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]).

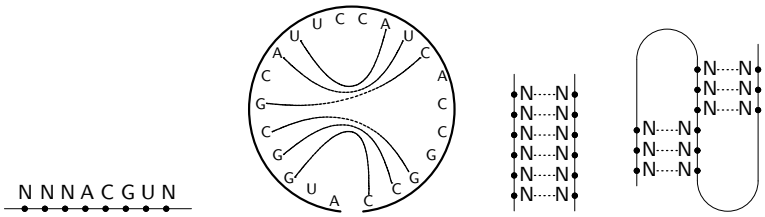
Another approach defines a signature as a set of interrelated motifs. Occurrences of the signature are sought using simple pattern-matching techniques and exhaustive tree search. Such programs include RnaMot (cf. [5]), RnaBob (cf. [5]), PatScan (cf. [3]), Palingol (cf. [1]) and RnaMotif (cf. [9]). Although 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).

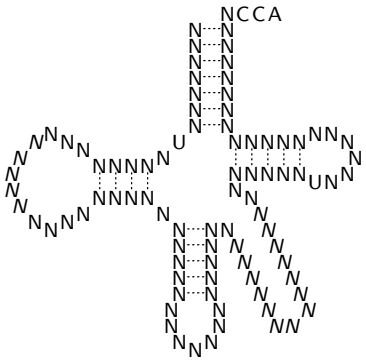
2 RNA motif problem

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*.

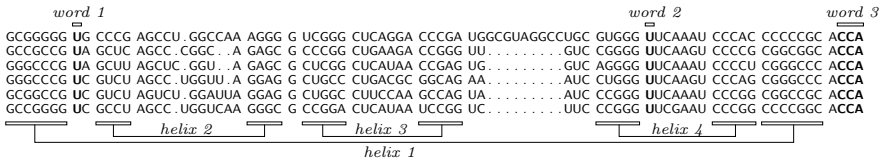
Example 1. Consider the sequence in Fig. 2(a). We want to find the motif that begins with ACGU, then has a spacer of three nucleotides and an helix of length



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

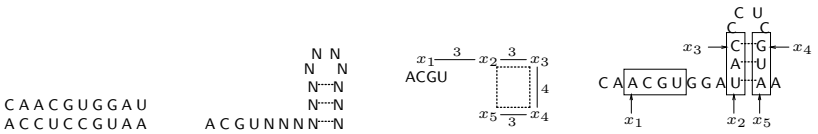


(e)



(f)

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:

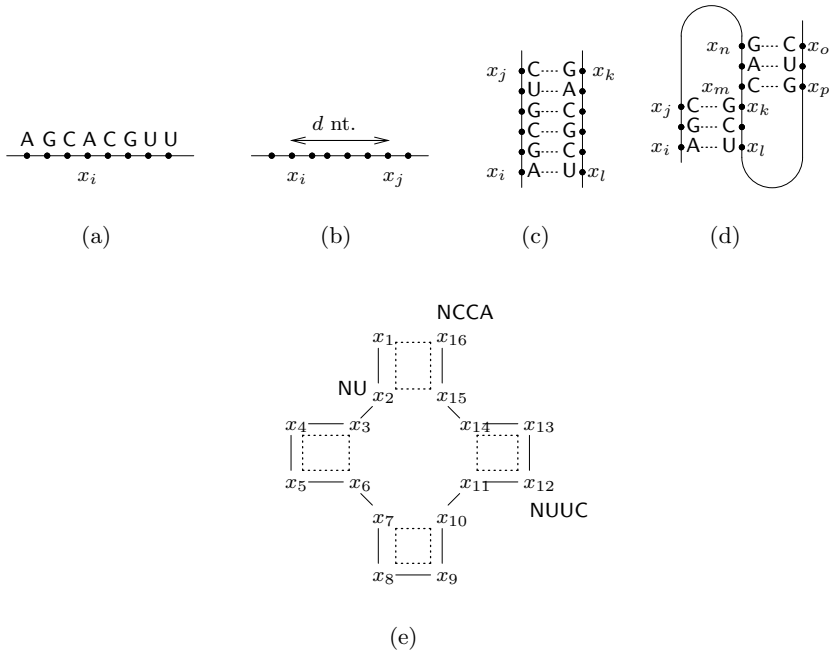
- The constraint *string(*word, x_i*)* (cf. Fig. 3(a)) takes a possibly ambiguous word (i.e. containing letters like N) and compares it to the subsequence that begins at the index x_i. The algorithm uses dynamic programming and gives affine cost to the gaps.
- The constraint *helix(*x_i, x_j, x_k, x_l*)* (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*]

foreach $x_i \in \mathcal{X}$ **do** SetBNC*(x_i) ;
$Q \leftarrow \mathcal{X}$; $c_\varnothing\_raised \leftarrow$ false ;
while $(Q \neq \varnothing)$ **do**
  $x_j \leftarrow Q.pop()$;
  foreach $x_i \in N(x_j)$ **do**
    SetBSupport(x_i, x_j) ; SetBSupport(x_j, x_i) ;
    if *(SetBNC*(x_i))* **then** $Q \leftarrow Q \cup \{x_i\}$;
  if *(SetBNC*(x_j))* **then** $Q \leftarrow Q \cup \{x_j\}$;
  if $(c_\varnothing\_raised)$ **then**
    $c_\varnothing\_raised \leftarrow$ false ;
    foreach $x_i \in \mathcal{X}$ **do**
      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.