Arc consistency for soft constraints

Thomas Schiex[1]

INRA, Toulouse, France,
Thomas.Schiex@toulouse.inra.fr,
http://www-bia.inra.fr/T/schiex

Abstract. Traditionally, local consistency is defined as a relaxation of consistency which can be checked in polynomial time. It is accompanied by a corresponding "filtering" or "enforcing" algorithm that computes in polynomial time, and from any given CSP, an equivalent unique CSP which satisfies the local consistency property.
The question whether the notion of local consistency can be extended to soft constraint frameworks has been addressed by several papers, in several settings [4, 14, 12]. The main positive conclusion of these works is that the notion of local consistency can be extended to soft constraints frameworks which rely on an idempotent violation combination operator. However, the question whether this can be done for non idempotent operators as eg, in the MAX-CSP problem, is not clear and has lead to several different notions of arc consistency [14, 16, 1, 11, 10]. Each of these proposals lacks several of the original properties of local consistency.
In this paper, we show that using a small additional axiom, satisfied by most existing soft constraints proposals (including MAX-CSP), it is possible to define a notion of arc consistency that keeps all the good properties of classical arc consistency but the unicity of the arc consistent closure. We show that this notion directly provides improved lower bounds. Stronger alternative definitions, that allow partial inconsistencies to propagate are also considered.

Introduction

Compared to other combinatorial optimisation frameworks, the CSP framework is essentially characterised by the ubiquitous use of so-called local consistency properties and enforcing algorithms among which arc consistency is certainly the most preeminent one.

The notion of local consistency can be easily characterised by a set of desirable properties: local consistency is a relaxation of consistency: for any consistent CSP, there is an equivalent non empty locally consistent CSP. Finding this equivalent locally consistent CSP, which is unique, is achievable in polynomial time by so-called enforcing or filtering algorithms. Several papers have tried to extend the classical notion of arc consistency to weighted constraint frameworks. In such frameworks, the aim is to find an assignment that minimises combined violations. The first work in this direction is probably [12] which defined arc consistency filtering for fuzzy CSP.

This extension was rather straightforward and one could think that this would also be the case for other frameworks such as the MAX-CSP framework, introduced in [15, 9], where the aim is to find an assignment which minimises the (weighted) number of violated constraints. This was not the case. Later works tried to extend arc consistency in a systematic way using axiomatic frameworks to characterise the properties of the operator used to combine violations:

- the Semi-Ring CSP framework was introduced in [4, 5]. In this work, the extension of arc consistency enforcing is induced by a generalisation of the fundamental relational operators such as projection, intersection and join. The essential conclusion of this work is that extended arc consistency works as far as the operator used to combine violations is idempotent. This includes the case of fuzzy CSP and also some other cases with partial orders. For MAX-CSP and other related cases, the algorithm may not terminate and may also provide non equivalent CSP.
- the Valued CSP framework was introduced in [14]. Here, the extension of the arc consistency property is essentially based on the notion of relaxation. The same conclusion as in the Semi-Ring CSP framework is reached for idempotent operators. For other frameworks such as the MAX-CSP framework, it is shown that the problem of checking the extended arc consistency property defines an NP-complete problem.

Parallel to these tentative extensions of arc consistency, several works such as [16, 1, 11, 10] tried to offer improved lower bounds for the MAX-CSP. These works abandon the idea of extending arc consistency in order to simply offer the most important service i.e., the ability to detect that a CSP has no solution of cost below a given threshold.

Globally, each of these proposals violates some of the desirable properties of local consistency. In this paper we show that it is possible, by the addition to the Valued CSP framework of a single axiom, to define an extended arc consistency notion that has all the desirable properties of classical arc consistency but the unicity of the arc consistent closure. It has also the nice property that in the idempotent operator cases, it reduces to existing working definitions and unicity is recovered. We show that a lower bound can easily be built from any of the arc consistent closures and that this lower bound generalises and improves on existing lower bounds [16, 1, 11, 10]. Finally, in the last section we consider stronger enforcing algorithms that may further improve the lower bounds by propagating partial inconsistencies and not only value deletions as in [11, 10].

1 Notations and definitions

A constraint satisfaction problem (CSP) is a triple $\langle X, D, C\rangle$. X is a set of n variables $X = \{1, \dots, n\}$. Each variable $i \in X$ has a domain of values $d_i \in D$ and can be assigned any value $a \in d_i$, also noted (i, a). C is a set of constraints. Each constraint $c \in C$ is defined over a set of variables $X_c \subset X$ by a subset of the Cartesian product $\prod_{i \in X_c} d_i$ which defines all consistent tuples of values. The cardinality $|X_c|$ is the arity of the constraint c. k will denote the largest arity of a CSP. We assume, without loss of generality, that at most one constraint is defined over a given set of variables. The set C is partitioned in two sets $C = C^1 \cup C^+$ where C^1 contains all unary constraints. The unary constraint on variable i will be denoted c_i. The projection of a tuple of values t over a set of variables $V \subset X$ is denoted by $t_{\downarrow V}$. This is extended to constraints: the projection $c_{\downarrow V}$ of a constraint c on a set of variables $V \subset X_c$ is the constraint defined on V and equal to the set of the projections on V of all tuples in c. A tuple $t \in c$ such that $t_{\downarrow\{i\}} = (a)$ is a support for value (i, a) on c. A tuple of values t satisfies a constraint c if $t_{\downarrow X_c} \in c$. Finally, a tuple of values over X is a solution iff it satisfies all the constraints of the CSP.

1.1 Valued CSP

Valued CSP (or VCSP) have been initially introduced in [14]. A valued CSP is obtained by associating a mathematical object, called a valuation, to each constraint of an original CSP. The set E of all valuations is assumed to be completely ordered and its maximum element is used for the most important constraints, namely hard constraints. When an assignment violates a set of constraints, one can compute its valuation by combining the valuations of all violated constraints using a dedicated operator (noted $\oplus$). This operator must satisfy a set of properties that are captured by a set of axioms defining a so-called "valuation structure".

Definition 1. *A* valuation structure *is defined as a tuple $\langle E, \oplus, \succcurlyeq \rangle$ such that:*

- *E is a set, whose elements are called valuations, which is totally ordered by $\succcurlyeq$, with a maximum element noted $\top$ and a minimum element noted $\perp$;*
- *$\oplus$ is a commutative, associative closed binary operation on E that satisfies:*
  - Identity*: $\forall a \in E, a \oplus \perp = a$;*
  - Monotonicity*: $\forall a, b, c \in E, (a \succcurlyeq b) \Rightarrow \big((a \oplus c) \succcurlyeq (b \oplus c)\big)$;*
  - Absorbing element*: $\forall a \in E, (a \oplus \top) = \top$.*

This structure of totally ordered commutative monoid with a monotonic operator is also known in uncertain reasoning, E being restricted to $[0, 1]$, as a "triangular co-norm" [6]. In the rest of the paper, we assume that the computation of $\succ$ and $\oplus$ are always polynomial in the size of their arguments. We also assume E is countable.

It is now possible to define valued CSPs. For the sake of generality, rather than considering that a valuation is associated with each constraint, as in [14], we will consider that a valuation is associated with each tuple of each constraint. As it is observed in [3], the two approaches are essentially equivalent.

Definition 2. *A valued CSP is a tuple $\langle X, D, C, S \rangle$ where $S = \langle E, \oplus, \succcurlyeq \rangle$ is a valuation structure, X is a set of n variables $X = \{1, \ldots, n\}$. Each variable $i \in X$ has a set of possible values $d_i \in D$. $C = C^1 \cup C^+$ is a set of constraints. Each constraint $c \in C$ is defined over a set of variables $X_c \subset X$ as a function from $\prod_{i \in X_c} d_i$ to E.*

An assignment A of values to some variables $V \subset X$ can be simply evaluated by combining, for all assigned constraints, the valuations of the tuple used:

Definition 3. *In a VCSP $\mathcal{P} = \langle X, D, C, S \rangle$, the valuation of an assignment t over $V \subset X$ is defined by:*

$$\mathcal{V}_{\mathcal{P}}(t) = \bigoplus_{c \in C, X_c \subset V} [c(t_{\downarrow X_c})]$$

The problem usually considered is to find a complete assignment with a minimum valuation. Globally, the semantics of a VCSP is a distribution of valuations over the assignments of X. The choice of axioms is quite natural and is usual in the field of uncertain reasoning. The ordered set E simply allows various impact of violations to be expressed. The commutativity and associativity guarantee that the valuation of an assignment depends only on *the set* of the valuations combined. The monotonicity of $\oplus$ guarantees that assignment valuations cannot decrease when constraint violations increase. For a more detailed analysis and justification of the VCSP axioms, we invite the reader to read [14, 10] which also emphasise the difference between idempotent and strictly monotonic $\oplus$ operators.

Definition 4. *An operator $\oplus$ is idempotent if $\forall v \in E, (v \oplus v) = v$. It is strictly monotonic if $\forall a, b, c \in E$, s.t. $(a \succ c) \wedge (b \neq \top)$, we have $(a \oplus b) \succ (c \oplus b)$*

As shown in [14], these two properties are incompatible as soon as $|E| > 2$. The only valuation structures with an idempotent operator corresponds to classical and fuzzy CSP, which use $\oplus = \max$ as the combination operator. Other soft CSP frameworks such as MAX-CSP, lexicographic CSP or probabilistic CSP use a strictly monotonic operator.

Arc consistency enforcing must yield an equivalent problem, the so-called arc-consistent closure. Several notions of equivalence have been introduced in [14, 10] that enable to compare pairs of VCSP with different valuations structure. In this paper, the notion of equivalence will only be used to compare pairs of VCSP with the same valuation structure and can be therefore simplified and strengthened. Two VCSP will be said equivalent iff they define the same distribution over complete assignments:

Definition 5. *Two VCSP $\mathcal{P} = \langle X, D, C, S \rangle$ and $\mathcal{P}' = \langle X, D, C', S \rangle$ are equivalent iff for any complete assignment A, we have:*

$$\mathcal{V}_{\mathcal{P}}(A) = \mathcal{V}_{\mathcal{P}'}(A)$$

2 Arc consistency revisited

In this section we first give a slightly unusual and very naive definition of classical arc consistency which will make the extension to soft constraints easier. We consider that the final domains of the CSP are defined by the unary constraints $c_i \in C^1$.

Definition 6. *A classical CSP is arc consistent if for each constraint $c \in C^+$ and for each variable $i \in X_c$, we have $(c_i \subset c_{\Downarrow\{i\}})$ and $(c_i \supset c_{\Downarrow\{i\}})$.*

Traditionally, a CSP is said to be arc consistent simply when $(c_i \subset c_{\Downarrow\{i\}})$. Our definition explicits the usual *implicit* assumption that tuples which use a deleted value are removed from the constraints i.e. that $(c_i \supset c_{\Downarrow\{i\}})$ too. Except for the fact that a CSP with empty domains may be arc consistent using this definition, it is equivalent to the usual definition (we consider that the possible emptiness of the arc consistent closure is a separate property). The first condition guarantees that each value has a tuple supporting it while the second condition guarantees that all tuples are made of viable values. So, globally, each value is supported by a viable tuple.

A corresponding enforcing algorithm can be defined by the iterated application, until quiescence, of two procedures, each enforcing one of the two conditions above: Projection enforces the fact that $(c_i \subset c_{\Downarrow\{i\}})$ by deleting values with no support. Extension enforces the fact that $(c_i \supset c_{\Downarrow\{i\}})$ by deleting tuples that contain a deleted value. The unary constraints c_i are used to mark deleted values. Termination follows from the fact that each iteration of AC, but the last, removes at least one value or tuple and there is only $O(e.d^k)$ of them. In the binary case, since Projection and Extension are in $O(d)$, each iteration of AC is in $O(e.d^2)$ and this shows that our naive arc consistency enforcing algorithm is in $O(e^2.d^4)$.

Algorithm 1 – Naive Arc Consistency enforcing

Projection(c, i, a);
$m \leftarrow 0$;
if $\nexists t \in c$ s.t. $t_{\Downarrow\{i\}} = (a)$ **then**
  Delete (a) from c_i;
  $m \leftarrow 1$;
return m;

Extension(i, a, c);
$m \leftarrow 0$;
if $(a) \notin c_i$ **then**
  foreach $t \in c$ s.t. $t_{\Downarrow\{i\}} = (a)$ **do**
    Delete t from c;
    $m \leftarrow 1$;
return m;

AC$(X, D, C^1 \cup C^+)$;
$m \leftarrow 1$;
while (m) **do**
  $m_c \leftarrow m_i \leftarrow 0$;
  foreach $c \in C^+$ **do**
    foreach $i \in X_c$ **do**
      foreach $a \in d_i$ **do** $m_c \leftarrow (m_c \vee$ Projection$(c, i, a))$;
  foreach $i \in X$ **do**
    foreach $c \in C^+$ s.t. $i \in X_c$ **do**
      foreach $a \in d_i$ **do** $m_i = (m_i \vee$ Extension$(i, a, c))$;
  $m \leftarrow m_i \vee m_c$;

AC returns a CSP which is equivalent to the original CSP since Projection and Extension only remove values or tuples which cannot participate to a solution. The final CSP is arc consistent since it is closed under both Projection and Extension. The poor time complexity of this naive algorithm, compared to the optimal $O(e.d^2)$ time complexity first achieved by AC-4 on binary CSP should be considered as the consequence of its intended simplicity: the aim of this paper is (only) to show that arc consistency can be extended to eg. MAX-CSP. We have chosen to describe an enforcing algorithm that will be easy to extend to VCSP. To achieve this, we will mainly have to guarantee that the extended versions of the elementary operations Projection and Extension preserve the semantics of the problem.

2.1 Arc consistency for soft constraints

We start with an introductory example. In the sequel of the paper, in order to illustrate the notions introduced on concrete examples, we will consider binary weighted MAX-CSP which correspond to valued CSP using the strictly monotonic valuation structure $\langle \mathbb{N} \cup \{+\infty\}, +, > \rangle$. To describe such problems, we use a graph representation where vertices represent values and weighted edges connect values from pairs of variables connected by a constraint. In this representation, we omit all edges weighted by 0 ($\perp$) and all weights equal to $+\infty$ ($\top$). Unary constraints (actual domains) are represented by weights associated with vertices, weights equal to 0 being omitted.

Let us consider the weighted MAX-CSP in figure 1(a). It has two variables called 1 and 2, each with two values a and b and a single constraint. The constraint forbids pair $((1,b),(2,b))$ with cost 1 and forbids pairs $((1,a),(2,a))$ and $((1,b),(2,a))$ completely (with cost $+\infty$). The pair $((1,a),(2,b))$ is authorised and the corresponding edge is therefore omitted.

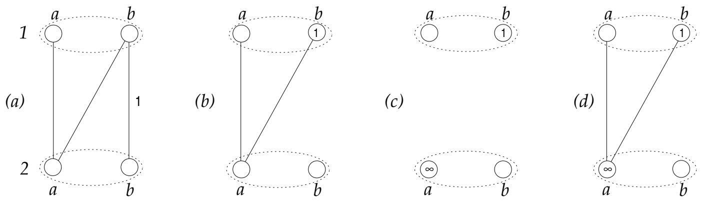

Fig. 1. Four equivalent MAX-CSP

If we assign the value b to variable 1, it is known for sure that a cost of 1 at least must be payed since the best support for $(1,b)$ has cost 1. Projection should make this explicit and induce a unary constraint on 1 that forbids $(1,b)$ with cost 1. However if we simply add this constraint to the MAX-CSP, as it has been proposed in [4] for those cases with an idempotent operator, the resulting CSP is not equivalent. The complete assignment $((1,b),(2,b))$ which has initially cost 1 would now have cost 2. In order to preserve equivalence, we must "compensate" for the induced unary constraint. This can be done by simply subtracting 1 from all the tuples that contain the value $(1,b)$. The corresponding equivalent CSP is described in figure 1(b): the edge $((1,b),(2,b))$ of cost 1 has disappeared (the associated weight is now 0) while the edge $((1,b),(2,a))$ is unaffected since it has an infinite cost. We can repeat this process for variable 2: all the supports of value $(2,a)$ have infinite cost. Thus we can add a unary constraint that completely forbids value $(2,a)$. In this specific case, and because the valuation $+\infty$ is such that $\forall v \in E, +\infty \oplus v = +\infty$, we can either compensate for this (figure 1(c)) or not (figure 1(d)). In both cases, an equivalent MAX-CSP is obtained. Actually, the weight associated with each of the edges connected to value $(2,a)$ cannot affect the semantics of the CSP since the value is completely forbidden. CSP 1(c) is interesting because it is is completely solved. However, we prefer the problem 1(d) because it makes the information explicit both at the domain and constraint level.

2.2 A new axiom for VCSP

To generalise this idea to other possible valuation structures, we have to be able to "compensate" for the information added by projection. This is made possible by the following additional axiom:

Definition 7. *A valuation structure $\langle E, \oplus, \succcurlyeq\rangle$ is* fair *if for any pair of valuations $u, v \in E$, $v \preccurlyeq u$, there exist a valuation $w \in E$ such that $v \oplus w = u$. w will be called a difference of u and v. The unique maximum difference of u and v will be noted $u \ominus v$.*

This simple axiom is actually satisfied by most existing concrete frameworks, including concrete with idempotent or strictly monotonic $\oplus$ operators.

- classical CSP can be defined as VCSP using the valuation set $E = \{0,1\}$, where $0 = \bot \prec 1 = \top$. The combination operator is max and we have $0 \ominus 0 = 0$, $1 \ominus 0 = 1$ and $1 \ominus 1 = 1$. So $\ominus = \max$ too. Note that 0 is another difference between 1 and 1, but it is not the maximum one.
- possibilistic CSP, which define a min-max problem dual to the classical max-min problem of fuzzy CSP, can be defined using the valuation set $\mathbb{N} \cup \{+\infty\}$, ordered as usual and using $\oplus = \max$. Here again, differences exist and one can show that $\ominus = \max$.
- weighted MAX-CSP can be defined using the valuation set $\mathbb{N} \cup \{+\infty\}$, ordered as usual and using $\oplus = +$. Here again, differences exist and one can show that $u \ominus v = u - v$ on finite valuations. Else, for all $v \in E$, we have $+\infty \ominus v = +\infty$.

Note that other concrete frameworks such as probabilistic CSP [7] or lexicographic CSP [8] can be reduced to weighted MAX-CSP as shown in [14] and can therefore benefit from the results given in this paper. This is particularly important for lexicographic CSP for which difference is not always defined.

Theorem 1. *Let $S = \langle E, \oplus, \succcurlyeq \rangle$ be a fair valuation structure. Then $\forall u, v, w, \in E, w \preccurlyeq v$, we have $(v \ominus w) \preccurlyeq v$ and $(u \oplus w) \oplus (v \ominus w) = (u \oplus v)$.*

Proof. By definition, $(v \ominus w) \oplus w = v$. From the monotonicity of $\oplus$, this proves that $(v \ominus w) \preccurlyeq v$ (this inequality becomes strict if $\oplus$ is strictly monotonic and $v \neq \top$). The second property follows from the commutativity and associativity of $\oplus$: we have $(u \oplus w) \oplus (v \ominus w) = u \oplus ((v \ominus w) \oplus w) = (u \oplus v)$. □

2.3 Soft Arc Consistency enforcing

It is now possible to define an extended version of the AC enforcing algorithm presented before. Our aim is to define a generic algorithm that will consistently work in all fair valuation structures. Remember that c_i denotes the unary constraint on variable i in the current CSP. So, $c_i(a)$ is the valuation associated with value (i, a) in the unary constraint. Similarly to value removals in classical AC enforcing, we will also use statements such as $c(t) \leftarrow v$ which means that the valuation associated with the tuple t in c will be set to v. We also require the following definitions:

Definition 8. *Let $v, w \in E$, be two valuations. If $v \neq \bot$ and $v \ominus v = v$, we say that v is* absorbing. *If $v \oplus w \neq w$ we say that v* affects *w.*

We assume that these properties can be tested in constant time. This is the case in all the concrete valuation structures considered. Note that $\top$ is absorbing in all valuation structures. When $\oplus$ is idempotent, then all valuations but $\bot$ are absorbing. We are now ready to extend our elementary procedures Projection and Extension.

Algorithm 2 – Extended procedures for Soft AC enforcing

```
Projection(c, i, a);
m ← 0;
v ← ⊤;
foreach t ∈ c s.t. t_{↓{i}} = (a) do
    v ← min(v, c(t));
if v affects c_i(a) then
    v' ← Choose(v);
    c_i(a) ← c_i(a) ⊕ v';
    m ← 1;
    foreach t ∈ c s.t. t_{↓{i}} = (a) do
        c(t) ← c(t) ⊖ v';
return m;
```

```
  Extension(i, a, c);
  m ← 0;
  v ← c_i(a);
  if v is absorbing then
      v' ← Choose(v);
      foreach t ∈ c s.t. t_{↓{i}} = (a) do
          if v' affects c(t) then
              c(t) ← c(t) ⊕ v';
              m ← 1;
1     c_i(a) ← c_i(a) ⊖ v';
  return m;
```

Let analyse these procedures. Both use a function Choose(v) that takes a valuation as argument and returns a valuation. Assume for now that Choose(v) is the identity function. In Projection, we first look for a cheapest support for the value (i, a) on constraint c and we keep its valuation in v. If (i, a) is used in an assignment, we will have to pay for this valuation at least. This gives a possible new unary constraint. Before adding this constraint, we first check if this would modify the CSP: we test if v affects the existing valuation $c_i(a)$ at the unary constraint level. If it does, we add the induced unary constraint to the CSP, we note that the CSP is modified and then we compensate for the addition by subtracting v from the valuations of all the supports of (i, a). This can always be done because the valuation structure is fair and by construction v is less than or equal to all the valuations of the supports.

In Extension, we first check if the valuation v of the value a is absorbing. This is needed for termination (see section 4). Then for each of the supports of (i, a) on c, we check if v can affect the valuation of the support and if so, we modify the tuple valuation. The line numbered 1 compensates for this at the unary level. This line is actually useless since v is absorbing but it will be useful in a later version of the procedure Extension where v may be non absorbing.

Theorem 2. *Given any fair VCSP $P = \langle X, D, C, S\rangle$, $i \in X$, $a \in d_i$ and $c \in C$ s.t. $i \in X_c$, the application of Projection(c, i, a) or Extension(i, a, c) to P yields an equivalent VCSP.*

Proof. Consider any complete assignment t. If $t_{\downarrow\{i\}} \neq (a)$ then the valuation of the assignment is unaffected by the application of Projection or Extension since these procedures only modify valuations of tuples containing value (i, a). If $t_{\downarrow\{i\}} = (a)$, we note $\alpha = c_i(a)$ and $\beta = c(t_{\downarrow X_c})$. The initial valuation of t can be written $\omega \oplus \alpha \oplus \beta$ where ω contains the combined valuations of all constraints except c and c_i. After the application of Projection, the valuation of t becomes $\omega \oplus (\alpha \oplus v) \oplus (\beta \ominus v) = \omega \oplus \alpha \oplus \beta$ by theorem 1 and is therefore unaffected. Similarly, the application of Extension yields a valuation of $\omega \oplus (\alpha \ominus v) \oplus (\beta \oplus v) = \omega \oplus \alpha \oplus \beta$ by theorem 1. □

Consider our initial AC algorithm using the new Projection and Extension procedures. If it terminates, it necessarily returns an equivalent problem. Termination is still unproved in the general case, however it is easy to prove it in the most important cases i.e. when $\oplus$ is either idempotent or strictly monotonic:

- Idempotent case: $\oplus$ is equal to max (see [14]) and obviously $\ominus$ = max too. The number of different valuations one may obtain by combining valuations with max is finite and equal to the number of different valuations that exist originally in the CSP. Then, each application of Projection or Extension may only increase valuations in domains or constraints. Since they can only take a finite number of values, the algorithm will stop.
- Strictly monotonic case: the only absorbing value is $\top$. For all other valuations $v \neq \top$, we have $v \ominus x \prec v$ for any $x \neq \bot$ (this follows from strict monotonicity). If we consider valuations inside constraints, each call to Projection may only reduce valuations while each call to Extension can only set constraint valuations to $\top$ which is the maximum element and as an absorbing value, cannot decrease. Globally, each iteration of the AC algorithm can

either lower constraint valuations or set them to $\top$. In the first case, the valuation cannot decrease below $\perp$ and in the second case they cannot be modified again. Termination is guaranteed because there is only a finite number of tuples inside constraints. Note that this proof is still valid if Choose returns any value which is smaller or equal to v and strictly larger than $\perp$.

So we now have an extended enforcing algorithm that terminates and yields an equivalent problem for all fair valuation structures with either an idempotent or strictly monotonic $\oplus$ operator. Note also that its specialisation to classical CSP described as VCSP corresponds to our classical AC enforcing algorithm. We can now define the corresponding arc consistency property as follows:

Definition 9. *A fair VCSP is arc consistent if for each constraint $c \in C^+$, for each variable $i \in X_c$, for each value $a \in d_i$ we have:*

- *the minimum valuation of the supports of (i, a) on c does not affect $c_i(a)$.*
- *if $c_i(a)$ is absorbing, then it does not affect any of the valuations of the supports of (i, a) on c.*

Our AC enforcing algorithm is correct by definition of the AC property which precisely corresponds to the properties enforced by Projection and Extension respectively. Note also that its specialisation to classical CSP described as VCSP corresponds to our classical AC definition.

We still have to prove that the algorithm is polynomial time. We haven't built a generic proof and have instead decided to limit ourself to the most significant case of binary weighted MAX-CSP. In this case, Projection is in $O(d)$ and Extension in $O(d)$ too. In AC each iteration performs $O(ed)$ calls to these procedures. Each iteration but the last either deletes a value, sets a tuple valuation to $0 = \perp$ or to $+\infty = \top$. There are only $e.d^2$ tuples and $n.d$ values so this naive AC enforcing algorithm is in $O(e^2.d^4)$ and is effectively polynomial time. Naturally, some work is needed to turn this theoretical result in a tuned time and space efficient algorithm.

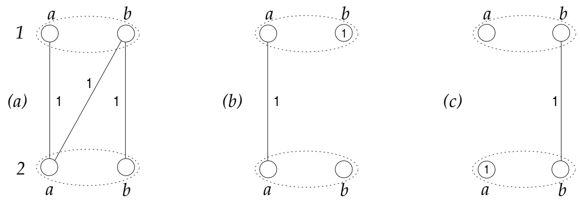

Fig. 2. A MAX-CSP and two different equivalent arc consistent closure

Finally, we observe that confluence of arc consistency enforcing is lost. This means that the arc consistent closure of a problem is not necessarily unique as the example of figure 2 shows. The MAX-CSP considered is similar to the problem of figure 1 except that non authorised tuples have all cost 1. If we start enforcing arc consistency by variable 1, we detect that $(1,b)$ has a minimum cost support of cost 1 and we obtain the CSP 2(b). This CSP is arc consistent and equivalent to the CSP 2(a). If we start by variable 2, we detect that $(2,a)$ has a minimum cost support of cost 1 and we obtain the CSP 2(c). This CSP is also arc consistent and equivalent to CSP 2(a) but is different from the CSP 2(b).

3 Lower bounds and comparison

In the VCSP framework, as in the CSP framework, each variable must be assigned a value. So, the following valuation is always a lower bound on the cost of an optimal solution:

$$\ell = \bigoplus_{i \in X} [\min_{a \in d_i} c_i(a)]$$

Computed after AC enforcing, this lower bound will be noted ℓ_{ac}. It depends on the arc consistent closure found which depends on the order constraints and variables are examined by AC enforcing. A CSP is said to be *maximally arc consistent* if it is arc consistent and if ℓ_{ac} is maximum over all equivalent arc consistent problems. The problem of enforcing maximum arc consistency is certainly important, but there are reasons to think that it is an NP-hard problem (see [13]). As an example, consider the CSP in figure 3. It has four variables with two values each and contains four constraints. Enforcing arc consistency of this problem yields the CSP on the right. The lower bound ℓ_{ac} is here equal to 4.

We now compare ℓ_{ac} to recent lower bounds for binary MAX-CSP. The first lower bound considered is the *reversible directed arc consistency counts* based lower bound (RDAC [11, 10]). It is parameterised by booleans indicating a direction per binary constraint. For each constraint c, and for each value a of the variable i indicated by the direction selected, the counter $w_{i,a}$ is incremented if the value has no support on the constraint. The induced lower bound is the sum of the minimum of the w_{ia} per variable. If we enforce AC and if we examine constraints in the same order as RDAC, always starting by the variable that RDAC examines, it is quite obvious that Projection will discover exactly the same counts as RDAC, as unary constraints. Additionally, it will examine the other side of each constraint and possibly infer more unary constraints which may lead to a strictly better lower bound. If we consider our example, exactly one value in each domain has a support of cost 0. So, RDAC will only be able to infer at most 4 counts of 1 for the eight values. Ideally, if the constraints are correctly oriented (eg. 2 constraints directed towards variable 1 and the 2 others towards variable 3), these 4 costs induce a lower bound of 2. However, a useless lower bound of 0 can also be inferred if the constraints are poorly oriented eg. if one constraint is oriented towards each variable.

The second lower bound considered is the *weighted arc consistency counts* based lower bound (WAC [1]). It is parameterised by one real number between 0 and 1 per constraint. It has been shown in [2] that it is a generalisation of RDAC. If α is the number associated with the constraint c, the counts inferred by WAC

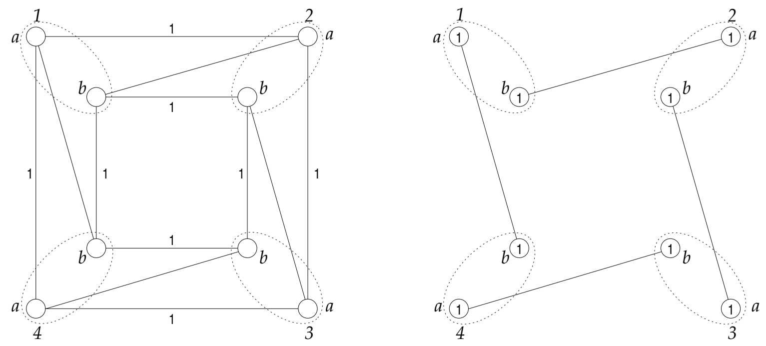

Fig. 3. A Max-CSP, and its arc consistent closure

will be α times the counts that RDAC would have inferred in one direction and $(1-\alpha)$ times the counts it would have inferred in the other direction. We can parameterise AC enforcing by modifying the Choose procedure so that instead of returning v it returns $\alpha.v$ on one side and $(1-\alpha).v$ on the other side and conclude that AC can always yield a lower bounds that is at least as good as WAC: we start by applying Projection once on each constraint/variable with the Choose procedure parameterised by the α. The CSP obtained exhibits the same costs as WAC as unary constraints but may be non arc consistent. If we finish enforcing arc consistency using the standard Choose procedure, we only increase costs and therefore increase the lower bound. If we again consider the problem of figure 3, WAC can ideally infer a lower bound of 2, as RDAC, using eg. $\alpha = \frac{1}{2}$ for all constraints.

So, our AC enforcing algorithm provides stronger lower bounds than recent proposals. Furthermore, it preserves equivalence and is defined for non-binary constraints.

4 Propagating local inconsistencies

In this section we consider the possibility, for procedure Extension, of processing valuations that are not absorbing. The corresponding procedure is called Extension+ and is simply deduced from Extension by removing the test whether $v = c_i(a)$ is absorbing (see algorithm 2). Propagation of non absorbing valuations has been forbidden in the basic version in order to guarantee termination: for non absorbing valuations, the use of Extension+(i, a, c) exactly cancels the effect of a previous call to Projection(c, i, a) which my lead to infinite loops. However, termination can always be forced if it is not natural. We find agreeable, in this section, to consider AC enforcing as a local search process in the space of equivalent CSP. The criterion optimised is the lower bound ℓ. A first neighbourhood can be defined by all possible applications of Projection and Extension. Its use corresponds to the AC enforcing algorithm 2: a greedy local search process always terminates and the locally optimal CSP reached is arc consistent.

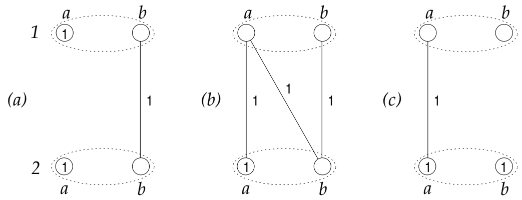

Fig. 4. Propagating non absorbing valuations may improve ℓ_{ac}

A second larger neighbourhood is defined by all possible applications of Projection and Extension+. It is particularly interesting in the context eg. of MAX-CSP where the propagation of non absorbing valuations may be the key to reach a maximum arc consistent closure. Consider the first MAX-CSP in figure 4. It includes one binary constraint that forbids pair $((1, b)(2, b))$ and two unary constraints that forbid values $(1, a)$ and $(2, a)$. The CSP is arc consistent and the corresponding lower bound ℓ_{ac} equals 0. If we apply Extension+ on value $(1, a)$ we get the equivalent CSP 4(b) which is not arc consistent. We can apply Projection on value $(2, b)$ and obtain the CSP 4(c) which is arc consistent with a corresponding lower bound $\ell_{ac} = 1$. The propagation of non absorbing valuations may allow the transport of local unary constraints from variable to variable. For some variables, this may be used to increase the minimum valuation $c_i(a)$ and therefore the lower bound ℓ_{ac}. The question whether a maximum arc consistent closure can always be reached from a given CSP using this second neighbourhood is open but obviously the first neighbourhood does not suffice. Further work is needed to determine if and how one could get similar results in an AC enforcing algorithm that always terminates.

Conclusion

In this paper, we have defined an extended version of arc consistency that applies to classical CSP, fuzzy CSP, weighted MAX-CSP and others. It extends the usual notion of equivalent arc consistent problem in a systematic way, including frameworks for which the notion was not available and this in the general case of non binary constraints. Several directions are still worth some attention. First of all, some work is obviously needed to turn our naive arc consistency enforcing algorithm in a tuned space and time efficient algorithm. This paper, combined with the existing arc consistency technology of AC-{6,7} should do the trick. The resulting algorithm should then be injected in tree-search algorithms for solving MAX-CSP such as PFC-MRDAC [11, 10]. More generally, the technique used in this paper to extend arc consistency should also allow the extension of most (if not all) existing local consistency properties of classical CSP to MAX-CSP and provide even stronger lower bounds.

Acknowledgements I want to sincerely thank G. Verfaillie for the stimulating discussions we always have and M. Lemaître for its comments on a first version of the paper.

References

[1] AFFANE, M. S., AND BENNACEUR, H. A weighted arc consistency technique for Max-CSP. In *Proc. of the 13th ECAI* (Brighton, United Kingdom, 1998), pp. 209–213.

[2] AFFANE, M. S., BENNACEUR, H., AND SCHIEX, T. Comparaison de deux minorants paramétriques pour Max-CSP. In *Actes de JNPC'99* (1999), pp. 95–102.

[3] 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.

[4] BISTARELLI, S., MONTANARI, U., AND ROSSI, F. Constraint solving over semirings. In *Proc. of the 14th IJCAI* (Montréal, Canada, Aug. 1995).

[5] BISTARELLI, S., MONTANARI, U., AND ROSSI, F. Semiring based constraint solving and optimization. *Journal of the ACM 44*, 2 (1997), 201–236.

[6] DUBOIS, D., AND PRADE, H. A class of fuzzy measures based on triangular norms. a general framework for the combination of uncertain information. *Int. Journal of Intelligent Systems 8*, 1 (1982), 43–61.

[7] FARGIER, H., AND LANG, J. Uncertainty in constraint satisfaction problems: a probabilistic approach. In *Proc. of ECSQARU '93, LNCS 747* (Grenade, Spain, Nov. 1993), pp. 97–104.

[8] FARGIER, H., LANG, J., AND SCHIEX, T. Selecting preferred solutions in Fuzzy Constraint Satisfaction Problems. In *Proc. of the 1st European Congress on Fuzzy and Intelligent Technologies* (1993).

[9] FREUDER, E., AND WALLACE, R. Partial constraint satisfaction. *Artificial Intelligence 58* (Dec. 1992), 21–70.

[10] LARROSA, J., MESEGUER, P., AND SCHIEX, T. Maintaining reversible DAC for Max-CSP. *Artificial Intelligence 107*, 1 (Jan. 1999), 149–163.

[11] LARROSA, J., MESEGUER, P., SCHIEX, T., AND VERFAILLIE, G. Reversible DAC and other improvements for solving max-CSP. In *Proc. of AAAI'98* (Madison, WI, July 1998).

[12] ROSENFELD, A., HUMMEL, R., AND ZUCKER, S. Scene labeling by relaxation operations. *IEEE Trans. on Systems, Man, and Cybernetics 6*, 6 (1976), 173–184.

[13] SCHIEX, T. Maximizing the reversible DAC lower bound in Max-CSP is NP-hard. Tech. Rep. 1998/02, INRA, July 1998.

[14] SCHIEX, T., FARGIER, H., AND VERFAILLIE, G. Valued constraint satisfaction problems: hard and easy problems. In *Proc. of the 14th IJCAI* (Montréal, Canada, Aug. 1995), pp. 631–637.

[15] SHAPIRO, L., AND HARALICK, R. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence 3* (1981), 504–519.

[16] WALLACE, R. Directed arc consistency preprocessing. In *Selected papers from the ECAI-94 Workshop on Constraint Processing*, M. Meyer, Ed., no. 923 in LNCS. Springer, Berlin, 1994, pp. 121–137.