Proceedings of WCB05
Workshop on
Constraint Based Methods for Bioinformatics

Rolf Backofen and Agostino Dovier

October 5, 2005, Sitges (Spain)

Program Committee

Rolf Backofen (co-chair), Jena Univ., Germany
Pedro Barahona, Univ. Nova de Lisbona, Portugal
Mats Carlsson, SICS, Uppsala, Sweden
Alessandro Dal Palù, Udine Univ., Italy
Agostino Dovier (co-chair), Udine Univ., Italy
Francois Fages, INRIA Rocquencourt, France
Enrico Pontelli, NMSU, USA
Sebastian Will, Jena Univ., Germany

External referees

Luca Bortolussi, Anke Busch, Carla Piazza.

Preface

Bioinformatics is a challenging research area where every major contribution can have significant impact on medicine, agriculture, and industry. Among the various problems tackled in this area are those related to the recognition, analysis, and organization of DNA sequences, those related to the structure of macromolecules (like the prediction of the spatial form of a polymer, given the sequence of monomers constituting it, or the detection of common RNA sequence/structure motifs), and those related to biological systems simulations (for metabolic or regulatory networks). All these problems can be naturally formalized using constraints over finite domains or intervals of reals. Moreover, Biological systems simulations can be easily designed using concurrent constraint programming.

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, nine of them have been judged deserved to be presented to the workshop.

We would like to thank the program committee and external referees for their contribution in reviewing the submitted papers. A special thank to Alessandro Dal Palù who greatly helped us in the editing process.

Rolf Backofen
Agostino Dovier

Contents

RNA Secondary Structure Design Using Constraint Handling Rules

Maryam Bavarian and Veronica Dahl

Logic and Functional Programming Group
Dept. of Computing Science
Simon Fraser University
Burnaby, B.C., Canada

Abstract. The need for processing biological information is rapidly growing, owing to the masses of new information in digital form being produced at this time. Old methodologies for processing it can no longer keep up with this rate of growth. We present a novel methodology for solving an important bioinformatics problem which has been proved to be computationally hard: that of finding an RNA sequence which folds into a given structure. Previous solutions to this problem divide the whole structure into smaller substructures and apply some techniques to resolve it for smaller parts, which causes them to be slow while working with longer RNAs (more than 500 bases). We prove that by using a set of simple CHR rules we are able to solve this problem and obtain approximate but still useful solution in O(n) time. We expect the results we present to be applicable, among other things, to in vitro genetics, by enabling the scientists to produce RNAs artificially from sequences; and to drug design, which typically progresses backwards from proteins to RNAs and finally to DNAs.

Keywords: RNA secondary structure, RNA secondary structure design, constraint handling rules, Watson-Crick base pairs.

1 Introduction

Over the past decade there has been a dramatic increase of collection rates for biological data, making the need for resorting to AI methods even more acute. Simultaneously, the intersection between logic programming and constraint reasoning has been maturing into extremely interesting methodologies, most notably Constraint Handling Rules, or CHR [15,16]. We have applied these methodologies first to human language processing, through implementing Property Grammars (a linguistic formalism based on constraints between sentence constituents rather than on the traditional notion of phrase structure [6,8]) in CHR [11], and through a parsing system for balanced parenthesis [7] and then to cognitive sciences, through generalizing these results into a general cognitive theory of concept formation [13] with applications to cancer diagnosis [5,4], to medical report interpretation [27] and to concept extraction [12].

The application to molecular biology of AI methods such as logic programming and constraint reasoning constitutes a fascinating interdisciplinary field which, despite being relatively new, has already proved quite fertile. Some examples of applying logic programming techniques to molecular biology problems are the description and analysis of protein structure [24], protein secondary structure prediction [23], drug design [19] and predicting gene functions [18].

In this paper, we present a novel methodology for solving one of the state-of-the-art problems in biology which is the problem of *RNA secondary structure design* namely finding an RNA sequence which folds into a given structure. Using the power beneath Constraint Handling Rules and adding some heuristics enabled us to solve this problem in linear time.

In the following sections, we first provide some background knowledge about the biological concepts involved. Next, we present a set of grammar rules for RNA secondary structure prediction first introduced in [3] and based on these rules we present our own solution for the inverse problem, namely RNA secondary structure design through CHR rules. Then we go through our results and compare them with other existing methods, and finally, we discuss possible future work and improvements to our system.

2 Background

2.1 Biological Concepts Invovled

RNA (ribonucleic acid) is a chemical found in cells which codes for amino acid sequences, serving as intermediate in the synthesis of protein[1]. Each RNA molecule is made up of four different compounds called nucleotides, each noted with one of the letters A (Adenine), C (Cytosine), G (Guanine) and U (Uracil). This strand of nucleotides is folded onto itself by pairings of the nucleotide A with U and C with G which are called *Watson-Crick* or *canonical base pairs*. Pairing also might happen between G and U but this is not very frequent. The nucleotides are often referred to, as bases and each pairing is called a *base pair*. The structure made by these base pairs is called *RNA secondary structure* which contain a number of structural patterns such as helix, hairpin loop, bulge loop and internal loop, etc (Figure 1).

One of the widely occurring structural motifs in RNAs is called *pseudoknot* which has been proved to play an important role regarding the functions of RNA. A simple pseudoknot is formed by pairing of some of the bases in a hairpin loop that are supposed to stay unpaired, with bases outside the loop (Figure 2). Adding generalized pseudoknots to the problem of RNA secondary structure prediction makes it NP-hard. however, by using some heuristics and restricting the format of pseudoknots, this problem can be solved in polynomial time ($O(n^4)$) [14]. Most methods that predict RNA secondary structure, for simplicity, disregard the possibility of having pseudoknots. According to these methods, for every two base pairs: $pair(i, j)$ and $pair(i_1, j_1)$, if i_1 is greater than i then j_1

[1] `www.nigms.nih.gov/news/science\_ed/chemhealth/glossary.html`

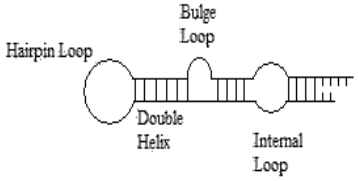


Fig. 1. Common motifs in RNA secondary structure are: hairpin loop, bulge loop, internal loop, etc.

should be less than j. This assumption prevents the formation of pseudoknots in the result structure.

The problem of RNA secondary structure prediction consists of determining which secondary structure will be adopted by a given sequence of nucleotides. Several methods have tackled this problem so far within two main approaches. The two most common approaches include finding *minimum free energy* [29] and *sequence comparison* [20,17].

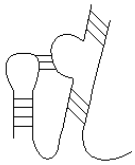


Fig. 2. A simple pseuodoknot

In this paper we focus on the problem of RNA secondary structure design, i.e.: given a secondary structure, we find a sequence which folds onto that structure. The motivations for solving this problem include the achievements made in *in vitro genetics* (done outside the living organism) field. Nowadays, scientists are capable of replicating any RNA in a test tube [9] which would eventually help in finding new paths for drug design or can have industrial use [2].

The RNA-SSD algorithm [2] by Andronescu et al. and RNAinverse from Vienna RNA package [26] have addressed the same problem. The RNA-SSD is based on stochastic local search procedure (SLS). This algorithm first assigns an initial sequence by using these probabilities: $P_G = P_C, P_A = P_U = 1/2 - P_C$. Next, it divides the structure into smaller substructures and then applies the SLS procedure on those substructures. After combining the substructures, as they are not independent from each other, the SLS procedure should be applied again. RNAinverse also initializes the sequence but for each step instead of applying SLS procedure, it will change a base randomly and then computes the free energy

for the result. If the free energy of the new sequence is reduced, it accepts the new sequence otherwise the change is not accepted. These two algorithms assume that there are no pseudoknots in the initial structure.

2.2 Constraint Handling Rules

Constraint handling rules (CHR) provide a simple bottom-up framework which has been proved to be useful for algorithms dealing with constraints [15,16]. Because logic terms are used, grammars can be described in human-like terms and are powerfully extended through (hidden) logical inference. The format of CHR rules is:

```
Head ==>Guard|Body
```

`Head` and `Body` are conjunctions of atoms and `Guard` is a test constructed from (Prolog) built-in or system-defined predicates. The variables in `Guard` and `Body` occur also in `Head`. If the `Guard` is the constant "true", then it is omitted together with the vertical bar. Its logical meaning is the formula $(Guard \rightarrow (Head \rightarrow Body))$ and the meaning of a program is given by conjunction. There are three types of CHR rules:

- *Propagation rules*, which add new constraints (body) to the constraint set.
- *Simplification rules* ,which also add as new constraints those in the body, but remove as well the ones in the head of the rule.
- *Simpagation rules*, which combine propagation and simplification traits, and allow us to select which of the constraints mentioned in the head of the rule should remain and which should be removed from the constraint set.

The rewrite symbols for the first two rules are respectively: `==>`, `<=>` and for simgation rules, the notation is `Head1\Head2<=>body`. Anything in `Head1` remains in the constraint set and anything in `Head2` is removed from the constraint set.

3 Our solution: chrRNA

To solve the problem of RNA secondary structure design, we made use of a set of simple Context Free grammar rules proposed in [3] for RNA secondary structure prediction:

$$S \rightarrow cSg|gSc|aSu|uSa|gSu|uSg$$

$$S \rightarrow aS|gS|uS|cS$$

$$S \rightarrow a|g|u|c$$

$$S \rightarrow SS$$

This set of relatively simple rules present a profound insight of the secondary structure of RNAs and have been used as a basis for some RNA secondary structure prediction algorithms [3]. However, the grammar generated by this set of rules is ambiguous and the reason for ambiguity is that there might be more

than one derivation for the same sequence, e.g. for the short sequence *cccg*, we have:

$$S \Rightarrow cS \Rightarrow ccSg \Rightarrow cccg$$

$$S \Rightarrow cSg \Rightarrow ccSg \Rightarrow cccg$$

We use CHR to implement this grammar in order to exploit the bottom-up characteristic of CHR rules, as well as keep track of ambiguous readings with no special overhead.

At the first step, we translated these rules into the format of CHR rules, adding only one extra rule to this set which ascertains that the bases right after a loop would not be able to be paired together. In our system the structure of the desired RNA (input data) is shown in the format of CHR constraints, e.g. for expressing that the base number 1 and the base number 43 in the sequence are paired together, we add the constraint `pair(1,43)` or if base number 3 is unpaired, the corresponding constraint would be `upair(3)`. One advantage of our input format to the input format used by RNAinverse and RNA-SSD is that it is capable of accepting pseudoknots in the input structure.

After constructing the CHR rules, the problem becomes that of assigning nucleotides to each position given the input constraints. One trivial solution is: randomly assign one of the Watson-Crick pairs (or GU pair) to each base pair and one of the four nucleotides (A, C, G and U) to the unpaired bases.

The problem with the random solution is that, although we follow the structure to build the sequence, since there is no preference criteria to select between the existing pairs, we might end up with a sequence that may not actually fold into the input structure. As mentioned earlier, the number of GC pairs has an important role in stabilizing a certain structure. For instance, if we assign base G to position 1 and base U to position 43 and if we have a base C in position 42, in the end, the structure might be `pair(1,42)` instead of `pair(1,43)`.

The solution we offer to this problem uses CHR rules combined with the probabilities that are believed to govern the proportion of base pairs within RNA sequences. We calculated these probabilities by comparing several RNAs together from Gutell lab's comparative RNA website [10], a database of known RNA secondary structures. After comparing 100 test cases with various length from 100 to 1500 bases, we found the following probabilities for each base pair:

$$P_{CG} = 0.53, P_{AU} = 0.35, P_{GU} = 0.12$$

The other probabilities which are of interest are the probabilities for an unpaired base to be one of A, C, G, or U. The results are as follows:

$$P_G = 0.18, P_A = 0.34, P_C = 0.27, P_U = 0.20$$

There is a simple explanation for the above probabilities. Base G can be paired with both base C and base U and as the number of GC pairs is important for RNA stabilization, the probability of an unpaired base to be base G would become smaller. Base U is not as important as base G for stabilization but still

can be paired with both base G and base A and that is why it has the second smallest probability to stay unpaired. Both base A and base C can only be paired with one nucleotide but as most base C's tend to be paired with base G's, the probability of an unpaired base to be base C would become less than base A.

3.1 Adding the probabilities

Inserting the probabilities into the rules is the most challenging part of the implementation. In our implementation, it is done by generating a random variable in the guard section of the rules, which is the only part that accepts Prolog predicates. This random variable then is tested according to the probabilities: for instance for the following rule if the random variable `I` is less than 0.53, it will assign a GC pair to `pair(X1,Y1)`. The `L` parameter in `s` contains the list of bases already added to the sequence and `find(M,N,I)` assigns a base pair to `M` and `N` based on the random variable `I`.

```
pair(X1,Y1)\s(X,Y,L)<=>X1 is X-1,Y1 is Y+1,random(I),find(M,N,I),
   append([X1:M,Y1:N],L,L1) | s(X1,Y1,L1).
```

As mentioned before, the other two algorithms do not accept pseudoknots in the input structure. However, our implementation provides the capability of handling structures with pseudoknots through the following rule. This rule finds pairs of nucleotides which do not meet the assumption made by common methods of RNA secondary structure prediction (section 2.1) and separates them into two strings and at the same time, according to the probabilities, assigns them one of the possible base pairs.

```
pair(X,Y)\pair(X1,Y1)<=> X < X1, X1 < Y, Y1 > Y,random(I),
           find(M,N,I)| s(X1,X1,[X1:M]),s(Y1,Y1,[Y1:N].
```

4 Results and comparisons

We have tested the chrRNA program with 100 RNA secondary structures (not containing pseudoknots) from the Gutell database [10]. The test cases were selected from both Prokaryote (Archaea and Bacteria) and Eukaryote. The results show that although the chrRNA program seems relatively simple compared to RNA-SSD and RNAinverse, it is still comparable in efficiency and even better in some cases.

For evaluation, firstly we transform the secondary structure of each test case to constraint format by using a Java based program and then run the chrRNA program for each of those. Using the probabilities makes the result of each run on a test case to be a completely new sequence. The sequence produced by chrRNA is then fed into an RNA folding server [28] and the output would be the structure of this new sequence. The two structures (the initial input structure and the output structure of the RNA folding server) are then compared together and the differences are marked.

The average error is estimated to be about 18%, meaning that 18% of the nucleotides might be paired with a nucleotide in a wrong position (in the original structure they might be either unpaired or be paired with another nucleotide). We have compared the performance of our program with two available software systems: RNA-SSD and RNAinverse. RNA-SSD has been implemented [1] and is available online[2] through a web application and RNAinverse is one of the programs inside Vienna RNA package[3].

RNAinverse and chrRNA were both executed on a Pentium 1.4GHZ Centrino with 512MB of RAM. Table 1 shows the results of the comparisons between chrRNA, RNAinverse and RNA-SSD based on their response time and accuracy (the results are divided according to the length of RNA sequences[4]). For

	Algorithm	Error	Time
a)	chrRNA	16%	2–5sec
	RNAinverse	1.6%	2–900sec
	RNA-SSD	0%	2–3600sec
b)	chrRNA	18%	5–7sec
	RNAinverse	2.2%	900–1800sec
	RNA-SSD	–	no answer
c)	chrRNA	19%	7–300sec
	RNAinverse	–	no answer
	RNA-SSD	–	no answer

answer

Table 1. Comparison between chRNA, RNA-SSD, and RNAinverse: a) for sequences of less than 300 bases, b) sequences between 300 and 500 bases and c) sequences between 500 and 1500 bases

sequences longer than 500 bases, RNAinverse run time seems to be increasing exponentially, e.g. for a sequence consisting of 500 bases, the run time was 30 minutes while for a 700 base sequence, it took 7 hours to finish. In contrast, for chrRNA the run time only varies in the order of seconds. The reason for such a difference between the expected running time of chrRNA and that of the other two algorithms is that in chrRNA we have only used rules which are executed linearly (O(n)) and there are no time consuming computations. The comparisons show that although for shorter RNAs (less than 500 bases), RNA-SSD and RNAinverse produce more accurate results than the results by chrRNA and the total run time is quite insignificant, however they fail to produce even a rough answer for longer sequences. According to this, it can be concluded that chrRNA

[2] `http://www.rnasoft.ca`

[3] `http://www.tbi.univie.ac.at/\char126ivo/RNA/`

[4] As indicated above, RNA-SSD is only available as a web application and was not able to give any answer for sequences longer than 300 bases within their time limit.

outperforms the other two for longer sequences by giving an approximate but still useful result within an acceptable amount of time.

5 Discussion and future work

We implemented an elegantly simple while powerful system based on CHR rules to predict RNA secondary structure design. According to the results, the two existing methods, RNA-SSD and RNAinverse, generate more precise results while designing shorter sequences by utilizing other non trivial rules such as energy rules into their methods which are not very easy to implement in our case of CHR rules. However, our method performs better when dealing with longer sequences, given that we do not need to break the initial structure into substructures, which for longer sequences results in either slowdown or failure. In addition to this, the capability of handling pseudoknots, makes this system more powerful in comparison to the other existing methods.

For improving our program, we are currently trying to add more rules to the limited set of trivial rules introduced in Section 3 . These rules find the positions inside the structure that might lead to error and solve them by assigning new bases. Some of these conditions are shown in Figure 3. These rules have improved our program for sequences of less than 300 bases from 16% error rate to 8%.

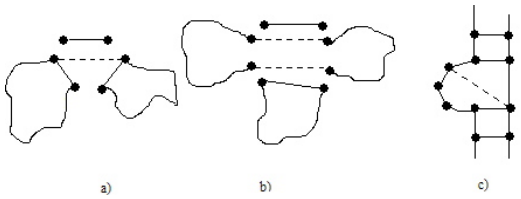


Fig. 3. Conditions to avoid

Another future improvement to our method would consist in also incorporating more non-trivial rules which are not easy to predict, through some machine learning methods such as *Inductive Logic Programming* (ILP) [21]. ILP is a relatively new methodology of automatically eliciting hidden knowledge with the help of a computer [22]. ILP has already been used in molecular bioinformatics [25]. Some examples of this application include gene function prediction [18], protein secondary structure prediction [23] and drug design [19]. By using this methodology and adding new rules, this system could probably show better results for shorter sequences as well as for longer ones.

We expect the results presented here to be invaluable for in vitro genetics, by enabling the scientists to produce RNAs virtually from sequences; and for drug design, which typically progresses backwards from proteins to RNAs to DNAs.

References

1. Andronescu, M., Aguirre-Hernandez, R.,Condon, A., Hoos, H. H. : RNAsoft: a suite of RNA secondary structure prediction and design software tools. Nucleic Acids Res., July 1, (2003) 31(13): 3416–3422
2. Andronescu, M., Fejes, A.P., Hutter, F., Hoos, H.H.,Condon, A. : A new algorithm for RNA secondary structure design. Journal of Mol. Bio. 336(3) (2004) 607–624
3. Baldi, P. : Bioinformatics: the machine learning approach. Cambridge, Mass. MIT Press (1998)
4. Barranco-Mendoza, A. : Stochastic and Heuristic Modelling for Analysis of the Growth of Pre-Invasive Lesions and for a Multidisciplinary Approach to Early Cancer Diagnosis. Ph.D. Thesis, Simon Fraser University, Burnaby,BC (2005)
5. Barranco-Mendoza, A., Persaoud, D.R. and Dahl, V. : A property-based model for lung cancer diagnosis. (accepted poster) RECOMB (2004) 27–31
6. Bes, G., Blache, P. : Proprieties et analyse d'un langage. In Proc. TALN99 (1999)
7. Bes, G., Dahl, V., Guillot, D., Lamadon, L., Milutinovici. I. and Paulo, J. : A parsing system for balanced parenthesis in NL texts. Poster and Demo at the Lorraine-Saarland Workshop on Prospects and Advances in the Syntax/Semantics Interface, Loria-Nancy (2003)
8. Blache, P., Balfourier, J. M. : Property Grammars: a Flexible Constraint-based Approach to Parsing. In Proc. IWPT-2001
9. Burke, J.M., Berzal-Herranz,A. : In vitro selection and evolution of RNA: applications for catalytic RNA, molecular recognition and drug discovery. FASEB J. **7** (1993) 106–112
10. Cannone J.J., Subramanian S., Schnare M.N., Collett J.R., D'Souza L.M., Du Y., Feng B., Lin N., Madabusi L.V., Muller K.M., Pande N., Shang Z., Yu N., and Gutell R.R. : The Comparative RNA Web (CRW) Site: An Online Database of Comparative Sequence and Structure Information for Ribosomal, Intron, and other RNAs. BioMed Central Bioinfo. 3:2 (2002)
11. Dahl, V., Blache, P. : Directly Executable Constraint Based Grammars. In Proc. Journees Francophones de Programmation en Logique avec Contraintes, Angers, France (2004) 149–166
12. Dahl, V., Blache, P. : Extracting noun phrases from arbitrary text using Property Grammars. (in preparation)
13. Dahl, V., Voll, K. : Concept Formation Rules: An Executable Cognitive Model of Knowledge Construction. In Proc. NLUCS'04 (2004) 28–36
14. Deogun, J. S., Donis, R., Komina, O., Ma. F. : RNA secondary structure prediction with simple pseudoknots. 2nd Asia-Pacific Bioinformatics Conference (2004) **29** 239 – 246ÊÊ
15. Fruhwirth, T. : User-Defined Constraint Handling. Poster, ICLP 93, Budapest, Hungary, MIT Press (1993)
16. Fruhwirth, T. : Theory and Practice of Constraint Handling Rules. Special Issue on Constraint Logic Programming (P. Stuckey and K. Marriot, Eds.), Journal of Logic Pro. 37(1-3) (1998) 95-138
17. Hofacker, I. L., Bernhart, S., Stadler, P. : Alignment of RNA base pairing probability matrices. Bioinformatics (2004) **20** 2222–2227
18. King, R.D. : Applying inductive logic programming to predicting gene function. AI Mag. **25-1** (2004) 57–68
19. King, R. D., Muggleton, S., Lewis, R.A., Sternberg, M. J. E. : Drug design by machine learning. Proc. Natl. Acad. Sci. **89** (1992) 11322–11326

20. Mathews, D., Turner, D. : Dynalign: An algorithm for finding the secondary structure common to two RNA sequences. Journal of Molecular Biology (2002) **317(2)** 191–203
21. Muggleton, S. : Inductive logic programming. Proceedings of the ILP-91 international workshop, Vianna de Castelo, Portugal, 2-4 March (1991)
22. Muggleton, S. : Inductive logic programming. Academic Press, London (1992)
23. Muggleton, S.,King,R.D., Sternberg.,M.J.E. : Protein secondary structure prediction using logic-based machine learning, Protein Eng. **5** (1992) 647–657
24. Rawling, C.J., Taylor. W.R., Nyakairo, J., Fox, J., Sternberg, M.J.E. : Reasoning about protein topology using the logic programming language PROLOG. J. Mol. Bio. **3-4** (1985) 151–157
25. Schulze-Kremer, S. : Molecular bioinformatics: algorithms and applications. Water de Gruyter, New York (1996) 142–157
26. Hofacker, I.L., Fontana, W., Stadler, P.F., Bonhoeffer, S., Tacker, M., Schuster, P. : Fast Folding and Comparison of RNA Secondary Structures. Monatshefte f. Chemie **125** (1994) 167–188
27. Voll, K. : Automatic interpretation of radiological reports. (thesis work on progress).
28. Zuker, M. : Mfold web server for nucleic acid folding and hybridization prediction. Nucleic Acids Res. 31 (13) (2003) 3406–3415
29. Zuker, M., Stiegler, P. : Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. Nucleic Acids Research (1981) **9** 133–148

Constraint Satisfaction Problems on DNA Strings

Luca Bortolussi[1] and Andrea Sgarro[2]

[1] Dept. of Maths and Computer Science, University of Udine, 33100 Udine, Italy.
bortolussi@dimi.uniud.it
[2] Dept. of Maths and Computer Science, University of Trieste, 34100 Trieste, Italy.
sgarro@units.it

Abstract. We explore the possibility of designing a constraint-based algorithm for constructing subsets of DNA words satisfying given constraints (the so-called DNA word design problem). In this direction, we use symbolic representation of sets of strings, and define a propagation algorithm on these representations. Some preliminary results are presented, together with several open problems.
Keywords*: DNA word design, Code Construction, Constraint Programming.*

1 Introduction

In the last ten years, a new computational paradigm emerged from a very uncommon place, i.e. wet labs of biologists. The fact that DNA contains all the basic information necessary to build very complex living organisms convinced Adlemann that it could also be used as a computational entity. In his milestone paper of 1994 [1], he proposed a computational model based on very simple manipulations of DNA that can be performed in a wet lab. This model is Turing-complete and bases its power on the massive parallelism achievable by using DNA. Moreover, one of the basic operations performed is the hybridization of complementary DNA strings. Specifically, DNA strings are oriented strings over the alphabet $\Sigma = \{a, c, g, t\}$, where a-t and c-g are complementary letters. Two of such strings are said to be complementary if they have the same length and if one can be generated by reversing the other and complementing each of its letters. Physically, complementary DNA strings can hybridize, i.e. they can attach one to the other, forming the famous double helix. Actually, hybridization can occur also between strings that are not perfect complements, but close to it. In DNA computations, data is coded by short strings of DNA in such a way that hybridizations occurring determine the output of the "algorithm" [8]. Therefore, one of the main concerns is to avoid that "spurious" hybridizations occur, leading straight to the so-called *DNA word design* problem.

DNA word design (cf. [7]) consists of identifying sets of DNA strings of a given length, called *DNA codes*, satisfying some constraints, usually that two of them have Hamming distance and reverse complement Hamming distance

greater than a certain threshold. Formally, given $x, y \in \Sigma^n$ and letting y^{RC} be the reverse complement of y, the *reverse complement Hamming distance* between x and y, $d_H^{RC}(x, y)$, is defined as the Hamming distance between x and y^{RC}, $d_H^{RC}(x, y) = d_H(x, y^{RC})$, where the Hamming distance is the usual one, counting the number of positions where the two strings differ.

In particular, the main concern of DNA word design is to identify maximal set of strings satisfying the above mentioned constraints. There is some theoretical work [5] that gives upper and lower bounds to the dimensions of such codes, and also some algorithms constructing such codes [10], that are based on stochastic local search.

The aim of the ongoing work we are presenting here is to give a constraint-based algorithm to build such sets of strings. We will tackle two different versions, both the optimization problem, i.e. find the maximal code, and the constraint satisfaction problem (CSP), i.e. find a set of a given size satisfying the constraints. At this stage of the work, we have made some working hypothesis that simplify the problem. The main one replaces the reverse complement Hamming distance by the simpler *reverse Hamming distance*. Given two strings $x, y \in \Sigma^n$ and indicating by y^R the reverse of y, this new distance is simply defined as $d_H^R(x, y) = d_H(x, y^R)$. A more detailed discussion of the properties of such distance can be found in [9]. Actually, this assumption is painless, as in [5] it is showed that there is a (constructive) one to one correspondence between these codes and the ones making use of the reverse complement distance.

The main problem we have to face, however, is related to the huge dimension of the search space into play. In fact, reasoning for simplicity with the CSP version of the problem, if we are looking for a code of size m, then we have m variables, whose domain is the space of all strings of length n, of size $|\Sigma|^n = 4^n$. Therefore, not only the space of possible solutions has a dimension of the order of 4^{nm} (which for the reasonable values of $n = 10$ and $m = 50$ is around 10^{300}, a gigantic size!), but also there is the problem of finding a way to (over)represent the feasible domains of the variables during the computation, as direct memorization is out of discussion. To tackle this problem we use a symbolic representation of these feasible sets of strings, pretty much in the line of how boolean functions are represented in symbolic model checking. Though these representations, which make use of direct acyclic graphs, can be exponentially large, they seem to perform quite well in practice. In addition, they automatically allow for an effective propagation algorithm that achieves global consistency. In Section 2 we introduce in more detail such mechanisms.

Needless to say, the enormous dimension of the search space pushes for a theoretical analysis of the problem in order to introduce as many constraints as possible. Moreover, an efficient strategy is also needed to choose the branches of the search tree during its exploration. The solution found so far are presented in Section 3. Finally, in Section 4 we present some results and we discuss the current weak points of this approach, outlining some possible solutions.

2 Symbolic Propagation

There are two versions of the DNA code design problem for strings of length n: the optimization one, looking for maximal size codes, and the CSP, looking for a code of fixed dimension m. Both these problems are parametrical w.r.t. the minimum distance D we impose between two strings belonging to it. This distance is

$$d(x, y) = \min\{d_H(x, y), d_H^R(x, y)\}, \tag{1}$$

i.e. the minimum between Hamming and reverse Hamming distance. In particular, in the CSP, we have m variables whose starting domain is the set Σ^n.

In the Introduction we mentioned that one of the main problems in attempting to use constraint-based methods for DNA code construction is the huge size of the domain of each variable. Therefore, there is the problem of storing somehow the feasible values that can be assigned to variables at every point of the search tree. Moreover, we need a way to perform efficiently the propagation of constraints introduced whenever a variable gets instantiated to an element of its domain. These constraints are of the form $d(X, s) \geq D$, where s is the value of the newly instantiated variable.

The techniques we use to tackle both these problems are taken from symbolic model checking [4]. In particular, we use a compact representation of set of strings by means of a particular kind of direct acyclic graphs (DAG) that resembles closely OBDD [3], and that will be called Generalized Decision Diagram (GDD). The very basic idea is that we can identify a set of strings S with its characteristic function $S = \chi_S : \Sigma^n \to \{0, 1\}$. Then we build a rooted DAG representing this function, where nodes are divided into two categories, terminal and non-terminal. There are just two terminal nodes, one labeled with 0 and one labeled with 1. Every non-terminal, or internal, node is labeled by a number from 1 to n. Every internal node has $|\Sigma| = K$ edges, labeled by the K different letters of the alphabet ($K = 4$ for DNA, but the following description is general). Edges always go from nodes with label i to nodes with label $j > i$ or to terminal nodes. Moreover, there is only one node with label one, and it is the root.

The main property of these graphs is that the concatenation of edge labels of a path from the root to the terminal node 1 represent a string belonging to the set S, i.e. evaluating its characteristic function to 1. Concatenating the label $a \in \Sigma$ of an edge exiting from a node with label i, correspond to assign a to the ith letter of the string being constructed.[3] On the other side, the labels of all paths leading to node zero correspond exactly to strings not belonging to S.

A necessary request in order to make sense out of the previous definition is that these GDD graphs must be in canonical form, where there are no redundant nodes (with all edges pointing to the same node) and no duplicated nodes (two

[3] We adopt the convention that if an edge goes from a node i to a node $j > i$, with $j - i > 1$, all position in the string between $i + 1$ and $j - 1$ are set to a wild character $*$, so that a path corresponds more precisely to a string in the augmented alphabet $\Sigma \cup \{*\}$.

nodes with the same label and with edges pointing to the same nodes). This form is unique and can be computed efficiently. Proof and algorithm are a straightforward adaption of the classical ones in [3].
The crucial feature of these representations is that, though containing an exponential number of nodes in the worst case (w.r.t. the number of levels or of different labels of internal nodes), they usually behave well, and have a small and tractable size. Moreover, there exists an efficient algorithm for constructing the GDD G representing the intersection of the sets accepted by two GDD G_1 and G_2, whose complexity is bounded by the product of the dimension of G_1 and G_2 (but can be made more efficient in practice by implementation tricks, cf. [2]). This algorithm is also a straightforward adaption of that presented by Bryant in [3] in the context of boolean functions.
In Figure 1 we show such a GDD for the set of binary strings of length 6 at Hamming distance $d \geq 3$ from the string `000000`. In general, all GDD representing set of strings at distance (either Hamming or reverse Hamming) D or more from a given string s have a shape similar to this GDD.

We use GDD for keeping track of the feasible domain of each non-instantiated variable at each level of the search tree. Every time we choose a branch in the search tree we instantiate a variable, that is to say, we add a string s to the current code. For that string, we construct the GDD for the set of strings at distance $d(X, s) \geq D$. This GDD is constructed by intersecting the two GDD representing the set of strings at Hamming distance at least D from s, and the set of strings at reverse Hamming distance $d_H^R(X, s) \geq D$. Then this resulting GDD is intersected with the GDD representing the feasible domain before the branching. The outcome of this computation is a GDD representing exactly the domain of the non-instantiated variables, given all the constraints (we have constraints of the form $d(X, Y) \geq D$, for every variable $X < Y$). Therefore, the use of symbolic graphical representations for the the variable domains not only allow a compact representation of exponentially big sets, but also, through the intersection algorithm, achieves a propagation which is *globally consistent*.

The problem with GDD is that there is no guarantee that their size stays small. In theory, some of them can have size exponential in n. Luckily, the experimental results run by combining together GDD representing constraints of the form $d(X, s) \geq D$ show that these intersections behave well, and never explode combinatorially. In particular, the GDD corresponding to the constraints introduced by the addition of a single string to the code all have the same, fixed, size and shape.[4] When we start combining these GDD together, to represent the intersection of the corresponding sets, the size of the GDD product rises, while the number of strings belonging to the intersection decreases. After combining a certain number of basic GDD, dependant on the length of the strings, the dimension of the intersection becomes sufficiently small in order to have a compact representation. In every case, the dimension of the bigger GDD in this process remain quite small. We conjecture that this depends on the fact that the

[4]The complementary sets of Hamming spheres, for a fixed threshold, are all isomorphic.

intersection of complementary sets of Hamming spheres, all with same radius, has a sufficiently high internal structure, and thus can be compressed efficiently.

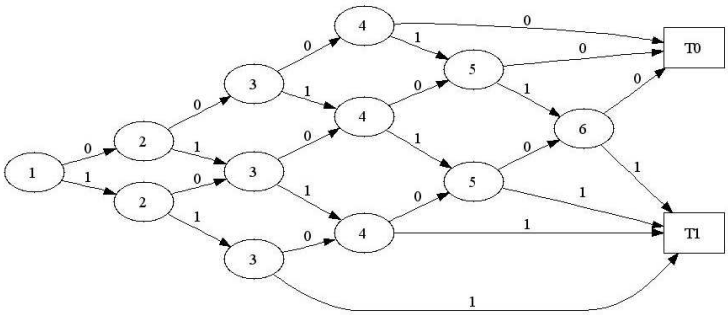


Fig. 1. A generalized OBDD representing the set of binary strings at Hamming distance $d \geq 3$, from the string 000000. Note that or the binary alphabet, GDD are exactly OBDD.

3 More Constraints and Heuristics

The code construction problem has many symmetries and, due to the big size of the search space, we have to break as many as we can. First of all, each set S can appear in all its $|S|!$ permutations. To avoid this, we have to introduce the lexicographical order between strings, and to impose the constraints $X_i < X_j$ for $i < j$.

In addition, given a code $\mathcal{C}$, there are a lot of equivalent codes obtained by transforming $\mathcal{C}$ using isometries of the string space, i.e. functions that are distance-preserving. In the case of our distance (1), these transformations must preserve the relation between conjugate indexes, i.e. couple of indexes whose sum is $n+1$ (for strings of length n). This is due to the fact that these couples of indexes are connected by the reverse transformation.

To break these symmetries, we should introduce constraints guaranteeing that a code $\mathcal{C}$ appears in just one possible way, i.e. forbidding all subsets $\mathcal{C}'$ isometric to $\mathcal{C}$. However, the identification of such set of constraints is still an open problem. What we do for now is fixing the minimum reverse Hamming distance of a word in the code from itself, and then set the value of the first variable to the smallest string in the lexicographical order with such a property. Moreover, if this minimal distance is $2K$ (reverse Hamming distance is always even[5]), then we impose

[5]This is very easy to see: consider, for simplicity, a string of even length, say $2n$, and write it uv, with $|u| = |v|$. Then $d_H^R(uv, uv) = d_H(uv, v^R u^R) = d_H(u, v^R) + d_H(v, u^R) = d_H(u, v_R) + d_H(v^R, u) = 2d_H(u, v^R)$.

the additional constraints $d_H^R(X, X) \geq 2K$, for all variables X. In this direction, another interesting open question is if every maximal code $\mathcal{C}$ contains a palindromic word, i.e. if the minimum self-reverse distance has to be always zero. This seems plausible, as these strings are the less committing in terms of the constraints they impose. In fact, for a palindromic word x, $d_H(x, y) = d_H^R(x, y)$, for each $y \in \Sigma^n$, hence the sets $d_H(x, y) \geq D$ and $d_H^R(x, y) \geq D$ coincide, and their intersection has the biggest possible cardinality.
At the moment, we do not use a symbolic representation of these added constraints, so our propagation algorithm achieves only a local consistency w.r.t. all constraints into play. Despite this, these constraints are exploited in the algorithmic procedure used to select the next string in a branching point. In fact, to choose a new word, we have to look at the paths terminating to node 1 in the current GDD. This can be done efficiently by doing a depth-first-search traversal of the tree that is the unfolding of the portion of the GDD containing these paths. Essentially, these new constraints become heuristics to prune parts of this tree during its exploration.

Another crucial point is the strategy used in the choice of the next word in a branching point. In fact, we want to find quickly a good solution, in order to make pruning effective. The pruning is executed if the size of the biggest code found is greater than the sum of the dimension of the constructed partial code and the dimension of the set of feasible strings (i.e. the dimension of the set accepted by the GDD, which can be counted simply by traversing it).
The strategy we use for the selection of the next string is based on the observation that in a code with minimum distance D, the minimum distance relative to most of its words is also D. Therefore, we select new words first from the subset of feasible strings at minimum distance from the partial code. Another interesting open question is if limiting the branching to this subset still guarantees that an optimal code will be eventually found.

4 Results and Future Work

We implemented our algorithm in **C**, and consequently run several tests. In particular, we tried to look for maximal codes of words of length between 4 and 10 in DNA alphabet, setting a time limit of one hour.
In all these cases, the algorithm finds very quickly a good solution, i.e. a code of pretty high size. Unfortunately, in one hour it never finishes the exploration of the search tree, even for strings of length 4. Moreover, it never finds a better solution than the initial one, even if it discovers other codes of the same size. Some results can be found in Table 1. Unfortunately, there is no literature about code construction w.r.t. distance 1, so we cannot compare easily our method with other approaches.

We also compared our program with $CLP(FD)$ of SICStus Prolog for the easier task of constructing Hamming codes. The results of the SICStus computations are taken from Dovier et alt. [6], and some comparisons can be found in

string length	dist. threshold	size of the solution found	time
4	2	40	0,18 sec
4	3	7	0,03 sec
6	3	59	99,00 sec
6	4	19	0,12 sec
6	5	6	0,03 sec
8	4	113	16,19 sec
8	5	30	4,57 sec
8	6	10	2,47 sec
10	6	58	35,10 sec
10	7	19	52,20 sec
10	8	9	105,18 sec

Table 1. Size of the best code found and time needed to find it, for different string lengths and thresholds for the simplified DNA word design problem.

Table 2. Here we can see that our engine runs around 50 times faster than the one of SICStus Prolog, though it preserves the same pattern of performance: where SICStus fails to find an answer in reasonable time, also our program suffers the same problem.

Taking a closer look to the behaviour of the search, we discovered that the main problem is that the pruning is not very effective. This means that the algorithm has to go very deeply in the tree to realize that a branch cannot contribute with a bigger code than the one found so far. Therefore, to create a more powerful pruning procedure, we need to work out a more clever estimate of the maximum code size, given a partial one.
In addition, the size of the actual search tree is too big to have any hope of exploring it all. The only chance to reduce its size is to find other constraints imposing the uniqueness of the maximal code. The combination of these new constraints and of more powerful pruning heuristics may be able to shrink the

string length	distance threshold	size of the code	existence of a solution	time in SICStus	time in our engine
6	3	8	Y	0,0	0,0
6	3	9	N	562,49	0,5
8	5	4	Y	0,0	0,0
8	5	5	N	3,85	0,03
10	3	64	Y	5,71	0,22
10	5	8	Y	0,05	0,01
10	5	9	Y	668,00	14,40

Table 2. Comparison of the performances between SICStus Prolog and out engine for the construction of Hamming codes.

portion of the tree to be visited to a small enough size. It is not clear, however, if, even with such additions, the algorithm would finish the exploration in a reasonable running time or if simply the task of using enumeration procedures for solving code construction problems is hopeless.

Nevertheless, there is a different and promising direction which we are currently beginning to explore: we are abandoning the target of a complete exploration of the search tree, and trying to integrate some stochastic ingredients allowing to visit just a small portion of the space, characterized by having an high probability of containing the optimal solution. Similarly, we could try to mix the branch and bound schema with local probabilistic search procedures. In this way, we may be able to create an algorithm giving good codes (even if not necessarily the optimal ones) in a reasonable time.

References

1. L. Adlemann, "Molecular Computations of Solutions of Combinatorial Problems". *Science*, Vol. 266, pp. 1021–1024, November 1994.
2. K. Brace, R. Bryant and R. Rudell, "Efficient implementation of a BDD package". *Proc. of the 27th ACM/IEEE conf. on Design Automation*, pp 40–45, 1991.
3. R. Bryant, "GraphBased Algorithms for Boolean Function Manipulation". *IEEE Transactions on Computers*, Vol. C35, No. 8, pp. 79–85, August 1986.
4. E. Clarke, O. Grumberg and D. Peled, *Model Checking*, MIT Press, 2000.
5. A. Condon, R.M. Corn, A. Marathe, "On Combinatorial DNA Word Design". *J. Computational Biology*, Vol. 8:3, pp. 201-220, 2001.
6. A. Dovier, A. Formisano, E. Pontelli, "A comparison of Constraint Logic Programming over Finite Domains and Answer Set Programming in tackling hard combinatorial problems", `http://www.di.univaq.it/~formisano/CLPASP/`.
7. G. Mauri, C. Ferretti, "Word Design for Molecular Computing: A Survey", *9th Int. Workshop on DNA Based Computers, DNA 2003*, 37-46 (electronic edition), 2003.
8. N. Pisanti, "A survey on DNA computing". *EATCS Bulletin* No. 64, pp. 188–216, 1998.
9. A. Sgarro and L. Bortolussi, "Codeword Distinguishability in Minimum Diversity Decoding". *Submitted to IEEE Trans. on Information Theory.*
10. D. Tulpan H. Hoos and A. Condon, "Stochastic Local Search Algorithms for DNA Word Design". *8th Int. Workshop on DNA Based Computers* , 2003.

Constraint Logic Programming for modeling a biological system described by a logical network

Fabien Corblin[1,2], Eric Fanchon[2], and Laurent Trilling[1]

[1] IMAG-LSR, Université Joseph Fourier BP 53, 38041 Grenoble Cedex 9, France
fabien.corblin@imag.fr
laurent.trilling@imag.fr
[2] LCM, Institut de Biologie Structurale Jean Pierre Ebel,
CEA-CNRS-Université Joseph Fourier,
41, rue Jules Horowitz, 38027 Grenoble Cedex 1, France
eric.fanchon@ibs.fr

Abstract. Adhesion between endothelial cells is an example of a biological system one would like to model. It is involved in the control of leukocyte migration across the endothelium of blood vessels. Some of the molecules participating to this process are known and a biochemical network can be drawn, but knowledge is still incomplete: the value of kinetic parameters are not known and other proteins and genes participating in adhesion have probably yet to be discovered. In order to tackle such problems and make progress, tools are needed that work at a qualitative level and that provide a large flexibility to allow biologists to ask various questions about the model.
Our approach is based on the following elements: (i) we use Thomas-Snoussi theory to abstract a system of ordinary differential equations into a discrete network, and an extension which considers so called singular states; (ii) a formal representation of this network using Constraint Logic Programming is developped. We show that constraints allow, with a single formal description of the model and the evolution rules, to perform various tasks: to infer and to check properties from observations and to perform qualitative simulations.

1 Introduction - Modeling objectives

With the development of high-throughput projects the quantity of molecular level data is exploding. It is now clear that biology is entering a new era in which all these molecular components have to be assembled into a *system* in order to reach new levels of understanding.

In general terms, our goal is to formalize 'verbal models' or, stated differently, build a formal model from a word description of a biological phenomenon. This means in practice that the knowledge is incomplete, that most information is not precise but qualitative, and that we may have to deal with several hypotheses. In such a state of partial knowledge we view modeling as a tool to formalize different competing hypotheses and explore their consequences; to help

in interpreting new data and use data to discriminate among competing models; to infer parameters and to devise maximally informative experiments. In short we look for rigorous methods to reason about models and data in the context of incomplete knowledge on complex systems.

One of our interests is the integration of biochemical reactions and genetic regulatory interactions in a single unified framework. It is possible in the case of genetic networks to describe the regulatory interactions by logical (or discrete) equations [7,8] without explicit reference to Ordinary Differential Equations (ODEs). The situation is different in the case of biochemical or signal transduction networks because the types of reaction are more diverse (phosphorylation, complexation, transport, *etc.*). So, although differential equations are not well suited to this knowledge level (where parameters are unknown), they are nevertheless useful as they can be transformed, at least in some cases, into a discrete model with the same logical structure. This allows a qualitative analysis of the dynamics and the exploration of the properties of a given model.

As suggested in [1], such a formal description of the biological system can be easily exploited via a Constraint Logic Programming (CLP) implementation. The advantages of the CLP approach are (i) that the implementation is expressed in a very similar way to the formal specification, thus guaranteeing the correctness of the implementation, (ii) that many different queries can be easily asked to this formal specification due to its logical form, for example: queries equivalent to simulation as well as queries equivalent to inference of model parameters in a context of incomplete knowledge.

These principles are illustrated by the study of endothelial cell-cell adhesion [5]. Once a logical formalization of the discrete model is at hand, we explore the properties of the system, and in particular we try to infer parameter values from the knowledge about the behavior of the system. Each behavioral information being a constraint that can be added to the constraints already known.

After a short introduction to the biological phenomenon, we describe the biological pathway and its components, then we give the ODE system describing this pathway. The theory of Thomas and Snoussi ([7,8]) leading to Asynchronous Multivalued Logical Models (AMLM) and the extension taking into account 'singular states' and sliding modes is introduced in section 3. We explain in section 4 how such logical description of regulatory models is constructed with the help of CLP language Prolog IV [2]. We give in section 5 an example of query.

2 The biological system

The phenomenon of cell-cell adhesion and its control by the cell is rather complex and our knowledge about it is far from complete. Blood vessels are lined with a monolayer of endothelial cells that form a barrier between blood and underlying tissues. Junctions between endothelial cells (adherens junctions) are composed of Vascular Endothelium cadherin proteins (VE-cadherin for short) embedded

in the plasmic membrane and β-catenin proteins bound to the cytoplasmic side of VE-cadherin. The migration of leukocytes through the monolayer breaks the junctions, causes clivage of VE-cadherins and consequently β-catenin are released in the cytoplasm [5]. The restoration process of the adherens junctions after this perturbation is the focus of our study. The biochemical structure of the system is informally illustrated in Figure 1.

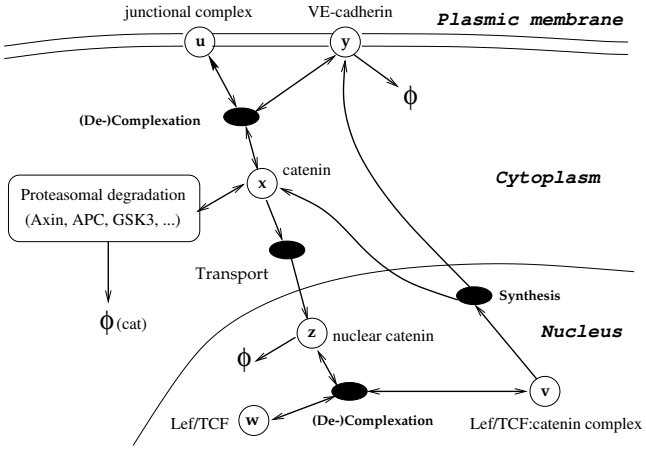


Fig. 1. Schematic and informal representation of the endothelial cell-cell adhesion system. This graph stresses the overall architecture of the system. Circles represent chemical species and black ovals chemical reactions. ϕ represents a degradation process. The molecules are distributed over three cellular locations: cytoplasm, nucleus and plasmic membrane. The concentration variables (x, y, z, u, v and w) are defined by the pairs (concentration variables, verbal definition) as follows: (x, cytoplasmic unphosphorylated β-catenin), (y, monomeric VE-cadherin in the membrane), (z, β-catenin in the cell nucleus), (u, complex of β-catenin with the VE-cadherin), (v, complex of Lef/Tcf and β-catenin), (w, Lef/Tcf transcription factor). Single-headed and double-headed arrows denote irreversible and reversible reactions, respectively.

From the graph of chemical reactions it is possible [4] to derive a system of ODEs giving the rate of variation of the concentration of each chemical species. After making several simplifying assumptions which we do not describe here (see [4]), we can isolate the following subsystem of 2 coupled variables x and z giving the temporal evolution of these variables (where the other four variables can be determined when a solution of the subsystem is known):

$$\begin{cases} \dot{x} = k_{\mathrm{x0}} + k_{\mathrm{x1}}.\sigma^{+}(z, s_{\mathrm{z1}}) - k_{\mathrm{t}}.\sigma^{+}(x, s_{\mathrm{x1}}) \\ \qquad -(m_{\mathrm{x1}} + m_{\mathrm{x2}}.\sigma^{+}(x, s_{\mathrm{x2}})).x \\ \dot{z} = k_{\mathrm{t}}.\sigma^{+}(x, s_{\mathrm{x1}}) - m_{\mathrm{z}}.z \end{cases} \tag{1}$$

Where $\sigma^+(x, s)$ represents an increasing sigmoid function with threshold s, the k parameters represent production kinetic constants, and the m parameters represent degradation kinetic constants. The order between the two thresholds for x is unknown and consequently two cases have to be considered: $s_{\mathrm{x1}} < s_{\mathrm{x2}}$ and $s_{\mathrm{x2}} < s_{\mathrm{x1}}$. The full system contains an additional threshold s_{z2} and the order between s_{z1} and s_{z2} must also be considered.

3 Thomas-Snoussi theory and singular state

Thomas [8] has developped a logical description for genetic regulatory networks called Asynchronous Multivalued Logical Models (AMLM). It allows to analyse *qualitatively* the dynamics of such networks. The logical equations are derived from Piecewise Linear Differential Equations (PLDEs) of the form:

$$\dot{x_j} = h(x_1, \ldots, x_i, \ldots) - m_j.x_j \tag{2}$$

giving the evolution of the concentration of the protein j, where h is a *sum* of products of (positive or negative) step functions depending on several variables x_i (possibly including x_j itself), and m_j is the degradation coefficient of the protein j. Equations 1 can be brought to this form by replacing sigmoids functions (σ) by step functions ($\mathfrak{s}$) defined as follows: $\mathfrak{s}^+(\mathsf{x}, \mathsf{s_x}) = 1$ if $\mathsf{x} \leq \mathsf{s_x}$ and 0 otherwise (with $\mathfrak{s}^-(\mathsf{x}, \mathsf{s_x}) = 1 - \mathfrak{s}^+(\mathsf{x}, \mathsf{s_x})$), $\mathsf{s_x}$ being the discrete value associated to the threshold s_{x}.

Now, following Thomas [8], we define discretization operators. If for example, a real variable a has two thresholds ($s_{\mathrm{Sup}}, s_{\mathrm{Inf}}$ with $s_{\mathrm{Sup}} > s_{\mathrm{Inf}}$) it is abstracted in a discrete variable $\mathsf{a} = d(a)$ as follows:

$\mathsf{a} = 0 \Leftrightarrow a < s_{\mathrm{Inf}}$
$\mathsf{a} = 1 \Leftrightarrow s_{\mathrm{Inf}} < a < s_{\mathrm{Sup}}$
$\mathsf{a} = 2 \Leftrightarrow s_{\mathrm{Sup}} < a$

In the framework of AMLM, a discrete state S of the system is defined by a vector (x_1, ..., x_j, ...) of discrete concentrations and a path is a sequence of states. The logical equations define the focal state (trend of the system) FS associated to each state S. Each component FS_i is given by $\mathsf{FS}_i = f_i(\mathsf{S})$, where the functions f_i are sums of terms: products of an integer coefficient and of step functions.

If the value of only one variable differs between FS and S, this variable changes value by one unit in the direction of FS. If n variables differ between FS and S, n transitions are considered (instead of just one) because, by the asynchronous hypothesis, *only one* variable can change value in a transition. In such a description the system evolution is non-deterministic.

We obtain the following system of logical equations from the equations (1) and for the discrete threshold orders $\mathsf{s}_{\mathrm{x}2} < \mathsf{s}_{\mathrm{x}1}$ and $\mathsf{s}_{\mathrm{z}1} < \mathsf{s}_{\mathrm{z}2}$:

$$\begin{cases} \mathsf{X} = & \mathsf{K}_{\mathrm{x},0}. & \mathfrak{s}^-(\mathsf{x},\mathsf{s}_{\mathrm{x}2}).\mathfrak{s}^-(\mathsf{z},\mathsf{s}_{\mathrm{z}1}) \\ & +\mathsf{K}_{\mathrm{x},0+1}. & \mathfrak{s}^-(\mathsf{x},\mathsf{s}_{\mathrm{x}2}).\mathfrak{s}^+(\mathsf{z},\mathsf{s}_{\mathrm{z}1}) \\ & +\mathsf{K}_{\mathrm{x},3}. & \mathfrak{s}^+(\mathsf{x},\mathsf{s}_{\mathrm{x}2}).\mathfrak{s}^-(\mathsf{x},\mathsf{s}_{\mathrm{x}1}).\mathfrak{s}^-(\mathsf{z},\mathsf{s}_{\mathrm{z}1}) \\ & +\mathsf{K}_{\mathrm{x},3+4}. & \mathfrak{s}^+(\mathsf{x},\mathsf{s}_{\mathrm{x}2}).\mathfrak{s}^-(\mathsf{x},\mathsf{s}_{\mathrm{x}1}).\mathfrak{s}^+(\mathsf{z},\mathsf{s}_{\mathrm{z}1}) \\ & +\mathsf{K}_{\mathrm{x},3-5}. & \mathfrak{s}^+(\mathsf{x},\mathsf{s}_{\mathrm{x}1}).\mathfrak{s}^-(\mathsf{z},\mathsf{s}_{\mathrm{z}1}) \\ & +\mathsf{K}_{\mathrm{x},3+4-5}. & \mathfrak{s}^+(\mathsf{x},\mathsf{s}_{\mathrm{x}1}).\mathfrak{s}^+(\mathsf{z},\mathsf{s}_{\mathrm{z}1}) \\ \\ \mathsf{Z} = & \mathsf{K}_{\mathrm{z}}. & \mathfrak{s}^+(\mathsf{x},\mathsf{s}_{\mathrm{x}1}) \end{cases} \tag{3}$$

The six discrete parameters K in this model depend on the real K kinetic parameters:

$$K_{\mathrm{x},0} = \tfrac{k_{\mathrm{x}0}}{m_{\mathrm{x}1}}, \qquad K_{\mathrm{x},1} = \tfrac{k_{\mathrm{x}1}}{m_{\mathrm{x}1}}, \qquad K_{\mathrm{x},3} = \tfrac{k_{\mathrm{x}0}}{m_{\mathrm{x}1}+m_{\mathrm{x}2}},$$

$$K_{\mathrm{x},4} = \tfrac{k_{\mathrm{x}1}}{m_{\mathrm{x}1}+m_{\mathrm{x}2}}, K_{\mathrm{x},5} = \tfrac{k_{\mathrm{t}}}{m_{\mathrm{x}1}+m_{\mathrm{x}2}}, K_{\mathrm{z}} = \tfrac{k_{\mathrm{t}}}{m_{\mathrm{z}}}$$

The discrete parameters are then defined as follows: $\mathsf{K}_{\mathrm{x,i}} = d_{\mathrm{x}}(K_{\mathrm{x,i}})$, $\mathsf{K}_{\mathrm{x,i+j}} = d_{\mathrm{x}}(K_{\mathrm{x,i}} + K_{\mathrm{x,j}})$, $\mathsf{K}_{\mathrm{x,i-j}} = d_{\mathrm{x}}(K_{\mathrm{x,i}} - K_{\mathrm{x,j}})$, $\mathsf{K}_{\mathrm{x,i+j-l}} = d_{\mathrm{x}}(K_{\mathrm{x,i}} + K_{\mathrm{x,j}} - K_{\mathrm{x,l}})$, $\mathsf{K}_{\mathrm{z}} = d_{\mathrm{z}}(K_{\mathrm{z}})$.

As explained in [8] and in [6] frontiers between adjacent states have to be taken into account. De Jong *et al* ([6]) have devised a general and rigourous way based on the concept of Filippov solutions to compute transitions between regular states (states not confined on a frontier) and singular states (states confined on one frontier or an intersection of several frontiers) or between singular states. A state component is said to be regular if it does not stand on a frontier; the dimension of a state is the number of regular components of this state. We have recast this extension of AMLM into a logical form suitable for a CLP implementation [3].

The *transition graph* of such a model is established from all the transition between domains.

Example: The transition graph for the equations 3 with the following parameters: $\mathsf{K}_{\mathrm{x},0} = \mathsf{K}_{\mathrm{x},0+1} = 1$, $\mathsf{K}_{\mathrm{x},3} = \mathsf{K}_{\mathrm{x},3+4} = \mathsf{K}_{\mathrm{x},3-5} = \mathsf{K}_{\mathrm{x},3+4-5} = 0$ and $\mathsf{K}_{\mathrm{z}} = 2$, is given in Fig. 2.

Note that in our state labelling scheme, even values represent regular components and odd values singular ones.

Some transitions can be easily explained. Let us consider the state $\mathsf{S} = [\mathsf{x},\mathsf{z}] = [0,2]$ which is a regular state. Then from the equation 3 its focal state is $\mathsf{FS} = [\mathsf{K}_{\mathrm{x},0+1}, 0] = [1,0]$. As a consequence x tends to augment as z tends to diminish. Then two transitions are possible: $[0,2] \rightarrow [0,1]$ and $[0,2] \rightarrow [1,2]$. Transitions begining in singular states are less easy to understand: intuitively, for

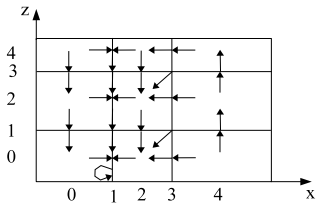


Fig. 2. Example of transition graph.

example, transition $[1,2] \rightarrow [1,1]$ exists partly because transitions $[0,2] \rightarrow [1,2]$ and $[2,2] \rightarrow [1,2]$ oppose themselves. □

4 Implementing AMLM taking into account 'singular states' in CLP

4.1 Organization of the CLP program

First we present the organization of the program and then some difficulties that we met. To follow the programming methodology "constraint first, then enumerate", we use intensively *reified* constraints to avoid non-determinism. It means we associated to each predicate `p(X)` a boolean parameter `B` such that `B` is equivalent to `p(X)` true. As the variables of AMLMs have finite discrete values, the solver of Prolog IV which is used the one dealing with interval of integers. As usual, some *redundant* constraints have been introduced to improve the infering capabilities of this solver.

In the following we outline the implementation. The main predicate is `multivalued_async_model(B, Model, Path)` which is true if `B` is equivalent to the fact "`Path` is a possible path (list of states) of the model `Model`". The definition of this predicate uses the predicate `successor(B, Model, State_i, State_s)` which is true if `B` is equivalent to the fact "`State_s` is a possible successor of the state `State_i` considering the model `Model`". Typically, the definition of `successor` uses reified constraints:

```
successor(B, Model, State_i, State_s) :-
          B = B1 bor B2,
          not(B1 band B2),
          less_equal_successor(B1, Model, State_i, State_s),
          greater_successor(B2, Model, State_i, State_s).
```

It expresses that the dimension of `State_s` is either less or equal to the dimension of `State_i` (`B1` true), or greater (`B2` true). The first condition enforces `B` to be the disjonction of `B1` and `B2`, and the second condition is a redundant constraint which enforces the exclusivity of the two cases. The predicates

`less_equal_successor` and `greater_successor` are defined in a similar way stating that `State_s` is a possible successor of `State_i` depending of one of the two cases.

For example, `greater_successor(B, Model, State_i, State_s)` is true if (i) `B` = `B1` and `B2`, (ii) `B1` is true iff `State_s` has more regular components than `State_i`, (iii) `B2` is true iff transition `State_i` → `State_s` exists according to `Model` and finally, (iv) `B` implies `State_i` is singular (which is a redundant constraint). Note that condition (iii) is established by constructing the minimal hyper-rectangle containing the focal states of all regular states which are adjacent to `State_s` [6].

Another predicate, *i.e.* `model(Idf_P, Model)`, is used to make the representation (variable `Model`) of the model identified by the variable `Idf_P` grouping an identifier (here `adhesion` for the adhesion model) and the list of parameters of the model. This representation is composed mainly by the discrete equations giving the qualitative evolution of the system.

4.2 Some implementation issues

We mention here some difficulties that we met when deriving a CLP program from the specification of AMLM extended to singular states.

A first difficulty is due to the fact that a direct translation of [6] would have lead to constraints of the form $\exists B|(B \equiv \exists X|C(X))$, meaning that the boolean B is true iff $C(X)$ is consistant (not to be confused with $\exists B, \exists X|B \equiv C(X)$). To overcome this difficulty we had to establish a new property $C'(U)$, where U is a variable on which X depends, such that $B \equiv \exists X|C(X) \quad \Leftrightarrow \quad B \equiv C'(U)$.

A second problem concerns the definition of the minimal hyper-rectangle which contains all focal states of regular states adjacent to a given state S. In order to express such an hyper-rectangle via a finite number of constraints, the number of adjacent states of S must be known (it is 2^{Order} where $Order$ is the number of singular components of S). In a first approach, we use quasi-parallel processes (freeze concept) waiting for order to post the necessary constraints. It means that queries must provide the knowledge of the order of states if we want to enumerate on a complete set of constraints. We currently develop another approach whose aim is to determine statically the order of the considered states.

A third difficulty concerns the definition of a component of a focal state. We need to express that the predicate `focal_comp(Xf, LB, LV_Xf)` is true if `LB` is a boolean list in which one and only one element is true (of index i), `LV_Xf` is a list of integers, `Xf` is the i^{th} element of `LV_Xf`. A first implementation is the following:

```
focal_comp(Xf, LB, LV_Xf) :-
                    index(1, LB, I),
                    index(Xf, LV_Xf, I).
```

where `index(V,L,I)` is true if `V` is the $\mathtt{I}^{\text{th}}$ value of `L`. Such an implementation allows to forbid `Xf` to take the j^{th} value of `LV_Xf` if $\mathtt{LB}_j$ is false. But it requires the use of union of intervals (at least in Prolog IV).

Example:

```
LB = [B1, B2, B3],
LV_Xf = [4,6,5],
focal_comp(Xf, LB, LV_Xf),
B2 = 0.
```

To get the result `Xf = cc(4,5)` with the above implementation of `focal_comp` the interval of `I` must be the union of intervals `[3,3]` and `[5,5]`. □

We had to find a solution which uses only standard intervals.

5 Examples of result

The program has been used for simulation knowing the order of the thresholds and the values of the K parameters but also to infer this knowledge by imposing constraints like known steady states, number of steady states or the minimal dimension of a state reached from a regular state.

To identify all the stationnary states (regular and singular) in the presented model, we state the following query:

```
model([adhesion, P], Model),
P = [_,[[tx,0], [tz,1]],_],  % Model is the presented model
multivalued_async_model(1, Model, [S,S]),
intsplit(S).
```

The second condition enforces `Model` to be the adhesion model for the discrete threshold orders $\mathsf{s}_{\mathrm{x}2} < \mathsf{s}_{\mathrm{x}1}$ and $\mathsf{s}_{\mathrm{z}1} < \mathsf{s}_{\mathrm{z}2}$. The last condition enumerates the possible steady states (the parameter values are not enumerated). We find in this way 13 possible steady states over the 25 states of the system. For the solution `S = [4,0]` we obtain without enumeration a completely instantiated list of parameters: $\mathsf{K}_{\mathrm{x},0} = \mathsf{K}_{\mathrm{x},0+1} = \mathsf{K}_{\mathrm{x},3} = \mathsf{K}_{\mathrm{x},3+4} = \mathsf{K}_{\mathrm{x},3-5} = \mathsf{K}_{\mathrm{x},3+4-5} = 2$ and $\mathsf{K}_{\mathrm{z}} = 0$. If we want to be certain of the existence of a steady state given in this way, we show by enumeration the existence of one set of parameters accepting this steady state. With this existence condition, the number of steady states goes from 13 possible to 11 actual steady states.

One of the most interesting query concerns *restoration paths*. Gulino and *al.* [5] showed that when junctions are destabilized by antibodies, VE-cadherins are destroyed and β-catenins are released in the cytoplasm. After this perturbation, a reformation of junction is observed. Then, a restoration path for the models defined by (3) is such that:

1. it begins by an initial state which is a perturbated state (PS) of the normal state of cell (i.e. a steady state, SS). More precisely x in PS is greater than x in SS (since β-catenins are released in cytoplasm),
2. it contains at least one state for which $\mathtt{z} = 4$, as it provides the possibility of synthesis of VE-cadherins necessary to the reconstruction of junctions,
3. and it finishes by the normal state of the cell (SS) for which $\mathtt{z} \neq 4$, because over-expression of the VE-cadherin gene in a normal state is not possible.

The constraint of existence of at least one restoration path for a given model, reduces the number of models from 600 to 40. We observe that all these models have the property $\mathsf{K}_z = 2$. The verification of this property has been obtained by a failure resulting of imposing to these 40 models the negation of the property ($\mathsf{K}_z \neq 2$). It is important to note that if only regular states are considered then some of these models do not accept restoration path. This shows that it is important to take into account singular states to model the behavior of this system.

Example: Among the models which accept a restoration path, one finds the one which is presented in the example of the section 3. For example the path `[[4,0],[4,1],[4,2],[4,3],[4,4],[3,4],[2,4],[1,4],[1,3],[1,2],[1,1],[1,0],[1,0]]` is a restoration path. This path would have not been observed if we had only considered ordinary AMLMs, *i.e.* not extended to singular states: clearly the normal state which finishes this path is singular. □

6 Some perspectives

There are numerous developments to this work, including efficiency, extension of models, interface and application to other biological systems.

Among them, one important issue is the languages that must be provided to the users of such a system: the language devoted to the expression of properties, the one which expresses results. We are thinking to a CTL-type (Computational Tree Logic) language in order to enforce complex properties. The design of a language for expressing results presents a real challenge. That is because biologists need such results like "these models satisfy these behaviours" expressed in an intentional way and not extensively as it is presently. This issue, related to automatic learning, is not an easy one but progress can certainly be achieved by first establishing with biologists the elementary components of the formulas of such a language.

References

1. J. Cohen. Classification of approaches used to study cell regulation: Search for a unified view using constraints and machine learning. ETAI, Machine Intelligence 18. Linköping Electronic Articles in Computer and Information Science ISSN 1401-9841, vol. 6 No. 25 (2001).

2. A. Colmerauer. Prolog – Constraints Inside, Manuel de Prolog, PROLOGIA, Case 919, 13288 Marseille cedex 09, France (1996).
3. F. Corblin. Rapport de projet de Master 2 Recherche. Inférence et simulation de réseaux biologiques logiques à l'aide de la PLC (Programmation Logique avec Contraintes). Université Joseph Fourier, Grenoble. (2005).
4. E. Fanchon, F. Corblin, L. Trilling, B. Hermant and D. Gulino. Modeling the Molecular Network Controlling Adhesion Between Human Endothelial Cells: Inference and Simulation Using Constraint Logic Programming. In Computational Methods in Systems Biology 2004, V. Danos and V. Schachter (Eds.), Lecture Notes in Bioinformatics 3082, 104–118 (2005).
5. B. Hermant, S. Bibert, E. Concord, B. Dublet, M. Weidenhaupt, T. Vernet and D. Gulino-Debrac. Identification of Proteases Involved in the Proteolysis of Vascular Endothelium Cadherin during Neutrophil Transmigration. The Journal of Biological Chemistry 278, 14002–14012 (2003).
6. H. de Jong, J.-L. Gouzé, C. Hernandez, M. Page, T. Sari, and J. Geiselmann. Qualitative Simulation of Genetic Regulatory Networks Using Piecewise-Linear Models. Bulletin of Mathematical Biology 66, 301–340 (2004).
7. E. H. Snoussi and R. Thomas. Logical Identification of All Steady States : The Concept of Feedback Loop Characteristic States. Bulletin of Mathematical Biology 55, 973–991 (1993).
8. R. Thomas and M. Kaufman. Multistationarity, the Basis of Cell Differentiation and Memory. II. Logical Analysis of Regulatory Networks in Term of Feedback Circuits. Chaos, 11, 180–195 (2001).

Constrained metabolic network analysis: discovering pathways using CP(Graph)

Gregoire Dooms, Yves Deville, Pierre Dupont

Department of Computing Science and Engineering
Université catholique de Louvain
B-1348 Louvain-la-Neuve - Belgium
{dooms,yde,pdupont}@info.ucl.ac.be*

1 Introduction

Biochemical networks – networks composed of the building blocks of the cell and their interactions are qualitative descriptions of the working of the cell. Such networks can be modeled as graphs. Metabolic networks are typical examples of such networks. They are composed of biochemical entities participating to reactions as substrates or products. Such a network can be modeled as a bipartite digraph which nodes are the biochemical entities and reactions and edges are the substrate or product link between an entity and a reaction.

Pathways are specific subsets of a metabolic network which were identified as functional processes of cells[1]. As these pathways are known to be working processes of the cell, they can be used to study the metabolic network. One type of metabolic network analysis consists in finding simple paths in the metabolic graph[2,3,4,5,6]. Here we focus on such analysis to discover pathways from a set of their reactions. A potential application is the explanation of DNA chip experiments using a CSP able to solve pathway discovery problems.

The study of the metabolic network is constantly evolving and most of the problems are solved with dedicated algorithms. This dedicated approach has the benefit of yielding very efficient programs to solve network analysis problems. This approach however has the drawback that it cannot be easily adapted to solve other problems or easily combined to solve combinations of various analyses.

In [7,8], we proposed to use constraint programming to solve constrained path finding problems in metabolic networks. This declarative paradigm allows to easily adapt programs or combine different programs. In order to provide a high level modeling language and as the data and results are graphs, we defined a graph computation domain for constraint programming [9].

The two following sections are devoted to a short introduction to CP(Graph) and a description of its application to metabolic network analysis.

*This research is supported by the Walloon Region, project BioMaze (WIST 315432). Thanks also to the EC/FP6 Evergrow project for their computing support.

2 The CP(Graph) Framework

CP(Graph) [9] features graph variables, node and arc variables, and set variables for nodes and arcs. They are depicted in figure 2. That figure shows the notation used for constants and variables in this paper. It also shows that a graph variable has an inherent constraints linking its arcs to its nodes.

Type	Representation	Constraint	Constants	Variables
Integer	$0, 1, 2, ...$		$i_0, i_1, ...$	$I_0, I_1, ...$
Node	$0, 1, 2, ...$		$n_0, n_1, ...$	$N_0, N_1, ...$
Arc	$(0, 1), (2, 4), ...$		$a_0, a_1, ...$	$A_0, A_1, ...$
Finite set	$\{0, 1, 2\}, \{3, 5\}$ ...		$s_0, s_1, ...$	$S_0, S_1, ...$
Finite set of nodes	$\{0, 1, 2\}, \{3, 5\}$ ...		$sn_0, sn_1, ...$	$SN_0, SN_1, ...$
Finite set of arcs	$\{(0, 3), (1, 2)\}$, ... (SN, SA)		$sa_0, sa_1, ...$	$SA_0, SA_1, ...$
Graph	SN a set of nodes SA a set of arcs	$SA \subseteq SN \times SN$	$g_0, g_1, ...$	$G_0, G_1, ...$
Weight functions	$\mathcal{N} \cup \mathcal{A} \rightarrow \mathbb{N}$		$w_0, w_1, ...$	–

Three kernel constraints suffice to express all MS-definable properties of graphs [10] as constraints:

$Arcs(G, SA)$ SA is the set of arcs of G.

$Nodes(G, SN)$ SN is the set of nodes of G.

$ArcNode(A, N_1, N_2)$ The arc variable A is an arc from node N_1 to node N_2. This relation does not take a graph variable into account as every arc and node has a unique identifier in the system. If A is determined, this constraint is a simple accessor to the tail and head of the arc A and similarly if both nodes are determined.

These constraints enable to express more complicated constraints such as $Reachables(G, N, SN)$ which states SN must be the set of nodes reachable from N in G (the transitive closure of the adjacency relation in G) or $Path(G, N_1, N_2)$ which holds if G is a path from node N_1 to node N_2. While these constraints can be expressed using kernel constraints, they are more efficient when implemented using dedicated global propagators (either for their consistency level or algorithmic complexity).

CP(Graph) enables to express constrained subgraph extraction problems such as the TSP or the equicut problem:

– Finding the TSP in graph g with weights w: minimize $Weight(G, w)$ s.t.

$$SubGraph(G, g) \wedge Cycle(G) \wedge Nodes(G) = Nodes(g)$$

– Graph partitioning problem: equicut of a graph g of even order: minimize $\#\left(Arcs(g) \setminus \left(Arcs(G_1) \cup Arcs(G_2)\right)\right)$ s.t.

$$SubGraph(G_1, g) \wedge SubGraph(G_2, g) \wedge Nodes(G_1) \cup Nodes(G_2) = Nodes(g) \wedge$$
$$\#Nodes(G_1) = \#Nodes(G_2) = \frac{1}{2}\#Nodes(g)$$

Related work: constraint programming was used to solve constrained path finding problems in [11] using a finite domain model (successor variables). Path variables were introduced in [12] to solve constrained path problems as part of a network design problem. A cost-based filtering technique for constrained shortest path problems was described in [13] and a global path constraint was presented in [14]. Graphs also play an important role in constraint programming in the design of propagators for global constraints: graph algorithms are used [15] and global constraints were modeled as networks of similar constraints [16,17]. The problem addressed in this paper is similar to queries addressed in the model checking approach of BIOCHAM [18]. While queries about reachability can be handled by both systems, it seems that some queries such as optimization problems can be expressed using CP(Graph) but not with CTL logic.

As it is, CP(Graph) allows constrained subgraph extraction. However, we are also working on constrained approximate subgraph isomorphism [19] by extending CP(Graph) with map variables [20,21].

3 Metabolic network analysis experiments

The general kind of analysis we wish to perform with CP(Graph) is pathway discovery by constrained subgraph extraction. One potential application of this type of analysis lies in assisted explanation of DNA chip experiments. In such experiments, the behavior of a sane cell and a mutant are compared in a given context (the substrate on which they are living or more generally their environment). This comparison is done at different times by extracting and amplifying the expressed RNA in the nucleus of the cells (this kills the cell). This RNA is then put on a DNA chip: an array of representative sequences of bases for a set of genes. The RNA binds to the chip in the locations which are specific to it. That array is then scanned to see the level of expression of each RNA strand. That RNA encodes for given enzymes which catalyze given reactions. Hence, the level of RNA can be translated into the information of which reactions were active in the cell at the time its RNA was extracted. Given this set of reactions, biologists would like to know which processes where active in the cell. If a CSP allows to recover known processes from sets of reactions, it could be adequate to discover the real processes given other sets of reactions. Hence, such a CSP could approximate the real processes at work in a cell from DNA chip results. These computational results could then be used to further guide other concrete experiments which are more expensive.

The current experiments focus on linear pathways by doing constrained shortest path finding. Future work comprise increasingly better characterizations of

the CSP (ie. more constraints which increase the rate of correct recovery) and a formulation of a CSP for pathways which contains branchings or cycles.

3.1 Prototype of CP(Graph) Implemented in Oz/Mozart and Gecode

We implemented a prototype of CP(Graph) in the Oz/Mozart [22] constraint programming framework. A set of nodes and a set of arcs are used to implement each graph variable. We also implemented this prototype over the Gecode generic constraint development environment [23]. In these prototypes, we implemented, among others, the global path propagator of [14].

3.2 Constrained Shortest Path Finding

As about half of the known pathways are simple paths [24], one type of experiment consists in trying to find these pathways by using constrained path finding in a directed graph (knowing a few nodes of the path). In [25], experiments were done first with a dedicated shortest path finding algorithm. Then some nodes (the pool metabolites, molecules like ATP or H_2O which are ubiquitous and take part in many reactions) were removed from the graph and the results compared with the previous ones. Some pathways, such as glycolysis, however use some of these metabolites as intermediates. In order to decrease the likelihood of selecting these nodes while still allowing to select them, all nodes were assigned a weight proportional to their degree. As pool metabolites have a very high degree, they are much less likely to be selected in the shortest paths.

Our experiment consists in redoing the former experiment with an additional constraint of mutual exclusion for certain pairs of reactions. These pairs are reverse reactions (the reaction from substrates to products and the one from products to substrates). Most of the time, these reactions are observed in a single direction in each species. Hence we wish to exclude paths containing both in our experiment. Such additional constraints like mutual exclusion are not always easily integrated in dedicated algorithms [25]. In CP(Graph) it just consists in posting a few additional constraints. If $n_1, ..., n_m$ are the included reactions and $(r_{i1}, r_{i2}), 0 < i \leq t$ the mutually exclusive pairs of nodes, the program looks like: miminize $Weight(G, w)$ s.t.

$$SubGraph(G, g) \wedge Path(G, n_1, n_m) \wedge \forall 0 < i \leq m : n_i \in Nodes(G) \wedge \\ \forall i \in [0, t] : r_{i1} \notin Nodes(G) \vee r_{i2} \notin Nodes(G)$$

In our experimental setting we first extract a subgraph of the original metabolic bipartite digraph by incrementally growing a fringe starting by the included nodes. Then, given a subset of the reactions of a reference pathway, we try to find the shortest constrained path in that subgraph. The first process of extraction of a subgraph of interest is done for efficiency reasons as the original graph is too big to be handled by the CSP (it contains around 16.000 nodes). The results

are presented in Table 1, it shows the increase of running time, memory usage and size of the search tree with respect to the size of the graph for the extraction of three illustrative linear pathways shown in [25]. All reactions are mandatory in the first experiment. The results of another experiment where one reaction out of two successive reactions in the given pathway is included in the set of mandatory nodes, is presented in Table 2.

The running time increases greatly with the size of the graphs. The program can however be stated in a few lines and first results obtained the same day the experiment is designed. The limitation on the input graph size does not guarantee to get the optimal shortest path in the original graph. This should however not be a major problem as biologists are most of the time interested in a particular portion of the metabolic graph. The rapidity of expression and resolution of such a NP(Hard) [13] problem outweights this size limitation.

Future work comprise two main aspects. The first is being able to cope with bigger graphs. We could design more efficient heuristics for labelling. The use of a cost-based filtering method could prune the size of the graph given an upper bound of the cost. Such an upperbound is available as soon as a first solution is found. Another solution would be to use an a-priori upper bound of the cost which would need to be increased or removed if no solution is found. The second aspect of our future work consists in finding which additional constraints are needed to recover known pathways as it was shown in [25] that non-constrained shortest paths are not able to recover all of them.

We are currently working on a extention of this approach to discovering pathways containing branchings or cycles. A first formulation we wish to test is the following: find the smallest graph containing all the seeds such that there is a seed from which all other nodes are reachable.

Given sn_s a set of nodes (seeds), minimize $Weight(G, w)$ subject to:

$$N \in sn_s, sn_s \subseteq Nodes(G) \wedge Reachable(G, N, Nodes(G))$$

Glycolysis (m=8)					Heme (m=8)					Lysine (m=9)				
Size	t	Time	Nodes	Mem	Size	t	Time	Nodes	Mem	Size	t	Time	Nodes	Mem
50	12	0.2	20	2097	50	22	0.2	32	2097	50	18	0.2	38	2097
100	28	2.5	224	2097	100	36	0.3	22	2097	100	40	4.7	652	2097
150	48	41.7	1848	4194	150	62	1.0	28	2097	150	56	264.3	12524	15204
200	80	55.0	1172	5242	200	88	398.8	7988	18874	200	70	-	-	-
250	84	127.6	4496	8912	250	118	173.3	2126	9961	250	96	-	-	-
300	118	2174.4	16982	60817	300	146	1520.2	21756	72876	300	96	-	-	-

Table 1. Comparison of the running time [s], number of nodes in the search tree and memory usage [kb], for the 3 pathways and for increasing original graph sizes. m is the number of node inclusion constraints and t the number of mutual exclusion constraints.

4 Conclusion

The problem of discovering the processes at work in a cell given a set of reactions can be modeled as a constrained subgraph extraction problem. But the formal

Glycolysis (m=5)					Heme (m=5)					Lysine (m=5)				
Size	t	Time	Nodes	Mem	Size	t	Time	Nodes	Mem	Size	t	Time	Nodes	Mem
50	12	0.2	22	2097	50	22	0.3	44	2097	50	18	0.1	16	2097
100	28	2.5	230	2097	100	36	0.9	78	2097	100	40	13.3	1292	3145
150	48	79.3	5538	6815	150	62	7.3	144	3145	150	56	260.4	8642	14155
200	80	39.9	1198	5767	200	88	57.3	950	5242	200	70	4330.5	74550	192937
250	84	323.6	5428	14680	250	118	36.0	350	8388	250	96	-	-	-
300	118	10470.8	94988	296747	300	146	-	-	-	300	96	-	-	-

Table 2. Same experiment as in Table1, but with one reaction node included every two ($m = 5$ instead of 8 or 9).

expression of this problem is not yet clear. In order to refine it, we first evaluate the various problem formulations on recovering known pathways. We hope the best solution to the reduced problem will be suitable to solve the more general problem of discovering real pathways.

Such an approach is difficult to achieve using dedicated algorithms as new algorithms must be designed each time a new problem formulation is to be evaluated [25]. A declarative approach is more practical as it just requires the formulation of the problem in a declarative language. Constraint programming is a declarative framework which has been successfully used to solve hard problems. CP(Graph) is a constraint programming computation domain suitable to express constrained subgraph extraction problems. It provides a higher level interface to define such problems and should be easier to use by bio-informaticians than classical finite domain or finite set computation domains.

This first application of CP(Graph) on constrained shortest path problems in metabolic networks shows that it is appropriate to express and solve these metabolic network extraction problems. We shall continue this work by moving to non linear pathways and trying to cope with bigger graphs.

Our future work includes an extention to pathways which contain cycles and branchings, the handling of larger graphs, and the experimental characterization of the formalization of the problem of recovering known pathways from the metabolic graph using CP(Graph).

References

1. Minoru, K., Susumu, G., Shuichi, K., Akihiro, N.: The KEGG databases at GenomeNet. Nucleic Acids Research **30(1)** (2002) 42–46
2. Jeong, H., Tombor, B., Albert, R., Oltvai, Z., Barabasi, A.: The large-scale organization of metabolic networks. Nature **406** (2000) 651–654
3. Kim, B., Yoon, C., Han, S., Jeong, H.: Path finding in scale-free networks. Phys Rev E Stat Nonlin Soft Matter Phys **65** (2002) 27101–27104
4. R. Alves, R.A. Chaleil, M.S.: Evolution of enzymes in metabolism: a network perspective. Journal of Molecular Biology **320** (2002) 751–770
5. van Helden, J., Naim, A., Mancuso, R., Eldridge, M., Wernisch, L., Gilbert, D., Wodak, S.: Representing and analyzing molecular and cellular function using the computer. Journal of Biological Chemistry **381(9-10)** (2000) 921–35

6. van Helden, J., Wernisch, L., Gilbert, D., Wodak, S.: Graph-based analysis of metabolic networks. In: Bioinformatics and genome analysis. Springer-Verlag (2002) 245–274
7. Dooms, G., Deville, Y., Dupont, P.: Recherche de chemins contraints dans les réseaux biochimiques. In Mesnard, F., ed.: Programmation en logique avec contraintes, actes des JFPLC 2004, Hermes Science (June 2004) 109–128
8. Dooms, G., Deville, Y., Dupont, P.: Constrained path finding in biochemical networks. In: Proceedings of JOBIM 2004. (2004) JO–40
9. Grégoire Dooms, Yves Deville, Pierre Dupont: CP(Graph): Introducing a Graph Computation Domain for Constraint Programming. In: Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming. Number LNCS ...., Springer-Verlag (2005)
10. Courcelle, B.: The monadic second-order logic of graphs. i. recognizable sets of finite graphs. Inf. Comput. **85** (1990) 12–75
11. Pesant, G., Gendreau, M., Potvin, J., Rousseau, J.: An exact constraint logic programming algorithm for the travelling salesman with time windows. Transp. Science **32** (1996) 12–29
12. Lepape, C., Perron, L., Regin, J.C., Shaw, P.: A robust and parallel solving of a network design problem. In: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming. Volume LNCS 2470. (2002) 633–648
13. Sellmann, M.: Cost-based filtering for shorter path constraints. In: Proceedings of the 9th International Conference on Principles and Practise of Constraint Programming (CP). Volume LNCS 2833., Springer-Verlag (2003) 694–708
14. Cambazard, H., Bourreau, E.: Conception d'une contrainte globale de chemin. In: 10e Journ. nat. sur la résolution pratique de problèmes NP-complets (JNPC'04). (2004) 107–121
15. Régin, J.: A filtering algorithm for constraints of difference in CSPs. In: Proc. 12th Conf. American Assoc. Artificial Intelligence. Volume 1. (1994) 362–367
16. Beldiceanu, N.: Global constraints as graph properties on structured network of elementary constraints of the same type. Technical Report T2000/01, SICS (2000)
17. Beldiceanu, N.: Global constraint catalog. Technical Report T2005-08, SICS (2005)
18. 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
19. Yves Deville, Grégoire Dooms, Stéphane Zampelli, Pierre Dupont: CP(Graph+Map): Constrained Approximate Subgraph Matching. In: INGI Research report 2005-07. (2005)
20. Gervet, C.: Interval propagation to reason about sets: Definition and implementation of a practical language. CONSTRAINTS Journal **1(3)** (1997) 191–244
21. Hnich, B.: Function Variables for Constraint Programming. PhD thesis, SICS, Sweden (2003)
22. Mozart Consortium: The mozart programming system version 1.2.5 (December 2002) `http://www.mozart-oz.org/`.
23. Gecode: Generic Constraint Development (2005) `http://www.gecode.org/`.
24. Lemer, C., Antezana, E., Couche, F., Fays, F., Santolaria, X., Janky, R., Deville, Y., Richelle, J., Wodak, S.J.: The aMAZE lightbench: a web interface to a relational database of cellular processes. Nucleic Acids Research **32** (2004) D443–D448
25. Croes, D.: Recherche de chemins dans le réseau métabolique et mesure de la distance métabolique entre enzymes. PhD thesis, ULB, Brussels (2005) (in preparation).

A new local consistency for weighted CSP applied to ncRNA detection

Christine Gaspin, Simon de Givry, Thomas Schiex, Patricia Thébault, Matthias Zytnicki

INRA – BIA Toulouse

Abstract. The recent discovery of numerous sequences of RNA that do not code for a protein gave rise to a renewed interest in bioinformatics and many programs for detecting RNA genes have been proposed. However, the complex structures of these RNA sequences make the problem of detecting an element of a given family NP-complete. We present here a framework called *weighted constraint satisfaction problem*, that can solve this kind of hard problem. We also introduce a new local consistency, particularly adapted to this kind of problem, that notably speeds up the search. We believe this framework can efficiently handle long sequences and more complex motifs than state-of-art programs.

1 Introduction

Our understanding of the role of RNA molecules has changed in recent years. Firstly considered as simply being the messenger that converts genetic information from DNA into proteins, RNA is now seen as a key regulatory factor in many of the cell's crucial functions, affecting a large variety of processes including plasmid replication, phage development, bacterial virulence, chromosome structure, DNA transcription, RNA processing and modification, development control and others. Consequently, the systematic search of non-coding RNA (ncRNA) genes, which produce functional RNAs instead of proteins, represents an important challenge.

RNA sequences can be considered as oriented texts (left to right) over the four letter alphabet {A, C, G, U} (cf. Fig. 1(a)). An RNA molecule can fold on itself by making a variety of interactions, the result being a structure which is essential for the biological function. The most prevalent interactions which stabilize folded molecules are stacking and hydrogen bonding (G–C, C–G, A–U and U–A bonds) between nucleotides. These interactions form what is usually called the secondary structure.

Every RNA secondary structure can be represented on a circular planar graph where the nucleotides of the sequence are represented as vertices and are connected by edges representing either (along the circle) covalent bonds between successive nucleotides in the RNA sequence or (inside the circle) hydrogen bonds between nucleotides from different regions (cf. Fig. 1(b)). Such a graph gives

rise to characteristic secondary structural elements such as helices (a succession of paired nucleotides, cf. Fig. 1(c)), and various kinds of loops (unpaired nucleotides surrounded by helices).

A more general definition of RNA structure allows for crossing edges in the representation graph, making it possible the representation of a type of helix usually called a pseudo-knot (cf. Fig. 1(d)) that cannot be described by a secondary structure. RNA structures can also include nucleotide triples inside triple helices.

It is possible to classify ncRNA into *families*. A family is a set of sequences that have envolved from a common ancestor. Members of a family usually have a similar biological function that is coded by a similar secondary structure, whereas the sequence itself is poorly conserved (cf. Fig. 1(f)).

Thus, the information contained both in the sequence itself and the structure can be viewed as a biological signal to exploit and search for. These common structural characteristics can be captured by a signature that represents the structural elements which are conserved inside a set of related RNA molecules.

Our aim is to find all the members of an ncRNA family described by its structure in a given sequence. Traditionally, two types of approaches have been used for RNA gene finding: signatures can be modelled as stochastic context free grammars (excluding pseudo-knots or complex structures) and then searched using relatively time consuming dynamic programming based parsers ([10,4]).

Another approach defines a signature as a set of interrelated motifs. Occurrences of the signature are sought using simple pattern-matching techniques and exhaustive tree search. Such programs include RnaMot (cf. [5]), RnaBob (cf. [5]), PatScan (cf. [3]), Palingol (cf. [1]) and RnaMotif (cf. [9]). Although most allow pseudo-knots to be represented, they have very variable efficiencies.

For sufficiently general signatures, this is an NP-complete problem that combines combinatorial optimization and pattern matching issues (cf. [12]). A first model using CSP has been proposed by [11]. It can detect virtually any kind of motif but a major drawback of this approach is the huge number of solutions. We present in this paper preliminary results that allow to order and select most promising RNA gene candidates using a score based on the energy of the molecule or the likelihood of the solution, in the framework of *weighted constraint satisfaction problem* (WCSP).

2 RNA motif problem

We call *motif* the elements of the secondary structure that define an RNA family. Given a sequence and a motif, our aim is to find all the occurrences of this motif in the sequence. To a first approximation, a motif can be decomposed into *strings* (cf. Fig. 1(a)) and *helices* (cf. Fig. 1(c)). Two elements can be separated by *spacers*.

Example 1. Consider the sequence in Fig. 2(a). We want to find the motif that begins with ACGU, then has a spacer of three nucleotides and an helix of length

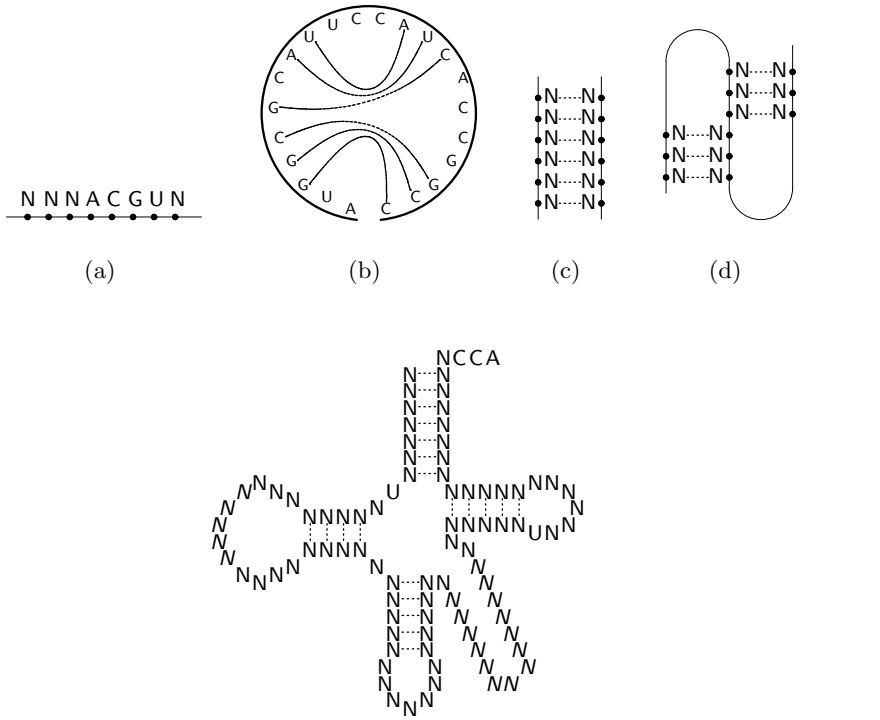
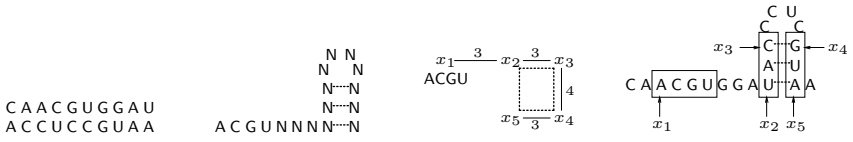


Fig. 1. Some motifs: (a) the string ACGU inside a long sequence, (b) an arbitrary secondary structure, (c) an helix, (d) a pseudo-knot, (e) a tRNA, where the nucleotides in italic are optional, (f) the common motif of several members of the tRNA family



(a) sequence (b) structure (c) model (d) solution

Fig. 2. The stages of the motif detection

3 with a loop of length 4 (described in FIG. 2(b)). The elements are in the boxes in FIG. 2(d): the first region is the string, the second is the first stem of the helix and the last, the second stem of the helix.

3 Preliminaries

Valuation structures are algebraic objects that allow to specify costs [7]. For WCSP, it is defined by a triple $\mathcal{S} = \langle E, \oplus, \leq \rangle$ where:

- $E = [0..k] \subseteq \mathbb{N}$ is the *set of costs*;
- $\oplus$ is the *addition* on E, defined by $\forall (a, b) \in \mathbb{N}^2, a \oplus b = \min\{a + b, k\}$;
- $\leq$ is the usual operator on $\mathbb{N}$.

It is useful to define the *subtraction* $\ominus$ of costs: $\forall (a, b) \in \mathbb{N}^2, a \ominus b = a - b$ if $a \neq k, k$ otherwise.

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 the finite domains of each variable and the size of the largest one is d,
- $\mathcal{C}$ is a set containing e constraints.

A constraint $c \in \mathcal{C}$ can be:

- a unary constraint $(c : D(x_i) \to E)$ and we call it c_i;
- a binary constraint $(c : D(x_i) \times D(x_j) \to E)$ and we call it c_{ij};
- an r-ary constraint $(c : D(x_{i_1}) \times \ldots \times D(x_{i_r}) \to E)$.

However, we will restrict ourselves to the *binary* case, where no constraint has an arity greater than 2. The following results could easily be extended to higher arity constraints.

Given a tuple $t \in D(x_{i_1}) \times \ldots \times D(x_{i_r})$, $c(t) = k$ means that c forbids the corresponding assignment. Another cost means the tuple is permitted by c with the corresponding cost. The cost of a *total* assignment $t \in D(x_1) \times \ldots \times D(x_n)$, noted $\mathcal{V}(t)$, is the sum over all the cost functions $c(t), c \in \mathcal{C}$,

We assume the existence of a unary constraint c_i for every variable, and a *zero*-arity constraint (i.e. a constant), noted $c_\varnothing$ (if no such constraint is defined, we can always define *dummy* ones: c_i is the null function, $c_\varnothing = 0$). This zero-arity constant gives a cost that should be paid for any assignment of the problem and thus represents a *lower bound* of the cost of a solution. An assignment t is *consistent* if $\mathcal{V}(t) < k$. Finding a consistent assignment is a NP-complete problem.

$k - 1$ represents the maximal acceptable cost. For the moment, it is fixed by trial and error but we hope to find a better method soon.

4 Model

Each variable represents the position of an element of the motif in the sequence. When the algorithm starts, the domain of every variable is the size of the sequence. The constraints are used to describe an element of the motif:

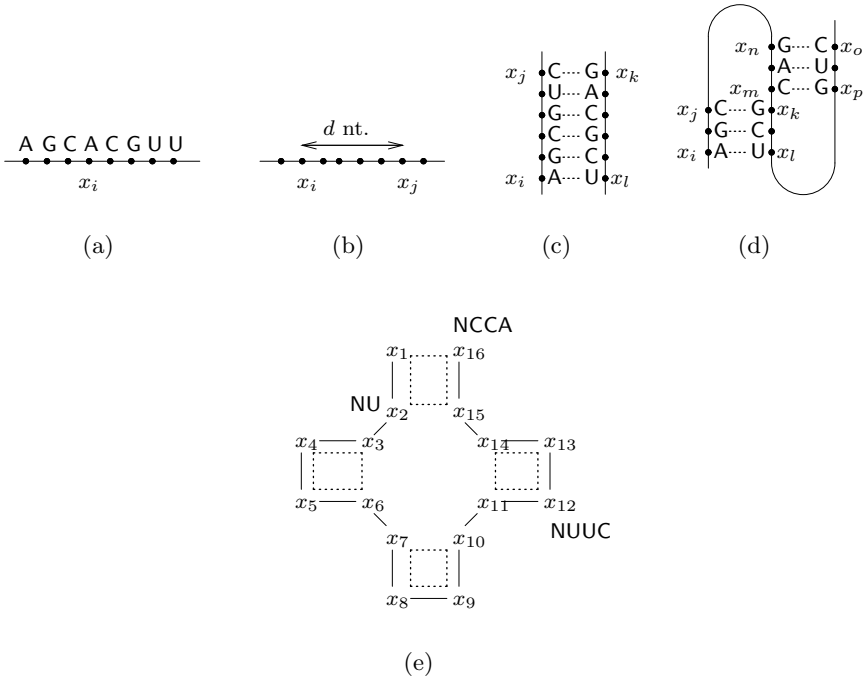
- The constraint *string(*word, x_i*)* (cf. Fig. 3(a)) takes a possibly ambiguous word (i.e. containing letters like N) and compares it to the subsequence that begins at the index x_i. The algorithm uses dynamic programming and gives affine cost to the gaps.
- The constraint *helix(*x_i, x_j, x_k, x_l*)* (cf. Fig. 3(c)) gives a cost to an helix of which the first stem is between x_i and x_j, and of which the second stem is between x_k and x_l. It also uses an *ad hoc* dynamic programming algorithm that takes into account affine gaps and the size of the helix to compute the score.
- The constraint *spacer(*$x_i, x_j, \mathsf{d_1}, \mathsf{d_2}$*)* (cf. Fig. 3(b)) gives a null score if $\mathsf{d_1} \leq x_2 - x_1 \leq \mathsf{d_2}$. Otherwise, the cost is a piecewise linear function that increases when x_1 and x_2 are getting too close or too far away from each other.

Within this model, finding all the solutions of the WCSP is finding all the potential members of the family described by the set of constraints. Their costs specify the adequation of the candidates to the given signature.

Example 2. Consider again the previous example. The motif could have been modeled like in Fig. 2(c) by five variables (x_1 to x_5), a *string* constraint (ACGU), four *spacer* constraints (the full lines) and a *helix* constraint (the dotted square). The algorithm should give the result described in Fig. 2(d). As the candidate perfectly matches the signature, its cost should by 0. Notice that a pseudo-knot can be described by two helices like in Fig. 3(d).

5 Some local properties

WCSPs are usually solved with a branch-and-bound tree of which each node is a partial assignment. To accelerate the search, local consistency properties are widely used to transform the sub-problem at each node of the tree to an equivalent, simpler one. The simplest local consistency property is *node consistency* (NC*, cf. [6]).



(a) (b) (c) (d)

(e)

Fig. 3. The constraints used to represent the motifs: (a) *string*(x_i, ACGU), (b) *spacer*(x_i, x_j, d), (c) *helix*, (d) *pseudo-knot* with two helices, (e) a tRNA

Definition 1. *A variable x_i is node consistent if:*

- $\forall v_i \in D(x_i), c_\varnothing \oplus c_i(v_i) < k$ *and*
- $\exists v_i \in D(x_i), c_i(v_i) = 0$ *(this value v_i is called the* unary support *of x_i).*

A WCSP is node consistent if every variable is node consistent.

This property can be enforced in time and space $\mathcal{O}(nd)$ for binary problems. Another famous stronger local consistency is the *arc consistency* (AC*, cf. [6]).

Definition 2. *The* neighbours $N(x_i)$ *of a variable x_i is the set of the variables x_j such that there exists a constraint that involves x_i and x_j. A variable x_i is arc consistent if:*

- $\forall v_i \in D(x_i), \forall x_j \in N(x_i), \exists w_j \in D(x_j), c_{ij}(v_i, w_j) = 0$ *(this value w_j is called the* support *of x_i in v_i w.r.t. c_{ij}) and*
- *x_i is node consistent.*

A WCSP is arc consistent if every variable is arc consistent.

Arc consistency can be enforced in time $\mathcal{O}(n^2d^3)$ and in space $\mathcal{O}(ed)$ for binary problems. Clearly, the space complexity is far too high for our kind of problem. This is why we use a weaker form of this property: *bound arc consistency* (BAC*).

Definition 3. *To apply bound arc consistency, we need to change the definition of a WCSP: the domains are now* intervals $\mathcal{I}$. *Each variable x_i can take all the values in $I(x_i) = [lb_i..ub_i]$ (lb_i is the* lower bound *of the interval of x_i and ub_i is its* upper bound*).*

A variable x_i is bound node consistent (BNC) if:*

- $(c_\varnothing \oplus c_i(lb_i) < k) \wedge (c_\varnothing \oplus c_i(ub_i) < k)$ *and*
- $\exists v_i \in I(x_i), c_i(v_i) = 0$.

A variable x_i is bound arc consistent if:

- $\forall x_j \in N(x_i), \exists (w_j, w'_j) \in I^2(x_j), c_{ij}(lb_i, w_j) = c_{ij}(ub_i, w'_j) = 0$ *and*
- *it is bound node consistent.*

A WCSP is bound arc consistent if every variable is bound arc consistent.

Theorem 1. *The algorithm 1 enforces BAC* in time $\mathcal{O}(ed^2 + knd)$ and in space $\mathcal{O}(n + e)$.*

Proof. **Correction:** We will consider the following invariants:

1. on line **2**, all variables are BNC*,
2. if x_i is not in Q, then $\forall x_j \in N(x_i), lb_i, ub_i, lb_j$ and ub_j have a support w.r.t. c_{ij}.

Algorithm 1: Algorithm enforcing BAC*

Procedure SetBAC*() [Enforce BAC*]

1 **foreach** $x_i \in \mathcal{X}$ **do** SetBNC*(x_i) ;
$Q \leftarrow \mathcal{X}$; $c_\varnothing\_raised \leftarrow$ false ;
2 **while** $(Q \neq \varnothing)$ **do**
$x_j \leftarrow Q.pop()$;
3 **foreach** $x_i \in N(x_j)$ **do**
4 SetBSupport(x_i, x_j) ; SetBSupport(x_j, x_i) ;
5 **if** *(SetBNC*(x_i))* **then** $Q \leftarrow Q \cup \{x_i\}$;
6 **if** *(SetBNC*(x_j))* **then** $Q \leftarrow Q \cup \{x_j\}$;
7 **if** $(c_\varnothing\_raised)$ **then**
$c_\varnothing\_raised \leftarrow$ false ;
8 **foreach** $x_i \in \mathcal{X}$ **do**
9 **if** *(SetBNC*(x_i))* **then** $Q \leftarrow Q \cup \{x_i\}$;

Function SetBNC*(x_i)*: boolean* [Enforce BNC*]
$changed \leftarrow$ false ;
while $(lb_i \leq ub_i) \wedge (c_\varnothing \oplus c_i(lb_i) \geq k)$ **do** $lb_i \leftarrow lb_i + 1$; $changed \leftarrow$ true ;
while $(lb_i \leq ub_i) \wedge (c_\varnothing \oplus c_i(ub_i) \geq k)$ **do** $ub_i \leftarrow ub_i - 1$; $changed \leftarrow$ true ;
ProjectUnary(x_i) ;
return *changed* ;

Procedure ProjectUnary(x_i) [Find the unary support of x_i]
$min \leftarrow \min_{v_i \in I(x_i)}\{c_i(v_i)\}$;
if $(min = 0)$ **then return** ;
$c_\varnothing\_raised \leftarrow$ true ;
foreach $v_i \in I(x_i)$ **do** $c_i(v_i) \leftarrow c_i(v_i) \ominus min$;
$c_\varnothing \leftarrow c_\varnothing \oplus min$;
if $(c_\varnothing \geq k)$ **then raise** *exception* ;

Procedure Project(x_i, v_i, x_j) [Find the support of $v_i \in \{lb_i, ub_i\}$ w.r.t. c_{ij}]
$min \leftarrow \min_{w_j \in I(x_j)}\{c_{ij}(v_i, w_j)\}$;
foreach $w_j \in I(x_j)$ **do** $c_{ij}(v_i, w_j) \leftarrow c_{ij}(v_i, w_j) \ominus min$;
$c_i(v_i) \leftarrow c_i(v_i) \oplus min$;

Procedure SetBSupport(x_i, x_j) [Find the supports of the bounds of x_i w.r.t. c_{ij}]
Project(x_i, lb_i, x_j) ; Project(x_i, ub_i, x_j) ;

First, ProjectUnary(x_i) finds the unary support of x_i and SetBNC*(x_i) loops until it finds the allowed bounds of x_i, so this function enforces BNC*. At the beginning of the algorithm, as the variables may not have this property, we call SetBNC*(x_i) for each variable x_i. Thus the second invariant is respected at the beginning of the algorithm.

This invariant may be broken by a projection from a binary constraint to a bound of an interval; this may either lead to the fact that one of the bound is now forbidden, or that a unary support (which was this bound) has disappeared. This is why SetBNC* is called on x_j and all its neighbours (lines **5** and **6**) after the projections of the line **4**.

The first invariant could also be broken when $c_\varnothing$ increases: a bound can now have a unary cost greater that $k - c_\varnothing$. This event can occur after the lines **5** and **6**. This explains the **if** beginning at line **7**.

Concerning the second invariant, it is true at the beginning of the algorithm as all the variables are enqueued. Afterwards, Project(x_i, v_i, x_j) finds the support of v_i w.r.t. c_{ij}, so SetBSupport(x_i, x_j) finds the supports of the bounds of x_i w.r.t. c_{ij}. Thus the line **4** enforces the second invariant.

This invariant can only be broken by SetBNC* and anytime this function is called, the corresponding variable is enqueued. Finally, at the end of the algorithm, the instance is BNC* (thanks to the first invariant) and every bound has a support w.r.t. to each constraint in which it is involved (thanks to the second invariant): the problem is now BAC*.

Time complexity: Thanks to [6], we know that Project and ProjectUnary take time $\mathcal{O}(d)$. Thus SetBSupport also takes time $\mathcal{O}(d)$ and the complexity of the line **1** is $\mathcal{O}(nd)$.

Each variable can be pushed in at most $\mathcal{O}(d)$ times into Q, thus the overall complexity of the line **6** is $\mathcal{O}(nd^2)$. The program enters in the loop of line **3** at most $\mathcal{O}(ed)$ times (given a constraint c_{ij}, the program can enter $\mathcal{O}(d)$ times because of x_i and $\mathcal{O}(d)$ times because of x_j) thus the overall complexity of lines **4** and **5** is $\mathcal{O}(ed^2)$. The line **7** can be true at most k times (otherwise the problem is detected as inconsistent) and the overall complexity of the line **9** is $\mathcal{O}(k \times n \times d)$. To sum up, this algorithm takes time $\mathcal{O}(nd^2 + ed^2 + knd) = \mathcal{O}(ed^2 + knd)$. However, as the **while** on line **2** can be true at most $\mathcal{O}(nd)$ times, the **foreach** on line **8** cannot loop more than $\mathcal{O}(n^2d)$ times and the complexity of the line **9** is not greater than $\mathcal{O}(n^2d^2)$. So the actual time complexity is $\mathcal{O}(ed^2 + \min\{k, nd\} \times nd)$, and if $k > nd$ then it is $\mathcal{O}(n^2d^2)$.

Space complexity: As described here, the algorithm has a space complexity of $\mathcal{O}(ed)$. However, the we can bring the complexity down to $\mathcal{O}(e)$ as suggested in [2], by using additionnal data structures.

6 Experimental results

We have tried to detect the well-known structure of tRNA [5] (cf. Fig. 3(e)), modeled by 16 variables, 15 spacers, 3 strings and 4 helices on parts of the

genome of *Saccharomyces Cerevisiæ* of different sizes and on the whole genome of *Escherichia coli*.

In our implementation, every constraint is separated into its *hard* part (that gives a cost of k) and its *soft* part (that gives a cost less than k). The hard part is first treated to remove rapidly some obviously inadequate values through 2B-consistency (cf. [8]). Then, AC* or BAC* process the soft part.

We used a 2.4Ghz Intel Xeon with 8 GB RAM to solve these instances. We compared our algorithm with the classic AC* on Fig. 4. For each instance of the problem, we write its size (10k is sequence of 10.000 nucleotides and the genome of *Escherichia coli* contains more than 4.6 millions nucleotides) and the number of solutions. We also show the number of nodes explored and the time in seconds spent. A "-" means the instance could not be solved due to memory reasons despite all the memory optimizations.

		AC*		BAC*	
Size	♯ solutions	nodes	time	nodes	time
10k	16	23	29	32	0
50k	16	35	545	39	0
100k	16	-	-	51	0
500k	16	-	-	194	1
1M	24	-	-	414	2
ecoli	140	-	-	1867	7

Fig. 4. Number of nodes explored and time in seconds spent to solve several instances of the ncRNA detection problem

The reason of the superiority of BAC* over AC* is twofold. First, AC* needs to store all the unary cost for every variable to project cost from binary constraints to unary constraint. Thus, the space complexity of AC* is at least $\mathcal{O}(nd)$. For very long domains (in our experiment, greater than 50.000 values), the computer cannot allocate sufficient memory and the program is aborted. For the same kind of projection, BAC* only needs to store the costs of the bounds of the domains, leading to a space complexity of $\mathcal{O}(n)$.

Second, the *distance* constraints dramatically reduce the size of the domains. Concretely, when a single variable is assigned, and when all the distance costs have been propagated, all the other domains have a size that is a constant with respect to d. As BAC* behaves particularly well with this kind of constraints, the instance becomes quickly tractable.

7 Conclusions and future work

In this paper we have applied the WCSP framework to the ncRNA detection problem and introduced a new local consistency called BAC*. We have given its complexity, which is lower than the commonly used algorithms and we have shown that maintaining BAC* is better than AC* for our problem.

In the future, we would like to implement other constraints such as hybridization and add several heuristics.

References

1. B. Billoud, M. Kontic, and A. Viari. Palingol: a declarative programming language to describe nucleic acids' secondary structures and to scan sequence database. *Nucleic Acids Research*, 24(8):1395–1403, 1996.
2. M. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154(1-2):199–227, 2004.
3. M. Dsouza, N. Larsen, R, and Overbeek. Searching for patterns in genomic data. *Trends in Genetics*, 13(12), 1997.
4. S. Eddy and R. Durbin. RNA sequence analysis using covariance models. *Nucleic Acids Research*, 22(11):2079–88, 1994.
5. D. Gautheret, L.J. Heyer, and R. Cedergren. Pattern searching/alignment with RNA primary and secondary structures: an effective descriptor for tRNA. *Computer Applications in the Biosciences*, 6(4):325–331, 1990.
6. J. Larrosa. Node and arc consistency in weighted CSP. In *Proc. AAAI'02*, 2002.
7. J. Larrosa and T. Schiex. Solving Weighted CSP by Maintaining Arc-consistency. *Artificial Intelligence*, 159(1-2):1–26, 2004.
8. O. Lhomme. Consistency techniques for numeric CSPs. In *Proceedings of the 13th IJCAI*, pages 232–238, 1993.
9. T. Macke, D. Ecker, R. Gutell, D. Gautheret, D. Case, and R. Sampath. Rnamotif, an RNA secondary structure definition and search algorithm. *Nucleic Acids Research*, 29(22):4724–4735, 2001.
10. Y. Sakakibara, M. Brown, R. Hughey, I. Mian, K. Sjölander, R. Underwood, and D. Haussler. Recent methods for RNA modeling using stochastic context-free grammars. In *CPM'94*, pages 289–306, 1994.
11. P. Thébault, S. de Givry, T. Schiex, and C. Gaspin. 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.
12. S. Vialette. On the computational complexity of 2-interval pattern matching problems. *Theoretical Computer Science*, 312(2-3):223–249, 2004.

Mendelian error detection in complex pedigree using weighted constraint satisfaction techniques

S. de Givry, I. Palhiere, Z. Vitezica and T. Schiex

INRA, Toulouse, France

Abstract. With the arrival of high throughput genotyping techniques, the detection of likely genotyping errors is becoming an increasingly important problem. In this paper we are interested in errors that violate Mendelian laws. The problem of deciding if Mendelian error exists in a pedigree is NP-complete [1]. Existing tools dedicated to this problem may offer different level of services: detect simple inconsistencies using local reasoning, prove inconsistency, detect the source of error, propose an optimal correction for the error. All assume that there is at most one error. In this paper we show that the problem of error detection, of determining the minimum number of error needed to explain the data (with a possible error detection) and error correction can all be modeled using soft constraint networks. Therefore, these problems provides attractive benchmarks for weighted constraint network solvers such as the dedicated WCSP solver `toolbar`.

1 Background

Chromosomes carry the genetic information of an individual. A position that carries some specific information on a chromosome is called a locus (which typically identifies the position of a gene). The specific information contained at a locus is the allele carried at the locus. Except for the sex chromosomes, diploid individuals carry chromosomes in pair and therefore a single locus carries a pair of allele. Each allele originates from one of the parents of the individual considered. This pair of alleles at this locus define the genotype of the individual at this locus. Genotypes are not always completely observable and the indirect observation of a genotype (its expression) is termed the phenotype. A phenotype can be considered as a set of possible genotypes for the individual. These genotypes are said to be compatible with the phenotype.

A pedigree is defined by a set of related individuals together with associated phenotypes for some locus. Every individual is either a founder (no parents in the pedigree) or not. In this latter case, the parents of the individual are identified inside the pedigree. Because a pedigree is built from a combination of complex experimental processes it may involve experimental and human errors. The errors can be classified as parental errors or typing errors. A parental error means that the very structure of the pedigree is incorrect: one parent of one individual is not actually the parent indicated in the pedigree. We assume that parental information is correct. A phenotype error means simply that the phenotype in the

pedigree is incompatible with the true (unknown) genotype. Phenotype errors are called Mendelian errors when they make a pedigree inconsistent with Mendelian law of inheritance which states that the pair of alleles of every individual is composed of one paternal and one maternal allele. The fact that at least one Mendelian error exists can be effectively proven by showing that every possible combination of compatible genotypes for all the individual violates this law at least once. Since the number of these combinations grows exponentially with the number of individuals, only tiny pedigree can be checked by enumeration. The problem of checking pedigree consistency is actually shown to be NP-complete in [1]. Other errors are called non Mendelian errors[1].

The detection and correction of errors is crucial before the data can be exploited for the construction of the genetic map of a new organism (genetic mapping) or the localization of genes (loci) related to diseases or other quantitative traits. Because of its NP-completeness, most existing tools only offer a limited polynomial time checking procedure. The only tool we know that really tackles this problem is PedCheck [10,11], although the program is restricted by a single error assumption. Evaluation of animal pedigrees with thousand of animals, including many loops (a marriage between two individuals which have a parental relation) (resulting in large tree-width) and without assuming the unlikely uniqueness of errors requires further improvements.

In this paper, we introduce soft constraint network models for the problem of checking consistency, the problem of determining the minimum number of errors needed to explain the data and the problem of proposing an optimal correction to an error. These problems offer attractive benchmarks for (weighted) constraint satisfaction. We report preliminary results using the weighted constraint network solver `toolbar`.

2 Modeling the problems

A constraint satisfaction network (X, D, C) [3] is defined by a set of variables $X = \{x_1, \ldots, x_n\}$, a set of matching domains $D = \{d_1, \ldots, d_n\}$ and a set of constraints C. Every variable $x_i \in X$ takes its value in the associated domain d_i. A constraint $c_S \in C$ is defined as a relation on a set of variables $S \subset X$ which defines authorized combination of values for variables in S. Alternatively, a constraint may be seen as the characteristic function of this set of authorized tuples. It maps authorized tuples to the boolean *true* (or `1`) and other tuples to *false* (or `0`). The set S is called the scope of the constraint and $|S|$ is the arity of the constraint. For arities of one, two or three, the constraint is respectively said to be unary, binary or ternary. A constraint may be defined by a predicate that decides if a combination is authorized or not, or simply as a set of combination of values which are authorized. For example, if $d_1 = \{1, 2, 3\}$ and $d_2 = \{2, 3\}$, two possible equivalent definitions for a constraint on x_1 and x_2 would be $x_1 + 1 > x_2$ or $\{(2, 2), (3, 2), (3, 3)\}$.

[1] Non Mendelian errors may be identified only in a probabilistic way using several locus simultaneously and a probabilistic model of recombination and errors [4].

The central problem of constraint networks is to give a value to each variable in such a way that no constraint is violated (only authorized combinations are used). Such a variable assignment is called a solution of the network. The problem of finding such a solution is called the Constraint Satisfaction Problem (CSP). Proving the existence of a solution for an arbitrary network is an NP-complete problem.

Often, tuples are not just completely authorized or forbidden but authorized and forbidden to some extent or at some cost. Several ways to model such problems have been proposed among which the most famous are the semi-ring and the valued constraint networks frameworks [2]. In this paper we consider weighted constraint networks (WCN) where constraints map tuples to non negative integers. For every constraint $c_S \in C$, and every variable assignment A, $c_S(A[S]) \in N$ represents the cost of the constraint for the given assignment where $A[S]$ is the projection of A on the constraint scope S. The aim is then to find an assignment A of all variables such that the sum of all tuple costs $\sum_{c_S \in C} c_S(A[S])$ is minimum. This is called the Weighted Constraint Satisfaction Problem (WCSP), obviously NP-hard. Several recent algorithms for tackling this problem, all based on the maintenance of local consistency properties [13] have been recently proposed [9,7].

2.1 Genotyped pedigree and constraint networks

Consider a pedigree defined by a set I of individuals. For a given individual $i \in I$, we note $pa(i)$ the set of parents of i. Either $pa(i) \neq \varnothing$ (non founder) or $pa(i) = \varnothing$ (founder). At the locus considered, the set of possible alleles is denoted by $A = 1, ..., m$. Therefore, each individual carries a genotype defined as an unordered pair of alleles (one allele from each parent, both alleles can be identical). The set of all possible genotypes is denoted by G and has cardinality $\frac{m(m+1)}{2}$. For a given genotype $g \in G$, the two corresponding alleles are denoted by g^l and g^r and the genotype is also denoted as $g^l|g^r$. By convention, $g^l \leq g^r$ in order to break symmetries between equivalent genotypes (e.g. 1|2 and 2|1). The experimental data is made of phenotypes. For each individual in the set of observed individuals $I' \subset I$, its observed phenotype restricts the set of possible genotypes to those which are compatible with the observed phenotype. This set is denoted by $G(i)$ (very often $G(i)$ is a singleton, observation is complete).

A corresponding constraint network encoding this information uses one variable per individual *i.e.* $X = I$. The domain of every variable $i \in X$ is simply defined as the set of all possible genotypes G. If an individual i has an observed phenotype, a unary constraint that involves the variable i and authorizes the genotypes in $G(i)$ is added to the network. Finally, to encode Mendelian law, and for every non founder individual $i \in X$, a single ternary constraint involving i and the two parents of i, $pa(i) = \{j, k\}$ is added. This constraint only authorizes triples (g_i, g_j, g_k) of genotypes that verify Mendelian inheritance *i.e.* such that one allele of g_i appears in g_j and the other appears in g_k. Equivalently:

$$(g_i^l \in g_j \wedge g_i^r \in g_k) \vee (g_i^l \in g_k \wedge g_i^r \in g_j)$$

For a pedigree with n individuals among which there are f founders, with m possible alleles, we obtain a final CN with n variables, a maximum domain size of $\frac{m(m+1)}{2}$ and $n-f$ ternary constraints. Existing problems may have more than 10,000 individuals with several alleles (the typical number of alleles varies from two to a dozen).

A small example is given in Fig. 1. There are $n = 12$ individuals and $m = 3$ distinct alleles. Each box corresponds to a male individual, and each ellipse to a female. The arcs describe parental relations. For instance, individuals 1 and 2 have three children 3,4, and 5. The founders are individuals 1,2,6, and 7 ($f = 4$). The possible genotypes are $G = \{1|1, 1|2, 1|3, 2|2, 2|3, 3|3\}$. There are 7 individuals (1,3,6,7,10,11, and 12) with an observed phenotype (a single genotype). The corresponding CSP has 12 variables, with maximum domain size of 6, and 8 ternary constraints. This problem is inconsistent.

A solution of this constraint network defines a genotype for each individual that respects Mendelian law (ternary constraints) and experimental data (domains) and the consistency of this constraint network is therefore obviously equivalent to the consistency of the original pedigree. As such, pedigree consistency checking offers a direct problem for constraint networks. In practice, solving this problem is not enough (i) if the problem is consistent, one should simplify the problem for further probabilistic processing by removing all values (genotypes) which do not participate in any solution, this specific problem is known as "genotype elimination", (ii) if the problem is inconsistent, errors have to be located and corrected.

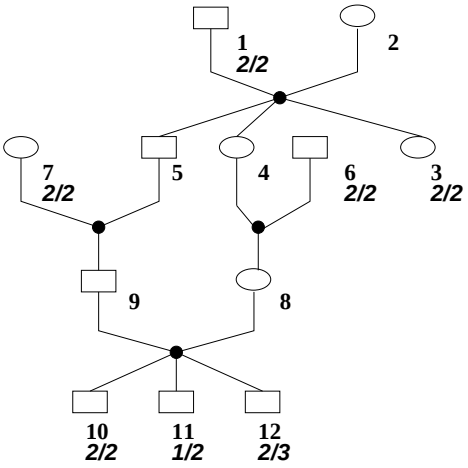


Fig. 1. Pedigree example taken from [11] with 12 individuals.

3 Error detection

From an inconsistent pedigree, the first problem is to identify errors. In our knowledge, this problem is not perfectly addressed by any existing programs which either identify a not necessarily minimum cardinality set of individuals that only restore a form of local consistency or make the assumption that a single error occurs. The first approach may detect too many errors that moreover may not suffice to remove global inconsistency and the second approach is limited to small datasets (even high quality automated genotyping may generate several errors on large datasets).

A typing error for individual $i \in X$ means that the domain of variable i has been wrongly reduced: the true (unknown) value of i has been removed. To model the possibility of such errors, genotypes in G which are incompatible with the observed phenotype $G(i)$ should not be completely forbidden. Instead, a soft constraint forbids them with a cost of 1 (since using such a value represents one typing error). We thereby obtain a weighted constraint network with the same variables as before, the same hard ternary constraints for Mendelian laws and soft unary constraints for modeling genotyping information. Domains are all equal to G.

If we consider an assignment of all variables as indicating the real genotype of all individuals, it is clear that the sum of all the costs induced by all unary constraints on this assignment precisely gives the number of errors made during typing. Finding an assignment with a minimum number of errors follows the traditional parsimony principle (or Ockham's razor) and is consistent with a low probability of independent errors (quite reasonable here). One solution of the corresponding WCSP with a minimum cost therefore defines a possible diagnostic (variable assigned with a value forbidden by $G(i)$ represent one error).These networks have the same size as the previous networks with the difference that all variables now have the maximum domain size $|G|$. The fundamental difference lies in the shift from satisfaction to optimization. The fact that only *unary* soft constraints arise here is not a simplification in itself w.r.t. the general WCSP since every n-ary weighted constraint network can be simply translated in an equivalent dual network with only unary soft constraints and hard binary constraints [8].

In the previous example of Fig. 1, the problem still has 12 variables, with domain size of 6. It has 8 hard ternary constraints and 7 soft unary constraints. The minimum number of typing errors is one. There are 66 optimal solutions of cost one, which can occur in any of the typed individuals except individual 10. One optimal solution is $\{(1, 2|2)$, $(2, 1|2)$, $(3, 2|2)$, $(4, 1|2)$, $(5, 2|2)$, $(6, 2|2)$, $(7, 2|2)$, $(8, 1|2)$, $(9, 2|2)$, $(10, 2|2)$, $(11, 1|2)$, $(12, 2|2)\}$ such that the *erroneous* typing 2|3 of individual 12 has been changed to 2|2.

3.1 Error correction

When errors are detected, one would like to optimally correct them. The simple parsimony criterion is usually not sufficient to distinguish alternative values.

More information needs to be taken into account. Errors and Mendelian inheritance being typically stochastic processes, a probabilistic model is attractive. A Bayesian network is a network of variables related by conditional probability tables (CPT) forming a directed acyclic graph. It allows to concisely describe a probability distribution on stochastic variables. To model errors, a usual approach is to distinguish the observation O and the truth T. A CPT $P(O|T)$ relates the two variables and models the probability of error.

Following this, we consider the following model for error correction: we first have a set of n variables T_i each representing the true (unknown) genotype of individual i. The domain is G. For every observed phenotype, an extra observed variable O_i is introduced. It is related to the corresponding true genotype by the CPT $P_i^e(O_i|T_i)$. In our case, we assume that there is a constant α probability of error: the probability of observing the true genotype is $1-\alpha$ and the remaining probability mass is equally distributed among remaining values.

For the individual i and its parents $pa(i)$, a CPT $P_i^m(T_i|pa(i))$ representing Mendelian inheritance connects T_i and its corresponding parents variables. Each parent having two alleles, there are four possible combinations for the children and all combinations are equiprobable (probability $\frac{1}{4}$). However, since all parental alleles are not always different, a single children genotype will have a probability of $\frac{1}{4}$ times the number of combination of the parental alleles that produce this genotype.

Finally, prior probabilities $P^f(i)$ for each genotype must be given for every founder i. These probabilities are obtained by directly estimating the frequency of every allele in the genotyped population. For a genotype, its probability is obtained by multiplying each allele frequency by the number of way this genotype can be built (a genotype $a|b$ can be obtained in two ways by selecting a a from a father and a b from the mother or the converse. If both alleles are equal there is only one way to achieve the genotype). The probability of a complete assignment $P(O,T)$ (all true and observed values) is then defined as the product of the three collections of probabilities (P^e, P^m and P^f). Note that equivalently, its log-probability is equal to the sum of the logarithms of all these probabilities.

The evidence given by the observed phenotypes $G(i)$ is taken into account by reducing the domains of the O_i variables to $G(i)$. One should then look for an assignment of the variables $T_i, i \in I'$ which has a maximum a posteriori probability (MAP). The MAP probability of such an assignment is defined as the sum of the probabilities of all complete assignments extending it and maximizing it defines an NP^{PP}-complete problem [12], for which no exact methods exist that can tackle large problems. PedCheck tries to solve this problem using the extra assumption of a unique already identified error. This is not applicable in large datasets either. Another very strong assumption (known as the Viterbi assumption) considers that the distribution is entirely concentrated in its maximum and reduces MAP to the so-called Maximum Probability Explanation problem (MPE) which simply aims at finding a complete assignment of maximum probability. Using logarithms as mentioned above, this problem directly reduces to

a WCSP problem where each CPT is transformed in an additive cost function. This allows to solve MPE using the dedicated algorithms introduced in `toolbar`.

Taken the pedigree example given in Fig. 1, we computed the maximum likelihood $P(O,T|e)$ for all possible minimal error corrections using `toolbar` with the MPE formulation. Note that only one correction is needed here to suppress all Mendelian errors. For a given error correction $T_j = g$, the given evidence e corresponds to reducing the domains of all the O_i *and* T_i variables to $G(i)$, except for T_j which is assigned to g. We used equifrequent allele frequencies $P^f(i)$ for the founders and a typing error probability $\alpha = 5\%$. The results are given in Table 1. The ratio $\frac{P(O,T|e)^{best}}{P(O,T|e)}$ is the ratio between the best found maximum likelihood for any correction and the maximum likelihood for a given correction. The results shown that either individual 11 or 12 is the most likely source of the error. Optimal corrections with respect to the Viterbi assumption are either $T_{11} = 2|2$, or $T_{11} = 2|3$, or $T_{12} = 1|2$, or $T_{12} = 2|2$.

Individual	Correction	$P(O,T\|e)$	$\frac{P(O,T\|e)^{best}}{P(O,T\|e)}$
1	1|2	$3.462e-10$	128.0
1	2|3	$3.462e-10$	128.0
3	1|2	$2.77e-09$	16.0
3	2|3	$2.77e-09$	16.0
6	1|1	$5.539e-09$	8.0
6	1|2	$2.77e-09$	16.0
6	1|3	$2.77e-09$	16.0
6	2|3	$2.77e-09$	16.0
6	3|3	$5.539e-09$	8.0
7	1|1	$5.539e-09$	8.0
7	1|2	$2.77e-09$	16.0
7	1|3	$2.77e-09$	16.0
7	2|3	$2.77e-09$	16.0
7	3|3	$5.539e-09$	8.0
11	2|2	$4.431e-08$	1.0
11	2|3	$4.431e-08$	1.0
11	3|3	$5.539e-09$	8.0
12	1|1	$5.539e-09$	8.0
12	1|2	$4.431e-08$	1.0
12	2|2	$4.431e-08$	1.0

Table 1. Likelihood ratios for possible error corrections for the problem given in Fig.1.

4 Conclusion

Preliminary experiments on the applicability of WCSP techniques to these problems have been made. The first real data sets are human pedigree data presented

in [10,11]. For all these problems, `toolbar` solves the consistency, error minimization and correction (MPE) problems very rapidly.

The real challenge lies in a real 129,516 individual pedigree of sheeps (*sheep4n*), among which 2,483 are typed with 4 alleles and 20,266 are founders. By removing uninformative individuals, it can be reduced to a still challenging 8,990 individual pedigree (*sheep4nr*) among which 2,481 are typed and 1,185 are founders. As an animal pedigree, it contains many loops and the min-fill heuristics gives a 226 tree-width bound. From the reduced pedigree *sheep4nr*, we removed some simple nuclear-family typing errors by removing parental and typing data of the corresponding erroneous children, resulting in a 8,920 individual pedigree (*sheep4r*) among which 2,446 are typed and 1,186 are founders. We solved the error detection problem (Section 3) using a decomposition approach. We used the hypergraph partitioning software *shmetis*[2] [6] using multilevel recursive bisection with imbalance factor equal to 5% to partition the *sheep4r* hypergraph into four parts. Among the 8,662 hyperedges (each hyperedge connects one child to its parents), 1,676 have been cut, resulting into four independent subproblems described in Table 2. This table gives the minimum number of typing errors (*Optimum*), and if non zero and equal to one, the list of individual identifiers such that removing one typing will suppress all Mendelian errors. The experiments were performed on a 2.4 GHz Xeon computer with 8 GB. The Table reports CPU time needed by `toolbar` and `PedCheck` [10,11] (*level 3*) to find the list of erroneous individuals. We used default parameters for `toolbar` except a preprojection of ternary constraints into binary constraints, a singleton consistency preprocessing [5], and an initial upper bound equal to two. Recall that `PedCheck` is restricted by a single error assumption.

Instance	N. of individuals	Optimum	Errors	`toolbar` CPU time	`PedCheck` CPU time
sheep4r_4_0	1,944	0	$\emptyset$	33.3 sec	27.1 sec
sheep4r_4_1	2,037	0	$\emptyset$	35.7 sec	28.2 sec
sheep4r_4_2	2,254	1	{126033}	124.2 sec	3 hours 51 min
sheep4r_4_3	2,685	1	{119506, 128090, 128091, 128092}	295.0 sec	7 hours 25 min

Table 2. Error detection for a large pedigree *sheep4r* decomposed into four independent weighted CSPs.

By removing two typings in *sheep4r* (individuals 126033 for *sheep4r_4_2* and 119506 for *sheep4r_4_3*), we found that the modified pedigree has no Mendelian errors.

These problems have been made available as benchmarks on the `toolbar` web site (carlit.toulouse.inra.fr/cgi-bin/awki.cgi/SoftCSP) for classical, weighted CSP and Bayesian net algorithmicists.

[2] `http://www-users.cs.umn.edu/~karypis/metis/hmetis/index.html`.

References

1. Aceto, L., Hansen, J. A., Ingólfsdóttir, A., Johnsen, J., and Knudsen, J. The complexity of checking consistency of pedigree information and related problems. *Journal of Computer Science Technology 19*, 1 (2004), 42–59.
2. Bistarelli, S., Fargier, H., Montanari, U., Rossi, F., Schiex, T., and Verfaillie, G. Semiring-based CSPs and valued CSPs: Frameworks, properties and comparison. *Constraints 4* (1999), 199–240.
3. Dechter, R. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.
4. Ehm, M. G., Cottingham Jr., R. W., and Kimmel, M. Error Detection in Genetic Linkage Data Using Likelihood Based Methods. *American Journal of Human Genetics 58*, 1 (1996), 225-234.
5. S. de Givry. Singleton consistency and dominance for weighted csp. In *Proc. of 6th International CP-2004 Workshop on Preferences and Soft Constraints*, page 15p., Toronto, Canada, 2004.
6. Karypis, G., Aggarwal, R., Kumar, V., and Shekhar S. Multilevel Hypergraph Partitioning: Applications in VLSI Domain. In *Proc. of the Design and Automation Conference* (1997).
7. Larrosa, J., de Givry, S., Heras, F., and Zytnicki, M. Existential arc consistency: getting closer to full arc consistency in weighted CSPs. In *Proc. of the 19^{th} IJCAI* (Aug. 2005).
8. Larrosa, J., and Dechter, R. On the dual representation of non-binary semiring-based CSPs. CP'2000 Workshop on Soft Constraints, Oct. 2000.
9. Larrosa, J., and Schiex, T. In the quest of the best form of local consistency for weighted CSP. In *Proc. of the 18^{th} IJCAI* (Aug. 2003), pp. 239–244.
10. O'Connell, J. R., and Weeks, D. E. PedCheck: a program for identification of genotype incompatibilities in linkage analysis. *American Journal of Human Genetics 63*, 1 (1998), 259–66.
11. O'Connell, J. R., and Weeks, D. E. An optimal algorithm for automatic genotype elimination. *American Journal of Human Genetics 65*, 6 (1999), 1733–40.
12. Park, J. D., and Darwiche, A. Complexity results and approximation strategies for map explanations. *Journal of Artificial Intelligence Research 21* (2004), 101–133.
13. Schiex, T. Arc consistency for soft constraints. In *Principles and Practice of Constraint Programming - CP 2000*, vol. 1894 of *LNCS*, pp. 411–424.

Disulfide bonds prediction using Inductive Logic Programming

Ingrid Jacquemin[1] and Jacques Nicolas[2]

[1] IRISA-INRIA, Symbiose Project, Campus University of Beaulieu, 35042 Rennes,
Ingrid.Jacquemin@irisa.fr
[2] Jacques.Nicolas@irisa.fr,
WWW home page: http://www.irisa.fr/

Abstract. We address the issue of predicting disulfide bonds in protein sequences. The knowledge of the state of cysteines is important since they are frequently involved in active sites or give information on the cellular location of proteins, and the formation of covalent links between them is now widely accepted as a key step in the 3D structure prediction task. We propose to tackle with the problem of prediction of interactions between cysteines in the framework of inductive logic programming (ILP). We obtain explicit rules of prediction, whose biological validity needs further investigation.
Contact: ijacquem@irisa.fr

1 The biological problem

The main properties of proteins may be determined from their three-dimensional structure. There are close to or even more than 1 million known sequences but only relatively few known folds (about 10 000 in the Protein Data Bank). Bridging this "knowledge gap" requires automatic methods capable of inferring folds from sequences. Our contribution to this ambitious goal addresses the prediction of particular interactions inside these macro-molecules. The tertiary folds of native proteins are defined by a large number of weak interactions. Proteins are also stabilized covalently by disulfide bonds formed by a pair of cysteine residues in the folded state. We focus on the prediction of these disulfide bonds. A disulfide bond is formed by the oxidative linkage of two cysteines through their thiol groups. In proteins some cysteines, called cystines, are oxidized and the others are called free cysteines. The correct prediction of this type of bonds could be of great help in reducing the complexity of the three-dimensional structure prediction problem, by constraining the global shape of the predicted protein, there exist a lot of articles like [5,7,15,2,21]. Errami *et al.*[8] describe a strategy to perform an exhaustive and objective statistical analysis of three-dimensional structures of proteins. They have shown that amino-acids implied in disulfide bonds are particularly well conserved because the conservation of a fold is due to the conservation of global physico-chemical properties instead of the conservation of weak interactions.

In such a context, two issues of increasing difficulty have to be distinguished:

1. Is it possible to predict the implication of a given cysteine is implied in a disulfide bridge only with the knowledge of its context?
2. Is it possible to predict which pairs of cysteines are connected with a disulfide bridge within a given protein?

A number of works have addressed the first issue. We have not the place to quote all the work but we can cite the most recent one: in 2004, Song *et al.* have worked on a simpler linear discriminator based on dipeptide composition of proteins ([24]). The prediction was performed with a new even larger dataset with 8114 cysteine-containing segments extracted from 1856 non-homologous proteins of well-resolved three-dimensional structures. The prediction accuracy of the oxidation form of cysteines scores is as high as 89%. We can also notice the best score of 90% in overall prediction accuracy obtained by [4] who use the SVM method based on multiple feature vectors (combining local sequences and global amino acid compositions) coupled with cysteine state sequences. Very few studies have worked on the second issue. Casadio *et al.*[3] have worked on the location of disulfide bridges when candidate cystines are known. Their method is based on a weighted graph representation of disulfide bridges. Each vertex represents oxidized cysteines and undirected edges are labeled by contact potential of the associated pair of cysteines. The most recent work is presented by Vullo *et al.*[31], they have developed an *ad-hoc* RNN architecture for scoring labeled undirected graphs that represent connectivity patterns. In this paper we propose a novel approach using the inductive logic programming (ILP) framework to work on both problems in an integrated way. Our goal is to infer specific patterns on physico-chemical properties of amino acids around cystines. We present in this paper only the first issue because the second one is on the way.

2 Progol

Inductive logic programming is at the intersection of machine learning and logic programming: examples of the training set are expressed as logical assertions and the result of learning is a logic program, i.e. a set of definite clauses. ILP has been applied successfully in a large number of applications in structural biology. Muggleton *et al.* have written several articles using ILP to learn protein three-dimensional fold signatures [6,30,26]. They have shown that their rules can be explained in terms of structural and/or functional concepts, such as active site location. A few programs for ILP are available. For prototyping, we have worked with the very flexible environment Aleph[25]. For more efficient programming we have used Progol[23,18], available at [20].
Structures of proteins are the result of complex interactions between sub-structures. For this reason, ILP algorithms, that learn relations between data and not only the prediction functions, which seems to be particularly relevant for this type of data. Another advantage of inductive learning, with respect to heterogeneous sets of data like protein sequences, is the by-product clustering into subgroups that is

achieved by the production of a set of prediction rules. The main difficulty about a large use of ILP techniques is that, like most combinatorial machine learning methods, ILP needs a careful tuning of many parameters. Solving a problem needs to choose a trade-off between the size of the hypothesis space to explore and the discriminating power of the corresponding level of expressiveness. One can distinguish three types of tasks towards this goal:
The first one consists of designing an adequate *representation of training instances together with the background knowledge*. Training instances are coded as logical facts, using the special purpose predicate *example* in Progol. Background knowledge is represented as a set of standard Prolog clauses, that is, definite clauses. It is an essential component in the learning strategy, containing the definition of potentially interesting relations, and integrity constraints. Background knowledge ensures the capacity of inferred relations both to predict new instances of the concept to be learned and to explain in an interpretable way the characteristics of this concept. This differs clearly from the methods such as neural networks where it is difficult to interpret classifiers.
The second task is the *specification of the hypothesis space*. This is achieved in Progol via head-and-body mode declarations (*modeh* and *modeb* predicates), together with types declarations. The general rules Body $\Rightarrow$ Head constructed by Progol are constrained to use only the indicated predicates, variables and constants. It then delineates a finite, generally huge space called Herbrand universe. This space is further reduced by applying *prune*, a user-definable predicate describing rules that are not valid in the previous space.
The last task concerns various parameters influencing the *strategy* and complexity of the search. For instance one can fix the maximum number of rules explored during the search, specify the maximum length of the body of the general rules which Progol constructs, allow rules to predict a small percentage of the negative instances (*noise* level) or set Progol to learn from positive examples only. For more details readers can report to the Progol manual[19].

3 The experimentation

3.1 Disulfide bonds databases and example representation

As usual in machine learning, the quality of initial data is crucial with respect to the quality of the results. Most authors extract disulfide bonds from proteins in the current release of SwissProt[27], for which intra-chains disulfide bonds are known. Since a few years, disulfide bond determination has been made easier [13], and a number of annotations progressively become available. However, one must clearly distinguish, between experimentally verified and predicted bonds (by similarity, potential, probable...). Actually, less than 12% of annotated disulfide bonds (in 16% of proteins) are determined experimentally ! In 2004, a number of efforts have lead to databases of disulfide patterns. In [14], a method based on SwissProt and Pfam[22] multiple alignments databases leads to 94499 disulfide patterns, but authors give no access to the corresponding database. The training database we used was provided by Dominique Tessier, who carefully selected a

set of chains from a pdb_select_25 dataset. They contain at least one disulfide bond annotation and are extracted from eukaryotic cells. In her paper[29], she points to the fact that signal peptides and subcellular locations are the primary descriptors that must be considered, if available, for the prediction of the bonded state of cysteines. We selected in her database extracellular proteins with intramolecular disulfide bonds. We worked on a training set containing about 722 positive (oxidized cysteines) and 47 negative (free cysteines) sequences. We have only 47 negative sequences because we worked with proteins becoming from the same environment which favours disulfide bonds. A number of descriptors are available, some of them being very discriminant such as the presence of signal peptides. We use a second database, available at [1], to validate our results. It was provided by C.Geourjon during the ACI GenoTo3D and contains 1129 positive sequences. We have taken out the redundant sequences of the two databases. However, we have restricted ourselves to the simplest information of the primary sequence. Indeed, we are not only interested in the prediction of the state of cysteines, but we try also to understand how pairing of cysteines is determined depending on the sequence. We have used the common assumption that the presence of a bond between two cysteines is correlated with the local environment of neighborings residues in the 3D spatial conformation of the protein. Most authors suggest that it might be sufficient to extract from the primary sequence a context window of 11 residues centered around cysteines. We have chosen a slightly higher value, representing disulfide bridges with pairs of windows of length 14 (1 window for each cysteine, 7 amino-acids on each side of the cysteine). This allows to take into account more residues that are close with respect to the 3D structure but more distant on the primary chain. From the set of positive context sequences, we have extracted 260 pairs of sequences corresponding to disulfide bonds.

As seen in figure 1, an example is thus represented as a predicate with one

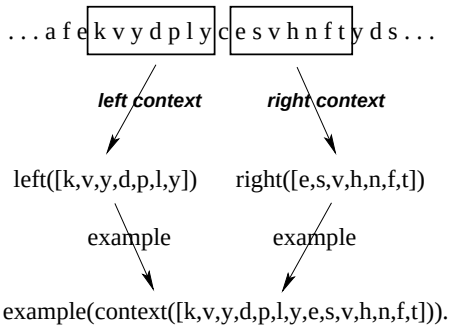


Fig. 1. Examples : selection of two windows on each side of a cysteine in a protein.

argument, $example(Context)$, where $Context$ describes a window of 14 amino-

acids, seven amino-acids on each side of cysteines. For positive instances, this window contains amino-acids around cystines whereas for negative examples this window represents amino-acids around free cysteines in the protein.

3.2 Hypothesis Space for the cysteine state prediction problem

The first issue we have addressed is to characterize the windows of amino-acids around oxidized cysteines. Our approach relies on finding detailed relevant properties of *sub-sequences* in such windows. Since our aim is to reveal a maximum number of potential relations between some complexes of residues, we combine logical and statistical aspects: the final property is a conjunction of patterns on sub-sequences, but the search of patterns themselves corresponds to studying the composition in sub-windows of fixed size. We start with some definitions precising the formal setting of the corresponding hypothesis space.

Definition 1. *Let S be a sequence on alphabet Σ, then $W_1...W_k \in \Sigma^k$ is a sub-sequence of S iff* $\exists u_1...u_{k+1} \in \Sigma^\star, S = u_1 W_1 ... u_k W_k u_{k+1}$

Definition 2. *Let S be a sequence on alphabet Σ, then a k-pattern is a conjunction of sub-sequences of properties $P_1\&...\&P_l$ of S, that is true iff*

$\exists v_1...v_k \in \Sigma^k$, a sub-word (ie. a sub-sequence without gap) of S and
$\exists L$ a labelling from $\{v_1, ..., v_k\}$ to $\{v_{11}...v_{lm(l)}\}$ such that
$\forall i \in [1, l], (v_{i1}...v_{im(i)}$ is a sub-sequence of $v_1...v_k$ and $P_i = p_{i1}...p_{im(i)} \Rightarrow$
$\forall j \in [1, m(i)], p_{ij}(v_{ij}))$., where p_{ij} denote elementary properties of letters (in the sequence of amino-acids).

Fig. 2. Sub-Window for detecting patterns of length 4 in the context of a cysteine

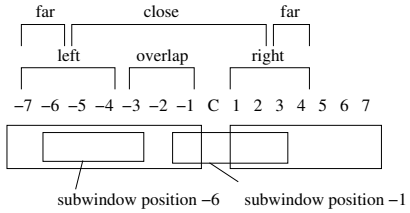


Fig. 3. Values of position attribute with respect to sub-window's starting position (context of size 14, sub-window of size 4)

The hypothesis space is made of conjunctions of k-patterns. In our experiments, we have fixed k to 4, and thus properties are checked on sub-windows of length 4 inside the cysteine context (see figure 2).

Sub-sequences of properties are predicates denoted by words $p_1...p_m$ of physico-chemical properties that are true iff the corresponding sub-sequence $v_1...v_m$ verify all the given physico-chemical properties. We have used for this purpose the Taylor diagram[28], which is a Venn diagram specifying relevant groups of amino-acids. Note that it is useful to add the whole class X of amino-acids to this diagram. Indeed, this allows to introduce "don't-care" positions in the model. Since this is useful only for reasonable sizes of subwords, this universal property will be added directly to the set of allowed arrangements of sub-sequences (see below the description of sub-sequence types in the background knowledge).
For example, consider the following context around a cysteine:
$example(AARDHEW, AGNGIFQ)$. Then, $[cccc \wedge ssss]$ or $[(+ + \& - -) \wedge (hs\&GG)]$, where c, s, +, -, h, G denote respectively properties charged, small, positive, negative, hydrophobic and amino-acid G, are valid conjunctions of 4-patterns, true on the subwords RDHE and AGNG of the example (at positions -5 and +1 with respect to the cysteine). *cccc* reflects the fact that all amino-acids of $RDHE$ are charged and $(+ + \& - -)$ reflects the fact that sub-sequences RH and DE are respectively positively and negatively charged.
In fact, keeping the exact position of each sub-window in the context might lead to overspecialized prediction rules. We have chosen to retain only a limited number of types for each sub-window, in a qualitative position attribute,which values belong to the set $\{overlap, close, far, left, right\}$ (see figure 3).

The corresponding hypothesis space is huge and we have to limit the number of conjuncts: given that each amino acid shares 4 elementary properties on the average, with sub-windows of size $k = 4$, and 6 possible types of sub-windows. Then, there exist 256 possible instantiations of properties for a given sub-window and 19 possible arrangements of sub-sequences, giving $5 * 256 * 19 = 24320$ possible typed 4-patterns. Even if we restrict ourselves to a maximum of 3 conjuncts, the size of the space is roughly $24320^3 = 1.5\ 10^{13}$.
Our algorithm explores this space with a two-level search. The first level aims at discovering a relevant subspace of the Herbrand universe, by sampling a restricted subset of hypotheses. The second level aims at discovering prediction rules, using this time the restricted Herbrand universe and the whole hypothesis space. Note that this methodology is not application-dependant and can be used as a general heuristics strategy in ILP. The first hypothesis space is restricted both at the level of k-patterns (no type is introduced) and at the level of the length of produced clauses (at most two). The level of noise in the first level is greater than the level of noise in the second stage, since the hypothesis space contains more general clauses and could therefore cover more negative instances.

The hypothesis space is described and organized in the background knowledge of the Progol program. Practically, sub-sequences of properties are enumerated using a *pattern* predicate that works on sub-windows of size k. For $k = 4$, several types of patterns are possible: *quatuor*, *quatuor1*, *quatuor2*, *quatuor3*, *quatuor4*, *triplet*, *triplet1* and *pair*. *Quatuor* is just a subword of size 4 and *quatuor1, quatuor2, quatuor3, quatuor4* introduce exactly one "don't-care" character at

position 1, 2, 3, 4 respectively; *triplet* denotes two sub-sequences of size 3 and 1 and *triplet1* denotes a sub-sequence of size 3 and one don't care; and *pair* denotes two sub-sequences of size 2 and 2. For instance, if s, p and h denote respectively properties small, polar and hydrophobic :

- *quatuor(h,h,s,p)* represents the pattern hhsp and means that all properties must appear in this exact order in the context around a cystine.
- *quatuor1(x,h,h,p)* (resp. *quatuor2(h,x,h,p), quatuor3(h,h,x,p), quatuor4(h,h,p,x)*) represents the pattern xhhp and means that all properties must appear in this exact order in the context, including any value at the first position (resp. at the second, third, fourth position)
- *triplet(h,h,h,s)* represents the pattern hhh&s, requiring in a sub-window of size 4 in the context around a cystine, both a hydrophobic sub-sequence of size 3 and a small amino-acid. Possible arrangements: *shhh*, *hshh*, *hhsh*, *hhhs*.
- *triplet1(h,h,h)* represents the pattern hhh&x and requires in a sub-window of size 4 in the context around a cystine a hydrophobic sub-sequence of size 3. Possible arrangements are *xhhh*, *hxhh*, *hhxh* and *hhhx*.
- *pair(h,h,p,p)* represents the pattern hh&pp and means that in a sub-window of size 4 in the context around a cysteine, one can find both a hydrophobic sub-sequence of size 2 and a polar sub-sequence of size 2. Possible arrangements are *hhpp*, *hphp*, *hpph*, *pphh*, *phph* and *phhp*.

The set of properties are coded using a tree reflecting the inclusion dependencies among properties. More precisely, we used the "single trick" representation that is often used in Machine learning, coding amino-acids and their properties within the same representation. Based on the set of trees property(hydrophobic(other/aliphatic/aromatic), polar(other/charged(positive/negative)), small(other/tiny)), amino-acids are fully instantiated trees and properties are partially instantiated trees. For instance, the amino-acid G is coded as *tree(hydrophobic(other),false, small(tiny))* and the property *small* is coded as *tree(X,Y,small(Z))*, where X, Y and Z are variables. During stage 2, the background knowledge is increased with the list of patterns extracted from induced rules. The definition of typed-pattern differs from pattern on two points : first, patterns are instantiated using the Herbrand universe of phase 1. Then, the sub-window type is computed during sub-window extraction.

4 Results

First, we have allowed Progol to find rules which can cover 0 to 8 negative instances (noise is allowed). We have obtained 32 rules and we have put them in our background knowledge(BK) for the second stage. Next, we allowed Progol to cover less negative examples, between 0 to 6 and we introduce the typed k-pattern definition defined in the section 3.2. We have obtained 27 rules which have been added to our BK for the third stage. For this one, noise is not permitted and Progol has to find rules using the rules it has found before or building

Table 1. Patterns obtained with Progol and their cover

Pattern P3	%DT	DT	CG	%CG	Pattern P3	%DT	DT	CG	%CG
h3hh3h1-close	19	137	218	19	h1tts	89	87	163	87
sshc hh3ht	35	173	280	37	h1ssh3 ssh1h1	91	65	107	88
sshc ppc	45	162	262	47	phhs phhsfar ptps	92	113	173	89
hh2c1h-close	52	90	142	53	sc2ss tsc2s	93	101	162	90
c2sss pchh	62	148	235	62	cth2p sph	94	75	112	91
h3sc pph1	66	100	178	66	sh1pp-far	95	52	110	92
ph3h3h2 ppsh3	73	171	290	74	h2ph1h h3ps	96	70	95	92
h2pc2p-close	76	43	56	75	h3hhh3 shh3-right spps	97	153	224	93
hh2t-close	78	62	114	77	h1hts-left	98	86	141	93
h3sc-left	79	139	217	80	tpps h2pp-close	98	53	74	94
h3h1h ssh3s	83	118	205	82	h1sth	98	80	153	94
c1pc2p-close	84	43	72	83	pph1s tpps-far sppp	99	18	37	94
shh3 sc1sp	87	139	191	85	h1stc1-right	99	26	49	94

h=hydrophobic h1=h(aliphatic) h2=h(aromatic) h3=h(other) t=tiny c=charged c1=c(positive) c2=c(negative) c3=c(other) p=polar s=small a=aliphatic +=positive %DT(resp.%CG)=sum of examples cover with D.Tessier(resp.C.Geourjon)'s database DT(resp.CG)=number of examples cover in the D.Tessier(resp.C.Geourjon)'s database

new ones. We have obtained 26 rules which can be seen in table 1. This table contains 5 columns: the first one represents rules that Progol has built, a conjunction of typed k-patterns and k-patterns. The second column represents the percentage of examples (positive) covered in our database (the D. Tessier's database); the third one represents the number of positive examples covered by each rule in our database and the fourth and the fifth columns represent the same thing as the second and the first one but on the C.Geourjon's database. The C.Geourjon's database is used to make our validation on other sequences. These two databases have no common examples. The first rule *h3hh3h1-close* obtained by Progol meens that we can find a pattern containing four hydrophobic amino-acids near cystines in 137 of our proteins: the pattern begins with a hydrophobic not aliphatic and not aromatic amino-acid, following by a hydrophobic, following by a hydrophobic not aliphatic and not aromatic and following by an aliphatic. This is a hydrophobic model because it contains only hydrophobic amino-acids and it is close from the cystine. It covers the same percentage of the two databases (19%). Most of the rules describe hydrophobic contexts and often include *small* amino-acids. *small* seems to be a favorable property for disulfide bonds because the protein becomes then more flexible. We can see that with only 17 rules, more than 90% of the databases are covered, which seems a very good result given the simplicity of the characterization.

5 Conclusion and Discussion

We have presented a new method for predicting the disulfide bonds in proteins. Background knowledge gives the advantage of inserting information for the pre-

diction. We have used two databases but with the C.Geourjon's one we have no negative examples, so we can not say if our rules cover false positive. It is difficult to distinguish a negative example because disulfide bonds depend on the environment of the protein. That is why we have considered proteins becoming from the same environment in the D.Tessier database. We are currently investigating the possibility to introduce the secondary structure of proteins, which can be useful to distinguish the models built by Progol. Patterns might be different according to their localisation in the fold. Protein structures are the result of complex interactions between secondary structure elements and the ability of ILP algorithms to learn relations is a key feature. In conclusion we think that the inductive logic programming can help to find explicit patterns around cystines. ILP has never been used to solve the problem of disulfide bonds in proteins. Turcotte *et al.*[26] have used ILP to find description of the major protein fold. They have shown that the rules constructed by Progol can sometimes identify conserved functional motifs. They concluded that relational background knowledge has demonstrable advantages for learning in the construction of fold descriptions. Since we have the same opinion we would like to put more information inside our background knowledge so as to obtain more specific rules on disulfide bonds. Now, we try to work on the second issue: is it possible to predict which pairs of cysteines are connected with a disulfide bridge within a given protein? To solve this problem we use the rules we have obtained to distinguish oxidized cysteines. Experiments are still in progress and will be presented in a futur paper.

References

1. LORIA, IBCP, LIF, IRISA, LIRMM, INRA (2003) http://www.loria.fr/ guermeur/ACIMD/
2. Backofen,R., Will,S.(2003) A Constraint-Based Approach to Structure Prediction for Simplified Protein Models That Outperforms Other Existing Methods, *ICLP 2003*, 49-71
3. Fiser,P., Casadio,R. (2001) Prediction of disulfide connectivity in proteins, *Bioinformatics*, **17**, 957-964
4. Chen,YC., Lin,YS., Lin,CJ., Hwang,JK. (2004) Prediction of the bonding states of cysteines using the support vector machines based on multiple feature vectors and cysteine state sequences, *Proteins: Structure, Function, and Bioinformatics*, **55(4)**, 1036-1042
5. Chuang,CC., Chen,CY., Yang,JM., Lyu,PC., Hwang,JK. (2003) Relationship between protein structures and disulfide-bonding patterns, *Proteins*, **53**, 1-5
6. Cootes,A.,P., Muggleton,S.,H., Greaves,R.,B., Sternberg,M.,J. (2001) Automatic determination of protein fold signatures from structured superposition, *Electronic Transactions in Artificial Intelligence*, **6-B2(026)**, 245-274
7. Creighton,T. (1993) Proteins: Structures and molecular properties, *Freeman*
8. Errami,M., Geourjon,C., Deléage,G. (2003) Conservation of amino acids into multiple alignments involved in pairwise interactions in three-dimensional protein structures, *Journal of Bioinformatics and Computational Biology*, **1(3)**, 505-520
9. Fariselli,P., Riccobelli,P., Casadio,R (1999) Role of evolutionary information in predicting disulfide-bonding state of cysteines in proteins, *Proteins*, **36**, 340-346

10. Fiser,A., Simon,I. (2000) Predicting the oxidation state of cysteines by multiple sequence alignment, *Bioinformatics*, **16**, 251-256
11. Ceroni,A., Frasconi P., Passerini,A., Vullo,A. (2003) Predicting the disulfide bonding state of cysteines with combinations of kernel machines, *Journal of VLSI SIgnal Processing*, **35(3)**, 287-295
12. Frasconi,P., Passerini,A., Vullo,A. (2002) A two−stage SVM architecture for predicting the disulfide bonding state of cysteines, *NNSP'02*
13. Gorman, J., Wallis, T. and Pitt, J. (2002) Protein disulfide bond determination by mass spectrometry, *Mass Spectrom. Rev.*, **21**, 183-216
14. Van Vlijmen, HW., Gupta, A., Narasimhan, LS. and Singh, J. (2004) A novel database of disulfide patterns and its application to the discovery of distantly related homologs, *J Mol Biol., Jan 23*, **335(4)**, 1083-1092
15. Krippahl,L., Barahona,P. (1999) Applying Constraint Programming to Protein Structure Determination, CP, 289-302
16. Martelli,P.,L., Fariselli,P., Malaguti,L., Casadio,R. (2002) Prediction of the disulfide-bonding state of cysteines in proteins at 88% accuracy, *Protein Science*, **11**, 2735-2739
17. Mucchielli-Giorgi,M.,H., Hazout,S., Tufféry,P. (2002) Predicting the disulfide bonding state of cysteines using protein descriptors, *Proteins*
18. Muggleton,S. (1995) Inverse Entailment and Progol, *New Generation Computing Journal*, **13**, 245-286
19. Muggleton,S., Firth,J. CProgol4.4: a tutorial introduction
20. Muggleton,S. (1998) Progol *http://www.doc.ic.ac.uk/s̃hm/progol.html*
21. Palù,A., Dovier,A. Fogolari,F. (2004) Constraint Logic Programming approach to protein structure prediction. *BMC Bioinformatics*, **5**, 186
22. Bateman,A., Coin,L., Durbin,R., Finn,RD, Hollich,V., Griffiths-Jones,S., Khanna,A., Marshall,M., Moxon,S., Sonnhammer,EL, Studholme,DJ, Yeats,C., Eddy,SR (2004) The Pfam protein families database *Nucl. Acids Res.*, **32**, 138-41
23. Roberts,S. (1997) An Introduction to Progol, *URL: ftp://ftp.cs.york.ac.uk/pub/ML_GROUP/Papers/progol-intro.ps.gz*
24. Song,JN., Wang,ML., Li,WJ., Xu,WB. (2004) Prediction of the disulfide-bonding state of cysteines in proteins based on dipeptide composition *Biochem Biophys Res Commun*, **318(1)**, 142-147
25. Srinivasan,A. (2003) The Aleph manual, *http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph*
26. Turcotte,M., Muggleton,SH., Sternberg,MJE. (2001) The effect of relational background knowledge on learning of protein 3D fold signatures, *Mach. Learn.*, 1-15
27. Boeckmann,B., Bairoch,A., Apweiler,R., Blatter,M.-C., Estreicher,A., Gasteiger,E., Martin,M.J., Michoud,K., O'Donovan,C., Phan,I., Pilbout,S., Schneider,M.(2003) The Swiss-Prot protein knowledgebase and its supplement TrEMBL in 2003 *Nucleic Acids Res.*, **31**, 365-370
28. Taylor,W.,R. (1986) The classification of amino acid conservation, *journal of theoretical biology*, **119**, 205-218
29. Tessier,D., Bardiaux,B., Larré,C., Popineau,Y. (2004) Data mining techniques to study the disulfide-bonding state in proteins: signal peptide is a strong descriptor, *Bioinformatics*, **20(16)**, 2509-2512
30. Turcotte,M., Muggleton,S.,H., Sternberg,M.,J.,E. (2001) Automated discovery of structural signatures of protein fold and function, *J. of Mol. Bio.*, **306**, 591-605
31. Vullo,A., Frasconi,P. (2004) Disulfide connectivity prediction using recursive neural networks and evolutionary information, *Bioinformatics*, **20(5)**, 653-659

Efficient Constraint-based Sequence Alignment by Cluster Tree Elimination*

Sebastian Will, Anke Busch and Rolf Backofen

Bioinformatics, Institute of Computer Science, Friedrich-Schiller-University
Ernst-Abbe-Platz 2, 07743 Jena, {*will,busch,backofen*}*@inf.uni-jena.de*

Abstract. Aligning DNA and protein sequences has become a standard method in molecular biology. Often, it is desirable to include partial prior knowledge and conditions in an alignment. The most common and successful technique for efficient alignment algorithms is dynamic programming (DP). However, a weakness of DP is that one cannot include additional constraints without specifically tailoring a new DP algorithm. Here, we discuss a declarative approach that is based on constraint techniques and show how it can be extended by formulating additional knowledge as constraints. We take special care to obtain the efficiency of DP for sequence alignment. This is achieved by careful modeling and applying proper solving strategies.

1 Introduction

Modern molecular biology is not possible without tools for the comparison of the macromolecules DNA, RNA, and proteins. It is most desirable to be able to specify additional restrictions for such similarity search whenever prior knowledge on the analyzed molecules is available. For example, consider the case of a biologist, who knows that certain regions in her sequences share a common local motif. Based on this knowledge, the rest of the sequences should be compared. Then, we need to optimize similarity under the additional constraint that parts of such regions should be matched to each other. Another striking example is the enhancement of RNA or protein comparison by employing knowledge on the structure of the macromolecules [10,3,1,5].

However in general, similarity searching tools on the web do not allow to take such prior knowledge into account automatically. The reason for this deficiency is of algorithmic nature. Only for certain special constraints, alignment algorithms have been discussed. In particular, there are approaches that incorporate anchor constraints [7] and precedence constraints [8]. We will later discuss how such constraints fit into our newly introduced framework as simple cases. Aligning sequences and (to some extent) sequences with additional structural information is commonly and most successfully performed by *dynamic programming (DP)* [9,11,3]. There is no straightforward and general way to extend a DP algorithm in order to take additional knowledge into account.

*This work is partially supported by the EU Network of Excellence REWERSE.

To overcome this, declarative formulations of the alignment problem have been proposed. Due to their use of constraints, such approaches can be extended to incorporate prior knowledge. For this aim, such knowledge is formulated as constraints and added to the model for unconstrained alignment. One such previous approach [6] is based on *integer linear programming (ILP)*. Since in ILP one can only use boolean variables, the ILP model of [6] for aligning two sequences of length n and m introduces $O(nm)$ variables for modeling the alignment edges. Due to the resulting complexity, one needs to introduce artificial restrictions on the possible alignment edges for solving the problem in practice. Furthermore, the solving strategy for ILP does not achieve the efficiency of DP for the unconstrained case. Another declarative approach [12] is based on constraint programming. The approach introduces quadratically many variables and constraints and remodels the given DP algorithm. As a consequence, only a rather restricted class of side constraints can be handled efficiently.

Here, we introduce a new constraint-based approach. The main challenge that we face with our approach is to compete with the very good efficiency of DP in the standard case and allow extension by introducing new constraints. We achieve the desired efficiency and adaption to additional constraints by modeling the alignment problem as a constraint optimization problem in the sense of [2,4] and then applying a special solution strategy, which is known as *cluster tree elimination (CTE)* [4].

2 A Constraint Model for Sequence Alignment

We develop a constraint model for sequence alignment of two sequences $a = a_1 \dots a_n$ and $b = b_1 \dots b_m$ that are both words of the alphabet Σ. To be more precise, we define an *alignment $\mathcal{A}$ of a and b* as an ordered matching of positions in a and b, i.e. as a subset of $\{1, \dots, n\} \times \{1, \dots, m\}$ such that for all $(i, j), (i', j') \in \mathcal{A}$:

1. $i = i'$ if and only if $j = j'$ and
2. $i < i'$ implies $j < j'$.

We call i and j matched by $\mathcal{A}$ if and only if $(i, j) \in \mathcal{A}$.

The *score of an alignment* $\mathcal{A}$, which we want to maximize, depends on the *similarity function on positions* $\sigma : \{1, \dots, n\} \times \{1, \dots, m\} \to \mathbb{R}$ and *gap cost* $\gamma \in \mathbb{R}$. It is defined as

$$\text{score}(\mathcal{A}) = (n + m - 2|\mathcal{A}|)\gamma + \sum_{(i,j)\in\mathcal{A}} \sigma(i, j). \tag{1}$$

The classical DP algorithm for sequence alignment is specified via the recursion equation $D_{i\,j} = \max\{D_{i-1\,j-1} + \sigma(i, j), D_{i-1\,j} + \gamma, D_{i\,j-1} + \gamma\}$ with initialization $D_{0\,0} = 0$, $D_{i\,0} = i\gamma$, and $D_{0\,j} = j\gamma$ for $1 \le i \le n$ and $1 \le j \le m$ and solves the problem in $O(nm)$ time.

Here, we model alignment as a constraint optimization problem in the framework that is described in a more general form in [4]. There, one defines variables

with finite domains and functions on these variables. In our special case, the solution of the problem is a valuation of the variables that maximizes the sum of the function values. Note that hard constraints c can be encoded in this framework by functions that yield $-\infty$ if the constraint is violated and 0 otherwise. Tacitly, our arithmetic is extended canonically in order to handle sums and maximizations involving infinity.

In our model, we represent alignments of a and b by finite domain variables X_i for $1 \leq i \leq n$ with domains $\mathrm{dom}(X_i) = \{0, \ldots, m\}$. Furthermore for technical reasons, we introduce the fixed variables $X_0 = 0$ and $X_{n+1} = m + 1$ and extend σ by defining $\sigma(n + 1, m + 1) = 0$. A given alignment $\mathcal{A}$ is uniquely encoded by a valuation $(X_0 = x_0, \ldots, X_{n+1} = x_{n+1})$ of variables $X_0, \ldots, X_{n+1}$ where 1.) $x_i = j$ if $(i, j) \in \mathcal{A}$ and 2.) $x_i = x_{i-1}$, for every i that is not matched in $\mathcal{A}$. Note that i and j are matched if and only if $x_i = j$ and $x_i > x_{i-1}$. For example, the valuation $\boldsymbol{x} = (0, 1, 2, 5, 6, 6, 6, 7, 8)$ of $X_0, \ldots, X_8$ corresponds to the alignment {(1,1),(2,2),(3,5),(4,6),(7,7)}, which can be represented alternatively by

$$\begin{array}{l} a_1\, a_2 - - a_3\, a_4\, a_5\, a_6\, a_7 \\ b_1\, b_2\, b_3\, b_4\, b_5\, b_6 - - b_7 \end{array}.$$

The only hard constraints on the variables X_i are $X_{i-1} \leq X_i$ for $1 \leq i \leq n+1$. They are modeled by functions

$$leq_i : \mathrm{dom}(X_{i-1}) \times \mathrm{dom}(X_i) \to \{-\infty, 0\}.$$

The scoring scheme is encoded via functions $f_i(X_{i-1}, X_i)$ for $1 \leq i \leq n + 1$ that are defined by

$$f_i(j', j) = \begin{cases} \sigma(i, j) + (j - j' - 1)\gamma & \text{if } j' < j \\ \gamma & \text{otherwise.} \end{cases}$$

Note that we correctly model alignments and their scores. Firstly, a valuation $(X_0 = x_0, \ldots, X_{n+1} = x_{n+1})$ represents an alignment $\mathcal{A}$ of a and b if and only if $\sum_{1 \leq i \leq n+1} f_i(x_{i-1}, x_i) + leq_i(x_{i-1}, x_i)$ is not $-\infty$. Secondly in this case, $\sum_{1 \leq i \leq n+1} f_i(x_{i-1}, x_i)$ equals the score of $\mathcal{A}$ (see Eq. 1).

3 Efficient Solving by Cluster Tree Elimination

Here, we sketch CTE and show its application to the our model. We demonstrate how direct application of CTE yields an $O(nm^2)$ algorithm. Then, by introducing modifications to the standard CTE approach, we improve the complexity to $O(nm)$ time.

For applying CTE, we first need a *cluster tree decomposition (CTD)* [4]. In such a decomposition, we distribute variables and functions to vertices (*clusters*) of a tree, such that 1.) each function occurs in exactly one cluster, 2.) if a function occurs in a cluster, then all variables of the function are assigned to the cluster as well, and 3.) for each variable the set of clusters that contain this variable induces a connected subtree.

Due to the definition, clusters that share variables are connected by edges. The shared variables are called *separator variables*. Figure 1 shows a cluster tree decomposition of our alignment model where edges are labeled by separator variables. We call the cluster consisting of X_{i-1}, X_i, f_i, and leq_i the cluster i. Note that in this figure (and the following ones) we omit the functions leq_i in our presentation.

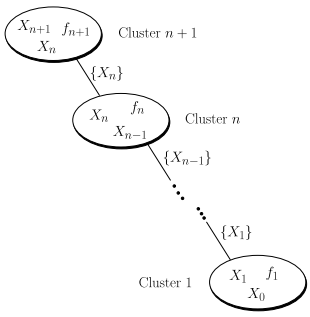


Fig. 1. CTD of pure sequence alignment.

CTE solves a constraint optimization problem by repeatedly exchanging messages between the clusters. The messages are functions that combine the functions of the cluster and marginalize them to the separator variables. Each message becomes a new function of the receiving cluster. From cluster i to cluster $i+1$, CTE sends a function g_i of the separator variable X_i. Beginning with cluster 1 it proceeds until cluster $n+1$ receives its message g_n. When sending a message from cluster i, this cluster is already augmented by a function g_{i-1}. Finally, it can be shown that $\max_{1 \leq j \leq m} (g_n(j) + f_{n+1}(j, m+1))$, which is the marginalization of the functions in cluster $n+1$ to the empty set of variables, is the maximal alignment score.

It remains to show how the messages g_i are computed. Due to the CTE algorithm, the message g_i is defined for $0 \leq j \leq m$ as

$$g_i(j) = \max_{0 \leq j' \leq m} \left(g_{i-1}(j') + f_i(j', j) + leq_i(j', j)\right). \tag{2}$$

Clearly, the standard approach takes $O(m^2)$ time for computing the function g_i. Since $O(n)$ messages are sent until the final alignment score can be computed, this results in an $O(nm^2)$ algorithm. Thereby, we have shown that the direct application of CTE to our constraint model yields a polynomial algorithm for sequence alignment.

Improving complexity. The complexity can be improved further if we employ the internal structure of the functions g_{i-1}, f_i, and leq_i. For this reason, we rewrite Equation 2 by the semantics of leq_i and expand the definition of f_i.

$$g_i(j) = \max_{0 \leq j' \leq j} \left(g_{i-1}(j') + \begin{cases} \sigma(i,j) + (j - j' - 1)\gamma & \text{if } j' < j \\ \gamma & \text{otherwise} \end{cases} \right).$$

Now, we can resolve the case distinction of f_i and move the constant $\sigma(i,j)$ out of the maximization. Then,

$$g_i(j) = \max \begin{cases} \sigma(i,j) + \max\limits_{0 \leq j' < j} \left(g_{i-1}(j') + (j - j' - 1)\gamma\right) \\ g_{i-1}(j) + \gamma. \end{cases}$$

A helper function $g^{\mathrm{m}}(j) = \max_{0 \le j' < j} (g_{i-1}(j') + (j - j' - 1)\gamma)$ can be defined recursively and then computed in $O(m)$ time by DP as

$$g^{\mathrm{m}}(0) = -\infty,\ g^{\mathrm{m}}(1) = g_{i-1}(0),\ \text{and for } j > 1\ g^{\mathrm{m}}(j) = \max \begin{cases} g^{\mathrm{m}}(j-1) + \gamma \\ g_{i-1}(j-1). \end{cases}$$

In consequence, the total computation of g_i is done in $O(m)$. This results in an $O(nm)$ time algorithm for the computation of the alignment score.[1]

4 Extension of the Sequence Alignment Model

Recently discussed constrained alignment approaches handled constraints like anchor constraints and precedence constraints. Such constraints can be encoded in our model straightforwardly and are handled by restricting the domains of variables, which even increases the efficiency of our algorithm. An *anchor constraints*, as discussed in [7], tells that position i in the first sequence can only be aligned to position j in the second sequence and furthermore, positions strictly left (resp. right) of i are aligned to positions strictly left (resp. right) of j. These conditions are expressed in our model by the constraints

$$X_{i-1} < j, X_{i+1} > j, \text{ and}$$
$$X_i = j \vee X_i = X_{i-1};$$

the latter implies $X_i \le j$. The constraints are directly propagated to the domains of X_i and X_{i-1} and do not increase the complexity of our constraint problem. Via the less than constraints the new domain information is further propagated to the domains of all variables.

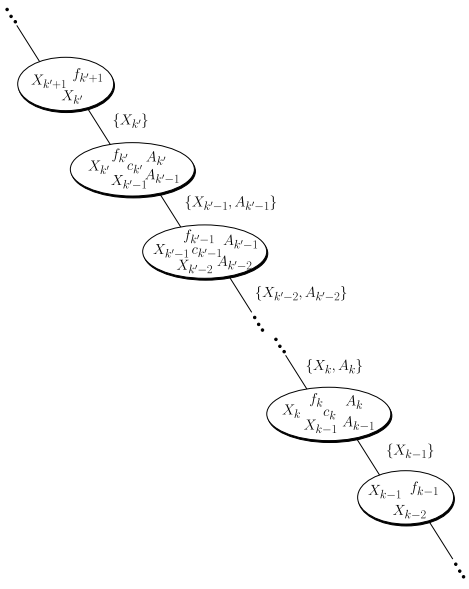


Fig. 2. CTD of an alignment with segment constraints.

A *precedence constraint*, handled in [8], tells that in the alignment position i of the first sequence is left (resp. right) of position j of the second sequence. In our model, corresponding conditions are encoded as $X_i \ge j$ (resp. $X_i \le j$). A further example for a trivial extension of the model is a condition like "position k is aligned to l or l'" (constraints: $X_k \in \{l, l'\}$ and $X_{k-1} < X_k$).

In this section, we discuss two more challenging extensions by example. Namely, the incorporation of prior knowledge on aligned segments and the extension to sequence structure alignment.

[1] $O(m)$ space can be achieved by further modifications to CTE.

Aligned Segments As example of constraining the alignment between segments in a and b, we consider the constraint that at least x% of the positions $\{k, \ldots, k'\}$ in a have to be matched with positions $\{l, \ldots, l'\}$ in b. For extending our model by this constraint, we add variables $A_{k-1}, \ldots, A_{k'}$ and for each $k \leq i \leq k'$ the function $c_i(A_{i-1}, A_i, X_{i-1}, X_i)$ that encodes the hard constraint

$$A_i = A_{i-1} + \begin{cases} 1 & \text{if } X_{i-1} < X_i \text{ and } l \leq X_i \leq l' \\ 0 & \text{otherwise.} \end{cases}$$

We fix $A_{k-1} = 0$. Since the variables A_i count the proper matches in the prefix segment {k,. . . ,i}, we can finally express the constraint by restricting the domains of A_i to $\{\max(0, \lceil \frac{x}{100}(k' - k + 1) \rceil - (k' - i)), \ldots, i - k + 1\}$.

Figure 2 shows the cut-out of the CTD that is affected by the extension of the model. CTE works essentially as in the standard case. For $k \leq i \leq k'$, CTE sends messages g_i depending on the separator variables that each can be computed in $O(m\bar{k})$ time where $\bar{k} = k' - k + 1$. Thus, the total complexity is $O(m(n - \bar{k}) + m\bar{k}^2)$. Note that, as assumed in this result, one can transfer the complexity improvement of the previous section to this case of constrained alignment. It suffices to look at the message g_i from the cluster that contains a variable A_i (and thus contains the variables X_i, X_{i-1}, and A_{i-1} by construction). The message g_i, which depends on values for X_i and A_i, is given (already using the semantic of leq_i and c_i) as

$$g_i(j, a) = \max_{0 \leq j' \leq j} \begin{cases} g_{i-1}(j', a - 1) + f_i(j', j) & \text{if } j' < j \text{ and } l \leq j \leq l' \\ g_{i-1}(j', a) + f_i(j', j) & \text{otherwise.} \end{cases}$$

One transforms further to

$$g_i(j, a) = \max \begin{cases} \sigma(i, j) + g^{\mathrm{m}}(j, a) \\ \gamma + g_{i-1}(j) \end{cases}$$

where we define

$$g^{\mathrm{m}}(j, a) = \max_{0 \leq j' < j} ((j - j' - 1)\gamma + g_{i-1}(j', a'))$$

where $a' = a$ if $l \leq j \leq l'$ and $a' = a - 1$ otherwise. Finally, g^{m} can be defined recursively as in the previous section as

$$g^{\mathrm{m}}(0, a) = -\infty, \; g^{\mathrm{m}}(1, a) = g(1, a'), \text{ and}$$
$$\text{for } j > 1 \; g^{\mathrm{m}}(j, a) = \max \begin{cases} g^{\mathrm{m}}(j - 1, a'') + \gamma \\ g_{i-1}(j - 1, a'') \end{cases}$$

where $a' = a$ if $l \leq 1 \leq l'$ and $a' = a - 1$ otherwise. Furthermore, $a'' = a$ if $l \leq j \leq l'$ and $a'' = a - 1$ otherwise.

We have demonstrated, that for this class of constraints the efficiency can be improved in the same way as in the case of unconstrained alignment. Intuitively, the additional constraints do not interfere with the nature of our score that enables the recursive decomposition.

Sequence Structure Alignment Here as additional input, we have two structures $P_a \subset \{1,\ldots,n\} \times \{1,\ldots,n\}$ and $P_b \subset \{1,\ldots,m\} \times \{1,\ldots,m\}$ and a function $\omega : \{1,\ldots,n\} \times \{1,\ldots,n\} \times \{1,\ldots,m\} \times \{1,\ldots,m\} \to \mathbb{R}$. A pair $(i_l, i_r) \in P_a$ (resp. $(j_l, j_r) \in P_b$) expresses a dependency, e.g. base pairing in RNA, between the positions i_l and i_r (resp. j_l and j_r). The function ω yields a score for aligning pairs of dependent positions.

The score of an alignment $\mathcal{A}$ is now defined in extension of Eq. 1 as

$$\text{score}(\mathcal{A}) + \sum_{\substack{(i_l,i_r)\in P_a,(j_l,j_r)\in P_b,\\ (i_l,j_l)\in\mathcal{A},(i_r,j_r)\in\mathcal{A}}} w(i_l, i_r; j_l, j_r).$$

Our alignment model can be extended by adding for each $(i_l, i_r) \in P_a$ functions $h_{i_l i_r}(X_{i_l-1}, X_{i_l}, X_{i_r-1}, X_{i_r})$ that are defined as

$$h_{i_l i_r}(j'_l, j_l, j'_r, j_r) = \begin{cases} \omega(i_l, i_r; j_l, j_r) & \text{if } j'_l < j_l,\ j'_r < j_r, \text{ and } (j_l, j_r) \in P_b \\ 0 & \text{otherwise.} \end{cases}$$

Figure 3 provides an example for $P_a = \{(k_l, k_r), (l_l, l_r)\}$ and arbitrary P_b, which demonstrates the general construction principle of such a CTD. Due to the base pair (k_l, k_r) (and analogously for (l_l, l_r)), the decomposition contains a node consisting of the variables X_{k_l}, X_{k_r} and their predecessors X_{k_l-1}, X_{k_r-1}, since these variables depend on each other via the function $h_{k_l k_r}$. This node is parent of two sub-trees. In its left sub-tree, we handle the alignment for positions between k_l and k_r and in the right sub-tree the alignment for the positions less than k_l. Due to the conditions for a CTD, the variable X_{k_l} has to be shared with nodes of the left sub-tree, since it is constrained to variables in the leftmost leave.

In this tree structure, CTE begins with the leave vertices and proceeds to the root. From each cluster, it sends a message to its parent cluster. The final alignment score is obtained from the root node.

5 Conclusion

We present the first declarative approach to sequence alignment that is equally efficient as the commonly used method of dynamic programming. However, due to the declarative nature of the presented algorithm, it is extensible by additional constraints. This extensibility subsumes and goes beyond earlier constrained alignment approaches. Especially, we have shown how certain prior knowledge and structure information can be incorporated into the alignment model. By applying cluster tree elimination to the resulting extended alignment problem, we solve it efficiently. Finally, we have demonstrated for the alignment problem how CTE could profit from intelligent reasoning on the constraint model. Thereby, we hint at possible improvements of a current constraint solving strategy.

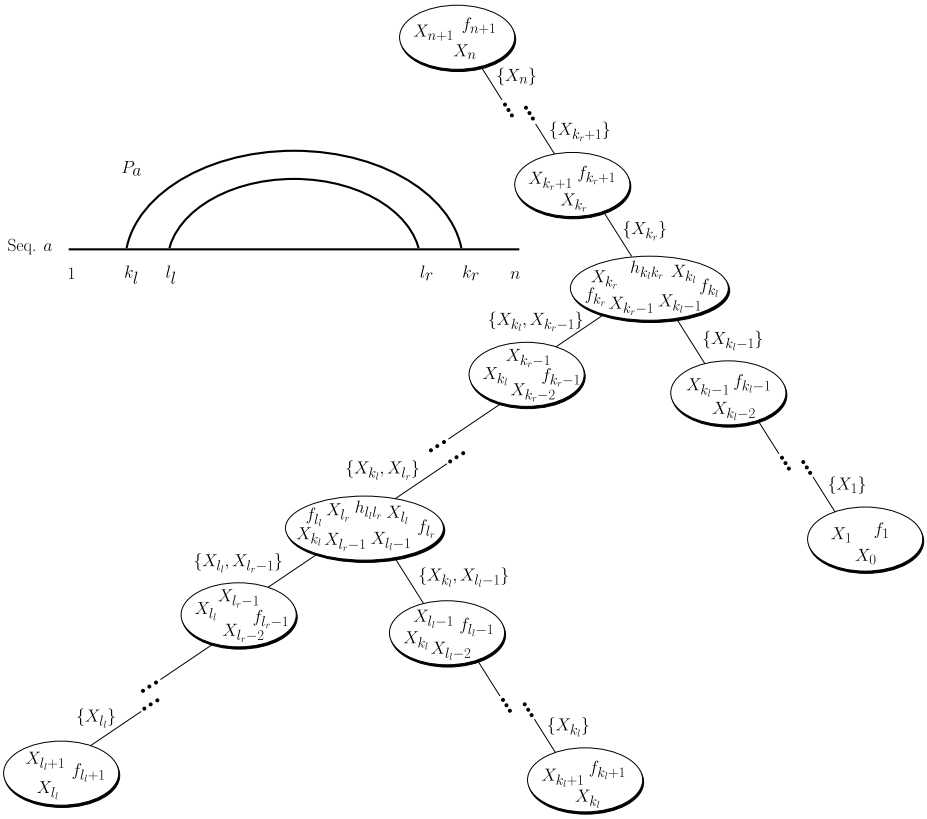


Fig. 3. Example sequence structure alignment CTD (see text for details).

References

1. Rolf Backofen and Sebastian Will. Local sequence-structure motifs in RNA. *Journal of Bioinformatics and Computational Biology (JBCB)*, 2(4):681–698, 2004.
2. Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, 1997.
3. Tao Jiang, Guohui Lin, Bin Ma, and Kaizhong Zhang. A general edit distance between RNA structures. *Journal of Computational Biology*, 9(2):371–88, 2002.
4. Kalev Kask, Rina Dechter, Javier Larrosa, and Avi Dechter. Unifying cluster-tree decompositions for reasoning in graphical models. *Artificial Intelligence*, 2005. forthcoming.
5. Giuseppe Lancia, Robert Carr, Brian Walenz, and Sorin Istrail. 101 optimal PDB structure alignments: a branch-and-cut algorithm for the maximum contact map overlap problem. In *Proc. of the Fifth Annual International Conferences on Compututational Molecular Biology (RECOMB01)*. ACM Press, 2001.
6. Hans-Peter Lenhof, Knuth Reinert, and Martin Vingron. A polyhedral approach to rna sequence structure alignment. In *Proc. of the Second Annual Interna-*

tional Conferences on Compututational Molecular Biology (RECOMB98), volume 5, pages 517–30. ACM Press, 1998.
7. Burkhard Morgenstern, Nadine Werner, Sonja J. Prohaska, Rasmus Steinkamp, Isabella Schneider, Amarendran R. Subramanian, Peter F. Stadler, and Jan Weyer-Menkhoff. Multiple sequence alignment with user-defined constraints at GOBICS. *Bioinformatics*, 21(7):1271–1273, 2005.
8. Gene Myers, Sanford Selznick, Zheng Zhang, and Webb Miller. Progressive multiple alignment with constraints. In *Proceedings of the first annual international conference on Computational molecular biology (RECOMB 1997)*, pages 220–225, 1997.
9. S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–53, 1970.
10. David Sankoff. Simultaneous solution of the RNA folding, alignment and protosequence problems. *SIAM J. Appl. Math.*, 45(5):810–825, 1985.
11. T.F. Smith and M.S. Waterman. Comparison of biosequences. *Adv. appl. Math.*, 2:482–489, 1981.
12. Roland H. C. Yap. Parametric sequence alignment with constraints. *Constraints*, 6(2/3):157–172, 2001.

Biological constraints for the consensus subsequence problem

Marco Zantoni[‡], Emiliano Dalla[†], Alberto Policriti[‡], and Claudio Schnaider[†]

[‡]Dipartimento di Matematica e Informatica, Università di Udine
[†]Laboratorio Nazionale CIB, Trieste
{zantoni,policriti}@dimi.uniud.it, {dalla,schneide}@lncib.it

Abstract. Motif discovery abstracts many problems encountered during the analysis of biological sequence data, where the sequences can be nucleotide or protein molecules and motifs represent short functionally important patterns. In this work we have in mind a new computational approach to the problem of looking for *Transcription Factor Binding Sites*: the search for genomic motifs responsible for the binding of Transcription Factors to Promoters and other regulative elements, the major event underlying gene expression control. We focus our attention on the problem of, given a set of strings, finding a substring common to the strings in the input set, allowing a *fixed layout* for mismatches in our output.

Introduction

The discovery/identification of short strings occurring approximately in a set of longer strings/sequences is one of the major tasks in today's computational biology. In our particular case we refer to these short strings as ***motifs***. In our initial setting the notion of "approximate occurrence" means that motifs must match a segment of (each) sequence with at most some specified number of mismatches.

Motif discovery abstracts many problems encountered during the analysis of biological sequence data, where the sequences can be nucleotide or protein molecules and motifs represent short functionally important patterns. In this work we have in mind a new computational approach to the problem of looking for *Transcription Factor Binding Sites* (TFBS): the search for genomic motifs responsible for the binding of Transcription Factors to Promoters and other regulative elements, the major event underlying gene expression control.

More specifically, in this paper we focus our attention on the problem of, given a set of strings, finding a substring common to (a significant portion of) the strings in the input set, allowing a *fixed layout* for mismatches in our output. The general strategy used to tackle the problem is based in the introduction of a data structure encoding *à la* Karp-Rabin substrings of the strings in the input set. Based on the observation that layout mismatches can be ordered (according to a suitable notion of ordering on strings), the construction of our data structure can be naturally carried out exploiting a constraint programming scheme.

An important byproduct of this approach—currently under investigation—is the possibility of extending our method to the more general cases (usually intractable) of the *Common (Sub)String Problem*.

We begin our presentation by recalling the *local multiple alignment* problem: an approach leading, under specific assumptions and statistical hypothesis, to a particular solution of our problem. In a more general setting, after finding a set of similar—multiply aligned—substrings of the same length of the strings in our input, we need to compute the *consensus sequence*: the "most representative string", according to some criteria, for the found set. This problem is, in general, NP-hard, therefore some constraint are needed to cut the search space. Motivated by the applications we have in mind and based on a biological background, we reduce the consensus problem to the *fixed-layout consensus problem*. From a technical point of view, we first show how a narrow biological relevant set of substrings can be chosen. Then, we show how the corresponding consensus sequence can be computed and, finally, we show how to enlarge the set towards a consensus-problem-like solution.

1 Basics and Literature

For a string s over Σ, we introduce the following notation:

- $|s|$ denotes the length of s
- $s[i]$ is the *i-th* character of the string s
- $s[i \dots i + l - 1]$ is the substring of ℓ characters starting from $s[i]$
- $s \trianglelefteq t$ denote that s is a substring of t

The Hamming distance between two strings of the same length is the number of symbols that disagree.

1.1 Local multiple alignment

Multiple sequence alignment is one of the well studied problems in computational molecular biology and has many applications. We focus our attention on multiple local alignment of nucleotide sequences, useful for finding conserved motifs, like TFBS, or greater regions putatively corresponding to entire promoters.

The local multiple alignment problem (also known as the *general consensus patterns problem*) consists, given a set of m strings, in locating a substring at fixed length ℓ from each string in the set, so that the *score* determined from the set of substrings is maximized/minimized.

Scoring Schemes. Let $\#_j(a) = |\{t_i : t_i[j] = a\}|$, that is the number of the appearances of letter a in the j-th column of t_i's. Let $f_j(a) = \frac{\#_j(a)}{n}$, the frequency of letter a in the j-th column of t_i's. Let $p(a)$ denote the frequency of letter a in the whole space (e.g. the genome), that is the background probability of a.
We can now define the following scoring schemes:

#LOG#-score: $score(t_1, \ldots, t_n) = \sum_{j=1}^{l} \sum_{a \in \Sigma} \#_j(a) \cdot \log \#_j(a)$;
IC-score: $score(t_1, \ldots, t_n) = \frac{1}{l} \sum_{j=1}^{l} \sum_{a \in \Sigma} f_j(a) \cdot \log \frac{f_j(a)}{p(a)}$;
SP-score (*sum-of-pairs*): $score(t_1, \ldots, t_n) = \sum_{j=1}^{l} \sum_{i<i'} dist(t_i[j], t_{i'}[j])$, where $dist(x, y)$ is the distance between letter x and letter y.

If we consider an arbitrary distance satisfying the triangle inequality, the problem is defined as the minimization problem instead of the maximization problem. Although scoring schemes are closely related, there is a large gap on the approximability.

By using the scoring scheme model hypothesis, we can formulate the problem as follows: given a set $\mathcal{F} = \{s_1, s_2, \ldots, s_m\}$ of strings, and an integer ℓ, find a substring t_i of length ℓ from each s_i, maximizing the score of $(t_1, \ldots, t_n)$, where $(t_1, \ldots, t_n)$ is a local multiple alignment, a local alignment, or simply an alignment. Multiple local alignment is NP-hard under each of these scoring schemes. In addition, multiple local alignment is APX-hard under the average information content scoring [2,3]: it implies that unless P=NP, there is no polynomial time algorithm whose worst case approximation error can be arbitrarily small (precisely, a polynomial time approximation scheme).

Stormo and Hartzell proposed a score based on the *average information content* (IC-score) and developed a (heuristic) iterative algorithm for finding an optimal score (widely used along with variants) [13,14]. By using the average information content scoring scheme, Lawrence and Reilly developed an EM (*expectation maximization*) algorithm [8], while Lawrence et al. developed a *Gibbs sampling* algorithm [7]. However, these algorithms do not guaranteed to find an optimal alignment (i.e. an alignment with the maximum score). Any theoretical guarantee is not given for the scores of the computed alignments.

The result in [1] suggests that the scoring schemes greatly influence the approximability and thus, should be considered as an important factor in approximation algorithms. In spite of the APX-hardness of local multiple alignment under IC-scoring, the results of computational experiments show that the Gibbs sampling algorithm is the best from a practical viewpoint.

1.2 Finding a common substring

A popular technique for finding motifs is to enumeratively test all strings over the sequence alphabet having length equal to the desired motif length. Enumerative algorithms produce all possible motifs for a set of sequences. This allows to evaluate, according to other criteria, the discovered motifs that possess a certain combinatorial property. For this reason, enumerative algorithms can provide input to other algorithms that filter motifs based on other properties.

We assume to use the Hamming distance. Formally, we define the motif enumeration problem as follows:

Definition 1. *The input of the motif enumeration problem is a set of strings, $\mathcal{F} = \{s_1, \ldots, s_m\}$, over an alphabet Σ such that $|s_i| \leq n$, $1 \leq i \leq m$, and integers ℓ and d such that $0 \leq d < \ell \leq n$. The solution is a set of motifs $\mathcal{M}_{\mathcal{F}} \subseteq \Sigma^l$ such*

that for each motif $\mathcal{C} \in \mathcal{M}_{\mathcal{F}}$ and each $s_i \in \mathcal{F}$, there exists a length ℓ substring of s_i, that is Hamming distance $\leq d$ from $\mathcal{C}$.

There are two major computational challenges to enumerating motifs. The first challenge is that the problem of deciding if $\mathcal{M}_{\mathcal{F}} = \emptyset$ is NP-hard; practical solutions are thus non-trivial. The second is that we are concerned with more than simply the decision, so we may have to produce output of exponential size.

This most naive form of search introduces a factor of $\Omega(|\Sigma|^{\ell})$ into the time complexity. The benefit of this type of enumeration is that it requires space bounded by a linear function of the size of the input. New ground was broken when Sagot [10] introduced a different approach that enumerates only those strings that are potential motifs, letting information from the sequences guide the enumeration. This more intelligent search remains within the (ℓ, d)-neighborhood of each sequence.

Definition 2 (neighborhood). *For a string $s \in \Sigma^n$, with $n \geq \ell$, the (ℓ, d)-neighborhood of s is the set*

$$\{t | t \in \Sigma^{\ell} \wedge d_H(s', t) \leq d \text{ for some substring } s' \text{ of } s \text{ with } |s'| = \ell\}.$$

For any string s, we use $N_{\ell,d}(s)$ to denote the (ℓ, d)-neighborhood of s. For a family $\mathcal{F}$ of strings, the (ℓ, d)-neighborhood of $\mathcal{F}$ is the set

$$\{t | t \in \Sigma^{\ell} \wedge \forall s \in \mathcal{F}, t \in N_{\ell,d}(s)\} = \bigcup_{s \in \mathcal{F}} N_{\ell,d}(s),$$

and is denoted $N_{\ell,d}(\mathcal{F})$.

We also define the value $N = \sum_{i=0}^{d} \binom{\ell}{i} (|\Sigma| - 1)^i$. The significance of N is that for a string s' with $|s'| = \ell$, $N = |N_{\ell,d}(s')|$.

The method of Sagot has a time complexity of $\mathcal{O}(\ell m^2 n N)$, a space complexity of $\mathcal{O}(\ell m^2 n)$. Furthermore, the algorithm in [10] is designed with a "quorum" parameter, so that a motif is only required to be common to some $q \leq m$ of the sequences.

1.3 Generating a consensus

Finding a consensus sequence representing the best approximation of all the similar results that have been obtained is crucial, in order to recover useful information from the examined (biological) sequences. The problem can be formulated as follows: find a substring or a small similar subsequence that is common to many of the strings in the set. We consider the Hamming distance d_H to define the concept of "similarity" among substrings.

The problems we are interested in include the *closest string* (CStrP) and the *closest substring* (CSStrP), with or without a threshold.

Given a set $\mathcal{F}$ of m strings, $\mathcal{F} = \{s_1, \ldots, s_m\}$, each of length $|s_i| = n$,
CStrP: find a string p that minimize the maximum distance (denote by d) between p and s, for all $s \in \mathcal{F}$.

Formally, $d = \max_{s \in \mathcal{F}} d_H(p, s) = \min_{p' \in \Sigma^n} \max_{s \in \mathcal{F}} d_H(p', s)$. We have to find the minimum d such that $N_{n,d}(\mathcal{F}) \neq \varnothing$.
CStrP(d): given an integer d, find a string p such that its maximum distance from a string s of $\mathcal{F}$ is d or less. That is to say, if $N_{n,d}(\mathcal{F}) \neq \varnothing$, we need to find $p \in N_{n,d}(\mathcal{F})$.
CSStrP(ℓ): given an integer ℓ, find a string p of length ℓ, and m substrings $s'_i \rhd s_i$ of length ℓ, such that the maximum distance between p and s'_i is the minimum among all possible substrings of $\mathcal{F}$. Formally, $d = \max_{s'_i} d_H(p, s'_i) = \min_{p' \in \Sigma^l, s'_i \rhd s_i, |s'_i| = \ell} \max_{s'_i} d_H(p', s'_i)$.
CSStrP(ℓ,d): given two integers ℓ and d, find a string p of length ℓ, and m substrings $s'_i \rhd s_i$, such that the minimum distance between p and s'_i is d or less.

Initially, guided by the needs of genomic research, statistical approaches were used to give solutions for the CStrP. The problem had been previously studied because of its connection with the area of coding theory, where it was proved to be NP-hard [5]. The CSStrP models the more general situation where the strings that must be compared do not have the same length, and one wants to find just parts of the string that are similar. For example, the usefulness of this approach in genomic research can be seen when performing cross-species sequences comparisons, where the dataset is made of orthologous regions that can share some common features while having different length due to differences in the genomic structure. The CSStrP, being a generalization of the CStrP, can be easily shown to be NP-hard. In terms of parameterized complexity, the main results for the CSStrP is that it cannot be solved in polynomial time, even when the distance parameter is fixed [4].

2 Our proposal

We begin this section recalling a classic in string manipulation which turns out to be the basic ingredient of our approach: the Karp-Rabin encoding. The idea is presented by a short review of the original algorithm and is completed with the relative complexity results.

2.1 Karp - Rabin algorithm

The algorithm proposed by Karp and Rabin in [6], solves the pattern discovery problem on the exact string matching background. This algorithm assumes that we can efficiently shift a vector of bits and that we can efficiently perform arithmetical operations on integers. To take advantage of this assumptions, we can see a string like an integer, mapping each character of Σ in a digit using a function f_m. For example, in the DNA context, $\Sigma = \{A, C, G, T\}$ can be mapped into $\Sigma' = \{0, 1, 2, 3\}$.

Definition 3. *For a text string T, let T_r denote the $\ell-$length substring of T starting at character r. We can now define the following function:*

$$H(T_1) = \sum_{i=1}^{i=\ell} |\Sigma|^{\ell-i} \cdot f_m(T[i]) \tag{1}$$

$$H(T_r) = |\Sigma| \cdot (H(T_{r-1}) - |\Sigma|^{\ell-1} \cdot f_m(T[r-1]) + f_m(T[r+\ell-1]) \tag{2}$$

The following theorem holds.

Theorem 1. *There is an occurrence of a pattern P starting at position r of T if and only if $H(P) = H(T_r)$.*

In [6], Karp and Rabin introduced a method called the *randomized fingerprint* method, that preserves the spirit of the above numerical approach, but allows to deal with larger numbers in an extremely efficient way. It is a *randomized* method because introduces a probability of error, but the probability that a false match occurs can be bounded as stated in the following theorem.

Theorem 2. *The Monte Carlo algorithm for pattern matching requires $\mathcal{O}(n + m)$ time and has a probability of error $\mathcal{O}(1/n)$.*

In this work, we use only the first part of the algorithm idea, because, working with approximate string matching, it is not simple to introduce an efficiently hashing funcion.

2.2 Formalization of the problem

Input: $\mathcal{F}$, ℓ, d and q, where $\mathcal{F} = \{s_1, \ldots, s_m\}$ is the set of strings (not necessarily of the same length), d is the number of errors allowed in comparisons and q is a parameter denoting the minimum size of the set of $\mathcal{F}$-elements containing a common substring of length ℓ with d errors. Without loss of generality, we consider input strings of the same length n.

Definition 4. *A solution for the* (l, d, q)-consensus problem *over $\mathcal{T}$ is S if and only if*

- *there exists a pattern p (*consensus*), $|p| = l$, s.t. $S \subseteq \{s \in \Sigma^l | d_H(s, p) \leq d\}$*
- *for all $s \in S$, there exists $s_i \in \mathcal{F}$ such that $s \trianglelefteq s_i$*
- *$|\{i | \exists s \in S, s \trianglelefteq s_i\}| \geq q$*

When trying to apply this approach to biological problems like the identification of known and putative TFBS, some considerations must be done in order to simplify the analysis. First of all, if a blind search has to be done and no previous knowledge is going to be used, it is useful to consider that, in general, Transcription Factors that regulate the expression of a group of genes involved in a given biological process tend to bind these genes' promoters in the same region, so the dimension of the sequences taken into account can be greatly reduced [12]. The

second useful assumption is that, given a motif (TFBS), only some of its characters (nucleotides) are important for the binding of the Transcription Factor; this feature, that is considered also by statistical approaches when creating the weighted matrices, allows to co-cluster different results, apparently corresponding to different motifs, simplifying and reducing the number of consensus TFBS generated by the algorithm. Thus, we propose a approximated algorithm based on the concept of localized nucleotide mutation: a protein can "accept" one or more mutations in a binding site, but always in the same positions.

Definition 5. *A solution for the* fixed-layout (ℓ, d, q)-consensus problem *over* $\mathcal{F}$ *is* S_{fl} *if and only if*

- $S_{fl} \subseteq \Sigma^{\ell}$
- *for all* $s' \in S_{fl}$*, there exists* $s_i \in \mathcal{F}$ *such that* $s' \trianglelefteq s_i$
- $|\{i | \exists s' \in S_{fl}, s' \trianglelefteq s_i\}| \geq q$
- *there exists a set of indexes* $fl = \{i_1, \ldots, i_d\}$, $1 \leq i_1 < \cdots < i_d \leq \ell$, *such that for all* s'_i *and* s'_j *in* S_{fl}, $s'_i[k] \neq s'_j[k] \Rightarrow k \in fl$.

Output: all the positions of the common subsequences of length ℓ with d errors founded in at least q sequences, with the addition of the constraint that the errors, if occur, are in the same position (called layout).

The *fixed-layout problem* is a reduction of the *consensus problem*, with the aim to find some useful biological information.

2.3 ScanPro

Due to the limited amount of information available about Transcription Factors-DNA interactions it is quite difficult to formalize specifications allowing to filter good from false positives results.

This algorithm aims to solve the *fixed-layout* (ℓ, d, q)*-consensus problem*, and then extend the results to obtain an approximate solution for the (ℓ, d, q)-*consensus problem* solution. The difference with solving immediately the former problem is in the generation of the consensus sequence and in the final complexity. Furthermore, thanks to the fixed-layout approach, it becomes possible to propose algorithms that exploit new biological constraints on protein/DNA 3D binding [11].

The most interesting goal is to find the best constraints that allow to compute the consensus sequence for a set of strings. It is well known that statistical approaches and hidden Markov models are based on *a priori* knowledge, that can help to find common subsequences with a certain structure (or biological function), but become unproductive when looking for new patterns.

Encoding. Given d, the maximum number of errors allowed in a string of ℓ characters, a generic error layout $fl = \{i_1, \ldots, i_d\}$, such that $1 \leq i_1 < \cdots < i_d \leq \ell$, is the set of the d positions where an error may occur during a comparison

between strings. Without loss of generality, we suppose that i_1 is strictly greater than 1, that is that an error can not be in first position; in this way there are $tfl = \binom{\ell-1}{d}$ error layouts called $FL = \{fl_1, \ldots, fl_{tfl}\}$.

We can divide error layouts into two classes: *basic* and *shifted*. The *basic layouts*, bfl, are characterized by having $i_d = \ell$ and are $\binom{\ell-2}{d-1}$ overall. *Shifted layouts* relating to an established basic layout bfl_x are denoted by $sfl_{x,j}$ and look like $\{i_1 - j, \ldots, i_d - j\}$, with $i_1 - j \geq 2$.

Fixed an error layout, the function $fl\_code : (\Sigma^\ell \times FL \rightarrow \mathbb{N})$ gives the encoding of a string of $\ell - d$ characters. For each string $s \in \mathcal{F}$, with $|s| = n$, and each basic layout $bfl_x = \{i_1, \ldots, i_d\}$, we have to encode all possible $n - \ell + 1$ substrigs of length ℓ in s, thus perform $(n - \ell + 1)(2\ell - 1)$ operations.

Considering a shifted layout $sfl_{x,j} = \{i_1 - j, \ldots, i_d - j\}$, with $j = 1, \ldots, i_1 - 2$, we can obtain the encoding of the substring $s[i, \ldots, i + \ell - 1]$ for the given layout by taking the encoding of $s[i - 1, \ldots, i + \ell - 2]$ for the error layout $sfl_{x,j-1}$, performing a left shift and adding the value relative to $s[i + \ell - 1]$, making only $2(n - \ell + 1)$ operations.

Since $|\Sigma| = 4$, we can map the alphabet on $\mathbb{Z}_4$ using only 2 bits for each character. Hence, an ℓ-characters-long string is mapped into a 2ℓ bits integer number, so any shift involves only 2 bits.

The function fl_code returns a value used as index of an array: in each position c of this array there is a pointer to a matrix M_c of dimension $tfl \times (m+1)$, with $m = |\mathcal{F}|$, whose elements are lists of positions from column 1 to m and in column $m + 1$ there is the "rank", that is the number of different strings $s \in \mathcal{F}$ in which we can find that specific motif. $M_k(i, j) \neq NULL$ if in s_j there exists a substring w such that $|w| = \ell$ and $fl\_code(w, fl_i) = k$. A rank is incremented when an element is put in a empty list in the relative row.

At the end, if this value is greater than the quorum, the motif is a solution for the *fixed-layout problem* and we have all positions of the occurrences. Using this occurrences, we are able to generate a consensus c_k, according to the majority string model. Now, thanks to the previous data structure, we have in constant time all positions of the substrings s', such that $d_H(c_k, s') \leq d$.

Definition 6. *Given a set S of r strings, $S = \{s_1, \ldots, s_r\}$, each of length $|s_i| = l$, we indicate by $\mathcal{N}(a, j, S)$, $a \in \Sigma$, $j \leq l$, the number of characters of type a in the j-th position of strings s_i of S.*

*A **majority string** for S is a string w, where for each $j \leq l$, $w[j]$ is one character with maximum frequency, i.e. $\mathcal{N}(w[j], j, S) = \max_{a \in \Sigma} \mathcal{N}(a, j, S)$.*

To obtain a most biological informative consensus, during its creation we use an extended alphabet $\Sigma' = \{A, C, G, T, R, Y, N\}$, where $R = A$ or G (purine), $Y = C$ or T (pyrimidine), and N is any character, in according to the IUPAC alphabet.

2.4 Constraints

Let us consider the formalization of the CSStrP: let $x_1, \ldots, x_\ell$ be variables over $|\Sigma|$, then $x_1 \ldots x_l$ is a string over the domain Σ^ℓ (enumeration of all motifs).

Let $s_{i,j}$ be the character j of the string i, and let $B_{i,j}$ be boolean variables, with $i = 1, \ldots, m$ and $j = 1, \ldots, n$. For all $x_1 \ldots x_l \in \Sigma^\ell$, we introduce the reified constraint $(x_k = s_{i,j+k-1}) \Leftrightarrow B_{i,j+k-1}$, with $1 \leq k \leq \ell$, $1 \leq i \leq m$, $1 \leq j \leq n - l + 1$, and impose that $B_{i,j} + \cdots + B_{i,j+\ell-1} \geq \ell - d$.

In the fixed-layout problem, we impose different constraints. Let $B_1, \ldots, B_\ell$ be boolean variables that fix the layout, thus $B_k \Rightarrow (x_k = s_{i,j+k-1})$, with $1 \leq k \leq \ell$, $1 \leq i \leq m$, $1 \leq j \leq n - l + 1$.

An improvement under investigation, concern the layout. Instead of fix it at the beginning, it would be use more efficient generate the best layout during the computation.

2.5 Further work

Future improvements, under analysis, concern the following topics.

1. Different computations for the consensus sequence, according to other distances or allowing an external consensus.
2. Introduction of a hashing function to obtain a randomized version of our algorithm.
3. Exploit the features of algorithm and data structure to change the number of errors allowing the "entry" of new errors.

3 Conclusions

Predicting promoter elements or extracting their consensus sequence are important steps towards the global comprehension of the mechanisms undergoing genes co-regulation. In order to achieve this goal, it is necessary to take into consideration aspects like the correct combination and the precise spatial organization of the regulatory sites, like demonstrated in recent papers [9,15]. The order and relative distances between the binding sites, thus, can no longer be considered negligible constraints, and whatever the method used to extract a consensus sequence, the prediction of precise promoters structure cannot be considered completed unless more biological knowledge is used during the prediction. It is our belief that, when working on human genomic data, no existing method for either promoter location prediction or promoter consensus extraction is very trustworthy at doing its job. Particularly troublesome to all approaches, whatever their aim, is the presence of too much noise, whether it comes from stretches of repeated sequences with no biological function or from the presence in the data set of more than one family of independent promoter motifs. The algorithms for extracting conserved sets of single or combined words in a sequence are growing increasingly more sophisticated and efficient, especially when using combinatorial approaches. Unfortunately, it is in the statistical evaluation of the motifs found that problems seem to persist, especially when errors are allowed and motifs may be composed of more than one element with adjacent parts standing at particular distances from one another. Purely combinatorial

approaches (also with a posteriori statistical evaluation of the quality of the results), such as Scanpro, allow a much more controlled, and sometimes precisely defined, analysis of the data, but often require a deep understanding of the algorithms as well as extended practice in its use.

References

1. T. Akutsu, H. Arimura, and S. Shimozono. On approximation algorithms for local multiple alignment. *RECOMB*, 2000.
2. S. Arora, C-Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation algorithms. *Proc. 33rd IEEE Symp. Foundations of Computer Science*, pages 14–23, 1992.
3. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, S.M. Spaccamela, and M.Protasi. *Complexity and Approximation. Combinatorial Optimization Problems and their Approximability Properties*. Springer-Verlag, 1999.
4. M.R. Fellows, J. Gramm, and R. Niedermeier. On the parameterized intractability of closest substring and related problems. *Lecture Notes in Computer Science*, pages 262–273, 2002.
5. M. Frances and A. Litman. On covering problems of codes. *Theor. Comput. Syst.*, 30:113–119, 1997.
6. R. M. Karp and M. O. Rabin. Efficient randomized patter-matching algorithm. *IBM J. Res. Dev.*, 31(2):249–260, 1987.
7. C. E. Lawrence, S.F. Altschul, and M.S. Boguski et al. Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science*, 262:208–214, October 1993.
8. C. E. Lawrence and A.A. Reilly. An expectation maximization (EM) algorithm for identification and characterization of common sites in unaligned biopolymer sequences. *PROTEINS: Structure, Function, and Genetics*, 7:41–51, 1990.
9. V.J. Makeev, A.P. Lifanov, A.G. Nazina, and D.A. Papatsenko. Distance preferences in the arrangement of binding motifs and hierarchical levels in organization of transcription regulatory information. *Nucleic Acids Research*, 31(20):6016–6026., October 2003.
10. M. F. Sagot. Spelling approximate repeated or common motifs using a suffix tree. *Lecture Notes in Computer Science*, 1380:111–127, 1998.
11. M.-F. Sagot, A. Viari, J. Pothier, and H. Soldano. Finding flexible patterns in a text - An application to 3D matching. *Computer Applications to the Biosciences*, 11:59–70, 1995.
12. E .Segal, N. Friedman, N. Kaminski, A. Regev, and D. Koller D. From signatures to models: understanding cancer using microarrays. *Nature Genetics*, 37:Suppl:S38–45. Review, June 2005.
13. G.D. Stormo. Consensus patterns in DNA. *Methods in Enzymology*, 183:211–221, 1990.
14. G.D. Stormo and G.W. Hartzell. Identifying protein-bindings sites from unaligned DNA fragments. *Nucleic Acids Research*, 86:1183–1187, 1989.
15. G. Terai and T. Takagi. Predicting rules on organization of cis-regulatory elements, taking the order of elements into account. *Bioinformatics*, 20(7):1119–1128, May 2004.

Author Index