A new local consistency for weighted CSP dedicated to long domains

ABSTRACT

The weighted constraint satisfaction problem (WCSP) is a soft constraint framework with a wide range of applications. Most current complete solvers can be described as a depth-first branch and bound search that maintain some form of local consistency during the search. However, the known consistencies are unable to solve problems with huge domains because of their time and space complexities. In this paper, we adapt the 2B-consistency, a weaker form of arc consistency well-known in classic CSPs, into the *bound arc consistency* and we provide several algorithms to enforce it.

General Terms

Algorithms

Categories and Subject Descriptors

F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Logic and constraint programming*; G.1.6 [**Numerical Analysis**]: Optimization—*Constrained optimization*

Keywords

Weighted Constraint Satisfaction Problem, Local Consistency, Bound Arc Consistency

1. INTRODUCTION

The weighted constraint satisfaction problem (WCSP) is a well-known extension of the CSP framework with many practical applications. Recently, several generalizations of the CSP's arc consistency (AC) have been proposed for soft constraints, like AC* in [6]. Unfortunately, compared to the classic AC, the time complexity always increases by a factor of d (the size of the largest domain), and the memory space is at least proportional to d. This makes these consistencies useless for problems with long domains like RNA detection or temporal constraints with preferences. We present here an extension of the 2B-consistency, first described for classic CSPs in [9], called *bound arc consistency*. Its time and space complexities are better than the complexities of AC* by an order of d.

Bound arc consistency (BAC*) is based on a interval representation of the sets of values and it can treat efficiently "simple" constraints, such as precedence: $f(v_1, v_2) = v_2 - v_1 - d$ if $v_2 - v_1 - d > 0, 0$ otherwise, that often show up in problems with long domains (like scheduling). We also propose an extension of this consistency that takes into account the semantics of the function, like monotonicity or convexity and we define *$\varnothing$-inverse consistency* that can boost the cost propagation on some conditions.

Finally, we compare BAC* with AC* on the problem of non-coding RNA detection and show the superiority of our consistency for this kind of problems.

SAC'06 April 23-27, 2006, Dijon, France

2. PRELIMINARIES

Valuation structures are algebraic objects that specify costs [13]. For WCSP [8], 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*, k can possibly be ∞; $\oplus$, the *addition* on E, is 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 binary 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, $\mathcal{C} = \{c_1, \ldots, c_e\}$ is the set of e constraints. A constraint $c \in \mathcal{C}$ can be either a unary constraint: $c : D(x_i) \rightarrow E$ (we call it c_i), or a binary constraint: $c : D(x_i) \times D(x_j) \rightarrow E$ (we call it c_{ij}).

We will restrict ourselves to *binary* WCSP, where no constraint has an arity greater than 2. Results can easily be extended to higher arity constraints. Furthermore, 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 constraints are defined, we can always define *dummy* ones: c_i is the null function over $D(x_i)$, $c_\varnothing = 0$).

Given a pair (v_i, w_j) (resp. a value v_i), $c_{ij}(v_i, w_j) = k$ (resp. $c_i(v_i) = k$) means that the constraint forbids the corresponding assignment. Another cost means the pair (resp. the value) is permitted by the constraint with the corresponding cost. The cost of an assignment $t = (v_1, \ldots, v_n)$, noted $\mathcal{V}(t)$, is the sum over all the cost functions: $\mathcal{V}(t) = \bigoplus_{i,j} c_{ij}(v_i, v_j) \oplus \bigoplus_i c_i(v_i) \oplus c_\varnothing$

An assignment t is *consistent* if $\mathcal{V}(t) < k$. The usual task

of interest is to find a consistent assignment with minimum cost. This is a NP-hard problem. Observe that, if $k = 1$, a WCSP reduces to classic CSP.

3. SOME LOCAL PROPERTIES

3.1 Existing local consistencies

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 the *node consistency* (NC*, cf. [6]).

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 $\mathcal{O}(nd)$. Another famous stronger local consistency is the *arc consistency* (AC*, cf. [12, 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. More formally: $\forall x_i \in \mathcal{X}, N(x_i) = \{x_j \in \mathcal{X} : c_{ij} \in \mathcal{C}\}$. 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.

On a binary WCSP, arc consistency can be enforced in time $\mathcal{O}(n^2d^3)$ and in space $\mathcal{O}(ed)$. The algorithm uses the operations ProjectUnary and Project described in Alg. 1 to enforce the supports of the values and the unary supports respectively. In practice, to reach the $\mathcal{O}(ed)$ space complexity, the algorithm uses extra costs differences data structures as suggested in [1].

3.2 Bound arc consistency

We present here a consistency which is weaker than AC*. It can be enforced with lower time and space complexities and it is called *bound arc consistency* (BAC*). To apply bound arc consistency, we need to change the definition of a WCSP: the domains are now *intervals* $\mathcal{I}$. The intervals initially range over all the possible values. We shall suppose that all the values of the variables are sorted by an arbitrary order and $\forall x_i \in \mathcal{X}, lb_i = \min\{D(x_i)\}$, $ub_i = \max\{D(x_i)\}$ (lb_i is the *lower bound* of the interval of x_i and ub_i is its *upper bound*).

Definition 3. 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.

Changing the representation of the set of the values to intervals alters the expressivity of the framework: it is not possible to describe that a value which is inside an interval has been deleted. But this allows us to decrease the space complexity as a domain is now represented by only two values. The Alg. 1 provides an algorithm to enforce this consistency.

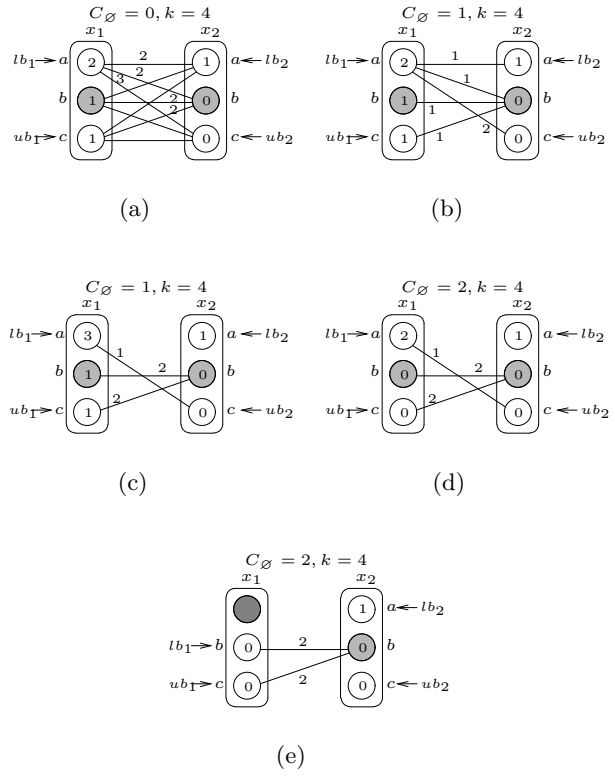


Figure 1: Steps to enforce BAC* with $\varnothing$IC

Example 1. Fig. 1(b) represents an instance of a small problem. It contains two variables (x_1 and x_2) with three possible values for each one (a, b and c), a unary constraint for each variable (the costs are written in the vertices) and a binary constraint (the costs are written on the edge that connects a pair of values; if there is no edge between two values, the cost is 0). k is arbitrarily set to 4 and $c_\varnothing$ is set to 0. The values are supposed to be sorted by the lexicographic order ($a \prec b \prec c$), thus $lb_1 = a$ and $ub_1 = c$ for x_1 and x_2. First, we notice that the value a of x_1 has no support. After a call of Project(x_1, a), we get Fig. 1(c). Then, we project a cost of 1 to the $c_\varnothing$ because x_1 has no unary support (cf. Fig. 1(d)). Finally, as $c_\varnothing \oplus c_i(lb_1)$ is equal to k, $x_1 = a$ is discarded and the lower bound of x_1 is updated to lb_1 (cf. Fig. 1(e)). This instance is BAC* but not AC* because $x_2 = b$ has no support. This proves that BAC* can be strictly weaker than AC*.

Theorem 1. *Algorithm 1 enforces BAC* in time* $\mathcal{O}(ed^2 + \min\{k, nd\} \times nd)$ *and in space* $\mathcal{O}(n + e)$ *(proof omitted).*

3.3 Strengthening BAC*

We may want to enforce a stronger local consistency that takes into account the constraint costs involving values *inside* the intervals. To keep a reasonable space complexity, this cost will be projected directly to $c_\varnothing$. Thus we add to the BAC* property the *$\varnothing$-inverse consistency* ($\varnothing$IC):

Definition 4. The constraint c_{ij} is $\varnothing$-inverse consistent if $\exists (v_i, w_j) \in D(x_i) \times D(x_j), c_{ij}(v_i, w_j) = 0$ (this pair (v_i, w_j)

Algorithm 1: Algorithm enforcing BAC*

Procedure *SetBAC*()*
  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*
  changed $\leftarrow$ false ;
  while $(bi_i \leq bs_i) \wedge (c_\varnothing \oplus c_i(bi_i) \geq k)$ **do**
    $bi_i \leftarrow bi_i + 1$; *changed* $\leftarrow$ true ;
  while $(bi_i \leq bs_i) \wedge (c_\varnothing \oplus c_i(bs_i) \geq k)$ **do**
    $bs_i \leftarrow bs_i - 1$; *changed* $\leftarrow$ true ;
  ProjectUnary(x_i) ;
  return *changed* ;

Procedure *SetBSupport(x_i, x_j)*
  Project(x_i, bi_i, x_j) ; Project(x_i, bs_i, x_j) ;

Procedure *ProjectUnary(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)*
  $min \leftarrow \min_{w_j \in I(x_j)}\{c_{ij}(v_i, w_j)\}$;
  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 *SetBSupport2(x_i, x_j)*
  ProjectBinary(x_i, x_j) ;
  Project(x_i, bi_i, x_j) ; Project(x_i, bs_i, x_j) ;

Procedure *ProjectBinary(x_i, x_j)*
  $min \leftarrow \min_{\substack{v_i \in I(x_i) \\ w_j \in I(x_j)}}\{c_{ij}(v_i, w_j)\}$;
  if $(min = 0)$ **then return** ;
  $c_\varnothing$_*raised* $\leftarrow$ true ;
  foreach $v_i \in I(x_i)$ **do**
    foreach $w_j \in I(x_j)$ **do**
      $c_{ij}(v_i, w_j) \leftarrow c_{ij}(v_i, w_j) \ominus min$;
  $c_\varnothing \leftarrow c_\varnothing \oplus min$;
  if $(c_\varnothing \geq k)$ **then raise** *exception* ;

is called the binary support of $c_\varnothing$). A WCSP is $\varnothing$-inverse consistent if every constraint is $\varnothing$-inverse consistent.

Remark that $\varnothing$IC is a generalization to a higher arity of the second requirement of the NC* property.

When BAC* finds a support w_j for lb_i w.r.t. c_{ij}, it projects the cost $c_{ij}(lb_i, w_j)$ to the unary constraint c_i. The constraint is now $\varnothing$IC (the binary support is (lb_i, w_j)), but this property is more relevant when enforced first: it directly increases the $c_\varnothing$. To enforce this local consistency, the procedure SetBSupport should be replaced by SetBSupport2.

Example 2. Let us resume with the problem on FIG. 1(a). If no cost is mentionned on an edge, it is by default 1. We can see on this instance that for any value of x_1 and for any value of x_2, the binary constraint yields to a cost not less than 1. In this case, BAC* would project some binary costs to the bounds but $\varnothing$IC directly projects all of this costs to $c_\varnothing$ (cf. FIG. 1(b)); this guarantees an increase of the lower bound.

THEOREM 2. *The algorithm enforcing BAC* with $\varnothing$IC takes time $\mathcal{O}(ed^3 + \min\{k, nd\} \times nd)$ and space $\mathcal{O}(n + e)$ (proof omitted).*

It could be possible to decrease the time complexity in d by using an appropriate structure that contains the sorted costs of a constraint. But this would increase the space complexity by a factor at least of d^2, which is unacceptable. Another possibility to have a faster algorithm is to use the semantics of the constraints to find the minimum of the function in less than $\mathcal{O}(d^2)$ time, when possible, to decrease the complexity. We need a definition to describe easily the cost propagation:

THEOREM 3. *If the minimum of the binary cost functions can be found in $\mathcal{O}(d)$ time, the complexity of BAC* with $\varnothing$IC becomes $\mathcal{O}(ed^2 + \min\{k, nd\} \times nd)$ with no memory space increase. If the minimum of unary and binary cost functions can be found in constant time, the complexity of BAC* with $\varnothing$IC becomes $\mathcal{O}(ed + \min\{k, nd\} \times n)$ with no memory space increase (proofs omitted).*

These results are particularly interesting for semi-convex functions (well-known in temporal constraints with preferences) because the minima can be found in constant time:

Definition 5. A function c_i (resp. c_{ij}) is *semi-convex* [5] iff: $\forall e \in E$, the set $\{v_i \in D(x_i) : c_i(v_i) > e\}$ (resp. $\{(v_i, w_j) \in D(x_i) \times D(x_j) : c_{ij}(v_i, w_j) > e\}$) is an interval. A function $x, y \mapsto f(x, y)$ is semi-convex w.r.t. a single variable y iff $\forall x, y \mapsto f(x, y)$ is semi-convex.

Informally speaking, semi-convex functions have only one peak. The unary semi-convex functions encompass monotonic functions and anti-functional constraints [4]. An example of semi-convex function w.r.t. a single variable is $x, y \mapsto x^2 - y^2$. It is semi-convex w.r.t. x but not to y.

If the costs functions are semi-convex w.r.t. *every variable*, like $x, y \mapsto x + y$, the minima can be found in constant time because they are located in the corner of the cost matrices. If the costs functions are semi-convex w.r.t. *a single variable*, the minimum cost is reached by a value on the edge of the cost matrix and so can be found in $\mathcal{O}(d)$ time.

4. COMPARISON WITH RELATED WORKS

4.1 2B-consistency

The definition of 2B-consistency, as defined in [9] for numeric non-binary CSP (NCSP) is:

Definition 6. $x \in \mathcal{X}$ is 2B-consistent if $\forall c : D(x) \times D(x_1) \times \ldots \times D(x_r) \in \mathcal{C}$ if: $\exists (v_1, \ldots v_r) \in D(x_1) \times \ldots \times D(x_r)$, $c(lb, v_1, \ldots, v_r)$ and $\exists (v_1, \ldots v_r) \in D(x_1) \times \ldots \times D(x_r)$, $c(ub, v_1, \ldots, v_r)$. A NCSP is 2B-consistent iff every variable is 2B-consistent.

$$
\begin{array}{c} \\ (x_2)\end{array}\!\!\begin{array}{c}\begin{array}{cccc} a & b & c & d \end{array}\\ \begin{bmatrix} 1 & 0 & 2 & 1 \\ 1 & 0 & 2 & 1 \\ 1 & 0 & 2 & 1 \end{bmatrix}\end{array}
\qquad
\begin{array}{c} \\ (x_3)\end{array}\!\!\begin{array}{c}\begin{array}{cccc} a & b & c & d \end{array}\\ \begin{bmatrix} 1 & 2 & 0 & 1 \\ 1 & 2 & 0 & 1 \\ 1 & 2 & 0 & 1 \end{bmatrix}\end{array}
$$

Figure 2: Two cost matrices

Obviously, a WCSP such that $k = 1$ which is BAC* is 2B-consistent.

Besides, it is possible to express a WCSP in classic CSP by *reifying* the costs [10].

Definition 7. Consider the WCSP $\mathcal{P} = \langle \mathcal{S}, \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ Let $\mathcal{P}' = \langle \mathcal{X}', \mathcal{D}', \mathcal{C}' \rangle$ be the classic CSP such that:

- the set $\mathcal{X}'$ of variables is $\mathcal{X}$ augmented with a cost variable x_E per constraint: x_E^{ij} for the binary constraint c_{ij}, x_E^i for the unary constraint c_i;
- the domain of x is $D(x)$ if x is in $\mathcal{X}$, E if x is a cost variable x_E; the set of the domains is $\mathcal{D}'$;
- the set $\mathcal{C}'$ of constraints contains: the reified constraints c'_{ij} defined by the set of tuples $\{(v_i, w_j, e) : v_i \in D(x_i), w_j \in D(x_j), e = c_{ij}(v_i, w_j)\}$; the reified constraints c'_i defined by the set of tuples $\{(v_i, e) : v_i \in D(x_i), e = c_i(v_i)\}$; an extra constraint c'_E that applies on the cost variables x_E: $\sum_{c_{ij} \in \mathcal{C}} x_E^{ij} + \sum_{c_i \in \mathcal{C}} x_E^i < k$.

The problem $\mathcal{P}'$ has a solution iff $\mathcal{P}$ has a solution. The aim of enforcing a property is usually to find inconsistencies as soon as possible. This leads to a definition of the *strength* of a consistency:

Definition 8. A property $\mathcal{T}$ is at least as strong as another property $\mathcal{T}'$ iff for any problem $\mathcal{P}$, when the enforcement of $\mathcal{T}'$ finds an inconsistency, then $\mathcal{T}$ finds an inconsistency too.

Consider now the little WCSP defined by three variables (x_1, x_2 and x_3) and two binary constraints ($c_{1,2}$ and $c_{1,3}$). $D(x_1) = \{a, b, c, d\}$, $D(x_2) = D(x_3) = \{a, b, c\}$ (we suppose $a \prec b \prec c \prec d$). The costs of the binary constraints are described Fig. 2. We set k to 2.

Let us show that the reified is 2B-consistent. Obviously, the bounds of every variable x_1, x_2 and x_3 have a support w.r.t. to the binary constraints. Moreover, the bounds of cost variables $x_E^{1,2}$ and $x_E^{1,3}$ also have a support: for example, it is $c_{1,2}(b, a)$ for the lower bound of $x_E^{1,2}$ and $c_{1,2}(c, a)$ for its upper bound. The extra constraint c'_E is also satisfied, hence the problem is 2B-consistent.

On the other hand, BAC* would detect an inconsistency by projecting the costs to x_1 and reducing little by little its domain. This shows that BAC* is at least not dominated by 2B-consistency for reified WCSPs. The fact that BAC* with $\varnothing$-inverse consistency always dominates 2B-consistency is still an open problem.

4.2 Range-based algorithm

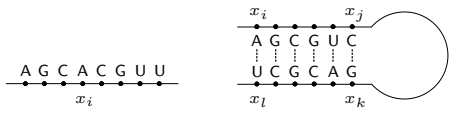
In [11], an algorithm called RMA is presented. It exploits a bound-based filtering algorithm to infer a lower bound, but only for the Max-CSP problem. A generalization in the WCSP framework of this work could be the *piecewise BAC**, where every domain is splitted into smaller domains, with a proper lower and upper bounds for each sub-domain. In the Max-CSP case, a domain is splitted into the consistent and the inconsistent part w.r.t. to a subset of constraints that applies on it. In this configuration, *piecewise BAC** performs at least as well as RMA.

5. EXPERIMENTAL RESULTS

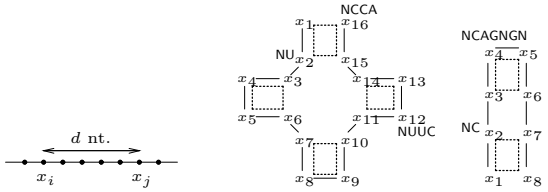
We have applied BAC* to the problem of non-coding RNA (ncRNA) detection. RNA sequences can be considered as oriented texts (left to right) over the four letter alphabet {A, C, G, U}. An RNA molecule can fold on itself through pairings between the nucleotides G–C, C–G, A–U and U–A. Such a folding gives rise to characteristic structural elements such as helices (a succession of paired nucleotides), and various kinds of loops (unpaired nucleotides surrounded by helices).

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.

We call *motif* the elements of the secondary structure that define a RNA family. To a first approximation, a motif can be decomposed into *strings* (cf. Fig. 3(a)) and *helices* (cf. Fig. 3(b)). Two elements can be separated by *spacers* (cf. Fig. 3(c)). These elements of description are modeled by soft constraints and the costs are given by editing distances (for strings and helices) or analytic function (for spacers).



(a) constraint $strings(x_i, \mathsf{ACGU})$ (b) constraint $helix(x_i, x_j, x_k, x_l)$



(c) constraint $spacer(x_i, x_j, d)$ (d) tRNA (e) IRE

Figure 3: A few motifs

Our aim is to find all the occurrences in the sequence that match the given motif, and the cost of these solutions. We have tried to detect the structure of tRNA [2] (cf. Fig. 3(d)), modeled by 16 variables, 15 spacers, 3 strings and 4 helices as well as an IRE motif [3] (cf. Fig. 3(e)) modeled by 8 variables, 7 spacers, 2 strings and 2 helices on parts of the genome of *Saccharomyces Cerevisiæ* of different sizes and on the whole genome of *Escherichia coli*. For tRNA, we used two different models, the first being much tighter than the second.

For each soft constraint, there is an hard constraint that prunes all the inconsistent values faster through 2B-consistency

tRNA, tight definition						
Size / ♯ solutions	10k / 16	50k / 16	100k / 16	500k / 16	1M / 24	ecoli / 140
AC* (nodes/time)	23 / 29	35 / 545	-	-	-	-
BAC* (nodes/time)	32 / 0	39 / 0	51 / 0	194 / 1	414 / 2	1867 / 7
tRNA, loose definition						
Size / ♯ solutions	10k / 84	50k / 84	100k / 84	500k / 111	1M / 164	ecoli / 702
AC* (nodes/time)	215 / 401	495 / 7041	-	-	-	-
BAC* (nodes/time)	347 / 0	1036 / 1	1775 / 2	8418 / 4	17499 / 8	83476 / 34
IRE						
Size / ♯ solutions	10k / 0	50k / 0	100k / 0	500k / 1	1M / 4	ecoli / 8
AC* (nodes/time)	0 / 3	0 / 57	0 / 223	-	-	-
BAC* (nodes/time)	0 / 0	0 / 0	0 / 0	20 / 0	44 / 2	237 / 8

Table 1: Number of nodes explored and time in seconds spent to solve several instances of the ncRNA detection problem

for classic CSPs. As the helix is a 4-ary constraint, we used a generalized bound arc consistency to propagate the costs. ∅IC has been enforced for spacers (which are semi-convex functions) but not for strings nor for helices. We used a 2.4Ghz Intel Xeon with 8 GB RAM to solve these instances. The results on our comparison between our algorithm and the classic AC* are displayed on FIG. 1. For each instance of the problem, we write its size (10k is sequence of 10.000 nucleotides of the genome of *Saccharomyces Cerevisiæ* 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.

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 $\mathcal{O}(n)$. A similar conclusion would have been drawn after a comparison between BAC* and Max-CSP algorithms like PFC-MRDAC (cf. [7]).

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.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a new local consistency for weighted CSPs, called *bound arc consistency*. It is specially devoted to problems with large domains and time and space complexities are lower than the well-known arc consistencies. An extension have been proposed for constraints with good characteristics, like semi-convex functions, and ∅IC seems particularly efficient for this kind of functions. Finally, we showed that maintaining BAC* is much better than AC* for the problem of ncRNA detection.

In the future, we will try to implement better heuristics for boosting the search.

7. REFERENCES

[1] M. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154:199–227, 2004.

[2] D. Gautheret, F. Major, and R. Cedergren. Pattern searching/alignment with RNA primary and secondary structures: an effective descriptor for tRNA. *Comp. Appl. Biosc.*, 6:325–331, 1990.

[3] J. Gorodkin, L. L. Heyer, and G. D. Stormo. Finding the most significant common sequence and structure motifs in a set of RNA sequences. *Nucleic Acids Research*, 25:3724–3732, 1997.

[4] P. V. Hentenryck, Y. Deville, and C.-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57(2–3):291–321, 1992.

[5] L. Khatib, P. Morris, R. Morris, and F. Rossi. Temporal constraint reasoning with preferences. In *Proc. IJCAI 2001*, pages 322–327, 2001.

[6] J. Larrosa. Node and arc consistency in weighted CSP. In *Proc. AAAI'02*, 2002.

[7] J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible DAC for Max-CSP. *Artificial Intelligence*, 17(1):149–163, 1999.

[8] J. Larrosa and T. Schiex. Solving Weighted CSP by Maintaining Arc-consistency. *Artificial Intelligence*, 159(1-2):1–26, 2004.

[9] O. Lhomme. Consistency techniques for numeric CSPs. In *Proc. IJCAI 1993*, pages 232–238, 1993.

[10] T. Petit, J.-C. Régin, and C. Bessière. Meta-constraints on violations for over constrained problems. In *Proc. ICTAI'00*, pages 358–365, 2000.

[11] T. Petit, J.-C. Régin, and C. Bessière. Range-based algorithm for Max-CSP. In *Proc. CP'02*, pages 280–294, 2002.

[12] T. Schiex. Arc consistency for soft constraints. In *Proc. CP'00*, pages 411–424, 2000.

[13] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *Proc. IJCAI 1995*, 1995.