy. Furthermore, the value of the energy difference between the source and destination nodes of most edges is 0.

4 Model checking properties of proteins

Temporal logics are formalisms for describing sequences of transitions between states. We restrict our attention to two well-known fragments of the *computation tree logic*

CTL*, namely, the *branching time* logic CTL and the *linear time* logic LTL [9]. CTL* formulae describe properties of computation trees and they are obtained by (repeatedly) applying Boolean connectives, *path quantifiers*, and *state quantifiers* to atomic formulae. The path quantifier **A** (resp., **E**) can be used to state that all paths (resp., some path) starting from a given state have some property. The state quantifiers are the next time operator **X**, which can be used to impose that a property holds at the next state of a path, the operator **F** (sometimes in the future), that requires that a property holds at some state on the path, the operator **G** (always in the future), that specifies that a property is true at every state on the path, and the until binary operator **U**, which holds if there is a state on the path where the second of its argument properties holds and, at every preceding state on the path, the first of its two argument properties holds.

CTL allows one to quantify over the paths starting from a given state. Unlike CTL*, it constrains every state quantifier to be immediately preceded by a path quantifier. In LTL one may only describe events along a single computation path. Its formulae are of the form $\mathbf{A}f$, where f does not contain path quantifiers, but it allows the nesting of state quantifiers. CTL and LTL have different expressive powers [9]. We chose to use both of them to benefit from their advantages. On the one hand, the complexity of model checking for CTL is linear in the number of states and edges of the transition system, while the model checking problem for LTL is PSPACE-complete. Furthermore, there are many tools for checking if finite state systems satisfy CTL formulae (see, e.g., SMV [14]). On the other hand, algorithms for on-the-fly model checking, a technique that allows one to contrast the state explosion problem trying not to build the entire transition system, mainly deals with LTL formulae. As a matter of fact, all the relevant properties of Protein Transition Systems we identified belong to the intersection of CTL and LTL.

Given a 2D Protein Transition System $M_P = (Q, T, L)$ and a temporal logic formula f expressing some desirable property of the system, the *model checking problem* consists in finding the set of all states in Q satisfying f:

$$[\![f]\!] = \{q \in Q : M_P, q \models f\}.$$

When a state does not satisfy a formula, model checking algorithms produce a counterexample that falsifies it, thus providing an insight to understand failure causes and important clues for fixing the problem.

We conclude the section by showing how meaningful properties of 2D Protein Transition Systems can be encoded in both CTL and LTL.

F1: Does it exist a path of length at most k that reaches a state with minimum energy?
CTL:$min\_en \vee EX\,min\_en \vee \cdots \vee \underbrace{EX \ldots EX}_{k} min\_en \equiv$

$$\bigvee_{i=0}^{k} E_1 X_1 \ldots E_i X_i min\_en.$$

LTL: $A(\neg min\_en \wedge X \neg min\_en \wedge XX \neg min\_en \wedge \cdots \wedge \underbrace{X \ldots X}_{k} \neg min\_en) \equiv$

$$A(\bigwedge_{i=0}^{k} X_1 \ldots X_i \neg min\_en).$$

Notice that the property expressed in LTL actually is the negation of property F1. However, it is sufficient to complement the set of states that satisfy this property to obtain the set of states satisfying F1.

F2: Is energy the minimum one? Alternatively, if energy is the maximum one, is it possible to reach a state with minimum energy without passing through states with intermediate energy?
CTL, LTL: $A(max\_en\ U\ min\_en)$.

F3: Is it possible to reach in one step a folding where the first half of the sequence is a helix of the form $rrllrr\ldots$?
Here we must distinguish between the case in which $m = \lfloor n/2 \rfloor$ is even and that in which it is odd.
If m is odd, we have:
CTL: $EX(\bigwedge_{i=2,i=2+4\cdot j,j\geq 0}^{m-1}(ith\_r \wedge i+1th\_r) \wedge \bigwedge_{i=4,i=4+4\cdot j,j\geq 0}^{m-1}(ith\_l \wedge i+1th\_l))$.

LTL: $AX(\bigvee_{i=2,i=2+4\cdot j,j\geq 0}^{m-1}(\neg ith\_r \vee \neg i+1th\_r) \vee$
$\bigvee_{i=4,i=4+4\cdot j,j\geq 0}^{m-1}(\neg ith\_l \vee \neg i+1th\_l))$.

If $m = 2+4\cdot j, j \geq 0$, we have:
CTL: $EX(\bigwedge_{i=2,i=2+4\cdot j,j\geq 0}^{m-1}(ith\_r \wedge i+1th\_r) \wedge \bigwedge_{i=4,i=4+4\cdot j,j\geq 0}^{m-1}(ith\_l \wedge i+1th\_l) \wedge mth\_r)$.

LTL: $AX(\bigvee_{i=2,i=2+4\cdot j,j\geq 0}^{m-1}(\neg ith\_r \vee \neg i+1th\_r) \vee$
$\bigvee_{i=4,i=4+4\cdot j,j\geq 0}^{m-1}(\neg ith\_l \vee \neg i+1th\_l) \vee \neg mth\_r)$.

If $m = 4+4\cdot j, j \geq 0$, we have:
CTL: $EX(\bigwedge_{i=2,i=2+4\cdot j,j\geq 0}^{m-1}(ith\_r \wedge i+1th\_r) \wedge \bigwedge_{i=4,i=4+4\cdot j,j\geq 0}^{m-1}(ith\_l \wedge i+1th\_l) \wedge mth\_l)$.

LTL: $AX(\bigvee_{i=2,i=2+4\cdot j,j\geq 0}^{m-1}(\neg ith\_r \vee \neg i+1th\_r) \vee$
$\bigvee_{i=4,i=4+4\cdot j,j\geq 0}^{m-1}(\neg ith\_l \vee \neg i+1th\_l) \vee \neg mth\_l)$.

F4: Is it true that every state which is at most k steps far from the current one has maximum energy, i.e., energy equal to 0?
CTL: $max\_en \wedge AX max\_en \wedge \cdots \wedge \underbrace{AX\ldots AX}_{k} max\_en \equiv$

$$\bigwedge_{i=0}^{k} A_1X_1\ldots A_iX_i max\_en.$$

LTL: $A(max\_en \wedge X max\_en \wedge \cdots \wedge \underbrace{X\ldots X}_{k} max\_en) \equiv$

$$A(\bigwedge_{i=0}^{k} X_1\ldots X_i max\_en).$$

In the next section we report the outcomes of some experiments where we model checked these (and other) properties on proteins of small dimension.

5 Experimental results and future developments

We implemented the proposed approach to the verification of properties of foldings in SICStus Prolog and we experimented it on some simple test cases. More precisely, we developed an algorithm for encoding 2D Protein Transition Systems, and then we implemented model checking algorithms to verify whether some specific 2D Protein Transition Systems satisfy or not a set of relevant properties, including F1-F4. We confined ourselves to test cases where protein length was at most 10. As for $F1$, for instance, we searched for states with energy equal to 0 that satisfy property $F1$ when $k = 1$, i.e., states with maximum energy that reach in one step a state with minimum energy. For n=8, given the string 111111111, where $min\_en = -4$, it came out that only 8 states fulfil the request. They are (every state is followed by the state testifying the satisfiabily of the property): $lrflflf \rightarrow llflflf$, $lfflflf \rightarrow llflflf$, $rlfrfrf \rightarrow rrfrfrf$, $rffrfrf \rightarrow rrfrfrf$, $flflfrl \rightarrow flflfll$, $flflffl \rightarrow flflfll$, $frfrflr \rightarrow frfrfrr$, and $frfrffr \rightarrow frfrfrr$. Similar experiments were performed in the cases of properties F2-F4. We used our tool to model check a number of other meaningful properties. As an example, we used it to check whether there exist states with an energy different from the minimum one that may reach in one step a state with a greater energy which, in its turn, may reach in a few steps (how many, it depends on the length of the protein) a state with minimum energy. The answer is positive. For example, for n=7, given the string 11111111, where $min\_en = -3$, the following state satisfies the property (the entire witness path is reported and every state is followed by its energy): $lrlfll(-2) \rightarrow lrlffl(0) \rightarrow lrllfl(-3)$. The existence of such paths shows that, in order to decrease the number of edges of the 2D Protein Transition System, it is not sound to cut edges connecting states where the source energy is lower than the destination energy because from the destination state we could rapidly reach states with minimum energy.

As for the future developments of our work, one of the main issues of model checking is the state explosion problem. In our case, a protein of length n gives rise to a transition system where the number of states is $\Theta(3^{n-1})$. This leads to both time and space problems. On-the-fly model checking [6,10] has been proposed to cope with the state explosion problem. This approach in many cases avoids the construction of the entire state space of the system, because the property to test guides the construction of the system. When a state falsifying the property under analysis is reached, the construction is stopped. Only in the worst case (when the property is satisfied) the entire system must be built. Exploiting on-the-fly model checking, we plan to apply our approach to proteins with a significant length.
Another technique proposed to control the state-explosion problem is symbolic model checking [14,4]. Symbolic model checking is based on the use of Ordered Binary Decision Diagrams (OBDDs) to compactly represent transition systems. In the worst case, the OBDD and the represented system have the same size. However, this is usually not the case when the transition system has some "regularities". We intend to study what happens if we use OBDDs to represent 2D Protein Transition Systems and, if possible, to exploit symbolic model checking techniques.
Finally, we plan to extend our approach to 3D-lattices and to switch to an energy model

that considers all the 20 kinds of aminoacids. In this context we intend to analyse the usefulness of our approach not only for the protein folding problem, but more in general for the study of folding/energy landscapes of proteins.

References

1. R.Backofen and S.Will. Excluding symmetries in constraint-based search. *Constraints*, 7(3):333–349, 2002.
2. R.Backofen and S.Will. A Constraint-Based Approach to Structure Prediction for Simplified Protein Models that Outperforms Other Existing Methods. *Proc. of ICLP 2003*, pp. 49-71, Springer Verlag, 2003.
3. M.Berrera, H.Molinari, and F.Fogolari. Amino acid empirical contact energy definitions for fold recognition in the space of contact maps. *BMC Bioinformatics*, 4(8), 2003.
4. E.M.Clarke, O.Grumberg, and D.A.Peled. *Model Checking*. The MIT Press, 1999.
5. P.Clote and R.Backofen. *Computational Molecular Biology*. John Wiley & Sons, 2001.
6. C.Courcoubetis, M.Y.Vardi, P.Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties *Formal Methods in System Design*, 1:275-288, 1992.
7. A.Dal Palù, A.Dovier and F.Fogolari. Constraint logic programming approach to protein structure prediction. *BMC Bioinformatics*, 5(186), 2004.
8. A.Dal Palù, A.Dovier and E. Pontelli. A Constraint Logic Programming Approach to 3D Structure Determination of Large Protein Complexes. *Proc. of LPAR'05*, pp. 48-63, 2005.
9. E.A. Emerson *Temporal and modal logic*. In Handbook of Theoretical Computer Science, Volume B (chapter 16), J. van Leeuwen ed., Elsevier Science Publisher, 1990.
10. J.C.Fernandez, C.Jard, T.Jeron, and G.Viho. Using on-the-fly verification techniques for the generation of test suites. In *Proceedings of the 1996 Workshop on Computer-Aided Verification*, LNCS 1102:348-359, 1996.
11. A.Kolinski and J.Skolnick. Reduced models of proteins and their applications. *Polymer*, 45:511–524, 2004.
12. K.F.Lau and K.A.Dill. A lattice statistical mechanics model of the conformational and sequence spaces of proteins. *Macromolecules*, 22:3986-3997, 1989.
13. N.Madras and G.Slade. The self avoiding walk. (Boston: Birkhäuser), 1993.
14. K.L.McMillan. Symbolic Model Checking: An Approach to the State Explosion Problem. *Kluwer Academic*, 1993.
15. J.T.Ngo and J.Marks. Computational complexity of a problem in molecular structure prediction. *Protein Engineering* 5:313-321, 1992.
16. A.Ponitz and P.Tittmann. Improved upper bounds for self-avoiding walks in Z^d. *Electronic J. Comb. 7*, 2000.
17. J.P. Quielle and J.Sifakis. Specification and verification of concurrent systems in CESAR. In *Logics and Models of Concurrent Systems, NATO ASI 13*. Springer, 1984.

Global Constraints for Discrete Lattices

Alessandro Dal Palù[1], Agostino Dovier[2], and Enrico Pontelli[3]

[1] Dip. Matematica, Univ. di Parma, alessandro.dalpalu@unipr.it
[2] Dip. Informatica, Univ. di Udine, dovier@dimi.uniud.it
[3] Dept. Computer Science, New Mexico State University, epontell@cs.nmsu.edu

Abstract. Constraint solving on discrete lattices has gained momentum as a declarative and effective approach to solve complex problems such as protein folding determination. In particular, [8] presented a comprehensive constraint solving platform (COLA) dealing with primitive constraints in discrete lattices. The purpose of this paper is to discuss some preliminary ideas on possible global contraints that can be introduced in a constraint system like COLA. The paper discusses various alternatives and provides preliminary results concerning the computational properties of the different global constraints.

1 Introduction

Discrete finite lattices are often used for approximated studies of 3D conformations of molecular structures. These models are used, in particular, to compute reasonable approximations of foldings of protein structures in 3D space [19,12,1]. Polymers are laid out in particular subsets of $\mathbb{N}^3$. These subsets are often described by the vectors that specify the set of neighbors of each point. Lattice models like FCC and chess knight are among them.

The protein folding problem in the context of discrete lattice structures has been studied as a *Constraint Optimization Problem* in the FCC lattice, using a simplified energy model in [2] and with a more precise energy model in [7]. In these approaches, each point P of the lattice is identified by a triplet of *finite domain variables* (P_x, P_y, P_z), where each variable separately describes a coordinate of the point. Following this approach, propagation algorithms operate on points by means of projection of domain information on individual coordinates. Considering each coordinate separately limits the power of *propagation* among points as single objects, rather than as triples of FD variables. In [8] we proposed the constraint solver called *COLA (COnstraint solver on LAttices)*, whose primitive domain considers lattice points as atomic values.

In this paper, we propose a study targeting the problem of dealing with global constraints in the general context of constraint solvers on lattice domain—and specifically in COLA. Global constraints are proven constructs that facilitate the declarative encoding of problems; at the same time, they allow the programmer to express knowledge about relationships between variables, that can be effectively employed by the search algorithm to prune infeasible parts of the solution search space. We introduce different global constraints, and we study the complexity of their satisfiability and of the associated propagation process.

We hope this paper will inspire further interest in this problem and promote discussion about suitable global constraints for discrete lattice structures and efficient implementation techniques.

2 Lattices and COLA

A *discrete lattice* (or, simply, a *lattice*) is a graph (P,E), where P is a set of triples $(x,y,z) \in \mathbb{N}^3$, connected by undirected edges (E). Given $A = (x,y,z) \in P$, we will denote x,y,z with A_x, A_y, A_z respectively.

Lattices contain strong symmetries and present regular patterns repeated in the space. If all nodes have the same degree δ, then the lattice is said to be δ-*connected*. Three examples of lattices are described next (and depicted in Fig. 1).

Definition 1. *A* cubic lattice (P,E) *is defined by the following properties:*

- $P = \{(x,y,z) \mid x,y,z \in \mathbb{N}\}$*;*
- $E = \{(A,B) \mid A,B \in P, sqeucl(A,B) = 1\}$.

where $sqeucl(A,B) = (B_x - A_x)^2 + (B_y - A_y)^2 + (B_z - A_z)^2$.

The cubic lattice is 6-connected—see Fig. 1(a).

Definition 2. *An* `FCC` *lattice* (P,E) *is defined by the sets:*

- $P = \{(x,y,z) \mid x,y,z \in \mathbb{N} \wedge x+y+z \text{ is even}\}$*;*
- $E = \{(A,B) \mid A,B \in P, sqeucl(A,B) = 2\}$.

Thus, in an `FCC` lattice we consider the 3D space organized in cubes, each side having length 2, and where the center point of each face is also admitted. The practical rule to compute the points belonging to the lattice is to check whether the sum of the point's coordinates (x,y,z) is even. Pairs of points at Euclidean distance $\sqrt{2}$ are linked and form the edges of the lattice; their distance is called *lattice unit*. Observe that, for lattice units, it holds that $|x_i - x_j| + |y_i - y_j| + |z_i - z_j| = 2$. The `FCC` lattice is 12-connected—see Fig. 1(b).

Definition 3. *A* chess knight *lattice is defined as follows:*

- $P = \{(x,y,z) \mid x,y,z \in \mathbb{N}\}$*;*
- $E = \{(A,B) \mid A,B \in P, sqeucl(A,B) = 5\}$.

Each edge allows a move like a knight on a chessboard, i.e., 2 units in one direction, 1 in another direction, 0 in the third direction. The chess knight lattice is 24-connected—see Fig. 1(c).

In COLA, a *domain* D is described by a pair of lattice points $\langle low(D), up(D)\rangle$. The domain D defines a set of lattice points in the 3D *box* identified by the two opposite vertices $low(D)$ and $up(D)$—i.e.,

$$Box(D) = \left\{ (x,y,z) \in P \;\middle|\; \begin{array}{l} low(D)_x \le x \le up(D)_x, low(D)_y \le y \le up(D)_y, \\ low(D)_z \le z \le up(D)_z \end{array} \right\}$$

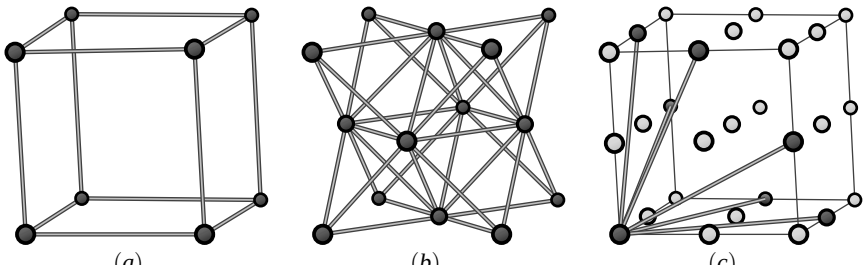


Fig. 1. Basic Component of a Cubic, FCC, and Chess Knight Lattices

COLA handles the domain operations of *intersection*, *union*, and *dilation*, described in [8]. In modeling a constraint satisfaction problem, each variable represents an entity to be placed in a point in the lattice space. The variable V is associated to a *domain* $D^V = \langle low(D^V), up(D^V) \rangle$.

Let V_1, V_2 denote two lattice variables, let $B_1 = Box(D^{V_1})$ and $B_2 = Box(D^{V_2})$, let $d \in \mathbb{N}$, and let P_1, P_2 be two lattice points. The following constraints are admitted in COLA:

$$\begin{aligned}
\texttt{DIST_LEQ}(V_1, V_2, d) &\Leftrightarrow \exists P_1 \in B_1, \exists P_2 \in B_2 \text{ s.t. } norm_\infty(P_1, P_2) \leq d \\
\texttt{EUCL}(V_1, V_2, d) &\Leftrightarrow \exists P_1 \in B_1, \exists P_2 \in B_2 \text{ s.t. } sqeucl(P_1, P_2) = d \\
\texttt{EUCL_LEQ}(V_1, V_2, d) &\Leftrightarrow \exists P_1 \in B_1, \exists P_2 \in B_2 \text{ s.t. } sqeucl(P_1, P_2) \leq d \\
\texttt{EUCL_GEQ}(V_1, V_2, d) &\Leftrightarrow \exists P_1 \in B_1, \exists P_2 \in B_2 \text{ s.t. } sqeucl(P_1, P_2) \geq d
\end{aligned}$$

where $norm_\infty(A, B) = \max\{|B_x - A_x|, |B_y - A_y|, |B_z - A_z|\}$.

The work presented in [8] describes the implementation of these concepts in a concrete constraint solving system, capable of performing *bounds consistency* on the previously described constraints. The COLA solver has been applied to the problem of solving the protein folding problem in the FCC lattice [8], producing interesting results for proteins of length up to 100.

3 Global constraints

The main contribution of this paper is the identification of which *global constraints* should be introduced in COLA to enhance its declarative nature and facilitate the efficient resolution of complex problems. In particular, we expect our design to be general and applicable to other constraint solvers on lattice domains. In order to be able to perform forms of consistency which are more accurate than bounds consistency, we assume that the finite domain associated to each variable is a finite set of lattice points, instead of a simple box, as in COLA.

Intuitively, a global constraint is a non-binary constraint. More formally, given n variables $X_1, \ldots, X_n$, respectively having domains $D^{X_1}, \ldots, D^{X_n}$, a *global constraint* C on the variables $X_1, \ldots, X_n$ can be defined as a subset $C \subseteq D^{X_1} \times \cdots \times D^{X_n}$.

For each global constraint C, we are interested in verifying two properties [4]:

– *consistency (CON):* $C \neq \emptyset$
– *generalized arc consistency (GAC):* $\forall i \in \{1,\ldots,n\}\, \forall a_i \in D^{X_i}$

$$\exists a_1 \in D^{X_1} \cdots \exists a_{i-1} \in D^{X_{i-1}} \exists a_{i+1} \in D^{X_{i+1}} \cdots \exists a_n \in D^{X_n}\ (a_1,\ldots,a_n) \in C$$

In the specific case where the constraint C is binary, i.e., it involves only two variables X_1, X_2, the GAC notion is known as *arc consistency (AC)*: C is arc-consistent iff

$$\forall a_1 \in D^{X_1}\, \exists a_2 \in D^{X_2}.\ (a_1,a_2) \in C \wedge \forall a_2 \in D^{X_2}\, \exists a_1 \in D^{X_1}.\ (a_1,a_2) \in C$$

Related to the notion of *GAC* is the notion of *filtering*, i.e., the problem of removing values from the domains of variables in order to obtain an equivalent constraint which is *GAC*. If the filtering is computationally too expensive, one can run fast approximated algorithms, that eliminate some values in some domains obtaining an equivalent constraint C', which is not, however, ensured to be GAC.

Other properties are of interest, e.g., generalized bounds consistency and directional arc/bounds consistency [10]. Nevertheless, in this paper, we only deal with the two properties described above.

By definition (assuming the domains non empty), GAC implies CON. Thus, if we prove that testing GAC is polynomial, the same will hold for CON. If CON is NP-complete, then GAC will be NP-hard. Let us proceed with the analysis of different global constraints in the context of lattices.

3.1 `alldifferent`

The `alldifferent` constraint [20] is probably the most well-known global constraint used in constraint programming. Its semantics is as follows: if $X_1,\ldots,X_n$ are variables with domains $D^{X_1},\ldots,D^{X_n}$, then

$$\texttt{alldifferent}(X_1,\ldots,X_n) = (D^{X_1} \times \cdots \times D^{X_n}) \setminus \left\{(a_1,\ldots,a_n) \in (D^{X_1} \times \cdots \times D^{X_n}) : \exists i,j.\ (1 \leq i < j \leq n \wedge a_i = a_j)\right\}$$

It is well-known that testing the CON and GAC properties, as well as performing GAC filtering for the `alldifferent` constraint can be done in polynomial time. These problems can be solved, for example, by adapting algorithms for bipartite graph matching (the first contribution in this direction is [18]).

The `alldifferent` constraint has a significant role in the modeling of the protein folding problem on discrete lattices [8]—e.g., to express the fact that a point in the lattice cannot be used to accommodate two distinct amino acids.

3.2 `contiguous`

The `contiguous` global constraint is used to describe the fact that a list of variables represent lattice points that are adjacent (in terms of positions in the lattice graph). Let E be the set of edges in a lattice, and let $X_1,\ldots,X_n$ be a list of variables (respectively, with domains $D^{X_1},\ldots,D^{X_n}$). The `contiguous` constraint can be defined as follows:

$$\texttt{contiguous}(X_1,\ldots,X_n) = (D^{X_1} \times \cdots \times D^{X_n}) \setminus \left\{(a_1,\ldots,a_n) \in (D^{X_1} \times \cdots \times D^{X_n}) : \exists i.\ (1 \leq i < n \wedge (a_i,a_{i+1}) \notin E)\right\}$$

Testing the GAC of `contiguous` can be done in polynomial time. In fact, the `contiguous` constraint is equivalent to the conjunction of the $n-1$ binary constraints of the form $C_{i,i+1}$, with $i \in \{1,\ldots,n-1\}$, such that

$$C_{i,i+1} = (D^{X_i} \times D^{X_{i+1}}) \setminus \{(a_i, a_{i+1}) : a_i \in D^{X_i} \wedge a_{i+1} \in D^{X_{i+1}} \wedge (a_i, a_{i+1}) \notin E\}$$

The graph induced by these constraint is acyclic. Thus, under these conditions AC implies GAC [11]. This result can also be justified as follows. Let us assume that for every $i \in \{1,\ldots,n-1\}$, $C_{i,i+1}$ is arc consistent. Choose $i \in \{1,\ldots,n\}$ and choose $a_i \in D^{X_i}$.

- Since $C_{i-1,i}$ is AC, then it exists $a_{i-1} \in D^{X_{i-1}}$ such that $(a_{i-1}, a_i) \in C_{i-1,i}$. Apply back the same process until $C_{1,2}$ is reached.
- Since $C_{i,i+1}$ is AC, then it exists $a_{i+1} \in D^{X_{i+1}}$ such that $(a_i, a_{i+1}) \in C_{i,i+1}$. Apply the same process forward until $C_{n-1,n}$ is reached.

Going backward and forward we have collected a set of elements such that $(a_1,\ldots,a_i,\ldots,a_n) \in C$, thus proving that $\texttt{contiguous}(X_1,\ldots,X_n)$ is GAC.

Observe also that, if there exists $C_{i,i+1}$ that is not AC, then C is not GAC. In fact, that would imply that there is $a_i \in D^{X_i}$ s.t. $\forall b_{i+1} \in D^{X_{i+1}}$ we have that $(a_i, b_{i+1}) \notin E$. This means, in particular, that for all $b_1 \in D^{X_1}$, …, $b_{i-1} \in D^{X_{i-1}}$, $b_{i+1} \in D^{X_{i+1}}$, …, $b_n \in D^{X_n}$, we have that $(b_1,\ldots,b_{i-1},a_i,b_{i+1},\ldots,b_n) \notin C$.

Since AC for binary constraints can be tested in polynomial time, the same holds for GAC. Polynomiality of CON follows.

The `contiguous` is particularly relevant when modeling protein folding problems as it allows one to state that the sequence of amino acids composing the primary sequence of a protein should remain contiguous in the discrete lattice.

3.3 `saw`

The `saw` constraint is used to require that each assignment to the variables $X_1,\ldots,X_n$ represents a self-avoiding walk (SAW) in the lattice. More formally, the constraint can be defined as follows:

$$\texttt{saw}(X_1,\ldots,X_n) = \texttt{contiguous}(X_1,\ldots,X_n) \cap \texttt{alldifferent}(X_1,\ldots,X_n)$$

The `saw` constraint can be used, for example, to model the fact that the primary sequence of a protein can not create cycles when placed in the 3D space.

Testing the CON property for `saw` is clearly in NP. We prove that it is NP-complete by reduction of the NP-complete Hamiltonian Cycle (HC) problem on a particular class of planar graphs, called *special planar graphs* in [6]. The proof consists of two steps. First we show how to embed each special planar graph G in a graph G' whose nodes and edges are in a cubic lattice. Then we "enlarge" G' (replacing every edge by two edges connected by a new node) obtaining a new graph G'' that we use to define variables and domains for an equivalent `saw` problem. We sketch here the proof.

A special planar graph $G = (N,E)$ [6] is composed of a number of *loops*, each containing only nodes with degree 3, along with *paths* of length 2 connecting nodes that belong to distinct loops (see Fig. 2). Let $n = |N|$.

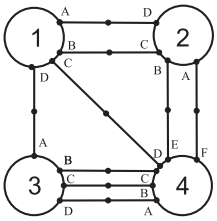


Fig. 2. An example of special planar graph G

It is easy to see that if there is a loop in the graph containing an odd number of nodes, then no HC exists for G. For this reason, we concentrate on graphs containing only loops with even numbers of nodes. Observe that loops have to contain at least four nodes. We assume, moreover, that G contains at least 2 loops (otherwise the result is obviously true).

Let us define how to obtain G' from G. For each loop i, of size $2n_i$, and consisting of the nodes $p_1, \ldots, p_{2n_i}$, we generate a subgraph of G', called *gadget*, which contains $2n_i \times (n_i + 2)$ nodes arranged as follows. These nodes are obtained using a clockwise enumeration of the loops—the starting point is irrelevant. There is a core of the gadget made of a loop of $2n_i$ nodes, arranged as a $2 \times n_i$ rectangle. From each of the core nodes there is a path leading to the nodes $p_1, \ldots, p_{2n_i}$. Those nodes are called the *output* nodes of the gadget. In Figure 3 we report a gadget for a 8-nodes loop.

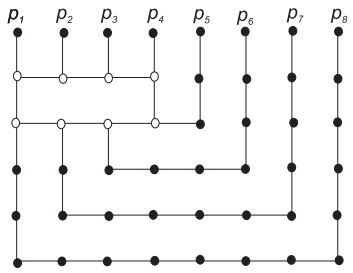


Fig. 3. Example of gadget for a loop of size 8

Let us fix one of the dimensions of the cube (w.l.o.g., $z = 0$) and let us work on the resulting 2-dimensional plane. Let us consider an arbitrary enumeration of the loops $\ell_1, \ldots, \ell_k$ of G and let us align the gadgets on the plane according to such ordering. In particular, all the output nodes of the gadget have $y = 0$, and they are adjacent in the x dimension (see Fig. 4). All other nodes of the gadget have $y > 0$.

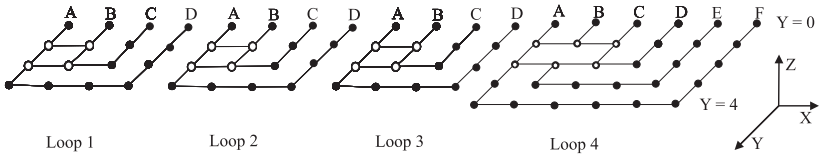


Fig. 4. The aligned gadgets for graph G in Fig. 2

Consider, in lexicographical ordering, the loop pairs $\langle \ell_a, \ell_b \rangle$, $a < b$, that are connected by edges in E. We wish to create copies of the output nodes of ℓ_a and ℓ_b in a separate plane (to avoid intersection of edges) and recreate on this plane the connection structure of G.

Let us illustrate the process for all the pairs $\langle \ell_1, \ell_{b_1} \rangle, \ldots, \langle \ell_1, \ell_{b_h} \rangle$ such that there are edges between loop 1 and loop b_i in G.

In the plane i, we add copies of the "relevant" output nodes for loop 1 and b_i. E.g., if an output node of loop 1 or loop b_i is at coordinates $(x,y,0)$, and such node is part of an edge connecting these two loops, then a copy of such node will be created at coordinates (x,y,i). Furthermore, copies of such nodes are also placed in all the intermediate planes (planes $0 \leq z \leq i-1$), and edges connecting these copies are created—i.e., edges of the type $((x,y,z),(x,y,z+1))$. The edges between the nodes of distinct loops of G are simulated by paths in the plane i. The output nodes of the gadgets 1 and b_i respect a clockwise traversal of the loops 1 and b_i. Since G is planar, we can connect using non-intersecting paths in plane i (see Fig. 5). As a strategy, start with the rightmost output node of loop 1.

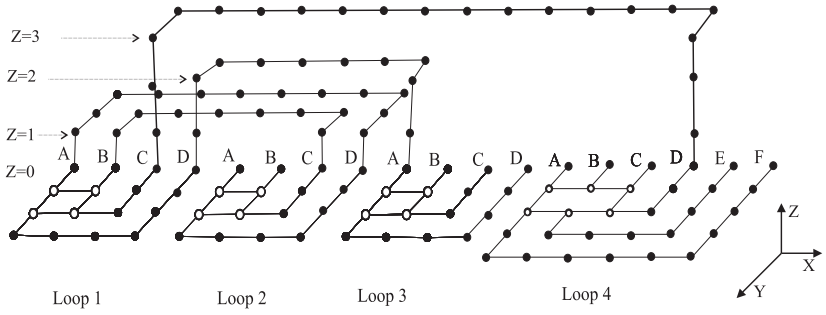


Fig. 5. The encoding of the edges outgoing from loop 1 in Fig. 2

The process is repeated for all the other pairs of connected loops (incrementing the plane levels). In Fig. 6 we show the graph G' corresponding to the graph G of Fig. 2.

A rough estimate of the size of the resulting graph is the following. The number of x indices used is $O(n)$ (gadgets outputs are the same as loop lengths). The number of

values of y used is $O(n)$ for the gadgets plus $O(|E|) = O(n^2)$. The number of values of z used is again $O(|E|) = O(n^2)$. Thus the global "box" containing G' contains $O(n^5)$ points.

Since the graph G' maintains the topology (and at most introduces new nodes with degree 2 on paths), it holds that the Hamiltonian Cycle problem on G' has a solution iff G has. Basically, G' is a copy of G where the edges linking distinct loops are stretched (by adding new nodes of degree 2). We will refer these sequences of edges as *loop2loop*. loop2loops have always length at most 4 (out of the gadgets) plus 2 (within the gadgets). We will use the same terminology in G''.

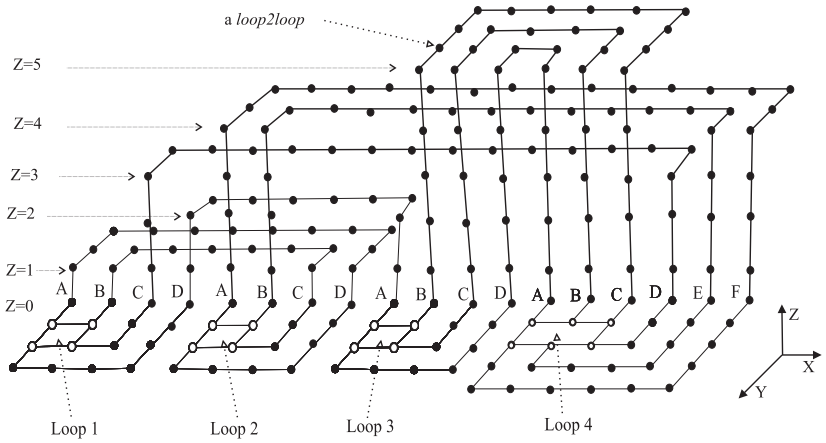


Fig. 6. The graph G' obtained from G in Fig. 2

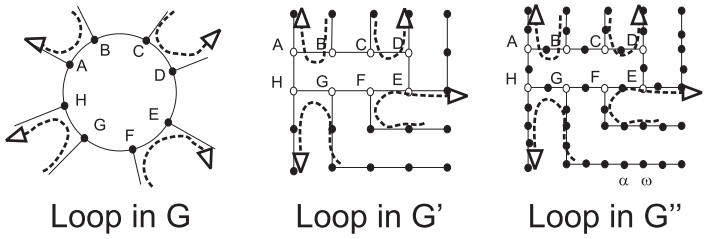


Fig. 7. Loops in G, G', and G''. Observe that the Hamiltonian paths in G and G' cannot touch half of the extra nodes added in loops in G''

We need an additional step to encode the HC problem using the `saw` constraint. The basic idea is that a self-avoiding walk is an Hamiltonian Path. The problem is that in G' representatives of elements of N may lie at distance 1 in spite of them not being connected by an edge in E. If we introduce variables and assign to their domains the nodes of N', self-avoiding walks on the nodes of N' can have trajectories that are not allowed in G.

We thus define the graph $G'' = (N'', E'')$ as follows:

- for every edge $((x,y,z),(x+1,y,z))$ introduce in N'' the nodes $a = (2x, 2y, 2z)$, $b = (2x+1, 2y, 2z), c = (2x+2, 2y, 2z)$, and the edges $(a,b),(b,c)$.
- for every edge $((x,y,z),(x,y+1,z))$ introduce in N'' the nodes $a = (2x, 2y, 2z)$, $b = (2x, 2y+1, 2z), c = (2x, 2y+2, 2z)$, and the edges $(a,b),(b,c)$.
- for every edge $((x,y,z),(x,y,z+1))$ introduce in N'' the nodes $a = (2x, 2y, 2z)$, $b = (2x, 2y, 2z+1), c = (2x, 2y+2, 2z+2)$, and the edges $(a,b),(b,c)$.

The graph G'' is a copy of G', in which each edge is substituted by a subgraph of the form (edge, new node, edge). Let $m = |N''|$. In Fig. 7, on the right it is depicted a fragment of G'' obtained from the fragment of G in the center of the figure. Observe that edges in E'' connect nodes at Euclidean distance 1. This property can be exploited by the `saw` constraint to simulate the graph connectivity.

Consider again Figure 7 from left to right. Let us assume that G admits an Hamiltonian Cycle. Any Hamiltonian Cycle traverses the loop in a way similar to the one depicted. There is a corresponding Hamiltonian Cycle traversing the loop in G'. A corresponding path exists in G''; however it is not Hamiltonian since *half of the nodes of the loop* cannot be traversed by that path. We must take care of that designing our encoding.

Let L be the global number of nodes in loops of G divided by 2. Define the variables $X_1 \dots X_{m-L}$, and for $1 < i < 2m$ let $D^{X_i} = N''$. For the variables X_1 and X_{m-L} we specify a singleton domain as follows. Identify in G'' two consecutive nodes of degree 2 in a loop2loop. Let us call them α and ζ (see also Fig. 7—right). Then $D^{X_1} = \{\alpha\}, D^{X_{m-L}} = \{\zeta\}$. The definition of the CSP is completed by the constraint $\texttt{saw}(X_1, \dots, X_{m-L})$.

Theorem 1. *G' admits an Hamiltonian Cycle iff G'' admits a self avoiding walk with $m-L$ nodes starting from α and ending in ζ.*

Proof. ($\rightarrow$) Let us assume that G' has an HC. The same cycle can be mimicked on the extended graph G''. All nodes in loop2loops are traversed by this path. Instead, for each loop, a number of points which is half of the number of points of the original loop in G is not traversed by the path. Then, the cycle has length $m-L$.

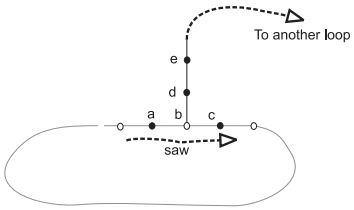
Since α and ζ have degree 2 and are in a loop2loop, the cycle must contain the edge (α, ζ). Removing such edge from the path we obtain a SAW of length $m-L$ starting in α and ending in ζ.

($\leftarrow$) Let $\alpha, p_2, \dots, p_{L-m-1}, \zeta$ be a a SAW consisting of $m-L$ nodes of G'' starting from α and ending in ζ. Since $|N''| = m$, exactly L nodes of N'' are left out by this SAW.

1. If the SAW enters and exists all the loops as in Fig. 7, then it will leave out L nodes and it corresponds to an Hamiltonian Path in G' starting in α and ending in ζ. Since

α and ζ are consecutive nodes of degree 2, it is sufficient to add the edge (ζ, α) to find the cycle which corresponds to an Hamiltonian Cycle in G.

2. Since α and ζ are in the same loop2loop it is impossible that a SAW starting in α and ending in ζ does not enter any loops, unless $m - L = 1$, which cannot be true by construction of G''.
3. It remains to analyze the case in which the SAW traverses a loop in a way different from that of point 1. Assuming that it exists, we will find a contradiction with its length $m - L$. For these SAWs, there is at least one loop2loop left out from a path traversing one loop, as in the following figure:



We have already seen that loops2loops are of length ≥ 6 in G' (13 in G''). Let e and d be the first two nodes of the loop2loop left out. These two nodes cannot be crossed by the SAW. As a matter of fact, if d or e are reached by a SAW, there is no way to come back to ζ without repeatedly visiting the same nodes.
On the other hand, the SAW inside the loop visits both the points a and c adjacent to the entering point b of the analyzed loop2loop. With respect to a SAW of the form dealt with in point 1, the SAW visits one additional point in the loop but looses two points outside the loop. This happens for every loop and for every loop2loop excluded by the SAW. Thus, more than L points of G'' are left out. This is a contradiction. □

This reduction is polynomial, thus the CON of `saw` global constraint is NP-complete, and, consequently, GAC is NP-hard.

Observe that the proof has been carried out using the cubic lattice. It is easy to modify the mapping for other 3D lattices.

3.4 `alldistant`

When we model biologically motivated problems (e.g., protein folding) on a discrete lattice, we often observe that the `alldifferent` global constraints is not sufficiently expressive. In particular, we often require that values assigned to a group of variables are sufficiently spread in the lattice, ensuring a minimal distance between each pair of points assigned to the variables. This is required, for example, to address the fact that different amino acids of a protein have different volume occupancy.

In the `alldistant` constraint, given n variables $X_1, \ldots, X_n$, with respective domains $D^{X_1}, \ldots, D^{X_n}$, and n numbers $c_1, \ldots, c_n$, we are looking for a solution $X_1 = p_1, \ldots, X_n = p_n$ such that, for each pair $1 \leq i, j \leq n$, we have that p_i and p_j are located at distance at

least $c_i + c_j$. More formally:

$$\begin{aligned}\texttt{alldistant}(X_1,\ldots,X_n,c_1,\ldots,c_n) = (D^{X_1}\times\cdots\times D^{X_n}) \setminus \\ \{(a_1,\ldots,a_n)\in(D^{X_1}\times\cdots\times D^{X_n}) : \\ \exists i,j.\ 1\le i<j\le n \wedge sqeucl(a_i,a_j) < (c_i+c_j)^2\}\end{aligned}$$

Note that if we consider the `alldistant` with $c_1 = \frac{1}{2},\ldots,c_n = \frac{1}{2}$ then we achieve the same effect as `alldifferent`.

We show how to reduce the BIN-Packing problem to the consistency problem for the `alldistant` constraint. Let us consider n items of size $c_1,\ldots,c_n$ and k bins (bin 0, ..., bin $k-1$) of capacity B. W.l.o.g., let us assume that, for all $i\in\{1,\ldots,n\}$, it holds that $c_i \le B$ (otherwise the problem is trivially unsatisfiable).

We reduce the problem using only one dimension of the lattice (assume, e.g. that all y and z coordinates are fixed to 0). We consider consecutive lattice collinear points. For the sake of simplicity, we consider lattice points $(0,0,0)$, $(1,0,0)$,$(2,0,0),\ldots$ and we refer to them simply as $0,1,2,\ldots$[4]

The reduction is defined as follows. Let us introduce n lattice variables $X_1,\ldots,X_n$. For $i\in\{1,\ldots,n\}$ the domain D^{X_i} is defined as

$$D^{X_i} = \bigcup_{j=0}^{k-1}[4jB+c_i\,..\,4jB+2B-c_i]$$

For example, consider the instance: $c_1=4, c_2=3, c_3=5, c_4=1, B=7, k=2$. Then $D_1 = [4\,..\,10]\cup[32\,..\,38]$, $D_2 = [3\,..\,11]\cup[31\,..\,39]$, $D_3 = [5\,..\,9]\cup[33\,..\,37]$, $D_4 = [1\,..\,13]\cup[29\,..\,41]$.

Intuitively, each interval $[0\,..\,2B]$, $[4B\,..\,6B]$, $[8B\,..\,10B]$, ... corresponds to a bin. Each assignment of the variable X_i in D_i is such that all values $[X_i - c_i, X_i + c_i]$ are included in exactly one of the above intervals (intuitively, the item i is assigned to the bin corresponding to such interval). If the values of X_i and X_j are in two different intervals, then $|X_i - X_j| > 2B \ge c_i + c_j$.

We show that there is a solution for the instance of the BIN-packing problem if and only if there is a solution for the CSP $\texttt{alldistant}(X_1,\ldots,X_n,c_1,\ldots,c_n)$. In the above example, a solution of the CSP is $X_1=4$, $X_2=11$, $X_3=33$, $X_4=40$, from which one can conclude that we should place items 1 and 2 in bin 0 and items 3 and 4 in bin 1.

For one direction, assume that the CSP admits a solution σ. Consider all the variables taking values in $\sigma(X_i)\in[4Bj\,..\,4Bj+2B]$. Assume that those variables are $X_1^j,\ldots,X_{m_j}^j$, and assume that $\sigma(X_1^j)<\cdots<\sigma(X_{m_j}^j)$. This means that

- $\sigma(X_1^j) \ge 4Bj + c_1^j$ (constraint on the domain),
- $\sigma(X_2^j) \ge 4Bj + c_1^j + (c_1^j + c_2^j) = 4Bj + 2c_1^j + c_2^j$ (`alldistant` constraint),
- $\sigma(X_3^j) \ge 4Bj + c_1^j + (c_1^j + c_2^j) + (c_2^j + c_3^j) = 4Bj + 2c_1^j + 2c_2^j + c_3^j$ (`alldistant` constraint),

[4] For some lattice structures, it may be necessary to choose a different subset of points. The proof can be adapted by choosing, e.g., a collinear set of lattice points (some scaling of coefficients may be needed).

– and so on, until $\sigma(X^j_{m_j}) \geq 4Bj + 2(c^j_1 + c^j_2 + c^j_{m_j-1}) + c^j_{m_j}$

Moreover, for the constraint on the domain, it holds that $\sigma(X^j_{m_j}) \leq 4Bj + 2B - c^j_{m_j}$. This means that $2(c^j_1 + c^j_2 + \cdots + c^j_{m_j}) \leq 2B$. Thus, put all items associated to the considered variables to bin j to obtain a solution of the bin packing.

The vice versa is similar. Given a solution of the bin packing, for each bin j, consider the items $\texttt{item}^j_1, \texttt{item}^j_2, \ldots, \texttt{item}^j_{m_j}$ assigned in to the bin j. Then set $\sigma(X^j_1) = 4Bj + c^j_1, \sigma(X^k_2) = 4Bj + 2c^j_1 + c^j_2, \ldots, \sigma(X^j_{m_j}) = 4Bj + 2(c^j_1 + c^j_2 + c^j_{m_j-1}) + c^j_{m_j}$.

NP completeness of GAC follows, as usual. The problem of filtering is open, and it could be investigated, e.g., through adaptation of the *sweep algorithms* used in [3].

3.5 rigid block

It is a frequent situation, when dealing with protein structure determination, to have knowledge of local features of the structure, e.g., presence of secondary structure components (such as α-helices and β-strands); thus, we may wish to be able to express the fact that a collection of points have to be located in the discrete lattice according to a predefined pattern.

This notion can be represented using another type of global constraint, called *rigid block constraint*. A rigid block defines a layout of points in the space that has to be respected by all admissible solutions. Let $X_1, \ldots, X_n$ be a list of variables (having, respectively, domains $D^{X_1}, \ldots, D^{X_n}$), and let $B = B_1, \ldots, B_n$ be a list of lattice points—that, intuitively, describe the desired layout of the rigid block. $\mathsf{block}(X_1, \ldots, X_n, B)$ is a k-ary constraint, whose solutions are assignments of lattice points to the variables $X_1, \ldots, X_n$, that can be obtained from B modulo *translations* and *rotations*.

More precisely, we define a *rotation* of a lattice point $p = (p_x, p_y, p_z)$ as $rot(\phi, \theta, \psi)(p) = X \cdot Y \cdot Z \cdot p^T$, where

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}, \; Y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \; Z = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Although the rotation angles ϕ, θ, ψ are real valued, only few combinations of them define automorphisms on the lattice in use. The total numbers of distinct automorphisms r depends on the lattice—e.g., in the cubic lattice, we have that $r = 16$, and in the FCC we have that $r = 24$.

We extend the definition of rotation to the case of lists of lattice points, $rot(\phi, \theta, \psi)(B)$, where B is a list of points and the result is a list in which every element of B is rotated according to the previous definition.

Given a list of points B, we define the concept of *templates* as the set:

$$\mathsf{Templ}(B) = \left\{ rot(\phi, \theta, \psi)(B) : \begin{array}{c} \exists \phi, \theta, \psi.\ rot(\phi, \theta, \psi)(B) \text{ is an} \\ \text{automorphism on the lattice} \end{array} \right\}$$

which contains the distinct 3-dimensional rotations of the points B in the lattice. Note that, for a given list of points (B), the cardinality of $\mathsf{Templ}(B)$ is at most r. We say

that $\ell = (\ell_x, \ell_y, \ell_z)$ is a *lattice vector* if the translation by ℓ of lattice points generates an automorphism on the lattice. Note that, for some asymmetric lattices, it is possible that lattice vectors do not exist.

Let ℓ be a lattice vector; with $\mathsf{Shift}[\ell]$ we denote a mapping that translates a rigid block according to the vector ℓ. Formally, for each $i = 1, \ldots, k$, $\mathsf{Shift}[\ell](B)[i] = B_i + \ell$. Shifts are used to place a template into the lattice space, preserving the orientation and the distances between points.

A rigid block constraint $\mathsf{block}(X_1, \ldots, X_n, B)$ is then defined as the set:

$$\left\{ (a_1, \ldots, a_n) \in D_1 \times \cdots \times D_n : \exists \ell \, \exists P. \begin{pmatrix} P \in \mathsf{Templ}(B) \wedge \\ \mathsf{Shift}[\ell](P) = (a_1, \ldots, a_n) \end{pmatrix} \right\}$$

With a fixed rotation of the block, CON is linear in the size of the smallest variable domain (a simple intersection of possible translations for each domain has to be performed). GAC is polynomial as well, since it is sufficient to repeat the CON test for each domain.

Propagation of this kind of constraint is studied in a wider context in [16]. Moreover, the idea of considering rigid blocks to model substructures of proteins has also been introduced in [9].

4 Conclusions and future work

In this paper we presented a preliminary study of various global constraints that can be used to provide declarative encoding of problems in discrete lattices. The introduction of global constraints has been motivated by problems derived from the use of constraint solving in discrete lattices to solve the protein folding determination problem. We propose different types of constraints and investigate their computational properties.

A number of issues are open and deserve consideration. First of all, it is interesting to investigate the relative expressive power of the different constraints—e.g., to understand the importance of having one type of global constraints versus the others. It is also important to gain a clear understanding of the computational properties of the different global constraints, with particular attention to the complexity of verifying the properties CON and GAC and the cost of performing filtering. Throughout the paper we hinted at the high complexity (e.g., NP-completeness) of some of these problems; in such cases, it will be important to detect approximated polynomial filtering algorithms, that can be effectively introduced in a constraint solver like COLA.

Furthermore, we plan to study the constraints among rigid blocks (e.g., parallelism, angles between them, or proximity between them as proposed by Krippahl and Barahona).

Acknowledgments

This work is partially supported by FIRB Project RBNE03B8KK and by PRIN Project 2005015491.

References

1. R. Backofen. The protein structure prediction problem: A constraint optimization approach using a new lower bound. *Constraints*, 6(2–3):223–255, 2001.
2. R. Backofen and S. Will. A Constraint-Based Approach to Structure Prediction for Simplified Protein Models that Outperforms Other Existing Methods. *Proc. of ICLP 2003*, Springer Verlag, 2003.
3. N. Beldiceanu and M. Carlsson. Sweep as a Generic Pruning Technique Applied to the Non-overlapping Rectangles Constraint. LNCS 2239, 2001.
4. C. Bessiere, E. Hebrard, B. Hnich, and T. Walsh. The complexity of global constraints. In Proceedings AAAI'04, San Jose CA, 2004.
5. P. Clote and R. Backofen. *Computational Molecular Biology: An Introduction*. John Wiley & Sons, 2001.
6. P. Crescenzi et al. On the Complexity of Protein Folding. *Journal of Computational Biology*, 5:3 423–466, 1998.
7. A. Dal Palù, A. Dovier and F. Fogolari. Constraint logic programming approach to protein structure prediction. *BMC Bioinformatics*, 5(186), 2004.
8. A. Dal Palù, A. Dovier and E. Pontelli. A Constraint Logic Programming Approach to 3D Structure Determination of Large Protein Complexes. *Proc. of LPAR 2005*.
9. A. Dal Palù, S. Will, R. Backofen and A. Dovier. Constraint Based Protein Structure Prediction Exploiting Secondary Structure Information. *Proc. of Italian Conference on Computational Logic CILC 2004*.
10. R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.
11. E. C. Freuder. A sufficient condition for backtrack-bounded search. JACM 29(1)24–32, 1982.
12. W. E. Hart and A. Newman. The Computational Complexity of Protein Structure Prediction in Simple Lattice Models. *Handbook on Algorithms in Bioinformatics*, CRC Press, 2003.
13. G. Kant. Drawing Planar Graphs using the Canonical Ordering. Tech. Rep. RUU-CS-92-33, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands, 1992
14. L. Krippahl and P. Barahona. Applying Constraint Propagation to Protein Structure Determination. *Proc. of CP 2003*, LNCS 1713:289–302, 1999.
15. L. Krippahl and P. Barahona. PSICO: Solving Protein Structures with Constraint Programming and Optimisation. *Constraints* 7:317–331, 2002.
16. L. Krippahl and P. Barahona. Propagating N-Ary Rigid-Body Constraints. *Proc. of CP 2003*, LNCS 2833:452–465, 2003.
17. L. Krippahl and P. Barahona. Applying Constraint Programming to Rigid Body Protein Docking. *Proc. of CP 2005*, LNCS 3709:373–387, 2005.
18. J.-C. Regin. A filtering algorithm for constraints of difference in CSPs. R.R. LIRMM 93–068, 1993.
19. J. Skolnick and A. Kolinski. Reduced models of proteins and their applications. *Polymer*, 45:511–524, 2004.
20. W. J. van Hoeve. The Alldifferent Constraint: a Survey. *Sixth ERCIM Workshop on Constraints*, 2001.

Suffix arrays and weighted CSPs

Matthias Zytnicki, Christine Gaspin, Thomas Schiex

INRA Toulouse – BIA

Abstract. In this paper, we describe a new constraint that uses some interesting data structure, the suffix array, well-known in pattern matching. We show how it helps answering the question of non-coding RNA detection in bio-informatics, and more precisely, finding the best hybrid in a duplex constraint.

1 Introduction

Thanks to the recent major advances in molecular biology, the problem of the detection of non-coding RNA (ncRNA) is now a hot topic in bio-informatics (cf. [1] for review). A ncRNA is usually represented by a sequel of letters, or *nucleotides*: A, C, G and T. An ncRNA also contains *interactions* —mainly A–T and C–G— that are essential to its biological function. In this paper, we will suppose we know the *structure* of a ncRNA family. The structure is the set of information located on a ncRNA that discriminate for a given biological function. Our aim is the following: how can I get all the candidates matching a given structure, in a sequence that may contain several billions of nucleotides? Put in other words, knowing an interaction map and some nucleotide positions, which regions of my sequence match this map and contain these nucleotides?

Among the proposed formalisms used to solve this problem, one of the most famous ones uses statistical information in a context-free grammar that describes this structure [5]. However, some complex ncRNA families cannot be described within this formalism and [6] showed that only NP-hard formalisms may correctly describe them. This favors a CSP model of the problem and such a work has been been done in [7].

However, usual queries give hundred of thousands of solutions and, in practice, it is impossible to exploit this huge amount of solutions. Obviously, by looking more carefully at the solutions, some are better than others and it would be useful to give only the best ones to the user. This is why we used the weighted CSP formalism to solve the ncRNA detection problem.

One interesting element of structure that we would like to model is the duplex. It is the ability from the ncRNA to *hybridize*, i.e. to develop a stretch of interactions with a DNA strand, or another RNA. We would like to embed this element of structure into a global constraint. Since this problem is very similar to the approximate string matching, many formalisms have already been proposed (cf. [2] for review) to compute the underlying algorithm. Most of them are based on a dynamic programming algorithm that computes a kind of *edit distance* between a word and the subsequences of a long sequence. To save time and space, these algorithms have been ported to different structures, such as the suffix tree. This structure makes it possible to focus the search on the most promising regions and dramatically speeds up the search. Recently, some papers [3,4] also proposed the *enhanced suffix array* —or *suffix array* for short— to solve

this kind of problem, with an enhancement that provides several advantages compared with the suffix tree, with no drawback.

In this paper, we present a new global constraint that checks whether there exists a word that matches a subsequence of a given long sequence, with a possible given number of errors, using a suffix array.

2 The WCSP model

The weighted CSP (WCSP) [8] framework is an extension of the CSP, that makes it possible to express *preferences* among solutions thanks to *soft constraints*. It has already been applied to resource allocation, scheduling, combinatorial auction, CP networks and probabilistic reasoning.

The *valuation structure* $\mathcal{S} = \langle E, \oplus, \leq \rangle$ specifies the costs, where: $E = [0..k] \subseteq \mathbb{N}$ is the *set of costs*, k, which is the highest cost, can possibly be ∞, and it represents an *inconsistency*; $\leq$ is the usual operator on $\mathbb{N}$; $\oplus$, the *addition* on E, is defined by $\forall (a,b) \in \mathbb{N}^2, a \oplus b = \min\{a+b, k\}$. A WCSP is a tuple $\mathcal{P} = \langle \mathcal{S}, \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where: $\mathcal{S}$ is the valuation structure; $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of *n variables*; $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ is the set of possible *values* of each variable, or *domains*, and the size of the largest one is d; $\mathcal{C} = \{c_1, \ldots, c_e\}$ is the set of e soft constraints.

An assignment t on the set of variables $Y \subseteq \mathcal{X}$ is a function that associates to each variable of Y one of its possible value: $t = (y_1 \leftarrow v_1, \ldots, y_m \leftarrow v_m)$. A soft constraint c_i involves a list of r_i variables $var(c_i) = (y_1, \ldots, y_{r_i})$ (r_i is the *arity* of c_i), and it associates to every assignment t of the involved variables a cost $c_i(t)$ in E. Given a constraint c_i and an assignment t of $var(c_i)$, $c_i(t) = k$ means that the constraint forbids the corresponding assignment. Another cost means the assignment is permitted by the constraint with the corresponding cost. The cost of a total assignment (i.e. of all the variables) is the sum of the costs of all the cost functions. A total assignment t is a *solution* if its cost is less than k.

In our model, as described in [7], the variables represent the *positions* on the sequence of the elements of structure. The initial domain of the variables will therefore be equal to the size of the sequence. The constraints enforce the presence of the wished elements of structure between the specified variables. Within this model, a solution is a position for each variable, such that all the elements of structure specified by the constraints can be found. Our aim is to find all the solutions of the problem, given a maximum cost k.

We will not describe here all the constraints used, and we will focus on the duplex constraint. This constraint ensures that there exists a set of interactions between our sequence (the *main sequence*) and another given sequence (the *target sequence*). It has two parameters: the target sequence and the maximum number of errors in the interaction set. Similarly to the edit distance, the number of errors of a hybridization is the number of nucleotides that do not interact with any other nucleotide, plus the number of pairs of nucleotides associated through a non-allowed interaction. This will be the cost given by the constraint. The duplex constraint involves four variables: x_i, x_j, y_k and y_l. x_i represents the start position of the main stem, x_j represents its end position, whereas y_k and y_l represent the start and end positions of the target stem. To solve the

problem, we use a depth-first branch-and-bound algorithm that maintains a extension of 2B-consistency adapted to soft constraints, called *bound arc consistency* (BAC*, [9]).

In our implementation, the duplex constraint remains idle until the x variables are assigned, and BAC* is only enforced on the y variables. Waiting for the x variables to be assigned means that we know the word on the main sequence that will take part in the interaction. Thus, our aim here will be to design an algorithm that efficiently finds the minimum number of errors between the given word and any subsequence of the target stem, bounded by the y variables.

3 Suffix arrays

The suffix tree is a tree with edges labeled with words. This data structure has been widely used in pattern matching algorithms. Given a text, the paths from the root node of its suffix tree and its terminal nodes enumerate all the suffixes of this text (cf. **Fig.** 1(a) for the string AAACA). It is a particularly convenient data structure, since it requires linear space w.r.t. the size of the text, takes linear time to build, and searching whether a word is contained in this text requires time proportional to this word (and is independent in the size of the text).

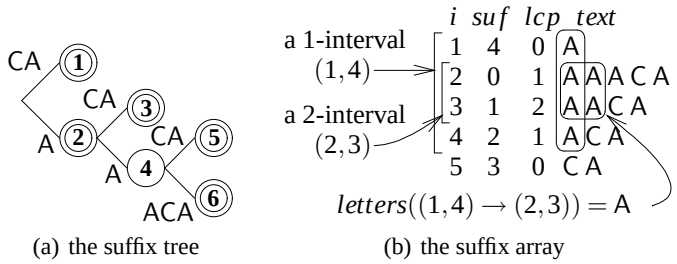
However, given a sequence T of size m, its suffix array S also requires a linear time to build, takes as much time to find a word, but requires a bit less space, and lead to less cache misses, thanks to the array structure [4]. Basically, a suffix array is an array where all the suffixes of a text are sorted through lexicographic order (cf. **Fig.** 1(b)). Of course, only the position $suf[i]$ of the first letter of each suffix i is stored. Additional information is also stored on each line of the array. First, the size of the longest common prefix (denoted lcp) between the suffix of line i and line $i-1$ is inserted on line i (by convention, $lcp[0] = 0$).

(i, j) is called a *l-interval* iff: $lcp[i] < l$, $\forall k \in (i, j], lcp[k] \geq l$, $\exists k \in (i, j], lcp[k] = l$, and $lcp[j+1] < l$. These l-intervals can be compared with the nodes of the suffix tree. For example, the interval $(2,3)$, on figure 1(b), is correlated with the node **4** of the suffix tree, and since this node represents the two-letters word AA, $(2,3)$ is a 2-interval. An interval represents an interior node if $i \neq j$, and it a leaf otherwise. Using linear space, we can build in linear time a function that, given an l-interval, gets their child l'-intervals (i', j'). With this function in hand, we can simulate a suffix tree with our suffix array. Using the same notations, we will denote $letters(i, j)$ the subsequence $T[suf[i]..suf[i]+l]$, and $letters((i, j) \rightarrow (i', j'))$ the subsequence $T[suf[i]+l..suf[i]+l']$. In our example, since the intervals $(1,4)$ and $(2,3)$ are correlated with the nodes **2** and **4** respectively, $letters(2,3)$ is AA and $letters((1,4) \rightarrow (2,3))$ is A.

4 An algorithm for approximate matching

4.1 First algorithm

This algorithm takes as an input the suffix array S, a word w of size n and a maximum edit distance $maxErr$. It returns the minimum distance between w and any subsequence of T, or $maxErr+1$ if this distance is greater than $maxErr$. It uses a hybridization cost



(a) the suffix tree (b) the suffix array

Fig. 1. Two representations of the suffixes of AAACA

matrix c_{hyb}, that, given two nucleotides, returns the hybridization penalty (0 being a perfect hybridization). c_{ins} is the penalty cost for a non-hybridized nucleotide.

The main function, getApproximateWord(), works as follows. On line **1**, we consider an l-interval, between the lines i and j in the array. We suppose that we have matched $pref_w$ letters of w so far, and we have encountered $nbErr$ errors. The function getChildren() returns in constant time (by using some appropriate data structure) all the child intervals of (i, j). The line **3** checks whether the considered child l'-interval is an interior node or a leaf. In the former case, we try to match $letters((i, j) \rightarrow (i', j'))$ with the remaining unmatched letters of w through the function getApproximateWord() on line **4**. If the flag f of an element returned by this function is set to true(line **5**), then all the letters of w have been matched and we may have a solution. Otherwise (line **6**), we have to continue the exploration. For that, we store the current configuration (including the bounds and the lcp of the current interval, and the number of errors found so far) in a stack, that we will examine afterwards. If the current child interval is a leaf (line **7**), then there is only one possibility to match the remaining unmatched letters of w. Thus, we simply call getCandidates() and only keep the solutions that match all the letters of w on line **8**.

Let us now explain the function getCandidates(). It gets two strings, w and b, of size s and t respectively, and tries to match them. It also takes $nbErr$ as a parameter, which gives the maximum allowed distance between w and b. Basically, it is a simple Needleman-Wunsch dynamic programming algorithm. The only difference is that it returns a list containing all the solutions with a cost less than $nbErr$ that are located on the last row or on the last column of the dynamic programming matrix. If it is on the last row, then b has been totally matched with a prefix of w; if it is on the last column, w has been totally matched with a prefix of b. Each element of the solution list contains the number of matched letters of the prefix, the score of the match, and a boolean that states whether the solution is on the last row or on the last column.

Algorithm 1: Functions used for approximate search

Function getApproximateWord(**suffix array** S, **string** w, **int** $maxErr$): **int**
  $stack.push(0, n-1, 0, 0)$; $min \leftarrow maxErr + 1$;
  while $(\neg stack.empty())$ **do**
1 $(i, j, pref_w, nbErr) \leftarrow stack.pop()$;
2 **for** $(i', j') \in$ getChildren(i, j) **do**
3 **if** $(i' \neq j')$ **then**
4 $list \leftarrow$ getCandidates$(letters((i,j) \rightarrow (i', j')), w[pref_w..m-1], maxErr - nbErr)$;
  while $(\neg list.empty())$ **do**
  $(len, score, f) \leftarrow list.pop()$;
5 **if** (f) **then** $min \leftarrow \min\{nbErr + score, min\}$;
6 **else** $stack.push(i', j', pref_w + len, nbErr + score)$;
  else
7 $list \leftarrow$ getCandidates$(letters((i,j) \rightarrow (i', i')), w[pref_w..m-1], maxErr - nbErr)$;
  while $(\neg list.empty())$ **do**
  $(len, score, f) \leftarrow list.pop()$;
8 **if** (f) **then** $min \leftarrow \min\{nbErr + score, min\}$;
  return min ;
Function getCandidates(**string** w, **string** b, **int** $nbErr$): **list** (**int**, **int**, **bool**)
  for $i \in [0..s]$ **do** $mat[i][0] = i$; **for** $j \in [1..t]$ **do** $mat[0][j] = j$;
  for $i \in [1..s]$ **do** **for** $j \in [1..t]$ **do**
  $mat[i][j] \leftarrow \min \begin{cases} mat[i-1][j-1] + c_{\text{hyb}}(w[i-1], b[j-1]), \\ mat[i-1][j] + c_{\text{ins}}, mat[i][j-1] + c_{\text{ins}} \end{cases}$;
  for $j \in [0..t]$ **do** **if** $(mat[s][j] \leq nbErr)$ **then** $list.add(j, mat[s][j],$ true$)$;
  for $i \in [0..s]$ **do** **if** $(mat[i][t] \leq nbErr)$ **then** $list.add(i, mat[i][t],$ false$)$;
  return $list$;

4.2 Optimizations

We also have implemented several optimizations. First, we observed that the exploration often visits several times the same l-intervals with exactly the same configuration, or even with less interesting configurations (they contain more errors, with the same number of matched letters). Obviously, some work is unnecessarily done. To avoid it, without using too much space, we store at each node the last configuration that visited it.

Second, we propagate information between the y variables. For example, if we have some information about the y_k variable, then we may shrink the domain of the y_l variable, knowing the size of w, and the number of allowed errors.

Then, we tried to take advantage of the information given by the WCSP. For instance, the solver might have reduced the bounds of the y_k variable (which represents the beginning of the duplex in the target sequence), and this information should be used to prune some branches of the suffix array. To achieve this dynamical pruning, we added on each l-interval (i, j) the smallest interval of T that contains the subsequence $letters(i, j)$, so that the values of y_k that have been deleted by the WCSP solver will never be explored by the suffix array.

A first, rough, evaluation of the worst time complexity of our algorithm is $O((m + maxErr)^{m+maxErr+1}\sigma^{maxErr})$, where σ is the size of the alphabet. On real life examples, where the main and the target sequences contain several millions of nucleotides, enforcing this constraint usually takes not more than a few seconds in the whole execution of the program. This is all the more encouraging as our program finds *all* the solutions of the problem.

5 Conclusions and future work

In this paper we have presented a new constraint, dedicated to bio-informatics problems (or, more generally, to text-based problems), that uses suffix arrays, in an attempt of combining constraints with pattern matching algorithms. In the future, we would like to compare our method with other existing ones, and provide for an empirical evaluation of our approach. You can use the tool that implements the described framework at `carlit.toulouse.inra.fr/Darn/index.php`.

References

1. Eddy, S.: Non-coding RNA genes and the modern RNA world. Nature Reviews **2** (2001)
2. Gusfield, D.: Algorithms On Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge Univ. Press (1997)
3. Abouelhoda, M., Kurtz, S., Ohlebusch, E.: The enhanced suffix array and its application to genome analysis. In: Second Workshop in Algorithms in Bioinformatics. (2002)
4. Abouelhoda, M., Kurtz, S., Ohlebusch, E.: Replacing suffix trees with enhanced suffix arrays. Journal of Discrete Algorithms (2004) 53–86
5. Eddy, S., Durbin, R.: RNA sequence analysis using covariance models. Nucleic Acids Research **22** (1994) 2079–88
6. Vialette, S.: On the computational complexity of 2-interval pattern matching problems. Theoretical Computer Science **312** (2004) 223–249
7. Thébault, P., de Givry, S., Schiex, T., Gaspin, C.: Combining constraint processing and pattern matching to describe and locate structured motifs in genomic sequences. In: 5th Workshop On Modelling and Solving Problems With Constraints. (2005)
8. Larrosa, J., Schiex, T.: Solving weighted CSP by maintaining arc-consistency. Artificial Intelligence **159** (2004) 1–26
9. Zytnicki, M., Schiex, T., Gaspin, C.: A new local consistency for weighted CSP dedicated to long domains. In: SAC 2006. (2006)

Supertree Construction with Constraint Programming: recent progress and new challenges

Patrick Prosser

Department of Computing Science, University of Glasgow, Scotland. pat@dcs.gla.ac.uk

1 Introduction

One goal of biology is to build the *Tree of Life* (ToL), a representation of the evolutionary history of every living thing. To date, biologists have catalogued about 1.7 million species, yet estimates of the total number of species ranges from 4 to 100 million. Of the 1.7 million species identified only about 80,000 species have been placed in the ToL [10]. There are applications for the ToL: to help understand how pathogens become more virulent over time, how new diseases emerge, and to recognise species at risk of extinction [10,7]. One approach to building the ToL is to combine smaller trees into "supertrees". Phylogenetic trees have been created for relatively small sets of species [14]. These trees are then combined together into supertrees.

In 2003 Ian Gent, Barbara Smith, Christine Wu Wei, and myself reported the first constraint programming model for supertree construction [3]. This was essentially a proof of concept, showing that constraint programming could address this problem in principle although our implementation was somewhat inefficient. This has recently been re-implemented using a faster constraint programming toolkit (JChoco, a java constraint programming tool [5]) and has allowed us to look at larger problems and get a better idea of the limits of this encoding. Furthermore, with this new implementation we are able to demonstrate the flexibility of our model, something that should be expected when using a versatile technology such as constraint programming.

The remainder of this article is organised as follows. First, I reintroduce the problem of supertree construction and briefly present the constraint encoding of [3]. Next, I present a study that attempts to reproduce the results of building a relatively large supertree of sea birds, reported by Kennedy and Page in [6]. I then describe a richer version of the supertree problem, where ancestral dates are included within species trees [12] and show how the constraint model can be modified to address this. Finally, we look at what limits these models, what we might do to break through those limits, and then draw to a conclusion.

2 Previous Work

The problem is to combine leaf labelled species trees, where there is an intersection in the leaf labels of those trees. The trees must be combined whilst respecting all the arboreal relationships in each tree. An example of this is shown in Figure 1, as a rectangular cladogram displayed using Rod Page's TREEVIEW [9]. The two input trees are A and B from [6]. One of the first techniques for supertree construction is the OneTree algorithm of Ng and Wormald [8] and is based on the build algorithm of [1]. OneTree

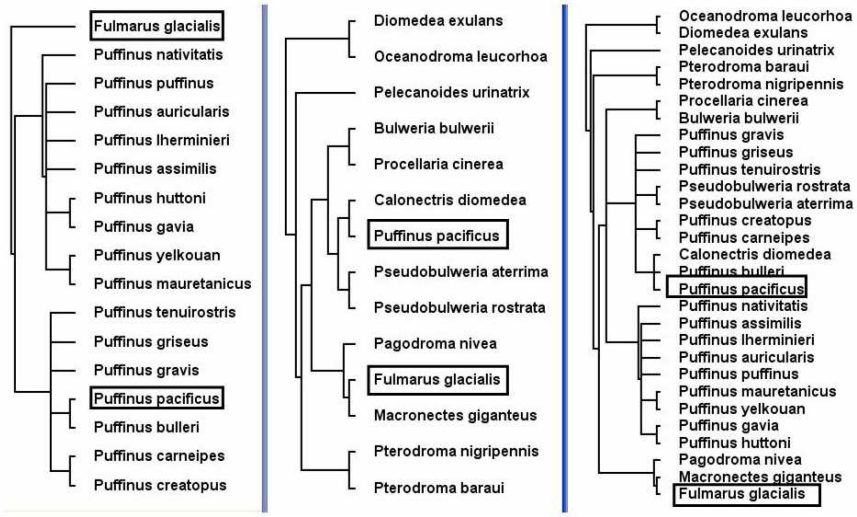


Fig. 1. Two species trees made up of sea birds, and on the right a supertree that combines both. Shared species are highlighted in boxes. The trees correspond to A and B in [6].

is based on the observation that in a tree any three leaf nodes define a unique relation with respect to their most recent common ancestor (mrca[1], such that $mrca(a,b)$ is the interior node furthest from the root that has both leaf nodes a and b as descendants. Given three leaf nodes (labelled a, b, and c) one of four relations must hold[2]:

$$(1) \quad mrca(a,b) > mrca(a,c) = mrca(b,c)$$
$$(2) \quad mrca(a,c) > mrca(a,b) = mrca(c,b)$$
$$(3) \quad mrca(b,c) > mrca(b,a) = mrca(c,a)$$
$$(4) \quad mrca(a,b) = mrca(a,c) = mrca(b,c)$$

This is shown pictorally in Figure 2. Using the terminology from [8] we can say that in (1), (2) and (3) we have the triples $(ab)c$, $(ac)b$, and $(bc)a$[3] and in (4) we have the fan (abc). Prior to applying the OneTree algorithm two (or more) species trees are broken up into triples and fans via the BreakUp algorithm [8], and the supertree is then constructed (if possible) using this as input.

[1] Note that mrca is sometimes refered to as lca, for least or lowest common ancestor.

[2] It is assumed labels a, b, and c are all different. $mrca(a,b)$ delivers an actual interior node in the tree and that if $mrca(a,b)$ is equal to x and $mrca(b,c)$ equals y, $x > y$ if node x exists at a greater depth in the tree than y, and $x = y$ if and only if x and y are the very same node. Note also that if relation (4) is omitted trees are forced to be binary.

[3] ... where $(xy)z$ can be read as "x is closer to y than z"

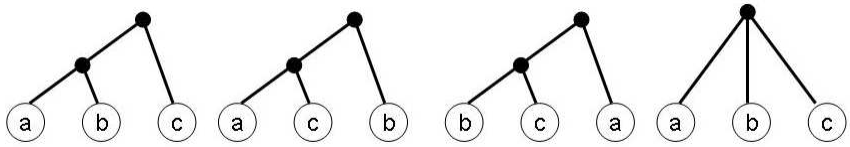


Fig. 2. The four possible relationships between three leaf nodes in a tree: i.e. the three triples $(ab)c$, $(ac)b$, and $(bc)a$, and the fan (abc).

In [3] we presented the first constraint programming model for this problem. This was based on the rather simple observation that any rooted tree is *ultrametric*[4]. That is, if interior nodes of a tree are labelled with their depth in that tree then any path from the root to a leaf node must be a strictly increasing sequence, and in [4] this is called a min-ultrametric tree. Further, in [4] it is proved that an ultrametric tree has an ultrametric matrix. In an ultrametric matrix M, for any three indices i, j, k where $i \neq j \wedge i \neq k \wedge j \neq k$ one of three relations must hold

$$M_{i,j} > M_{i,k} = M_{j,k}$$
$$M_{i,k} > M_{i,j} = M_{j,k}$$
$$M_{j,k} > M_{i,j} = M_{i,k}$$
$$M_{i,j} = M_{i,k} = M_{j,k}$$

In an ultrametric tree T, and its corresponding ultrametric matrix M, given two leaf nodes i and j in T, $mrca(i,j)$ will have the same value (and that might possibly be depth in that tree) as $M_{i,j}$. In [4] it is also proved that an ultrametric matrix has a corresponding ultrametric tree, and the proof given is constructive and is therefore an algorithm.

Our constraint encoding starts by producing an $n \times n$ matrix M of constrained integer variables, each with a domain 1 to $n-1$. Amongst the trees to be combined there are exactly n species and each species is mapped to an integer. The array M is symmetric such that $M_{i,j}$ is the same constrained integer variable as $M_{j,i}$ and all diagonal elements $M_{i,i}$ are preset to zero. An ultrametric constraint is blanketed across the array. By that I

[4] A metric on a set of objects is given by the assignment of a real number $d(x,y)$ to every pair of objects x and y such that $d(x,y)$ has the following properties:

- $d(x,y) > 0$ for $x \neq y$
- $d(x,y) = 0$ for $x = y$
- $\forall x,y\ [d(x,y) = d(y,x)]$
- $\forall x,y,z\ [d(x,y) \leq d(x,z) + d(y,z)]$ (triangular inequality)

To be ultrametric we have the additional property: $\forall x,y,z\ [d(x,y) \leq max(d(x,z), d(y,z)]$.

mean that for all i, j, k where $1 \leq i < j < k \leq n$ the following constraint is posted:

$$\begin{aligned} M_{i,j} > M_{i,k} = M_{j,k} \vee \\ M_{i,k} > M_{i,j} = M_{j,k} \vee \\ M_{j,k} > M_{i,j} = M_{i,k} \vee \\ M_{i,j} = M_{i,k} = M_{j,k} \end{aligned}$$

The species trees are then broken up, using the BreakUp algorithm [8], into triples and fans. If in a tree we have the fan $(s_0, s_1, ..., s_m)$, i.e. species s_0 to s_m have the same most recent common ancestor, then the set of ${}^mC_3$ 3-fans $\{(s_0, s_1, s_2), (s_0, s_1, s_3), ..., (s_0, s_1, s_m), ..., (s_{m-2}, s_{m-1}, s_m)\}$ are produced by our BreakUp algorithm. These triples and 3-fans are then used to break disjunctions in the above constraint. The M variables are then the decision variables, and a solution is found. Assuming a solution exists, the resultant supertree is constructed from the ultrametric matrix using the algorithm in chapter 17 of [4].

3 A Supertree of Sea birds

One obvious limiting factor of our constraint model is its sheer size. It generates $O(n^2)$ constrained integer variables and $O({}^nC_3)$ ultrametric constraints as above. A study was performed to determine just how far this model could be pushed. The model was re-coded in JChoco, a free java constraint solver [5], and can be downloaded from [13]. An attempt was made to reconstruct the supertree produced by Kennedy and Page [6]. The data set is seven species trees of sea birds, identified as A through to G. This is shown in Table 1. A table entry gives the number of species involved in a pair of trees, and along the diagonal the number of species in an individual tree. For example, combining tree A (17 species) with tree B (14 species) results in a supertree with 29 species. Therefore A and B have 2 species in common. A table entry in closed round brackets shows that the two trees are incompatible. In particular, trees A and C are incompatible, B and C are incompatible, C and G are incompatible, as are D and G. An entry of a dash (-) means that the data set was too large to model.

	A	B	C	D	E	F	G
A	17	29	(32)	47	-	31	46
B	29	14	(29)	42	-	30	40
C	(32)	(29)	20	50	-	34	(44)
D	47	42	50	30	-	44	(56)
E	-	-	-	-	90	-	-
F	31	30	34	44	-	16	(38)
G	46	40	(44)	(56)	-	(38)	30

Table 1. Size of species trees and supertrees for the 7 trees in [6]. The diagonal gives the size of individual trees. Off the diagonal is given the size of the supertree, in brackets if incompatible, and a dash if too large to model.

In the Auk paper [6] Kennedy and Page went back to the underlying evidence to successfully combine all of these trees. I cannot do that, and from Table 1 it can be seen that the best that can be done is to combine trees A, B, D and F[5]. The trees can be constructed in a number of ways, i.e. by adding A to B to get supertree AB, adding D to F to get supertree DF, and then combining supertrees AB and DF. This was in fact done, however there is a risk associated with this. Supertree AB is not unique, and neither is DF. Furthermore AB and DF may be incompatible! This was demonstrated by Sanderson, Purvis, and Henze in [11]. The input program was modified such that it takes as input a file containing names of trees to be combined, i.e. the encoding takes as input a forest. This forest is then broken up into triples and 3-fans before solving. The resultant supertree is shown in Figure 4 at the end of this paper. This took about 20 seconds to produce on a 3 GHz processor. The supertree has 69 species.

4 Ancestral Divergence Dates

In [12] ancestral divergence dates are added to the interior nodes of trees, where dates may be relative or explicit. The RANKEDTREE algorithm (proposed in [2]) takes as input two species trees where interior nodes are assigned integer values such that if the divergence of species A and B predates the divergence of species X and Y then the most recent ancestor of A and B will be assigned a value less than the most recent common ancestor of species X and Y.

This is trivial to incorporate into the constraint model. If we assume that trees have already been ranked, and some or all interior nodes have been labelled, then for each pair of species (X,Y) in the leaf set of a tree we get the value of $mrca(i,j)$ where i and j are the integer indices corresponding to species X and Y respectively. The constraint integer variable $M_{i,j}$ is then set to the value of $mrca(i,j)$ if and only if $mrca(i,j)$ is labelled. The tree is then broken up into its triples and 3-fans and these constraints are then used as disjunction breakers. In [13] this has been implemented as the RBuild (Rank Build) method.

In fact, we can go one step further. We associate a decision variable $D_{i,j,k}$ with each ultrametric constraint and post the following constraints:

$$\begin{aligned}
D_{i,j,k} = 1 &\leftrightarrow M_{i,j} > M_{i,k} = M_{j,k} \\
D_{i,j,k} = 2 &\leftrightarrow M_{i,k} > M_{i,j} = M_{j,k} \\
D_{i,j,k} = 3 &\leftrightarrow M_{j,k} > M_{i,j} = M_{i,k} \\
D_{i,j,k} = 4 &\leftrightarrow M_{i,j} = M_{i,k} = M_{j,k}
\end{aligned}$$

Therefore, rather than instantiate the variables in M we instantiate the disjunction breaking decision variables D. As a consequence of this, in a solution variables in M may have ranges of values. This is demonstrated in Figure 3. On the left and right we have two ranked species trees of cats taken from [12] (where cat names were given three-letter

[5] Table 1 can be considered as an adjacency matrix A of a graph where an entry $A_{i,j}$ not in closed round brackets means that there is an edge (i,j) signifying that tree i is compatible with tree j. In the corresponding graph of A the largest clique has 4 vertices and those vertices are A, B, D and F.

abbreviations). On the right we have one of the 17 possible resultant supertrees. Note that the most recent common ancestor of PTE and LTI is labelled with the range [6..9]. In total 7 of the 17 solutions contain interior nodes with ranges. Without this 30 solutions are produced. This addresses one of the issues raised in [12], i.e. to enumerate all supertrees compactly. Our constraint model has been further modified such that a

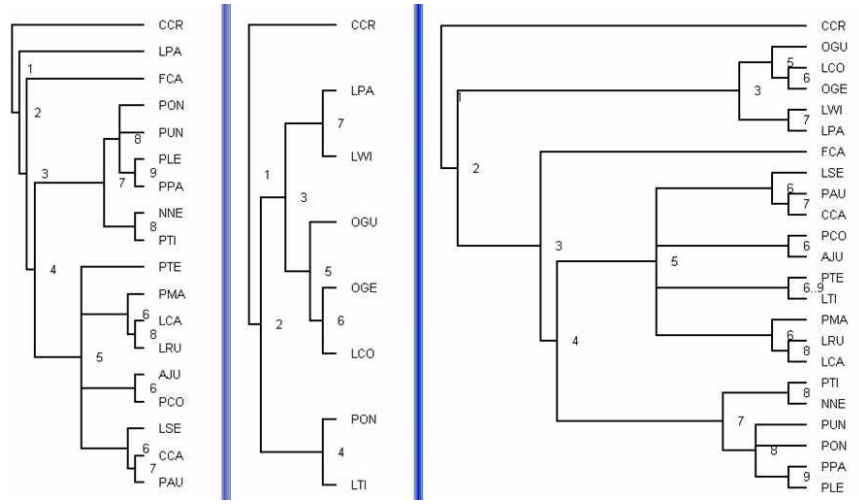


Fig. 3. Two ranked trees of cats taken from [12] and on the right one of the 17 possible supertrees, this one with the most recent common ancestor of PTE and LTI having a range of values.

penalty is taken when a decision variable $D_{i,j,k}$ takes a value 4, i.e. when a fan is selected. The penalties are then minimised to produce the supertree that contains the least number of fans. This might not be biologically sound but it has been implemented to demonstrate the versatility of the model. Again, this is available at [13].

5 Limitations, Future Work, Conclusion

Clearly the model is self limiting by its cubic size. There are $O(n^3)$ ternary constraints and the same number of variables when we address the optimisation problem (minimising fans). The largest trees we have built have about 70 species. One obvious next step is to make this model more compact, and this might be done by implementing a specialised ultrametric constraint that involves three variables. This constraint might propagate more efficiently than as at present (using toolkit primitives) and each of the constraints might take less space. However, we still have $O(n^3)$ of these constraints. Therefore the step after that might be to design an n-ary ultrametric constraint that takes as arguments the $n \times n$ array M.

Our model is now available, being re-implemented in java using JChoco [13]. We have been able to demonstrate the versatility of the constraint programming technol-

ogy, by taking a model that essentially does the same as OneTree, modified it to take a forest as input, dealt with ancestral divergence dates, been able to produce all solutions compactly, and address an optimisation problem (although this might not be biologically sound). However, the model is limited in what it can do by its sheer size, and this should be addressed soon.

Where to next? In [12] the authors pose the question "what common information is carried in all these supertrees?" I believe that constraint programming will be the technology to address this next challenge.

References

1. A.V. Aho, Y. Sagiv, T.G. Szymanski, and J.D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM J. Comput*, 10(3):405–421, August 1981.
2. O.R.P Bininda-Emonds. *Phylogenetic supertrees: Combining information to reveal the Tree of Life*. Springer, 2004.
3. Ian P. Gent, Patrick Prosser, Barbara M. Smith, and Christine Wu Wei. Supertree Construction with Constraint Programming. In *CP2003 (LNCS 2833)*, 2003.
4. Dan Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.
5. JChoco. http://choco.sourceforge.net/ A java library for constraint satisfaction problems, constraint programming, and explanation-based constraint solving, 2006.
6. M. Kennedy and R.D.M. Page. Seabird supertrees: Combining partial estimates of procellariiform phylogeny. *The Auk, A Quarterly Journal of Ornithology*, 119:88–108, 2002.
7. Gorgina M. Mace, John L. Gittleman, and Andy Purvis. Preserving the Tree of Life. *Science*, 300:1707–1709, 2003.
8. Meei Pyng Ng and Nicholas C. Wormald. Reconstruction of rooted trees from subtrees. *Discrete Applied Mathematics*, 69:19–31, 1996.
9. R.D.M. Page. TREEVIEW: An application to display phylogenetic trees on personal computers. *Computer Applications in the Biosciences*, 12:357–358, 1996.
10. Elizabeth Pennisi. Modernizing the Tree of Life. *Science*, 300:1692–1697, 2003.
11. M.J Sanderson, A. Purvis, and C. Henze. Phylogenetic supertrees: assembling the tree of life. *TREE*, 13:105–109, 1998.
12. C. Semple, P. Daniel, W. Hordijk, R.D.M. Page, and M. Steel. Supertree algorithms for ancestral divergence dates and nested taxa. *Bioinformatics*, 20(15):2355–2360, 2004.
13. CP Supertrees. http://www.dcs.gla.ac.uk/∼pat/supertrees Patrick Prosser's JChoco Supertree Builders.
14. TreeBASE. http://www.treebase.org/ TreeBASE: a database of phylogenetic knowledge, 2003.

Acknowledgements: Thanks to Pierre Flener for his help and encouragement.

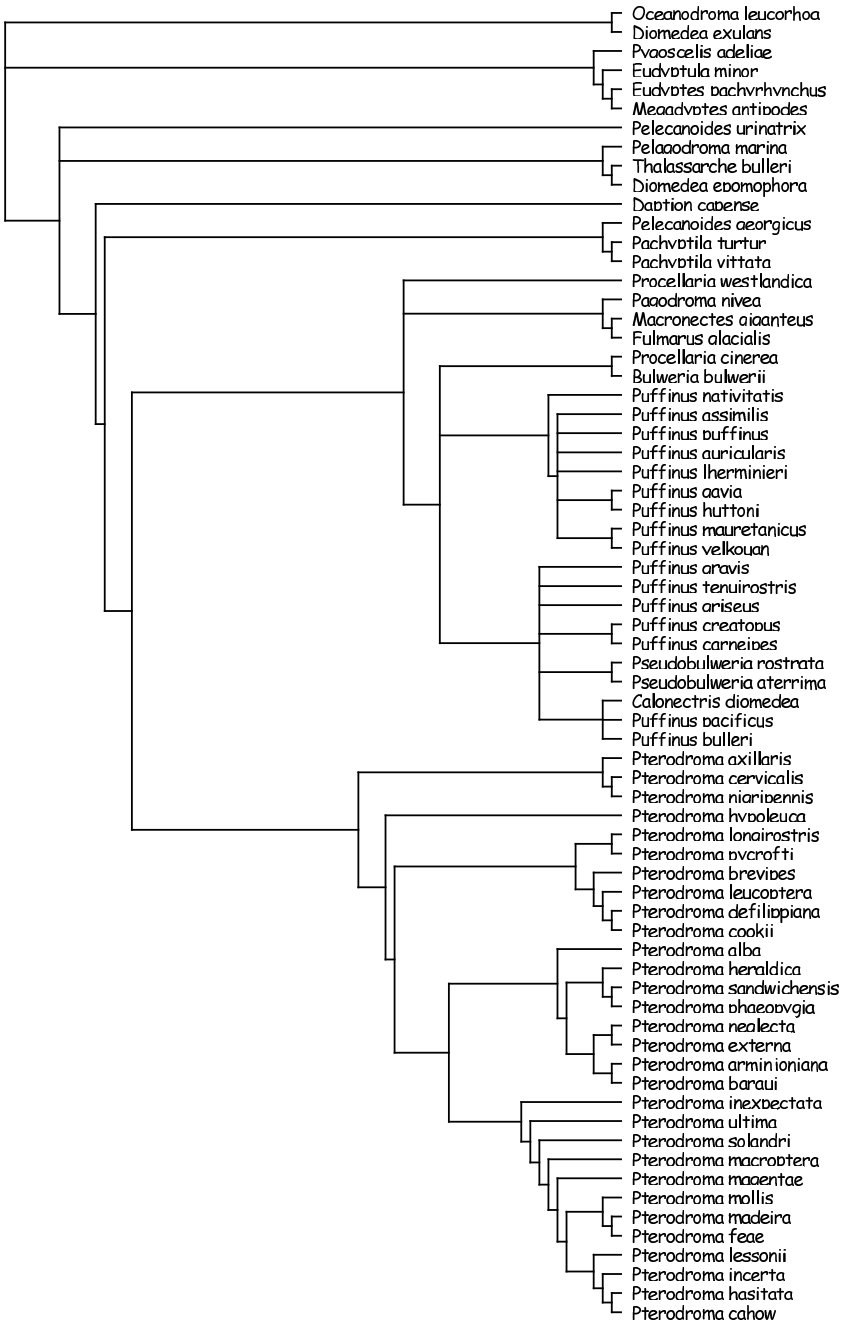


Fig. 4. The supertree constructed using the trees A, B, D, and F from the study in [6] via the forest build method in [13].

Counting Protein Structures by DFS with Dynamic Decomposition

Sebastian Will and Martin Mann

Chair for Bioinformatics, Institute of Computer Science, Albert-Ludwigs-University
Georges-Koehler-Allee, Geb. 106, D-79110 Freiburg, Germany
{will,mmann}@informatik.uni-freiburg.de

Abstract. We introduce depth-first search with dynamic decomposition for counting the solutions of a binary CSP completely. In particular, we use the method for computing the number of minimal energy structures for a discrete protein model.

1 Introduction

The number of minimal energy structures of proteins in a discrete model is an important measure, which is strongly related to protein stability. The enumeration of optimal and suboptimal structures has applications in the study of protein evolution and kinetics [12,20,27,26]. The prediction of protein structures in simplified protein models is a complex, NP-complete [6] combinatorial optimization problem that received lots of interest in the past, e.g. [16,28]. Importantly for our work here, it can be successfully modeled as Constraint Satisfaction Problem (CSP) [2,4].

Recently, counting solutions of a CSP and related problems gained a lot of interest over considering only satisfiability [1,9,18,22]. This is partly due to the increased complexity of counting compared to deciding on satisfiability [19]. For general CSPs and in particular for protein structure prediction, solving is NP-complete. However, the counting of CSP solutions is an even harder problem in the complexity class #P. This class was defined by Valiant [24] as the class of counting problems associated with nondeterministic polynomial time computations.

Standard solving methods in constraint programming like Depth-First Search (DFS) combined with constraint propagation are well suited for determining one solution, but leave room for saving redundant work when counting all solutions. Here, we present a method that is especially tailored for this case. Applied to the CSP formulation of structure prediction, it improves exhaustive counting and enumeration of optimal protein structures.

Basically, our new method *dynamically* decomposes the constraint (sub-)problems that emerge during the search into independent partial problems along connected components of the problem's associated constraint graph. Separate counting in the partial problems still allows to infer the number of solutions of the complete problem.

Instead of *statically* exploiting only properties of the initial constraint graph, dynamic strategies analyze the emerging constraint graphs during the search and employ their features. We believe this is a major advantage in many constraint problems. In

particular, if the initial constraint network is very dense (as in our structure prediction problem), static methods don't make an impact.

Decomposing into connected components and, more generally, utilizing the special structure of the constraint graph is discussed already for a long time. In the beginning, [13] proposed statically decomposing a CSP and solving the partial problems independently. As a more recent example, [9] introduced AND/OR search for solution counting, again this approach relies on static analysis of the constraint graph. To our knowledge, dynamic decomposition was discussed more thoroughly only for very special cases. [18] showed that adding this idea to counting models of 3-SAT by a Davis-Putnam procedure [8] results in a very successful new strategy. Similar ideas are discussed for SAT-solvers in [7].

As our main contribution, we demonstrate that the ideas of employing the graph structure dynamically are applicable to binary CSPs, even including certain global constraints, and are useful for constraint programming. In particular, this allows us to use the strategy in the complex problem of protein structure counting. Furthermore, we discuss several ideas going beyond previous approaches. For example, dynamic decomposition can yield a more compact representation of the solution space. We discuss how analyzing the constraint graph can further improve counting and how search strategies can be tailored in order to maximize the benefits from our strategy.

2 Dynamic Decomposition

2.1 Definitions

A *Constraint Satisfaction Problem (CSP)* is a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ of the variables $\mathcal{X} = X_1, \ldots, X_n$, their associated domains $\mathcal{D} = D_1, \ldots, D_n$, i.e. finite sets of values, and a finite set of constraints $\mathcal{C}$ on the variables in $\mathcal{X}$. A *solution* of the CSP is an assignment of each variable in $\mathcal{X}$ to one value in its associated domain. A variable X_i is *determined by* $\mathcal{D}$, iff its associated domain D_i in $\mathcal{D}$ is singleton. We call $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *solved*, iff each variable in $\mathcal{X}$ is determined by $\mathcal{D}$. The CSP is *failed*, iff at least one of the variables in $\mathcal{X}$ has an empty domain. A *subproblem of a CSP* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is a CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C}')$, where $\mathcal{C} \subseteq \mathcal{C}'$. A CSP $(\hat{\mathcal{X}}, \hat{\mathcal{D}}, \hat{\mathcal{C}})$ is called *partial problem of* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where $\hat{\mathcal{X}} \subseteq \mathcal{X}$ and $\hat{\mathcal{D}}$ and $\hat{\mathcal{C}}$ are restrictions of $\mathcal{D}$ and $\mathcal{C}$ to $\hat{\mathcal{X}}$, respectively. We call a CSP *n-ary*, iff each of its constraints is at most n-ary. For a constraint c, we denote by $X(c)$ the *set of variables of* c. The *constraint graph* of a binary CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is the undirected graph (V, E) defined by $V = \mathcal{X}$ and $E = \{(X_i, X_j) \in \mathcal{X}^2 | c \in \mathcal{C}, \{X_i, X_j\} \subseteq X(c), X_i \neq X_j\}$. Two partial CSPs $(\hat{\mathcal{X}}, \hat{\mathcal{D}}, \hat{\mathcal{C}})$ and $(\hat{\mathcal{X}}', \hat{\mathcal{D}}', \hat{\mathcal{C}}')$ of $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ are *independent*, iff $\hat{\mathcal{X}} \cap \hat{\mathcal{X}}' = \emptyset$ and there is no constraint c in $\mathcal{C}$, where $X(c)$ shares elements with $\hat{\mathcal{X}}$ and $\hat{\mathcal{X}}'$.

2.2 Counting DFS

The usual approach to counting solutions of a CSP is by DFS in combination with constraint propagation. As preparation to our approach, we present a recursive formulation, which we temporarily call Counting Depth-First Search (CDFS).

```
1: function CDFS(X, D, C)
2:     (D', C') ← PROPAGATE(X, D, C)
3:     if ISFAILED(X, D', C') then return 0
4:     else if ISSOLVED(X, D') then return 1
5:     else  c ← SELECT(X, D')
6:         return  CDFS( X, D', C' ∪ {c}) + CDFS( X, D', C' ∪ {¬c})
7:     end if
8: end function
```

In our formulation, $CDFS(\mathcal{X}, \mathcal{D}, \mathcal{C})$ yields the number of solutions to $(\mathcal{X}, \mathcal{D}, \mathcal{C})$. Note that the function performs full propagation of constraints to the domains (also, entailed constraints of $\mathcal{C}$ are removed in $\mathcal{C}'$) in line 2. The tests for failure and determination by the propagated domains $\mathcal{D}'$ are in line 3 and 4. Line 6 allows the algorithm an arbitrary branching selection; often one selects a variable X_i from $\mathcal{X}$ and a value $d \in \mathcal{D}_i$ and enumerates by $c = (X_i \equiv d)$. Finally, the solution count of each subproblem adds to the total number of solutions in line 7.

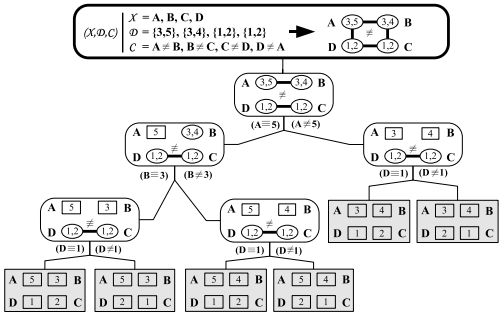


Fig. 1. DFS search tree traversed by CDFS.

We provide an example CSP and a corresponding search tree for CDFS solution counting in Figure 1. Each node corresponds to a propagated subproblem of the initial problem given in the root and is visualized as a constraint graph.

2.3 Dynamically Decomposing DFS

Even in the minimal example of Figure 1, the main problem of CDFS is visible. The partial problem on variables *C* and *D* is solved redundantly in each of the search branches. This could be saved due to the independence of the two partial problems on variables *A* and *B* and variables *C* and *D*. Our new method *Decomposing Depth-First Search (DDFS)* avoids such unnecessary work.

```
1: function DDFS(X, D, C)
2:     (D', C') ← PROPAGATE(X, D, C)
3:     if ISFAILED(X, D', C') then return 0
4:     else if ISSOLVED(X, D') then return 1
5:     else  s ← 1                                   ▷ initialize counter
```

```
 6:        𝔇 ← DECOMPOSE(𝒳, 𝒟′, 𝒞′)
 7:        for all (𝒳̂, 𝒟̂, 𝒞̂) ∈ 𝔇 do
 8:            c ← SELECT(𝒳̂, 𝒟̂)
 9:            s = s · ( DDFS(𝒳̂, 𝒟̂, 𝒞̂ ∪ {c}) + DDFS(𝒳̂, 𝒟̂, 𝒞̂ ∪ {¬c}) )
10:        end for
11:        return s
12:    end if
13: end function
```

The code differs from CDFS only in lines 5 to 11, which correspond to the decomposition into independent partial problems. In line 6, we completely decompose the propagated CSP $(\mathcal{X}, \mathcal{D}', \mathcal{C}')$ into its pairwise independent partial problems.

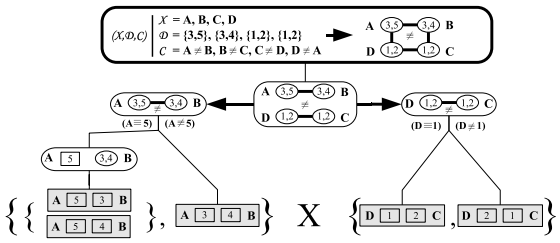


Fig. 2. Search tree traversed by DDFS.

Note that the independent partial problems correspond to the connected components in the constraint graph. Consequently, our decomposition can be computed in linear time by depth first traversal of the graph. As a technicality, we fuse all solved partial problems to an (arbitrary) unsolved partial problem. As a consequence, all remaining problems in $\mathfrak{D}$ are unsolved. In line 11, s is the product of the solution counts of all CSPs in $\mathfrak{D}$. Since $\mathfrak{D}$ is a complete decomposition of $(\mathcal{X}, \mathcal{D}', \mathcal{C}')$ into pairwise independent partial problems, s equals the number of solutions for $(\mathcal{X}, \mathcal{D}, \mathcal{C})$. Each solution of this CSP is a combination of a selection of one solution from each partial problem.

Using this extension the CSP in Figure 1 can be solved as given in Figure 2 with DDFS avoiding the redundant work. With only one decompositions and two branchings instead of five branchings the overall solution number can be determined.

Note that a simple modification of the counting algorithm yields a new way to retain the set of all solutions. Instead of adding and multiplying solution counts, we can build up a tree-like compact representation of the solution space. Examples are given at the bottom of Figure 2 and later in Figure 3c. Of course, the compression does not improve the theoretical worst-case space complexity. Nevertheless, the space savings are of equal size as potential reductions of the search tree by our method and can be large in practice. In order to finally enumerate the solutions, the compact representation is expanded.

2.4 Further Improvements

As a first important improvement, we can derive that a CSP has no solution as soon as one of its independent partial problems has no solution. This is also reflected in multiplying the solution counts. A simple improvement is to skip counting in further partial problems, whenever a partial problem returns no solutions.

By this, the order of the partial problems is critical for avoiding unnecessary work. Optimally, partial problems with high chance of failing are explored first. The ordering can be based on heuristics analogous to the variable selection for branching. Additional savings result from checking if all partial problems are satisfiable, before we start to count all solution of any partial problem.

The solution number for partial problems with an empty set of constraints can be derived directly without further decomposition or enumeration. In this case, the number of solutions can be determined as product of the domain sizes. This effect is already shown in Figure 2.

DDFS profits most from early and well balanced decompositions. Therefore, new strategies for variable and value selection are desirable that support good decomposition. Constraint graph based variable selection, e.g. detection of articulation points, can guide the variable selection and domain splitting instead of single value branching may lead to sturdy decompositions. Even deeper analysis of the constraint graph structure can guide the heuristics further. Many techniques are already investigated and proved beneficial for the static case [14,15,17,23]. For example, we could strive for the breaking of circles in the constraint graph in order to obtain a tree structure. Solutions of tree-structured CSPs can be enumerated more efficiently. Note that the detecting when the graph becomes a tree is for free if we already look for graph decompositions.

Albeit presented in this fashion, DDFS is not completely restricted to binary constraint graphs. Many widely used n-ary and global constraints (e.g. `AllDifferent`) can be used as well, if a suitable binarization is at hand [5,21]. The method can then employ the strong propagation of the global constraint and use the semantically equivalent set of binary constraints for checking dependencies in the constraint graph.

3 Application to Structure Prediction and Results

In [4], a constraint-based approach for exact structure prediction in the *HP-model* of the cubic and face-centered cubic lattice has been presented. In this simplified protein model, the amino-acids of the protein are classified into *hydrophobic (H)* and *polar (P)* ones and each is represented by a single point, its center of mass. It models water-soluble globular proteins. The folding of such proteins is mainly driven by hydrophic forces, that leads to the emergence of compact hydrophobic cores. A *structure* in this model is a placing of the H/P-monomers to nodes of the lattice (e.g. 3D-cubic lattice), such that successive monomers are lattice neighbors and each node is only occupied once (*self-avoidance*). For a structure the set of positions of all H-monomers is called its *H-core* and corresponds to the hydrophobic core of real proteins. The energy is calculated as shown in Figure 3a by counting HH-contacts. The example structure for the sequence HPPHPPHPHP in Figure 3b has an energy of -2.

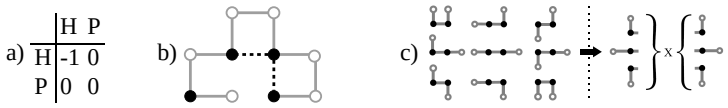


Fig. 3. a) Energy function b) Structure of square lattice HP-model (H-monomer: black, P-monomer: white, structure back bone: grey, HH-contact: dotted) c) Compression of the structure space representation for PHHP from 9 structures down to 3 by 3 partial structures.

The prediction of *optimal* structures (with minimal energy) can be formulated as CSP and was named *Constraint-based Protein Structure Prediction (CPSP)* [2,4]. It is a fast approach for enumerating all these structures for a given HP-sequence using DFS.

Its main idea is the pre-calculation of *compact H-cores*. These can be used in the approach since optimal structures tend to form maximally compact H-cores. The construction of compact H-cores is a hard problem by itself, which was solved using constraint-programming too [3,25]. Since the compact point sets of a given size are sequence-independent, they can be pre-calculated and used for the last sequence-specific part of CPSP. For the remaining task, it is necessary to search for self-avoiding walks with the restriction that H-monomers are placed in a given H-core. Therefore, we introduce a variable for each sequence position with lattice nodes as values. H-monomers are constrained to H-core positions, P-domains are left with a finite domain of non-H-core positions. The self-avoiding walk condition can be expressed by a global `AllDifferent` constraint and a sequence of neighbor constraints, which can be modeled as $X_i - X_{i+1} = N_i$. There, X_i represents the variable for the ith sequence position and N_i contains all possible lattice specific neighbor vectors[1].

CPSP is effectively solved using DFS and so CDFS for solution counting can be applied too. As mentioned before the number of optimal structures of a protein is an important measure. It provides information about the character of the energy landscape and degeneracy and can be used for their further investigation [11,10,12,20,27,28].

As discussed before, a semantically equal set of binary inequality constraints can be used to represent the global `AllDifferent` constraint in the constraint graph. DDFS was applied using problem specific heuristics in addition to node degree and articulation point identification. A first prototypical implementation uses ILOG Solver 6.1™. We present some results from this program in the following table.

test suite	branch	fail	time	pos. time	decomp.
T33	7.8	0.7	1.5	4.7	42
T54	7.7	0.9	1.7	5.2	26

We investigated two test suites T33/T54 with random HP-sequences of length 33/54 in the cubic lattice. To show the contraction of the search tree the ratio of branchings CDFS/DDFS is given in column *branch*. It can be reduced by decomposition up to a factor of 8 in average with the presented average number of decompositions. Due to a

[1] In practice, lattice positions and the neighborhood N are indexed by integers such that standard constraint solvers for finite domains over integers are applicable.

non-optimal partial problem ordering DDFS yields slightly more *fails* than CDFS. This can be avoided by the mentioned improvement of first checking for one solution for each partial problem before its complete investigation. The ratio of the mean time consumption of CDFS/DDFS in column *time* illustrates the reduced number of branching. Since the current implementation is not at all optimized, the time-behavior can certainly be improved. The time ratio in column *pos. time* is calculated using DDFS without versus with decomposition. It demonstrates that the possible speedup using DDFS for faster implementations is around 5 times. We expect further speedups and search tree reductions using better partial problem ordering and variable selection heuristics.

4 Discussion

We presented a general method Decomposing DFS (DDFS) for completely counting and enumerating the solutions of a binary CSP by dynamically exploiting decomposition of (sub-)CSPs. Furthermore, we demonstrated that the method can be generalized such that even global constraints can be used. As we could show, our strategy of dynamically decomposing the (sub-)problems into partial problems reduces the search tree significantly. Since partial problems can be efficiently detected using well established graph algorithms, this results in a speed up of the search. Beyond this, we discussed how the graph structure can guide the variable and value selection in order to achieve many balanced decompositions, e.g. by the identification of articulation points. Such considerations go beyond previous work on constraint graph decomposition.

The application of DDFS to the CPSP problem shows the large capabilities of the method. First results with a prototypic implementation already show a significant speedup. Improving our ability for counting and enumerating optimal structures has important implications for the investigation of protein evolution and the folding process.

We could give evidence that the more general approach of dynamically analyzing the constraint graph during the search and employing its special structure has a large potential for solution counting in constraint programming. To our conviction, exploring these possibilities even further is an interesting field for future research.

4.1 Acknowledgments

Martin Mann is supported by the EU project EMBIO (EC contract number 012835). Sebastian Will is partly supported by the EU Network of Excellence REWERSE (project reference number 506779).

References

1. Ola Angelsmark and Peter Jonsson. Improved algorithms for counting solutions in constraint satisfaction problems. In *Proc. of CP-2003)*, pages 81–95, Sep-Oct 2003.
2. Rolf Backofen and Sebastian Will. Fast, constraint-based threading of HP-sequences to hydrophobic cores. In *Proc. of CP'2001*, volume 2239, pages 494–508, 2001.
3. Rolf Backofen and Sebastian Will. Optimally compact finite sphere packings — hydrophobic cores in the FCC. In *Proc. of CPM2001*, pages 257–272, 2001.

4. Rolf Backofen and Sebastian Will. A constraint-based approach to fast and exact structure prediction in three-dimensional protein models. *J. of Constraints*, 11(1):5–30, 2006.
5. Roman Bartak. Theory and practise of constraint programming. In *CPDC2001*, pages 7–14. Wydavnictvo Pracovni Komputerowej, Gliwice, Poland, 2001.
6. B. Berger and T. Leighton. Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete. *JCB*, 5(1):27–40, 1998.
7. Armin Biere and Carsten Sinz. Decomposing sat problems into connected components. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:191–198, 2006.
8. Elazar Birnbaum and Eliezer L. Lozinskii. The good old davis-putnam procedure helps counting models. *Journal of AI Research*, 10:457–477, 1999.
9. Rina Dechter and Robert Mateescu. The impact of AND/OR search spaces on constraint satisfaction and counting. In *CP'2004*, 2004.
10. Ken A. Dill and Hue S. Chan. From levinthal to pathways to funnels. *Nature Structural Biology*, 4(1):10–19, 1997.
11. A. R. Dinner, A. Sali, and M. Karplus. The folding mechanism of larger model proteins: Role of native structure. *Proc. Natl. Acad. Sci. USA*, 93:8356–8361, 1996.
12. Christoph Flamm, Ivo L. Hofacker, Peter F. Stadler, and Michael T. Wolfinger. Barrier trees of degenerate landscapes. *Z.Phys.Chem*, 216:155–173, 2002.
13. Eugene C. Freuder and Michael J. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. In *Proc. of IJCAI-85*, pages 1076–1078, 1985.
14. G. Gottlob, N. Leone, and F.Scarcello. A comparison of structural CSP decomposition methods. *Artif. Intell.*, 124(2):243–282, 2000.
15. G. Gottlob, N. Leone, and F. Scarcello. Hypertree decomposition and tractable queries. *J. of Computer and System Sciences*, 64(3):579–627, 2002.
16. Peter Grassberger. Sequential Monte Carlo methods for protein folding. In *NIC Symposium 2004*, Juelich, oct 2004.
17. Philippe Jégou and Cyril Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *LSIS*, 2002.
18. R. J. Bayardo Jr. and J. D. Pehoushek. Counting models using connected components. In *Proc. of the 7th Nat'l Conf. on AI*, 2000.
19. Gilles Pesant. Counting solutions of CSPs: A structural approach. In *IJCAI-05*, page 260, 2005.
20. A. Renner and E. Bornberg-Bauer. Exploring the fitness landscapes of lattice proteins. In *2nd. Pacif. Symp. Biocomp.* Singapore, 1997.
21. F. Rossi and V. Dhar. On the equivalence of constraint satisfaction problems. In *ECAI90*, pages 550–556. Stockholm, Sweden, 1990.
22. Dan Roth. On the hardness of approximate reasoning. *Artif. Intelligence*, 82(1-2):273–302, 1996.
23. Marko Samer. Hypertree-decomposition via branch-decomposition. In *IJCAI'05*, pages 1535–1536, 2005.
24. Leslie G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.
25. Sebastian Will. Constraint-based hydrophobic core construction for protein structure prediction in the face-centered-cubic lattice. In *Proc. of PSB 2002*, pages 661–672, 2002.
26. M. Wolfinger, S. Will, I. Hofacker, R. Backofen, and P. Stadler. Exploring the lower part of discrete polymer model energy landscapes. *Europhysics Letters*, 74(4):725–732, 2006.
27. Richard Wroe, Erich Bornberg-Bauer, and Hue Sun Chan. Comparing folding codes in simple heteropolymer models of protein evolutionary landscape: robustness of the superfunnel paradigm. *Biophys J*, 88(1):118–31, 2005.
28. K. Yue, K. M. Fiebig, P. D. Thomas, H. S. Chan, E. I. Shakhnovich, and K. A. Dill. A test of lattice protein folding algorithms. *PNAS*, 92(1):325–9, 1995.

Author Index