Virtual arc consistency for weighted CSP

Martin Cooper, Simon de Givry, Marti Sanchez, Thomas Schiex, Matthias Zytnicki

▶ **To cite this version:**

Martin Cooper, Simon de Givry, Marti Sanchez, Thomas Schiex, Matthias Zytnicki. Virtual arc consistency for weighted CSP. Twenty-third AAAI Conference on Artificial Intelligence, Jul 2008, Chicago, United States. pp.6. hal-02752851

HAL Id: hal-02752851
https://hal.inrae.fr/hal-02752851

Submitted on 3 Jun 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008)

Virtual Arc Consistency for Weighted CSP

M. Cooper
IRIT - Univ. Paul Sabatier, Toulouse, France
cooper@irit.fr

S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki
INRA, UR 875, Toulouse, France
{degivry,tschiex}@toulouse.inra.fr

Abstract

Optimizing a combination of local cost functions on discrete variables is a central problem in many formalisms such as in probabilistic networks, maximum satisfiability, weighted CSP or factor graphs. Recent results have shown that maintaining a form of local consistency in a Branch and Bound search provides bounds that are strong enough to solve many practical instances.

In this paper, we introduce Virtual Arc Consistency (VAC) which iteratively identifies and applies *sequences* of cost propagation over rational costs that are guaranteed to transform a WCSP in another WCSP with an improved constant cost. Although not as strong as Optimal Soft Arc Consistency, VAC is faster and powerful enough to solve submodular problems. Maintaining VAC inside branch and bound leads to important improvements in efficiency on large difficult problems and allowed us to close two famous frequency assignment problem instances.

Introduction

Graphical model processing is a central problem in AI. The optimization of the combined cost of local cost functions, central in the valued CSP framework (Schiex, Fargier, & Verfaillie 1995), captures problems such as weighted MaxSAT, Weighted CSP or Maximum Probability Explanation in probabilistic networks. It has applications in *resource allocation*, *combinatorial auctions*, *bioinformatics*...

Dynamic programming approaches such as bucket or cluster tree elimination have been largely used to tackle such problems but are inherently limited by their guaranteed exponential time and space behavior on graphical models with high tree width. Instead, Branch and Bound allows to keep a reasonable space complexity but requires good (strong and cheap) lower bounds on the minimum cost to be efficient.

In the last years, increasingly better lower bounds have been designed by enforcing local consistencies for WCSP. Enforcing is done by the iterated application of Equivalence Preserving Transformations (EPTs, (Cooper & Schiex 2004)) which extend the usual local consistency operations used in pure CSP. A similar trend has followed in MaxSAT where equivalence preserving "inference rules" are now used in all recent solvers (Heras, Larrosa, & Oliveras 2007). Such EPTs move integer weights between cost functions of different arities and may be able to eventually increase the cost function of empty scope (a constant) to a non naive value. This value provides an obvious lower bound on the minimum cost which can be maintained during search.

At the arc consistency level, the unrestricted chaotic application of arc EPTs does not usually converge to a unique fix-point (Schiex 2000) and may even not terminate. Different heuristic restrictions that terminates have been therefore introduced leading to different variants of soft arc consistency such as AC*, DAC*, FDAC*, EDAC* (de Givry *et al.* 2005)... The recent definition of Optimal Soft Arc Consistency (OSAC (Cooper, de Givry, & Schiex 2007)) showed that it is possible to precompute, in polynomial time, a *set* of rational cost EPTs that maximizes the constant cost function obtained. This algorithm (based on linear programming) provides strong lower bounds but seems too costly to be maintained during tree-search.

In this paper, we introduce Virtual Arc Consistency which uses an instrumented classical arc consistency algorithm to produce *sequences* of rational cost EPTs that increase the lower bound. Although not as powerful as OSAC, VAC is often much faster and still capable of directly solving submodular cost function networks. It is one of the key ingredient that allowed us to close two hard frequency assignment problems which have remained open for more than 10 years.

Preliminaries

A weighted CSP (WCSP) is a quadruplet (X, D, W, m). X and D are sets of n variables and domains, as in a standard CSP. The domain of variable i is denoted D_i. For a set of variables $S \subset X$, we note $\ell(S)$ the set of tuples over S. W is a set of cost functions. Each cost function (or soft constraint) w_S in W is defined on a set of variables S called its scope and assumed to be different for each cost function. A cost function w_S assigns costs to assignments of the variables in S i.e.: $w_S : \ell(S) \rightarrow [0, m]$. The set of possible costs is $[0, m]$ and $m \in \{1, \ldots, +\infty\}$ represents an intolerable cost. Costs are combined by the bounded addition $\oplus$, defined as $a \oplus b = \min\{m, a + b\}$ and compared using $\geq$. Observe that the intolerable cost m may be either finite or infinite. A cost b may also be subtracted from a larger cost a using the operation $\ominus$ where $a \ominus b$ is $(a - b)$ if $a \neq m$ and m otherwise.

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

For simplicity, we assume that every constraint has a different scope. For binary/unary cost functions, we use simplified notations: a binary cost function between variables i and j is denoted w_{ij}. A unary cost function on variable i is denoted w_i. We assume the existence of a unary cost function w_i for every variable, and a nullary (constant) cost function, noted $w_\varnothing$. If $m = 1$, note that the WCSP is equivalent to the CSP (cost 1 being associated to forbidden tuples). To make clear that in this case, cost functions represent constraints, they will be denoted as c_S instead of w_S.

The cost of a complete assignment $t \in \ell(X)$ in a problem $P = (X, D, W, m)$ is $Val_P(t) = \bigoplus_{w_S \in W} w_S(t[S])$ where $t[S]$ denotes the usual projection of a tuple on the set of variables S. The problem of minimizing $Val_P(t)$ is an optimization problem with an associated NP-complete decision problem.

Enforcing a given local consistency property on a problem P consists in transforming $P = (X, D, W, m)$ in a problem $P' = (X, D, W', m)$ which is equivalent to P ($Val_P = Val_{P'}$) and which satisfies the considered local consistency property. This enforcing may increase $w_\varnothing$ and provide an improved lower bound on the optimal cost. Enforcing is achieved using Equivalence Preserving Transformations (EPTs) moving costs between different scopes.

Algorithm 1 gives two elementary EPTs. Project() works in the scope of one cost function w_S. It moves an amount of cost α from w_S to a unary cost function $w_i, i \in S$, for a value $a \in D_i$. If the cost α is negative, cost moves from the unary cost function w_i to the cost function w_S: this is called an extension. To avoid negative costs in P', one must have $-w_i(a) \leq \alpha \leq \min_{t \in \ell(S), t[\{i\}]=a}\{w_S(t)\}$. Similarly, UnaryProject() works on a subproblem defined by one variable $i \in X$. It moves a cost α from the unary cost function w_i to the nullary cost function $w_\emptyset$ (with $-w_\varnothing \leq \alpha \leq \min_{a \in D_i}\{w_i(a)\}$ in order to keep costs positive).

Algorithm 1: The two EPTs Project and UnaryProject

Procedure Project(w_S, i, a, α)
  $w_i(a) \leftarrow w_i(a) \oplus \alpha$;
  foreach ($t \in \ell(S)$ such that $t[\{i\}] = a$) **do**
    $w_S(t) \leftarrow w_S(t) \ominus \alpha$;

Procedure UnaryProject(i, α)
  $w_\varnothing \leftarrow w_\varnothing \oplus \alpha$;
  foreach ($a \in D_i$) **do** $w_i(a) \leftarrow w_i(a) \ominus \alpha$;

All previous arc level local consistencies definitions (AC, DAC, FDAC, EDAC) can be seen as well defined polynomial time heuristics trying to get closer to a closure maximizing $w_\varnothing$. Allowing for rational costs, it has been shown that such an optimal closure can be found by Optimal Soft Arc Consistency (OSAC, (Cooper, de Givry, & Schiex 2007)). This requires solving a large linear program and has therefore been limited to preprocessing. We introduce in the next section an alternative cheaper mechanism based on classical AC which can be maintained during search.

Virtual Arc Consistency

Given a WCSP $P = (X, D, W, m)$, we define Bool(P) as the CSP $(X, D, \overline{W})$ where $c_S \in \overline{W}$ iff $\exists w_S \in W$ with $S \neq \varnothing$ such that $\forall t \in \ell(S)$ $(t \in c_S \Leftrightarrow w_S(t) = 0)$. Bool($P$) is a classical CSP whose solutions (if any) have cost $w_\varnothing$ in P.

Definition 1 *A WCSP P is Virtual Arc Consistent (VAC) if the arc consistent closure of Bool(P) is not empty.*

If a WCSP P is not VAC, then Bool(P) is inconsistent and it is known that all solutions of P have a cost strictly higher than $w_\varnothing$. More interestingly, by simulating the application of classical AC on Bool(P), one can built a sequence of EPTs which are guaranteed to increase $w_\varnothing$. Consider for example the problem in Figure 1(a). This problem is a boolean binary WCSP also defined as a Max-SAT problem with clauses $\bar{x}; x \vee \bar{y}; x \vee z; y \vee \bar{z}$. It is EDAC. Note that Bool(P) is represented by the same figure if one assumes that $m = 1$ (1 represents "forbidden").

In a first phase, we enforce arc consistency on Bool(P). The result is depicted in Figure 1(b): since the value (x, t) is forbidden, values (y, t) and (z, f) have no support on x and can be deleted (marked with cost 1). For each deletion, we remember the "source" of the deletion by a grey arrow pointing to the variable that offered no support. Value (z, t) can then be deleted thanks to the deletion of (y, t) and variable z wipes-out. Since the WCSP has integer costs, we can deduce a lower bound of 1 for the original problem P.

A second phase retraces the steps of the first phase, starting from the wiped-out variable. Assume that an unknown quantum of cost λ can be moved to $w_\emptyset$ from the variable z that wiped-out. This would require costs of at least λ at each value of variable z to project on $w_\varnothing$. Following the arrows, we know these costs can be obtained by projection from the cost functions w_{yz} and w_{xz} respectively. The costs $w_{yz}(f, t)$ and $w_{xz}(f, f)$ being non-zero in P, we will be able to take the costs here and there is no need to trace them back further. The other required costs must be obtained from w_y and w_x and extended. A weight of λ has to be traced back further via w_{xy} to w_x. This process halts when all the required weights are non-zero in the original problem P. We are now able to count the number of requests for a cost of λ on every non-zero cost. These counts are shown in italic in Figure 1(c). Here the maximum number of requests is reached on $w_x(t)$ with 2 requests. Since $w_x(t) = 1$ in P, the maximum amount we can assign to λ is therefore $\frac{1}{2}$.

In a third phase, we apply all the traced-back arc EPTs in reverse order to the original WCSP P with $\lambda = \frac{1}{2}$. The process is illustrated in Figures 1(d) to 1(g) where extended and projected costs are shown in bold. The result is an equivalent WCSP with $w_\emptyset = \frac{1}{2}$. Being equivalent to the original P, this process can be repeated. Here, no wipe-out occurs in the new Bool(P), the problem is VAC. Because the original problem has integer costs, we can infer a lower bound of 1.

The following theorem shows that if establishing arc consistency in Bool(P) produces an inconsistency, then it is always possible to increase $w_\emptyset$ by a sequence of soft arc consistency operations (and conversely).

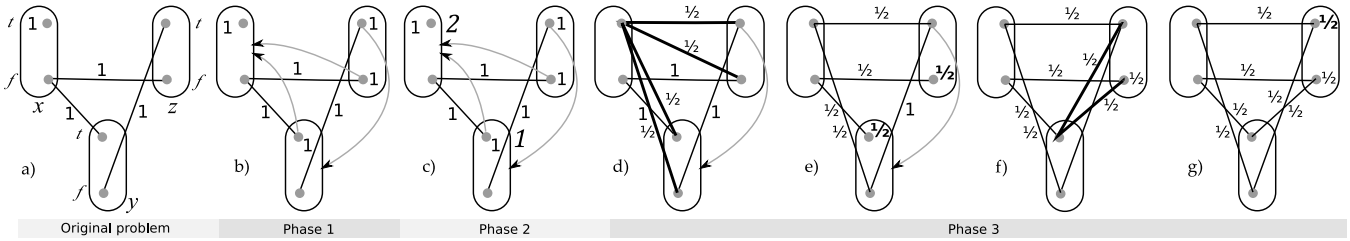

Figure 1: A WCSP where VAC enforcing produces a better lower bound than EDAC.

Theorem 1 *Let P be a WCSP. There exists a sequence of arc EPTs which when applied to P leads to an increase in $w_\emptyset$ iff the arc consistency closure of Bool(P) is inconsistent.*

Proof: $\Rightarrow$: Let $O_1, \ldots, O_t$ be a sequence of arc EPTs in P which produces an equivalent WCSP with an increased $w_\emptyset$. Let $O'_1, \ldots, O'_t$ be the corresponding arc EPTs with the weight being projected or extended always equal to 1 and assuming $m = 1$. Applying this sequence of operations to Bool(P) (a WCSP with $m = 1$) inevitably leads to a domain wipe-out and hence inconsistency.

$\Leftarrow$: Let $O_1, \ldots, O_t$ be a sequence of arc EPTs in Bool(P) which leads to a domain wipe-out. We can assume without loss of generality that no two of the operations O_i, O_j are strictly identical, since the same arc EPTs never needs to be applied twice in a classical CSP. Each O_i corresponds to a Project or UnaryProject operation when Bool(P) is viewed as a WCSP with $m = 1$. Let O'_i be the corresponding soft EPT in P applied with a weight δ/e^i, where δ is the minimum non zero weight occurring in P. We divide by $e = |W|$ each time, since a weight may need to be divided into smaller quantities to be extended to all constraints involving the same variable (or projected to all variables in the same constraint scope). After applying $O'_1, \ldots, O'_t$ to P we necessarily have an increase of $w_\emptyset$ larger than $\delta/e^t > 0$. □

VAC is easily shown to be stronger than EAC and can solve submodular problems, a non trivial polynomial language of WCSP (Cohen *et al.* 2006). Assuming an arbitrary ordering on every domain, a cost function w_S is submodular iff $\forall t, t' \in \ell(S), w(\max(t, t')) + w(\min(t, t')) \leq w(t) + w(t')$ where $\max$ and $\min$ represent point-wise applications of $\max$ (resp. $\min$) on the values of t, t'. This class includes functions such as $\sqrt{x^2 + y^2}$ or $(x \geq y?(x-y)^r : m)$ with $(r \geq 1)$, useful in bioinformatics (Zytnicki, Gaspin, & Schiex 2006) and capturing simple temporal CSP with linear preferences (Khatib *et al.* 2001).

Theorem 2 *Let P be a WCSP whose cost functions are all submodular and which is VAC, then an optimal solution can be found in polynomial time and has cost $w_\varnothing$.*

Sketch of proof: in Bool(P), the cost function submodularity becomes $c(t) \wedge c(t') \Rightarrow c(\max(t, t')) \wedge c(\min(t, t'))$ which means that all the relations of Bool(P) are both min and max-closed. Given that the original WCSP is VAC, Bool(P) is arc consistent and max-closed and therefore completely solved (Jeavons & Cooper 1995). A solution can be found in polynomial time by taking maximum values. Its cost in the original WCSP is $w_\varnothing$ and therefore optimal. □

Because Project and UnaryProject preserve submodularity (Cooper 2008), enforcing VAC on a submodular problem solves the problem. Since the domain order has no influence on VAC, it can solve arbitrarily permutated submodular problems (Schlesinger 2007) without domain reordering.

Enforcing VAC

In this section we restrict ourselves to binary WCSP, but VAC enforcing can be applied to problems of arbitrary arities using GAC instead of AC (the killer structures would be a constraint scope in this case). As our previous example shown, VAC enforcing is a three-phase process. The first phase consists in applying an instrumented classical AC algorithm on Bool(P), denoted as Instrumented-AC and not described here because of its simplicity. If no wipe-out occurs, the problem is already VAC and 0 is returned. Otherwise, it should return the wiped-out variable. The instrumentation should collect two types of information: for every value (i, a) deleted by lack of support on a constraint c_{ij}, the data structure killer(i, a) must contain the variable j. Furthermore, the value (i, a) itself must be pushed in a queue P. These two data structures have a space complexity of $O(ed)$ and $O(nd)$ respectively which do not modify the time and space complexity of optimal AC algorithms.

The second phase is described in Algorithm 2. It exploits the queue P and the killer data structure to rewind the propagation history and collect an inclusion-minimal subset of value deletions that is sufficient to explain the domain wipe-out observed. For this a boolean $M(i, a)$ is set to true whenever the deletion of (i, a) is needed to explain the wipe-out. This phase also computes the quantum of cost λ that we will ultimately add to $w_\varnothing$. Using the previous killer structure, it is always possible to trace back the cause of deletions until a non zero cost is reached: this will be the source where the cost of λ must be taken. However, in classical CSP, the same forbidden pair or value may serve multiple times. In order to compute the value of λ, we must know how many quanta of costs are requested for each solicited source of cost in the original WCSP, at the unary or binary level. For a tuple (a pair or a value) t_S of scope S, such that $w_S(t_S) \neq 0$, we use an integer $k(t_S)$ to store the number of requests of the quantum λ on $w_S(t_S)$. Using the queue P guarantees that the deleted values are explored in anti-causal order: a deleted value is always explored before any of the deletions that caused its deletion. Thus, when the cost request for a

given tuple is computed, it is based on already computed counts and it is correct. Ultimately, we will be able to compute λ as the minimum of $\frac{w_S(t_S)}{k(t_S)}$ for all t_S s.t. $k(t_S) \neq 0$.

Initially, all k are equal to 0 except at the variable i_0 that has been wiped-out where one quantum is needed for each value (line 1). A value (i, a) extracted from P (line 2) has been deleted by lack of support on the variable $\mathsf{killer}(i, a) = j$. If cost is needed at (i, a) (line 3), this lack of support can be due to the fact that:

1. the pair (a, b) is forbidden by c_{ij} in Bool(P) which means that $w_{ij}(a, b) \neq 0$ (line 5). The traceback can stop, the number of quanta requested on pair (a, b) (line 6) and λ (line 7) are updated accordingly.
2. otherwise, value (j, b) was deleted and $k((i, a))$ quanta of costs are needed from it. Note that if different values of i request different amounts of quanta from (j, b), just the *maximum* amount is needed since one extension from (i, a) to w_{ij} provides cost to all $w_{ij}(a, b)$. To maintain this maximum, we use another data structure, $k_i((j, b))$ to store the number quanta requested by i on (j, b). We therefore have $k((i, a)) = \sum k_j((i, a))$. Here, if the new request is higher than the known request (line 8), $k_i((j, b))$ (line 9) and $k((j, b))$ (line 10) must be increased accordingly. If there is no unary cost $w_j(b)$ explaining the deletion, this means that (j, b) has been deleted by AC enforcing and we need to traceback the deletion of (j, b) inductively (line 11). Otherwise, the traceback can stop at (j, b) and λ is updated (line 12).

The last phase is described in Algorithm 3 and actually modifies the original WCSP by applying the sequence of EPTs identified in the previous phase in the reverse order, thanks to the queue R. As Theorem 1 shows, the new WCSP will have an improved $w_\varnothing$.

Algorithm 2: VAC - Phase 2: Computing λ

```
Initialize all k, k_j to 0, λ ← m ;
i_0 ← Instrumented-AC() ;
if (i_0 = 0) then return;
foreach a ∈ D_{i_0} do
1   k((i,a)) ← 1, M(i,a) ← true;
    if (w_i(a) ≠ 0) then M(i,a) ← false, λ ← min(λ, w_i(a)) ;
while (P ≠ ∅) do
2   (i,a) ← P.Pop() ;
3   if (M(i,a)) then
        j ← killer(i,a); R.Push(i,a) ;
4       foreach b ∈ D_j do
5           if (w_ij(a,b) ≠ 0) then
6               k((i,a),(j,b)) ← k((i,a),(j,b)) + k((i,a)) ;
7               λ ← min(λ, w_ij(a,b) / k((i,a),(j,b))) ;
8           else if (k((i,a)) > k_i((j,b))) then
9               k_i((j,b)) ← k((i,a)) ;
10              k((j,b)) ← k((j,b)) + k((i,a)) − k_i((j,b)) ;
11              if (w_j(b) = 0) then M(j,b) ← true ;
12              else λ ← min(λ, w_j(b) / k((j,b))) ;
```

Algorithm 3: VAC - Phase 3: Applying EPTs

```
while (R ≠ ∅) do
    (j,b) ← R.Pop() ;
    i ← killer(j,b) ;
    foreach a ∈ D_i s.t. k_j((i,a)) ≠ 0 do
        Project(w_ij, i, a, −λ × k_j((i,a)));
        k_j((i,a)) ← 0 ;
    Project(w_ij, j, b, λ × k((j,b))) ;
UnaryProject(i_0, λ) ;
```

Because of the $k((i, a), (j, b))$ data structure, the algorithm has a $\mathcal{O}(ed^2)$ space complexity. It is possible to get rid of these binary counters by observing that quanta requests on $w_{ij}(a, b)$ can come only from i or from j. $k((i, a))$ quanta are requested by i if $\mathsf{killer}(i, a) = j$ and $M(i, a)$ is true, and symmetrically for (j, b). Thus, $k((i, a), (j, b))$ can be computed in constant time from the killer, M and unary k data structures. This yields a $\mathcal{O}(ed)$ space complexity.

One iteration of the algorithm has a time complexity of $O(ed^2)$. This is true for the first phase as long as an optimal AC algorithm is used since the instrumentation itself is $O(nd)$. The 2nd phase is $O(nd^2)$ since there are at most nd values in P and the loop at line 4 takes $O(d)$. Using the $O(ed)$ space trick, the same $O(nd^2)$ complexity applies to the last phase. Because λ may become smaller and smaller at each iteration, the number of iterations could be unbounded. To implement VAC, we used a threshold ε. If more than a given number of iterations never improve $w_\varnothing$ by more than ε then VAC enforcing stops prematurely. This is called VAC$_\varepsilon$. The number of iterations is then $O(\frac{m}{\varepsilon})$. When one iteration does not increase the lower bound by more than ε, one bottleneck (that fixed the value of λ) is identified and the unary and binary costs associated to a related variable are ignored in Bool(P) at following iterations.

In order to rapidly collect large cost contributions, we replaced Bool(P) by a relaxed but increasingly strict variant Bool$_\theta(P)$. A tuple t is forbidden in Bool$_\theta(P)$ iff its cost in P is larger than θ. After sorting the list of non zero binary costs $w_{ij}(a, b)$ in a fixed number k of buckets, the decreasing minimum cost observed in each bucket define a sequence of thresholds $(\theta_1, \ldots, \theta_k)$. Starting from θ_1, iterations are performed at a fixed threshold until no wipe-out occurs. Then the next value θ_{i+1} is used. After θ_k, a geometric schedule defined by $\theta_{i+1} = \frac{\theta_i}{2}$ is used and stopped when $\theta_i \leq \varepsilon$.

Experimental results

In this section we present experimental results of VAC$_\varepsilon$ on randomly generated and real problems using `toulbar2`[1]. Our implementation uses fixed point representation of costs. To achieve this, all initial costs in the problem are multiplied by $\frac{1}{\varepsilon}$ assumed to be integer. To exploit the knowledge that the original problem solved is integer, the branch and bound pruning occurs as soon as $\frac{\lceil w_\varnothing \times \varepsilon \rceil}{\varepsilon} \geq ub$ where $ub = m$ is the global upper bound (best known solution cost).

[1] http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro.

The experiments were performed on a 3 GHz Intel Xeon with 16 GB. Our solver includes a certain number of features including a last conflict driven variable selection heuristic, variable elimination during search and dichotomic branching. When VAC_ε is used we have fixed its default setting to $\varepsilon = \frac{1}{10000}$. It is also possible to maintain VAC_ε during search. Because of the overhead of each iteration of VAC_ε, which implies a reconstruction of $\text{Bool}_\theta(P)$, the convergence of VAC_ε is stopped prematurely during search, using a final θ larger than during preprocessing. This enforces VAC only when it is capable of providing large improvements in the lower bound. No initial upper bound is used on the random instances.

Randomly generated instances The first set of instances are random Max-CSP. We used the problems generated in (Cooper, de Givry, & Schiex 2007). These are Sparse Tight, Dense Tight, Complete Tight (ST, DT, CT with 32 variables, 10 values, 50 instances per class) where VAC_ε and OSAC preprocessing yield non trivial lower bounds. The following table shows the time and the quality of the lower bound (lb) after preprocessing by EDAC, VAC_ε and OSAC:

preprocessing	ST		DT		CT	
	lb	time	lb	time	lb	time
EDAC	16	<.01s	18	<.01s	40	<.01s
VAC_ε	25	.06s	28	.09s	49	.25s
OSAC	27	10.5s	32	2.1s	74	631s

As expected OSAC always provides the strongest lb. VAC_ε computes a lb which is 8% (ST) to 33% (DT) weaker than OSAC and is one to two orders of magnitude faster.

To evaluate the efficiency of VAC_ε on submodular problems, we generated random permutated submodular problems. At the unary level, every value receives a $0/1$ cost with identical probability. Binary submodular constraints can be decomposed in a sum of so-called *generalized interval functions* (Cohen *et al.* 2004). Such a GI function is defined by a fixed cost (we used 1) and one value a (resp. b) in each involved variable. We summed together d such GI functions using random values a and b uniformly sampled from the domains to generate each submodular binary cost function. The domains of all variables were then randomly permutated to "hide" submodularity.

The following table shows that maintaining VAC_ε allows to rapidly outperforms EDAC on these problems. Problems have from 100 to 150 variables, 20 values, from 900 to 1450 constraints, and 10 instances per class. The CPU time used for solving these problems, including the proof of optimality, is reported below (a dash for $> 10^4$ seconds):

n vars:	100	110	120	130	140	150
EDAC	17	85	135	433	1363	-
VAC_ε	0.31	0.34	0.37	0.36	0.77	1.17

Radio Link Frequency Assignment Problem (RLFAP) (Cabon *et al.* 1999) consists in assigning frequencies to a set of radio links in such a way that all the links may operate together without noticeable interference. Some RLFAP instances can be naturally cast as binary WCSPs.

We first compared VAC_ε to OSAC and EDAC for preprocessing only. RLFAP instances are distributed either in their original formulation or preprocessed by a combination of dominance analysis and singleton AC. A subindex $_r$ in the name of instances below identifies the reduced preprocessed instances (equivalent to the original ones).

The table below shows that VAC_ε can be one to two orders of magnitude faster than OSAC and gives almost the same lb on the $\texttt{graph11}_r$ and $\texttt{graph13}_r$ instances.

preprocessing		$\texttt{scen07}_r$	$\texttt{scen08}_r$	$\texttt{graph11}_r$	$\texttt{graph13}_r$
lb	EDAC	10000	6	2710	8722
	VAC_ε	29498	35	2955	9798
	OSAC	31454	48	2957	9798
time	VAC_ε	211s	86s	3.5s	29s
	OSAC	3530s	6718s	492s	6254s

We then tried to solve the same problems by maintaining simultaneously EDAC (useful for variable ordering heuristics) and VAC_ε. During search, VAC_ε was stopped at $\theta = 10/\varepsilon$. The table below reports the results on the open instances `graph11` and `graph13` (see fap.zib.de/problems/CALMA) which are solved to optimality for the first time both in their reduced and original formulation, given the best known upper bound and on the `scen06`. The table gives for each problem the number of variables, total number of values, number of cost functions, cpu time for EDAC alone, number of developed nodes with VAC, cpu-time with VAC, mean increase of the lower bound observed after one VAC iteration (lb/iter) and total number of VAC iterations (niter). We observed that the value k (number of cost requests) at each VAC iteration can be high, reaching a mean value of 16 in some resolutions of `graph` instances.

	nb. var.	nb. val	nb. c.f	EDAC cpu	VAC nodes	VAC cpu	lb/ iter	nb. iter
$\texttt{gr11}_r$	232	5747	792	-	1536	18.2s	2.5	973
gr11	340	12820	1425	-	$2 \cdot 10^5$	217min.	6.63	$2.6 \cdot 10^5$
$\texttt{gr13}_r$	454	13153	2314	-	32	62s	4.8	1893
gr13	458	17588	4815	-	114	254s	0.4s	9486
sc06	82	3274	327	39min.	$2 \cdot 10^6$	155min.	96	$3 \cdot 10^6$

In the **uncapacitated warehouse problem** (UWLP, problem description and WCSP model in (Kratica *et al.* 2001) and (de Givry *et al.* 2005) respectively). We tested VAC_ε preprocessing on instances capmq1-5 (600 variables, 300 values max. per variable and 90000 cost functions) and instances capa, capb and capc (1100 variables, around 90 values per variable and 101100 cost functions). We report solving time in seconds in the following table:

	mq1	mq2	mq3	mq4	mq5	a	b	c
EDAC	2508	3050	2953	7052	7323	6179	-	-
VAC_ε	2279	3312	2883	4024	8124	3243	4343	2751
CPLEX_ε	622	1022	415	1266	2357	3	4.5	13

Instances where also solved using the ILP solver CPLEX 11.0 and a direct formulation of the problem. Note that given the floating point representation of CPLEX and the large range of costs in these problems, the proof of optimality of CPLEX is questionable here. OSAC results are not given because LP generation overflows on these instances.

Related works

As OSAC (Cooper, de Givry, & Schiex 2007), VAC aims at finding an arc level EPT reformulation of an original problem with an optimized constant cost. OSAC identifies a *set* of arc EPTs which applied simultaneously yield an optimal reformulation. VAC just finds *sequences* of arc EPTs which applied sequentially, improve this constant cost.

The idea of using classical local consistency to build lower bounds in WCSP or Max-SAT is not new. On Max-CSP, (Régin *et al.* 2001) used independent arc inconsistent subproblems to build a lower bound. Similarly for Max-SAT, (Li, Manyà, & Planes 2005) used minimal Unit Propagation inconsistent subproblems to build a lower bound. These approaches do not use EPTs but rely on the fact that the inconsistent subproblems identified are independent and costs can be simply summed up. But they lack the incrementality of local consistencies.

In Max-SAT again, (Heras, Larrosa, & Oliveras 2007) also used Unit Propagation inconsistency to build sequences of *integer* EPTs but possibly strictly above the arc level, generating higher arity weighted clauses (cost functions). Our approach remains at the arc level by allowing for rational costs. It can be seen as a non trivial extension (capable of dealing with non boolean domains and non binary constraints) of the roof-dual algorithm based on a max-flow algorithm used in quadratic pseudo-boolean optimization (Boros & Hammer 2002). It is similar to the "Augmenting DAG" algorithm independently proposed by (Schlesinger 1976) for preprocessing 2-dimensional grammars, recently reviewed in (Werner 2007).

Conclusion

This paper shows how classical arc consistency can be used to identify a sequence of arc EPTs that can improve the constant cost function. By using rational weights, Virtual Arc Consistency is capable of providing improved lower bounds that rely only on arc EPTs and therefore provide the incrementality beneficial for maintenance in a tree search.

Our algorithm for enforcing Virtual AC is still preliminary. Only part of the work that is performed at each iteration is useful for further iterations. Since Bool(P) is relaxed after every iteration, dynamic AC algorithms could probably be very useful here. Ordering heuristics inside the instrumented AC algorithm should also be studied since the cost increase obtained at each iteration of VAC depends on the proof of inconsistency of Bool(P) built by AC. Although fundamentally different from a flow algorithm, VAC could perhaps take advantage of the improvements obtained in efficient max-flow algorithms. Finally, we have presented our algorithm in the framework of weighted binary CSP, but our approach is also applicable to non-binary cost functions and to other valuation structures opening the door to other domains such as Max-SAT and MPE in Bayesian Networks. Instrumenting existing hard global constraints to enforce VAC is another attractive direction of research.

Acknowledgments We would like to thank Tomas Werner for discussion on the closely related Augmenting DAG algorithm. This research has been partly funded by the Agence Nationale de la Recherche (STALDECOPT project).

References

Boros, E., and Hammer, P. 2002. Pseudo-Boolean Optimization. *Discrete Appl. Math.* 123:155–225.

Cabon, B.; de Givry, S.; Lobjois, L.; Schiex, T.; and Warners, J. 1999. Radio link frequency assignment. *Constraints* 4:79–89.

Cohen, D.; Cooper, M.; Jeavons, P.; and Krokhin, A. 2004. A Maximal Tractable Class of Soft Constraints. *Journal of Artificial Intelligence Research* 22:1–22.

Cohen, D.; Cooper, M.; Jeavons, P.; and Krokhin, A. 2006. The complexity of soft constraint satisfaction. *Artificial Intelligence* 170(11):983 – 1016.

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

Cooper, M.; de Givry, S.; and Schiex, T. 2007. Optimal soft arc consistency. In *Proc. of IJCAI-07*, 68–73.

Cooper, M. 2008. Minimization of locally-defined submodular functions by Optimal Soft Arc Consistency. *Constraints* 13(4).

de Givry, S.; Zytnicki, M.; Heras, F.; and Larrosa, J. 2005. Existential arc consistency: Getting closer to full arc consistency in weighted csps. In *Proc. of IJCAI-05*, 84–89.

Heras, F.; Larrosa, J.; and Oliveras, A. 2007. MiniMaxSat: A New Weighted Max-SAT Solver. In *Proc. of SAT'2007*, number 4501 in LNCS, 41–55.

Jeavons, P., and Cooper, M. 1995. Tractable constraints on ordered domains. *Artificial Intelligence* 79(2):327–339.

Khatib, L.; Morris, P.; Morris, R.; and Rossi, F. 2001. Temporal constraint reasoning with preferences. In *Proc. of the 17th IJCAI*, 322–327.

Kratica, J.; Tosic, D.; Filipovic, V.; and Ljubic, I. 2001. Solving the Simple Plant Location Problems by Genetic Algorithm. *RAIRO Operations Research* 35:127–142.

Li, C.-M.; Manyà, F.; and Planes, J. 2005. Exploiting Unit Propagation to Compute Lower Bounds in Branch and Bound Max-SAT Solvers. In *Proc of CP-05*, number 3709 in LNCS, 403–414.

Régin, J.-C.; Petit, T.; Bessière, C.; and Puget, J.-F. 2001. New Lower Bounds of Constraint Violations for Over-Constrained Problems. In *Proc. of CP-01*, number 2239 in LNCS, 332–345.

Schiex, T.; Fargier, H.; and Verfaillie, G. 1995. Valued constraint satisfaction problems: hard and easy problems. In *Proc. of IJCAI-95*, 631–637.

Schiex, T. 2000. Arc consistency for soft constraints. In *Proc. of CP-00*, volume 1894 of *LNCS*, 411–424.

Schlesinger, M. 1976. Sintaksicheskiy analiz dvumernykh zritelnikh signalov v usloviyakh pomekh (Syntactic analysis of two-dimensional visual signals in noisy conditions). *Kibernetika* 4:113–130.

Schlesinger, D. 2007. Exact Solution of Permuted Submodular MinSum Problems. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, number 4679/2007 in LNCS, 28–38.

Werner, T. 2007. A Linear Programming Approach to Max-sum Problem: A Review. *IEEE Trans. on Pattern Recognition and Machine Intelligence* 29(7):1165–1179.

Zytnicki, M.; Gaspin, C.; and Schiex, T. 2006. A new local consistency for weighted CSP dedicated to long domains. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, 394–398.