Noname manuscript No.
(will be inserted by the editor)

High order consistencies for Weighted CSPs

**Hiep Nguyen · Thomas Schiex ·
Christian Bessiere · Simon de Givry**

Received: date / Accepted: date

Abstract Keywords Weighted CSP · Cost Function Networks · high order consistencies consistency · path inverse consistencies · max-restricted path consistencies

1 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 [17], 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 can be 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, Depth First 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 on Cost Function Networks (CFNs). The most simplest are arc consistencies such as AC*, DAC*, FDAC* or EDAC* [13], inspired from the arc consistency in hard constraint networks. They have a slight enforcing time but do not provide large lower-bounds as expected.

F. Author
first address
Tel.: +123-45-678910
Fax: +123-45-678910
E-mail: fauthor@example.com

S. Author
second address

Beyond arc consistencies, up to now, only few strong consistencies, have been proposed for WCSPs such as complete k-consistency [6] and tuple Consistency [9] .... In this paper, we will show that strong soft consistencies can be defined for WCSPs by extending hard high order consistencies used for CSPs such as RPC, PIC, maxRPC, NIC, SAC, k-inverse consistency,... Among them, the group of triangle-based consistencies consisting of RPC, PIC, maxRPC are most interested because they have a pruning power strong enough and their computational cost is much cheaper than the others by only dealing with subnetworks of three variables.

By exploiting the combined costs involving unary, binary and ternary terms inside triangles of variables, the new strong consistencies allow to improve the lower bound compared to soft arc consistencies. As a result, they provide significant speedups on some specific problems.

2 Background

2.1 CSPs

A constraint satisfaction problem (CSP) is a triple (X, D, C) where X is a set of n variables, D is a set of n domains ($D_i \in D$ for variable $i \in X$), and C is a set constraints. Each constraint $c_S \in C$ defined over a set S of variables specifies the authorized assignments τ of values for variables in S, denoted by $\tau \in C$. S and $|S|$ are the scope and the arity of the constraint. For simplicity, $c_{\{i\}}$, $c_{\{i,j\}}$ are replaced by c_i, c_{ij}. The constraints c_i, c_{ij}, c_S with $|S| > 2$ are respectively called unary, binary and non-binary. Given a set of variables S, $\ell(S)$ denotes the set of assignments (tuples) of values for variables in S. Given a variable $i \in S$ and a subset $S' \in S$, $\tau[i]$ and $\tau[S']$ respectively denote the projection of tuple τ on i and S. A tuple τ is consistent if it satisfies all the constraints whose scope is included in the scope of τ. A solution is a consistent complete assignment. The problem is consistent if it has at least one solution.

Definition 1 (Local consistencies) Given a CSP $P = (X, D, C)$.

- P is arc consistent (AC) if $\forall i \in X$, $\forall a \in D_i$, $\forall c_S \in C$ such that $i \in S$, there exists a tuple $\tau \in \ell(S)$ such that $\tau[i] = a$ and $\tau \in c_S$. Such a tuple τ is called the support of value (i, a) in the constraint c_S.
- P is restricted path consistent (RPC, [4]) iff it is AC and $\forall i \in X$, $\forall a \in D_i$, $\forall c_{ij} \in C$ on which a has only one support $b \in D_j$, $\forall k$ linked to i and j by c_{ik}, c_{jk}, there exists a value $c \in D_k$ such that $(a, c) \in c_{ik}$ and $(b, c) \in c_{jk}$.
- P is path inverse consistent (PIC, [11]) iff it is AC and $\forall i \in X$, $\forall a \in D_i$, $\forall j, k$ such that i, j, k are linked one-by-one by binary constraint, there exists a value $b \in D_j, c \in D_k$ such that $(a, b) \in c_{ij}, (a, c) \in c_{ik}$ and $(b, c) \in c_{jk}$.
- P is max-restricted path consistent (maxRPC, [8]) iff it is AC and $\forall i \in X$, $\forall a \in D_i$, $\forall c_{ij} \in C$, a has a support $b \in D_j$ such that $\forall k$ linked to both i, j by c_{ik}, c_{jk}, there exists a value $c \in D_k$ such that $(a, c) \in c_{ik}$ and $(b, c) \in c_{jk}$.
- Let Φ be a hard consistency Φ and CSP P. Φ–closure of P is a CSP which is equivalent to P (has the same set of solutions as P) and satisfies Φ

2.2 Weighted CSPs

Weighted CSPs (WCSPs, [16]) extends CSPs by associating costs to tuples. A WCSP is a tuple (X, D, C, m) where X and D are respectively the sets of variables and domains as in classical CSPs. C is a set of cost functions. Each cost function $c_S \in C$ assigns costs to tuples $\tau \in \ell(S)$ i.e. $c_S : \ell(S) \to [0..m]$ where $m \in \{1, \dots, +\infty\}$. The addition and subtraction of costs are bounded operations, defined as $a +_m b = \min(a + b, m)$, $a -_m b = a - b$ if $a < m$ and m otherwise. The combined cost of a tuple τ over scope W in a WCSP P is the sum of costs $Val_P(\tau) = \bigoplus_{S \in W} c_S(\tau[S])$, where $\oplus$ is $+_m$. τ is inconsistent if $Val_P(\tau) = m$, and consistent otherwise. When $Val_P(\tau) = 0$, τ is completely consistent. The solution of P is a complete tuple τ with a minimum $Val_P(\tau)$.

We assume the existence of a unary cost function c_i for every variable i, and a nullary cost function, noted $c_\varnothing$. This constant positive cost defines a lower bound on the cost of every solution. A WCSP P can be transformed into an equivalent problem P' ($Val_P(t) = Val_{P'}(t)\ \forall t$) with a possibly increased lower bound $c_\varnothing$ on the optimal cost by using so-called equivalence-preserving transformations (EPTs) which shift costs between cost functions. The EPTs `Shift`$(\tau_S, c_{S'}, \alpha)$ (Algorithm 1) moves an amount of cost α between a cost function $c_{S'}$ and a tuple τ over S such that $S \subset S'$. The conditions (2) and (3) guarantee that the operation will not create any negative cost in the problem. `Shift` respectively implies 3 Soft Arc Consistency operations (SAC operations [7]): `Project` (from $c_{S'}$ to τ), `Extend` (from τ to $c_{S'}$) and `UnaryProject` (from i to $c_\varnothing$) when $\alpha > 0$, $\alpha < 0$ and $\alpha > 0$, $S' = \{i\}$, $|S| = 0$.

Algorithm 1: Operation for shifting costs in WCSPs

```
1 Procedure Shift($\tau_S, c_{S'}, \alpha$)
      // condition: (1)$S \subset S'$, (2)$c_S(\tau) \oplus \alpha \geq 0$, (3)$c_{S'}(\tau') \geq \alpha : \forall \tau' \in \ell(S'), \tau'[S] = \tau$
2     $c_S(\tau) \longleftarrow c_S(\tau) +_m \alpha$ ;
3     foreach $\tau' \in \ell(S'), \tau'[S] = \tau$ do
4         $c_{S'}(\tau') \longleftarrow c_{S'}(\tau') -_m \alpha$;
```

Soft consistencies are techniques that aim to strengthen the lower bound $c_\varnothing$. The simplest one, node consistency (NC [12]), requires that $\forall i \in X$, $\forall a \in D_i$ $c_i(a) + c_\varnothing < m$ and there exists a value $a \in D_i$ such that $c_i(a) = 0$. The arc consistencies presented below are inspired from hard arc consistency. For simplicity, we restrict ourselves to binary WCSPs. The definitions of soft arc consistencies in non-binary WCSPs are introduced in [7, **?**,**?**]

Definition 2 (Soft arc consistencies) Given a binary WCSP $P = (X, D, C, m)$ and an order $<$ of variables.

- P is arc consistent (AC) iff $\forall i \in X$, $\forall a \in D_i$ and $\forall c_{ij} \in C$, there exists $b \in D_j$ such that $c_{ij}(a, b) = 0$. b is called the support for (i, a) in c_{ij}.

- P is directional arc consistent (DAC) w.r.t $<$ iff $\forall i$, $\forall a \in D_i$, $\forall c_{ij}$ such that $i < j$, there exists a value $b \in D_j$ such that $c_{ij}(a,b) + c_j(b) = 0$. b is called the full support for (i,a) in c_{ij}.
- P is full directional arc consistent (FDAC) w.r.t $<$ if it is arc consistent and directional arc consistent w.r.t to $<$.
- P is existential arc consistent (EAC) iff $\forall i \in X$, there exists a value $a \in D_i$ such that $c_i(a) = 0$ and $\forall c_{ij} \in C$, there exists $b \in D_j$ such that $c_{ij}(a,b) + c_j(b) = 0$. Value a is called the existential arc consistent support of i.
- P is existential directional arc consistent (EDAC) w.r.t $<$ iff it is existential arc consistent and full directional arc consistent w.r.t $<$.
- Bool(P) is a CSP $(X, D, \overline{C}, 1)$ such that $\exists \overline{c}_S \in \overline{C}$ iff $\exists c_S \in C$, $S \neq \varnothing$ and $\overline{c}_S(\tau) = 1 \Leftrightarrow c_S(\tau) \neq 0$. P is virtual arc consistent (VAC) iff the AC-closure of Bool(P) is non-empty.

A binary WCSP is AC*, DAC*, FDAC*, EAC*, EDAC* if it is NC and respectively AC, DAC, FDAC, EAC, EDAC. The notions generalized arc consistencies (GACs) are used in the case of non-binary WCSPs.

3 Soft high order consistencies

From soft arc consistencies, we have generalized 6 soft variants, also called 6 "softness" levels, for each hard consistency: "simple" ("non-directional"), "directional", "full directional", "existential", "existential directional" and "virtual". In this section, we introduce the soft consistencies extended from hard RPC, PIC and maxRPC. In addition to soft ACs, they guarantee the extensibility of arc supports on extra third variables at a so-called "witnesses".

Definition 3 (Witnesses) Given a value (i,a), a pair of values (i_a, j_b) and a variable k linked both to i and j.
- A simple witness of (i_a, j_b) on k is a value $c \in D_k$ such that $c_{ik}(a,c) + c_{jk}(b,c) + c_{ijk}(a,b,c) = 0$.
- A full witness of (i_a, j_b) on k is a value $c \in D_k$ such that $c_k(c) + c_{ik}(a,c) + c_{jk}(b,c) + c_{ijk}(a,b,c) = 0$.

Definition 4 (Extensibility of a value on a triangle) A triangle is a triple of variables (i,j,k), noted Δ_{ijk}, that are linked one-by-one by binary cost function. Given a value (i,a) and a triangle (i,j,k).
- (i,a) is simply extensible on triangle (i,j,k) if there exists a simple arc support for (i,a) in c_{ij} which is simply extensible on k.
- (i,a) is fully extensible on triangle (i,j,k) if there exists a full arc support for (i,a) in c_{ij} which is fully extensible on k.

Definition 5 (Extensibility of a pair of values on a variable) Given a pair of values (i_a, j_b) and a variable k linked both to i and j.
- (i_a, j_b) is simply extensible on k if there exists a simple witness on k for it.
- (i_a, j_b) is fully extensible on k if there exists a full witness on k for it.

Definition 6 (Extendibility of a pair of values) For a pair of values (i_a, j_b) and an order $<$ on the variables. (i_a, j_b) is:
- simply extensible if it is simply extensible on $\forall k$ linked to both i and j.
- fully extensible if it is fully extensible on $\forall k$ linked to both i and j.
- directionally-fully extensible if it is fully extensible on $\forall k > i$, linked to both i and j.
- semi-fully extensible if it is simply extensible on every variable $k < i$ and is fully extensible on every $k > i$ such that k is linked both to i and j.

Notice that the full extensibility implies the semi-full extensibility. The semi-full extensibility implies the directional-full and simple extensibility. Conversely, both directional-full and simple extensibility do not imply any other extensibility. Consider the example in Figure 1 to better understand the different extensibility of pairs of values.

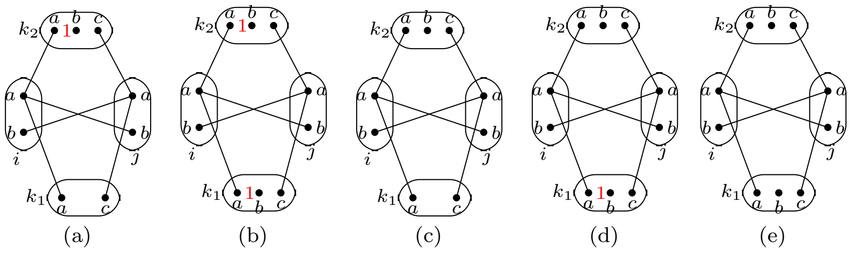


Fig. 1 Example of different extensibilities of the pair of values (i_a, j_a). $k_1 < i < j < k_2$. In WCSP(a), (i_a, j_a) is not simply extensible on k_1. In WCSP(b), (i_a, j_a) is simply extensible (on both k_1, k_2) but is not directionally-fully extensible (because it is not fully extensible on k_2). In WCSP(c), (i_a, j_a) is directionally-fully extensible w.r.t k_2 but is not semi-fully extensible (because it is not simply extensible on k_1). In WCSP(d), (i_a, j_a) is semi-fully extensible (fully extensible on k_1 and simply extensible on k_2) but is not fully extensible (because it is not fully extensible on k_1). In WCSP(e), (i_a, j_a) is fully extensible (on both k_1, k_2).

The idea of soft RPC consistencies is to only check the extensibility for pairs of values (i_a, j_b) which will make a value soft arc inconsistent if their binary cost increases. Indeed, such pairs of values are the unique supports for involving values in c_{ij}. If a value (i, a) has only one simple support (j, b) in c_{ij} and this support (i_a, j_b) is not extensible on some third variable k, every 3-values tuple over $\{i, j, k\}$, involving (i_a, j_b), has a positive combined cost. Because (j, b) is the unique arc support of (i, a), every complete tuple involving (i, a) has a positive cost evaluation. Thus, the unary cost $c_i(a)$ can increase by an equivalent of transformation.

Definition 7 (Soft restricted path consistencies (Soft RPCs)) Given a WCSP $P = (X, C, D, m)$ and an order "<" on the variables.

- P is RPC if it is AC and $\forall i \in X, \forall a \in D_i, \forall c_{ij} \in C$ on which (i, a) has only one simple arc support $b \in D_j$, (i_a, j_b) is simply extensible.

- P is directional RPC (DRPC) if it is DAC and $\forall i \in X, \forall a \in D_i, \forall c_{ij} \in C$ such that $i < j$ and (i,a) has only one full arc support $b \in D_j$, (i_a, j_b) is directionally-fully extensible.
- P is full directional RPC (FDRPC) if it is FDAC and $\forall i \in X, \forall a \in D_i, \forall c_{ij} \in C$ such that (1) if $i > j$ and (i,a) has only one simple arc support $b \in D_j$ then (i_a, j_b) is simply extensible, or (2) if $i < j$ and (i,a) has only one full arc support $b \in D_j$ then (i_a, j_b) is semi-fully extensible.
- P is existential RPC (ERPC) if $\forall i \in X$, there exists a value $a \in D_i$ such that (1) $c_i(a) = 0$, (2) i_a has a full arc support in every cost function and (3) $\forall c_{ij} \in C$ on which (i,a) has only one full arc support $b \in D_j$, (i_a, j_b) is fully extensible. Such a value (i,a) is the ERPC support for i.
- P is existential directional RPC (FDRPC) if it is ERPC and FDRPC.
- P is virtual RPC (VRPC) if the RPC-closure of Bool(P) is non-empty.

VRPC is defined based on the classical CSP Bool(P) and the hard RPC. The other consistencies of RPC are different each other by (1) the strength of supports (simple or full) (2) the strength of witnesses (simply, fully, directionally-fully, semi-fully extensible) and (3) the application area (every domain value or one value per domain, every or in some specific cost function).

Example 1 Consider WCSPs in Figure 1.

- WCSP(a) is VRPC because the RPC closure of Bool(P) is not empty, containing values $(i_b), (j,b), (k_1,a), (k_1,c), (k_2,a), (k_2,c)$. However, it is not RPC because the unique support (i_a, j_a) of (i,a) on c_{ij} is not simply extensible on k_1
- WCSP(b) is RPC: both (i_a, j_a) and (i_b, j_b) (respectively the unique simple arc support of $(i,a),(j,a)$ in c_{ij} and of $(i,b),(j_b)$ in c_{ij}) are simply extensible on k_1 and k_2 at simple witnesses (k_1, b) and (k_2, b) respectively. However, it is not DRPC because the unique full arc support (i_a, j_a) of (i,a) in c_{ij} is not fully extensible on $k_1 > i$.
- WCSP(c) is DRPC because both (i_a, j_a) and (i_b, j_b) (respectively the unique full arc support of (i,a) in c_{ij} and of (i,b) in c_{ij}) are fully extensible on $k_2 > i$ at (k_2, b). Variable $k_1 < i$ is not interested by DRPC for i. However, it is not FDRPC because the unique full support (i_a, j_a) of value (i,a) in c_{ij} is not simply extensible on k_1.
- WCSP(d) is FDRPC where the supports (i_a, j_a) and (i_b, j_b) are fully extensible on k_2 at (k_2, b) and simply extensible on k_1 at (k_1, b). At the same time, it is ERPC where (i,b), (j,b), (k_1,a), (k_2,a) are ERPC supports for variables i, j, k_1 and k_2.

The idea of soft path inverse consistencies is to guarantee the extensibility of domain values on triangles of variables. For all triangles (i,j,k) sharing two variables i,j of a cost function c_{ij}, PICs require that for each k, one of arc supports of (i,a) in c_{ij} is extensible on k. The arc supports of (i,a) that are extensible on different k can be different.

Definition 8 (Soft path inverse consistencies (Soft PICs)) Given a WCSP $P = (X, C, D, m)$ and an order "<" on the variables.

- P is PIC if it is AC and $\forall i \in X, \forall a \in D_i, \forall \Delta_{ijk}$, (i,a) is simply extensible on Δ_{ijk}.
- P is directional PIC (DPIC) if it is DAC and $\forall i \in X, \forall a \in D_i, \forall \Delta_{ijk}$ such that $i < j, i < k$, (i,a) is fully extensible on Δ_{ijk}.
- P is full directional PIC (FDPIC) if it is FDAC and $\forall i \in X, \forall a \in D_i, \forall$ triangle Δ_{ijk}, (i,a) is fully extensible on Δ_{ijk} if $i < j, i < k$ and simply extensible on Δ_{ijk} otherwise.
- P is existential PIC (EPIC) if $\forall i \in X$, there exists a value $a \in D_i$ such that (1) $c_i(a) = 0$, (2) i_a has a full arc support in every cost function and (3) (i,a) is fully extensible on every triangle.
- P is existential directional PIC (EDPIC) if it is EPIC and FDPIC.
- P is virtual PIC (VPIC) if the PIC-closure of Bool(P) is non-empty.

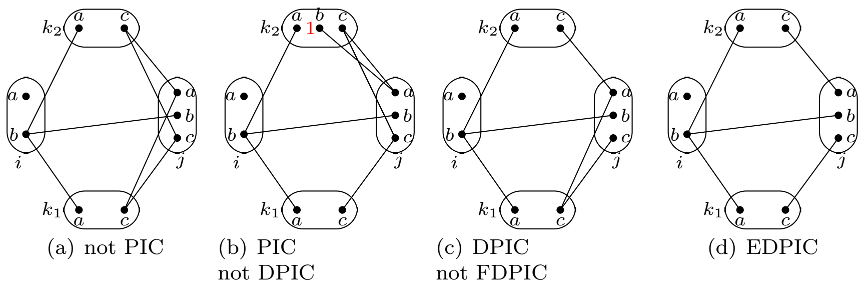
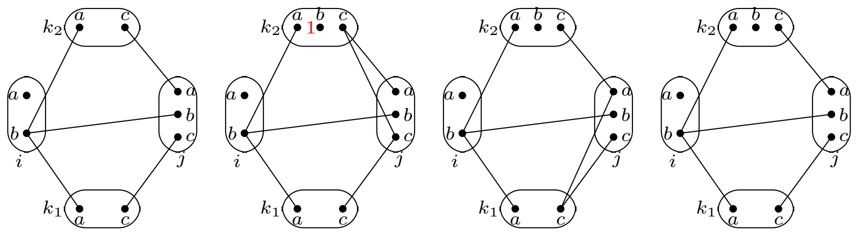


Fig. 2 Example of soft PIC consistencies. $k_1 < i < k_2 < j$ and $\exists$ $c_{ij}, c_{ik_1}, c_{ik_2}, c_{jk_1}, c_{jk_2}$. The WCSP(a) is not PIC because value (i,b) is not simply extensible to triangle (i,j,k_1). The WCSP(b) is PIC but is not DPIC because value (i,b) is not fully extensible to triangle (i,j,k_2) with $i < j, i < k_2$. The WCSP(c) is DPIC (because every value in D_i can be fully extended to (i,j,k_2) only which is interested by DPIC for i) but it is not FDPIC (because value (i,b) is not simply extensible to triangle (i,j,k_1)). The WCSP(d) is FDPIC where every variable is simply extensible to 2 triangles and i is fully extensible to (i,j,k_2). The WCSP(d) is also EPIC where $(i,a),(j,a),(k_1,a),(k_2,a)$ are respectively EPIC supports of i,j,k_1,k_2.

Stronger than PICs, soft max-restricted path consistencies (soft maxRPCs) check the existence of an extensible arc support for each value in each binary cost function whatever the number of arc supports the value has. In constrast to soft PICs, maxRPCs require the extensibility of the same arc support for each value in each binary cost function at the same time on all third variables. If value (i,a) has no such extensible arc support in some binary cost function c_{ij}, each support (i_a, j_b) of (i,a) in c_{ij} is not extensible in some extra variable k, i.e. the combined cost of every tuple (i_a, j_b, k_c) is positive. Thus, the binary cost of every arc support of (i,a) in c_{ij} can increase by an equivalence preserving transformation and then (i,a) will no longer be arc consistent.

Definition 9 (Soft max-restricted path consistencies (Soft maxRPCs)) Given a WCSP $P = (X,C,D,m)$ and an order "<" on the variables.

- P is maxRPC if it is AC and $\forall i \in X, \forall a \in D_i, \forall c_{ij} \in C$ there exists a simple arc support $b \in D_j$ such that (i_a, j_b) is simply extensible.
- P is directional maxRPC (DmaxRPC) if it is DAC and $\forall i \in X, \forall a \in D_i$, $\forall c_{ij} \in C$ such that $i < j$, there exists a full arc support $b \in D_j$ such that (i_a, j_b) is directionally-fully extensible.
- P is full directional maxRPC (FDRPC) if it is FDAC and for $\forall i \in D, \forall a \in D_i, \forall c_{ij} \in C$ (1) if $i > j$, there exists a simple arc support $b \in D_j$ such that (i_a, j_b) is simply extensible. (2) otherwise, if $i < j$, there exists a full arc support $b \in D_j$ such that (i_a, j_b) is semi-fully extensible.
- P is existential maxRPC (EmaxRPC) if $\forall i \in X$, there exists a value $a \in D_i$ such that (1) $c_i(a) = 0$, (2) i_a has a full arc support in every cost function and (3) $\forall c_{ij} \in C$, there exists a full arc support $b \in D_j$ such that (i_a, j_b) is fully extensible.
- P is existential directional maxRPC (EDmaxRPC) if it is EmaxRPC and FDmaxRPC.
- P is virtual maxRPC (VmaxRPC) if the maxRPC-closure of Bool(P) is non-empty



(a) EmaxRPC not maxRPC (b) maxRPC not DmaxRPC (c) DmaxRPC not FDmaxRPC (d) FDRPC EDmaxRPC

Fig. 3 Example of soft maxRPCs. $k_1 < i < k_2 < j$ and $\exists\ c_{ij}, c_{ik_1}, c_{ik_2}, c_{jk_1}, c_{jk_2}$. The WCSP(a) is not maxRPC because value (i,b) has no arc support in c_{ij} (between (i_b, j_a) and (i_b, j_c)) that is simply extensible on both k_1, k_2. The WCSP(b) is maxRPC but is not DmaxRPC because value (i,b) has no full arc support in c_{ij} (between (i_b, j_a) and (i_b, j_c)) that is fully extensible to k_2. The WCSP(c) is DmaxRPC (because every value in D_i has full arc support in c_{ij}, c_{ik_2} that is respectively fully extensible on k_2 and j. Triangle (i,j,k_1) is not interested by DmaxRPC for i). The WCSP(c) is not FDmaxRPC because value (i,b) has no full support in c_{ij} (between (i_b, j_a) and (i_b, j_c)) that is simply extensible on k_1. The WCSP(d) is both FDmaxRPC and EmaxRPC where $(i,a), (k_1,a), (j,a), (k_2,a)$ are respectively EPIC supports of variables i, k_1, k_2, j.

4 Comparison between soft domain consistencies

Enforcing soft consistencies aims to (1) increasing the lower bound $c_\varnothing$ (e,g,. VAC) or (2) moving costs from cost functions of higher arity to cost functions of lower arity. Thus, the power of virtual consistencies is evaluated by the quality of the lower bound and the power of soft domain consistencies (which define properties for values such as all ACs, RPCs, PICs and maxRPCs except for the virtual ones) is further evaluated by the capacity of increasing unary costs. A soft consistency A is called stronger than a soft consistency B if for every problem which already satisfies A, the weaker consistency B cannot improve it in terms of increasing unary costs and $c_\varnothing$. For a given WCSP P and a soft consistency A, let $c_\varnothing[P]$ denote the lower bound of P and $A(P)$ be the problem obtained after enforcing A in P.

Definition 10 (Strictly stronger relation) Given 2 soft consistencies A, B.
- A is stronger than B, noted by $A \geq B$, iff for every WCSP P that satisfies A, $B(P) = P$.
- A is stronger than B in terms of lower bound, noted by $A \geq_{c_\varnothing} B$, iff for every WCSP P that satisfies A, $c_\varnothing[B(P)] = c_\varnothing[P]$
- A is strictly stronger than B, noted $A > B$, iff $A \geq B$ and $\exists$ a WCSP P such that P satisfies B and $A(P) \neq P$
- A is strictly stronger than B in terms of lower bound, noted $A >_{c_\varnothing} B$, iff $A \geq_{c_\varnothing} B$ and $\exists$ a WCSP P such that P satisfies B and $c_\varnothing[A(P)] > c_\varnothing[P]$

Proposition 1 *Given 2 soft consistencies A and B. If $A \geq B$ then $A \geq_{c_\varnothing} B$.*

Proof The proof is trivial. Because $A \geq B$, $B(P) = P$ for every P that satisfies A. So we have $c_\varnothing[B(P)] = c_\varnothing[P]$ and thus $A \geq_{c_\varnothing} B$.

Similarly to the stronger and strictly stronger relations for hard consistencies, our relations for soft consistencies also have the transitive property.

Property 1 (Transitive) Given three soft consistencies A, B, C.
a. If $A \geq B$ and $B \geq C$ then $A \geq C$.
b. If $A > B$ and $B > C$ then $A > C$.
c. If $A > B$ and $B \geq_{c_\varnothing} C$ then $A \geq_{c_\varnothing} C$

Proof a. Let P be a WCSP that satisfies A. Because $A \geq B$ and P satisfies A, $B(P) = P$, i.e., P also satisfies B. Because $B \geq C$ and P satisfies B, $C(P) = P$. Thus, P satisfies A, $C(P) = P$, i.e., $A \geq C$.
b. (1) Because $>$ implies $\geq$, we have $A \geq B$ and $B \geq C$. So $A \geq C$ from the property (a). (2) Because $A > B$, there exists a WCSP P satisfying B and $A(P) \neq P$. Because P satisfies B and $B \geq C$, $C(P) = P$, i.e., P also satisfies C. Thus there exists P which is C and $A(P) \neq P$. So $A > C$.
c. Because $>$ implies $\geq$, we have $A \geq B$. Let P be a WCSP that satisfies A, P also satisfies B. Because $B \geq_{c_\varnothing} C$ and P satisfies B, $c_\varnothing[C(P)] = c_\varnothing[P]$. Thus, for every WCSP which satisfies A, $c_\varnothing[C(P)] = c_\varnothing[P]$. i.e., $A \geq_{c_\varnothing} C$.

To show that a soft consistency A is not stronger or not stronger in terms of lower bounds than B, it is enough to show that there exists a WCSP P in which A holds and B does better than A. Two consistencies A and B are incomparable iff A is not stronger than B and B is not stronger than A.

Definition 11 (Incomparable relation) Given 2 soft consistencies A, B.
- A and B are incomparable, noted $A \not\gtrless B$, iff $A \not\geq B$ and $B \not\geq A$

– A and B are incomparable in terms of lower bound, noted $A \not\gtrless_{c_\varnothing} B$, if $A \not\geq_{c_\varnothing} B$ and $A \not\leq_{c_\varnothing} B$

Graph 4 summarizes the relations among soft ACs, RPCs, PICs and maxRPCs. In a row of the graph are 6 soft consistencies associated with a same hard consistency and in a column are the ones at a same "softness" level. A directed path from a consistency A to B, without or with dashed arrow, respectively means that $A > B$ or $A >_{c_\varnothing} B$. If there does not exist any directed path between A and B, they are incomparable. First, we consider the relation between virtual consistencies and domain consistencies. Then, domain consistencies are considered according to the rows and the columns of the graph.

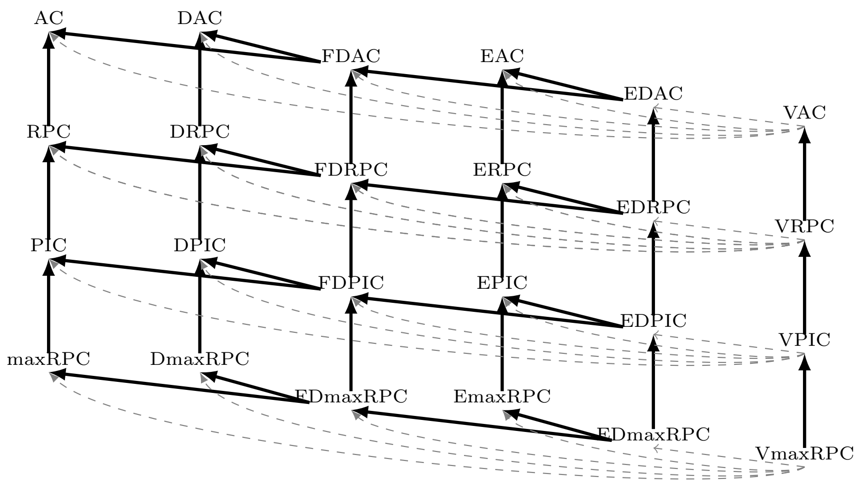


Fig. 4 Hasse diagram of relations between soft consistencies
$A \longrightarrow B : A > B$ $A \longrightarrow B \longrightarrow C$ implies $A \longrightarrow C$
$A \dashrightarrow B : A >_{c_\varnothing} B$ $A \longrightarrow B \dashrightarrow C$ implies $A \dashrightarrow C$

Theorem 1 *Given two hard consistencies $\overline{A}$, $\overline{B} \in \{AC, RPC, PIC, maxRPC\}$ and two virtual consistencies VA, VB respectively associated with $\overline{A}$, $\overline{B}$.*
a. VA $>_{c_\varnothing}$ A for every soft domain consistency A associated with $\overline{A}$.
b. If $\overline{A} > \overline{B}$ then
b1. VA > VB
b2. VA $>_{c_\varnothing}$ B for every soft consistency B ≠ VB associated with $\overline{B}$.

Proof a. We first prove that VA $\geq_{c_\varnothing}$ A by contradiction. Suppose that there exists a WCSP P satisfying VA and A still can increase $c_\varnothing$ of P from a variable $x_\varnothing$. All values and tuples whose costs have been necessary for increasing $c_\varnothing$ by A are also forbidden when enforcing $\overline{A}$ in the classic CSP Bool(P). So, if we eliminate these values and tuples in the same order that costs are moved by A in P , $x_\varnothing$ will be wiped-out in Bool(P). Thus P is not VA and the supposition is false. This means that VA $\geq_{c_\varnothing} A$. Secondly, Figure 12 shows a problem which satisfies every soft domain consistency of AC, RPC, PIC, maxRPC but does the virtual ones.

b1. Firstly, we prove that $VA \geq VB$. Let P be a WCSP which is VA. The $\overline{A}$ closure of $Bool(P)$ is not empty. Because $\overline{A} \geq \overline{B}$, the $\overline{B}$ closure of $Bool(P)$ will be not empty. Thus, P also satisfies VB, i.g., VB$(P) = P$. Now we prove that $VA > VB$, i.e., VmaxRPC > VPIC > VRPC > VAC. Figures 5, 6, 7 respectively show a WCSP which is VAC but not VRPC, VRPC but not VPIC, VPIC but not VmaxRPC and $c_\varnothing$ can be increased by 1 by the unsatisfied consistencies.

b2. We have VA > VB (Theorem 1(b1)) and $VB >_{c_\varnothing} B$ (Theorem 1(a)) that implies $VB \geq_{c_\varnothing} B$. From Property 1(c), $VA \geq_{c_\varnothing} B$. Now, we will prove that VA $>_{c_\varnothing} B$. Because $VB >_{c_\varnothing} B$, there exists a WCSP P such that P is B and VB can still increase the lower bound $c_\varnothing[P]$. This means that the $\overline{B}$−closure of $Bool(P)$ is empty. Because $\overline{A} > \overline{B}$, the $\overline{A}$−closure of $Bool(P)$ is also empty. Thus, $c_\varnothing[\text{VA}(P)] > c_\varnothing[P]$ while P satisfies B.

The following theorem will show that at each "softness" level, the soft maxRPC is strictly stronger than the soft PIC,the soft PIC is strictly stronger than the soft RPC,and the soft RPC is strictly stronger than the soft AC.

Theorem 2 (Vertical comparison)

a. maxRPC > PIC > RPC > AC.
b. DmaxRPC > DPIC > DRPC > DAC.
c. FDmaxRPC > FDPIC > FDRPC > FDAC.
d. EmaxRPC > EPIC > ERPC > EAC.
e. EDmaxRPC > EDPIC > EDRPC > EDAC.

Proof First, we have the stronger relation $\geq$ between consistencies by using their definition. At each softness level, the soft consistency of maxRPC implies PIC, PIC implies RPC and RPC implies AC. Second, we prove the strictly stronger relation between them by showing WCSPs in which the weaker consistencies hold while the stronger ones do not.

a. Figure 5 shows a WCSP which satisfies AC but does not satisfy RPC. Figure 6 shows a WCSP which satisfies RPC but does not satisfy PIC. Figure 7 show a WCSP which satisfies PIC but does not maxRPC. Thus maxRPC > PIC > RPC > AC.

b-e. The proof is similar to that for (a) by using Figure 5, 6 and 7.

The following theorem will show that associated with any hard consistency: (1) the existential directional consistency is strictly stronger than both the existential and the full directional ones, (2) the full directional consistency is strictly stronger than both the non-directional and the directional ones, (3) other pairs of consistencies are incomparable.

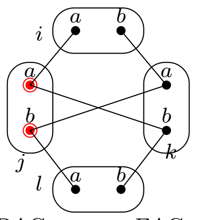
Theorem 3 (Horizontal comparison) *Given 2 hard consistencies $\overline{X}, \overline{Y} \in$ {AC, RPC, PIC, maxRPC}. Let X, DX, FDX, EX, EDX be the simple, directional, full directional, existential, existential directional consistency of $\overline{X}$; Y, DY, FDY be the simple, directional, full directional consistency of $\overline{Y}$.*

a. (column 2-1): $X \not\geq DY$
b. (column 3-1): $FDX > X, DX$
c. (column 4-1,2,3): $EX \not\geq Y, DY, FDY$
d. (column 5-3): $EDX > FDX, EX$

Proof a. $X \not\geq DY$: using Figures 8 and 9.
b. $FDX > X, DX$. The stronger relation $\geq$ is implied from the definition of the consistencies. $FDX > X$: Figure 8 shows a problem which is maxRPC, PIC, RPC, AC but is not FDmaxRPC, FDPIC, FDRPC, FDAC. $FDX > DX$: Figure 9 shows a problem which is DmaxRPC, DPIC, DRPC, DAC but is not FDmaxRPC, FDPIC, FDRPC, FDAC.
c. $EX \not\geq Y, DY, FDY$: using Figures 10 and 11.
d. $EDX > FDX$ and $EDX > EX$. The proof is trivial based on the definitions.

For any other pair of consistencies which is not covered by three previous theorems, the consistencies are incomparable.

- FDAC $\not\geq$ RPC,PIC,maxRPC, DRPC,DPIC,DmaxRPC: using Figures 5, 8 and 9.
- FDRPC $\not\geq$ PIC,maxRPC, DPIC,DmaxRPC: using Figures 6, 8 and 9.
- FDPIC $\not\geq$ maxRPC, DmaxRPC: using Figures 7, 8 and 9.
- EDAC $\not\geq$ (E/FD/D/-)(RPC/PIC/maxRPC): using Figures 5, 10 and 11.
- EDRPC $\not\geq$ EPIC,EmaxRPC, FDPIC,FDmaxRPC, DPIC,DmaxRPC, PIC,maxRPC: using Figures 6, 10 and 11.
- EDPIC $\not\geq$ EmaxRPC, FDmaxRPC, DmaxRPC, maxRPC: using Figures 7, 10 and 11.
- VAC $\not\geq_{c_\varnothing}$ (ED/E/FD/D/-)(RPC/PIC/maxRPC): using Figures 5 and 12.
- VRPC $\not\geq_{c_\varnothing}$ (ED/E/FD/D/-)(PIC/maxRPC): using Figures 6 and 12.
- VPIC $\not\geq_{c_\varnothing}$ (ED/E/FD/D/-)maxRPC: using Figures 7 and 12.



AC	DAC	FDAC	EAC	EDAC	VAC
~~RPC~~	~~DRPC~~	~~FDRPC~~	~~ERPC~~	~~EDRPC~~	~~VRPC~~
~~PIC~~	~~DPIC~~	~~FDPIC~~	~~EPIC~~	~~EDPIC~~	~~VPIC~~
~~maxRPC~~	~~DmaxRPC~~	~~FDmaxRPC~~	~~EmaxRPC~~	~~EDmaxRPC~~	~~VmaxRPC~~

Fig. 5 A WCSP which satisfies all arc consistencies but does not satisfy any soft RPC (hence does not satisfy any soft PIC, maxRPC). $j < k < i < l$. The problem does not satisfy any soft RPC because of variable i (the unique support (j_a, k_a) of (j, a) in c_{jk} is not simply extensible on i and the unique support (j_b, k_b) of (j, b) is not simply extensible on l.)

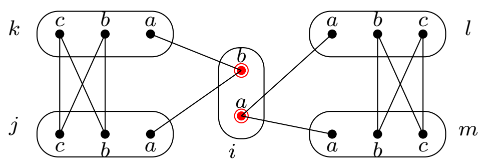


Fig. 6 A WCSP which satisfies all RPC consistencies but does not satisfy any PIC consistency. $i < j < k < l < m$. Every value of i satisfies RPC consistencies because it has more than 2 full (hence simple) arc supports in $c_{ik}, c_{ij}, c_{il}, c_{im}$. The problem does not satisfy any PIC consistency because of variable i (value (i, a) is not normally (hence not fully) extensible to triangle Δ_{ilm} while (i, b) is not simply (hence not fully) extensible to triangle Δ_{ijk})

RPC	DRPC	FDRPC	ERPC	EDRPC	VRPC
~~PIC~~	~~DPIC~~	~~FDPIC~~	~~EPIC~~	~~EDPIC~~	~~VPIC~~
~~maxRPC~~	~~DmaxRPC~~	~~FDmaxRPC~~	~~EmaxRPC~~	~~EDmaxRPC~~	~~VmaxRPC~~

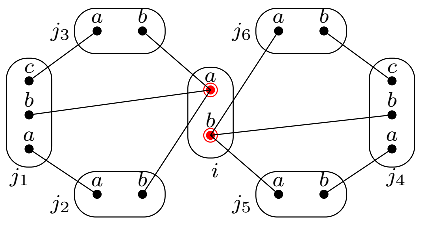
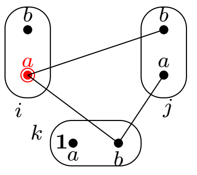


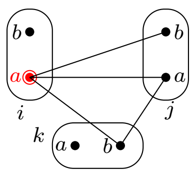
Fig. 7 A WCSP which satisfies all PIC consistencies but does not satisfy any maxRPC consistency. $i < j_1 < j_2 < j_3 < j_4 < j_5 < j_6$. There are only zero unary costs in this problem, thus simple and full supports (or witnesses) are identical. The problem is EDPIC since both $(i, a), (i, b)$ can be fully extended to all 4 triangles. However, the problem does not satisfy any maxRPC consistency because of variable i (no arc support of value (i, a) in c_{ij_1} can simultaneously be extended on $\Delta_{ij_1j_2}$ and $\Delta_{ij_1j_3}$. This is the same for value (i, b) in c_{ij_4}).

RPC	DRPC	FDRPC	ERPC	EDRPC	VRPC
PIC	PIC	FDPIC	EPIC	EDPIC	VPIC
~~maxRPC~~	~~DmaxRPC~~	~~FDmaxRPC~~	~~EmaxRPC~~	~~EDmaxRPC~~	~~VmaxRPC~~



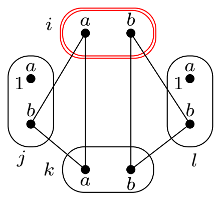
AC	~~DAC~~	~~FDAC~~	-	~~EDAC~~
RPC	~~DRPC~~	~~FDRPC~~	-	~~EDRPC~~
PIC	~~DPIC~~	~~FDPIC~~	-	~~EDPIC~~
maxRPC	~~DmaxRPC~~	~~FDmaxRPC~~	-	~~EDmaxRPC~~

Fig. 8 A WCSP which is non-directional consistent but is directional inconsistent. $i < j < k$. The problem is not DAC because value (i, a) has no full arc support in c_{ik}. Therefore, it does not satisfy FDAC, EDAC, FDRPC, EDRPC, FDPIC, EDPIC, FDmaxRPC, EDmaxRPC. However, the problem is maxRPC (hence PIC, RPC) because it is AC and every domain value is normally extensible to the triangle.



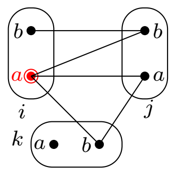
~~AC~~	DAC	~~FDAC~~	-	~~EDAC~~
~~RPC~~	DRPC	~~FDRPC~~	-	~~EDRPC~~
~~PIC~~	DPIC	~~FDPIC~~	-	~~EDPIC~~
~~maxRPC~~	DmaxRPC	~~FDmaxRPC~~	-	~~EDmaxRPC~~

Fig. 9 A WCSP which is directional consistent ($i > j > k$) but is non-directional inconsistent. The problem is not AC because (i, a) has no arc support in c_{ij}. However, the problem is DAC because every value of j and k has full arc support in c_{ji}, c_{ki}. Moreover, the problem is DmaxRPC (hence DPIC, DRPC) because every value of j and k can be fully extended on the triangle (in the triangle $\varDelta_{ijk}$, only the smallest variable k and c_{ki}, c_{kj} are interested by high order directional consistencies).



AC	DAC	FDAC	~~EAC~~	~~EDAC~~
RPC	DRPC	FDRPC	~~ERPC~~	~~EDRPC~~
PIC	DPIC	FDPIC	~~EPIC~~	~~EDPIC~~
maxRPC	DmaxRPC	FDmaxRPC	~~EmaxRPC~~	~~EDmaxRPC~~

Fig. 10 A WCSP which is full directional consistent but is existential inconsistent ($l < j < k < i$). The problem is not EAC (hence not ERPC, EPIC, EmaxRPC) because of value i (i_a has no full support in c_{ij} while i_b has no full support in c_{il}). However, the problem is FDmaxRPC (hence FDPIC, FDRPC) because it is FDAC and every value of i, k can be normally extended to both 2 triangles and every value of j, l can be fully extended to $\varDelta_{jik}$ and $\varDelta_{lik}$ respectively.



~~AC~~	~~DAC~~	~~FDAC~~	EAC	~~EDAC~~
~~RPC~~	~~DRPC~~	~~FDRPC~~	ERPC	~~EDRPC~~
~~PIC~~	~~DPIC~~	~~FDPIC~~	EPIC	~~EDPIC~~
~~maxRPC~~	~~DmaxRPC~~	~~FDmaxRPC~~	EmaxRPC	~~EDmaxRPC~~

Fig. 11 A WCSP which is existential consistent but is full directional inconsistent. $i > j > k$. The problem is not AC (hence is not RPC, PIC, maxRPC) because of value (i, a) (has no arc support in c_{ij}) and is not DAC (hence is not DRPC, DPIC, DmaxRPC) because of value (j, b) (has no full arc support in c_{ij}). However, the problem is EmaxRPC (hence EPIC, ERPC, EAC) where $(i, b), (j, a), (k, a)$ are respectively EmaxRPC supports of i, j, k.

5 Algorithms

In this section, we introduce the algorithm enforcing EDPIC, EDmaxRPC. EDRPC has not been implemented because of the costly maintainance of the number of arc supports per value in each cost function. Arc supports can be iteratively created and broken during moving costs between cost functions of different arities.

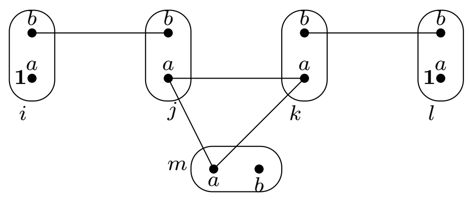


Fig. 12 A WCSP which is existential directional but is not virtual consistent $l < i < j < k < m$. The problem is not VAC (hence not VRPC, VPIC, VmaxRPC) because AC makes Bool(P) wiped-out at j or k. Conversly, the problem is EDmaxRPC where variables j, m, l are FDmaxRPC in Δ_{ijm} and i_b, j_a, k_a, l_b, m_a are EmaxRPC supports of variables.

AC	DAC	FDAC	EAC	EDAC	~~VAC~~
RPC	DRPC	FDRPC	ERPC	EDRPC	~~VRPC~~
PIC	DPIC	FDPIC	EPIC	EDPIC	~~VPIC~~
maxRPC	DmaxRPC	FDmaxRPC	EmaxRPC	EDmaxRPC	~~VmaxRPC~~

The common idea for enforcing for values of i in c_{ij} supports which are extensible on k is to move costs of triangles Δ_{ijk} (consisting of binary, ternary and possibly unary costs) to inconsistent values of i. The notation $\Delta_{ijk}(a,b,c) = c_{ij}(a,b) + c_{jk}(b,c) + c_{ik}(a,c) + c_{ijk}(a,b,c)$ denotes such combined cost, that is the sum of binary and ternary costs involved in tuple (i_a, j_b, k_c), where $c_{ijk}(a,b,c) = 0$ if c_{ijk} does not exist. Algorithm 2 present all the basic operations for moving costs inside triangles that will be used in our algorithm.

- $Extend2To3(i,a,j,b,c_{ijk},\alpha)$ extends a cost of α from a pair of values (i_a, j_b) to a ternary cost function c_{ijk}.
- $Project3To2(c_{ijk},i,a,j,b,\alpha)$ projects a cost of α from c_{ijk} on a pair (i_a, j_b).
- $Project3To1(c_{ijk},i,a,\alpha)$ projects a cost of α from c_{ijk} on a value (i,a).
- $Extend1To2(i,a,c_{ij},\alpha)$ extends a cost of α from a value (i,a) to c_{ij}.
- $Project2To1(c_{ij},i,a,\alpha)$ projects a cost of α from c_{ij} on a value (i,a)

The queues P, S, T store variables or cost functions which have had some change in value domain or in cost. They will be used for the propagation of changes in our enforcing algorithm.

- Q stores variables i such that some value of D_i has been deleted (Procedure $PruneVar$, line 24).
- P stores variables i such that some value of D_i has increased its cost from 0 (Procedure $Project3To1$ at line 10 and $Project2To1$ at line 15).
- With the same content as P (Procedure $Project3To1$ at line 11 and Procedure $Project2To1$ at line 16), S is an auxiliary queue for efficiently building the propagation queue R which contains variables that need to be checked for the existential consistency. These are all variables of S and their neighbors because: (1) for $i \in S$, the value in D_i that has increased its unary cost may be the existential support of i and (2) the existential support of neighboring variables j may be fully supported by this value.
- T contains binary costs functions c_{ij} that have been modified (because of an unary cost extension in Procedure $Extend1To2$, line 4) for which i, j and their common neighbors may have lost simple support/witness and need to be revised.

Algorithm 2: Elementary operations

```
1  Procedure Extend1To2($i, a, c_{ij}, \alpha$)
2      // precondition: $c_i(a) \geq \alpha$
3      foreach $b \in D_j$ do $c_{ij}(a,b) := c_{ij}(a,b) + \alpha$ $c_i(a) := c_i(a) - \alpha$;
4      $T \leftarrow T \cup \{c_{ij}\}$;
5  Procedure Extend2To3($i, a, j, b, c_{ijk}, \alpha$)
       // precondition: $c_{ij}(a,b) \geq \alpha$
6      foreach $c \in D_k$ do $c_{ijk}(a,b,c) := c_{ijk}(a,b,c) + \alpha$ $c_{ij}(a,b) := c_{ij}(a,b) - \alpha$
7  Procedure Project3To1($c_{ijk}, i, a, \alpha$)
8      // precondition: $\forall b \in D_j, c \in D_k, c_{ijk}(a,b,c) \geq \alpha$
9      foreach $b \in D_j, c \in D_k$ do $c_{ijk}(a,b,c) := c_{ijk}(a,b,c) - \alpha$ if $c_i(a) = 0 \wedge \alpha > 0$ then
10         $P := P \cup \{i\}$;
11         $S := S \cup \{i\}$;
12     $c_i(a) := c_i(a) + \alpha$;
13 Procedure Project2To1($c_{ij}, i, a, \alpha$)
       // precondition: $\forall b \in D_j, c_{ij}(a,b) \geq \alpha$
14     foreach $b \in D_j$ do $c_{ij}(a,b) := c_{ij}(a,b) - \alpha$ if $c_i(a) = 0 \wedge \alpha > 0$ then
15         $P := P \cup \{i\}$;
16         $S := S \cup \{i\}$;
17     $c_i(a) := c_i(a) + \alpha$;
18 Procedure isSmallest($i, \Delta_{ijk}$)
19     if $i < j$ and $i < k$ then return true else return false;
20 Procedure PruneVar($i$)
21     foreach $a \in D_i$ do
22         if $c_i(a) + c_\varnothing \geq m$ then
23             $D_i := D_i - \{a\}$;
24             $Q := Q \cup \{i\}$;
```

5.1 Enforcing PICs

5.1.1 Enforcing PIC supports

Simple PIC supports are enforced by Procedure $findPICSupport$ in Algorithm 3. To create a simple PIC support for a value i_a on Δ_{ijk}, binary and ternary costs involved in Δ_{ijk} are moved to i_a in such a way that there is a tuple (i_a, j_b, k_c) whose ternary and binary costs decrease to 0. The order for moving costs is presented in Figure 13. Firstly, binary costs c_{ij}, c_{ik}, c_{jk} are extended on ternary cost function c_{ijk} by the procedure $Extend2To3$ (line 10–10). Then, ternary costs c_{ijk} are projected on i_a by Procedure $Project3To1$ (line 10). The maximum possible cost projected on each value $a \in D_i$, stored in $P_i[a]$, is computed based on the available binary and ternary costs (line 3). Binary cost extensions E_{ij}, E_{ik}, E_{jk} are computed based on $P_i[a]$, the ternary and binary costs on two other sides of the triangle (line 5–9). Each extension is strong enough in the sense that a stronger extension cannot lead to a projection on i_a greater than $P_i[a]$. This extension is also minimum in the sense that a weaker extension would result in negative costs. The last condition guarantees that for each binary cost extension $E_{ij}(a,b), E_{ik}(a,c)$ or $E_{jk}(b,c)$, there exists a value k_c, j_b or i_a for the remaining variable such that the final resulting ternary cost $c_{ijk}(a,b,c) + E_{ij}(a,b) + E_{ik}(a,c) + E_{jk}(b,c) - P_i[a] = 0$. Therefore, binary cost extensions on ternary functions do not lead to the loss of ternary AC supports. Moreover, binary cost extensions do not lead to the loss of PIC supports because PIC supports involve only zero binary costs which cannot be used for extension.

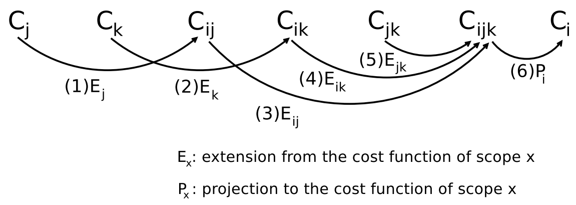

C_j
C_k
C_ij
C_ik
C_jk
C_ijk
C_i
(1)E_j
(2)E_k
(3)E_ij
(4)E_ik
(5)E_jk
(6)P_i
E_x: extension from the cost function of scope x
P_x: projection to the cost function of scope x

Fig. 13 The order of cost movements for enforcing simple or full PIC supports, where unary cost extensions are not included in the enforcement of simple PIC supports. The flows indicate the direction of cost movements and the numbers under the flows indicate the order in which the corresponding cost movements are performed

Full PIC supports are enforced by Procedure *findFullPICSupport* in Algorithm 3 in a way similar to Procedure *findPICSupport*. The difference is that unary costs of j, k are extended on binary functions c_{ij} and c_{ik} respectively, by the procedure *Extend1To2*, in order to create full PIC supports with zero unary costs (line 18, 18). After that, binary and ternary costs are moved to i_a in the same way as for enforcing simple PIC supports (line 18). The order in which costs are moved to enforce full PIC supports is also described in Figure 13. The unary costs of j, k are taken into account for the computation of $P_i[a]$ as well as for the computation of unary cost extensions E_j, E_k (line 13,15,17). Similarly to binary cost extensions, unary cost extensions are strong enough to lead a cost projection $P_i[a]$ without creating negative costs. This condition ensures that for any unary cost extension $E_j[b], E_k[c]$, there exists a value $a \in D_i$ such that the final resulting binary costs $c_{ij}(a,b)+E_j(b)-E_{ij}(a,b)$ and $c_{ik}(a,c)+E_k(c)-E_{ik}(a,c)$ are equal to 0. Therefore, unary cost extensions on binary functions cannot lead to the loss of binary AC supports. However, unary cost extensions on binary functions can lead to the loss of simple PIC supports, thus modified binary functions are stored in the list T in order to recall to enforce PIC supports for related values later.

Example 2 Consider the WCSP(a) in Figure 14. It has 4 variables $i < j < k < l$ and 5 binary cost functions $c_{ij}, c_{ik}, c_{il}, c_{jk}, c_{jl}$. Binary costs are represented by edges (red continuous line) and ternary costs are represented by hyper edges (blue dashed lines for c_{ijk} and green dashed lines for c_{ijl}). The absence of (hyper)edges indicates a zero cost. The initial problem is FDAC but not FDPIC because value (i,a) is not fully extensible on Δ_{ijk}. Now, consider enforcing

Algorithm 3: Algorithms enforcing PIC supports

```
1  Procedure findPICSupport($i, \Delta_{ijk}$)
2      foreach $a \in D_i$ do
3          $P_i[a] \leftarrow \min_{b\in D_j, c\in D_k} \Delta_{ijk}(a,b,c)$;
4      foreach $a \in D_i, b \in D_j$ do
5          $E_{ij}[a,b] \leftarrow \max_{c\in D_k}\{P_i[a] - c_{ijk}(a,b,c) - c_{ik}(a,c) - c_{jk}(b,c)\}$;
6      foreach $a \in D_i, c \in D_k$ do
7          $E_{ik}[a,c] \leftarrow \max_{b\in D_j}\{P_i[a] - c_{ijk}(a,b,c) - c_{jk}(b,c) - E_{ij}(a,b)\}$;
8      foreach $b \in D_j, c \in D_k$ do
9          $E_{jk}[b,c] \leftarrow \max_{a\in D_i}\{P_i[a] - c_{ijk}(a,b,c) - E_{ij}(a,b) - E_{ik}(a,c)\}$;
10     foreach $a \in D_i, b \in D_j$ do Extend2To3($i, a, j, b, c_{ijk}, E_{ij}[a,b]$) foreach
       $a \in D_i, c \in D_k$ do Extend2To3($i, a, k, c, c_{ijk}, E_{ik}[a,c]$) foreach $b \in D_j, c \in D_k$ do
       Extend2To3($j, b, k, c, c_{ijk}, E_{jk}[b,c]$) foreach $a \in D_i$ do
       Project3To1($c_{ijk}, i, a, P_i[a]$) ProjectUnary($i$);

11 Procedure findFullPICSupport($i, \Delta_{ijk}$)
12     foreach $a \in D_i$ do
13         $P_i[a] \leftarrow \min_{b\in D_j, c\in D_k} \Delta_{ijk}(a,b,c) + c_j(b) + c_k(c)$;
14     foreach $b \in D_j$ do
15         $E_j[b] \leftarrow \max_{a\in D_i, c\in D_k}\{P_i[a] - \Delta_{ijk}(a,b,c) - c_k(c)$;
16     foreach $c \in D_k$ do
17         $E_k[c] \leftarrow \max_{a\in D_i, b\in D_j}\{P_i[a] - c_{ijk}(a,b,c) - E_j[b]$;
18     foreach $b \in D_j$ do Extend1To2($j, b, c_{ji}, E_j[b]$)  foreach $c \in D_k$ do
       Extend1To2($k, c, c_{ki}, E_k[c]$)  findPICSupport($i, \Delta_{ijk}$);

19 Procedure findEPICSupport($i$)
20     $\alpha \leftarrow \min_{a\in D_i}\{c_i(a) + \sum_{\Delta_{ijk}, i>j \text{ or } i>k} \min_{b\in D_j, c\in D_k}\{\Delta_{ijk}(a,b,c) + c_j(b) + c_k(c)\}\}$;
21     if $\alpha > 0$ then
22         foreach $\Delta_{ijk}$ do
23             if $\neg$isSmallest($i, \Delta_{ijk}$) then findFullPICSupport($i, \Delta_{ijk}$)
               $R := R \cup \bigcup_{\Delta_{ijk}}\{j,k\}$;
24         UnaryProject($i, \alpha$);
```

full PIC supports for the values of i. Procedure $findFullPICSupport(i, j, k)$ computes the amounts of cost for projections/extensions: $P_i[a] = E_j[b] = 1$. Other projection/extension costs are zero. After extending a cost of 1 from (j, b) on c_{ij}, it will call Procedure $findPICSupport(i, j, k)$ and compute the amounts of cost projections/extensions as following:

$$P_i[a] = E_{ij}[a,b] = E_{ik}[a,a] = E_{jk}[a,b] = 1.$$

The resulting problem, presented in the sub-figure 14(d) is still not FDPIC because value (i, b) cannot be fully extended on triangle Δ_{ijl}. Then Procedure $findFullPICSupport(i, j, l)$ computes the following projection/extension costs:

$$P_i[b] = E_{ij}[b,b] = E_{il}[b,a] = E_{jl}[a,b] = 1.$$

The final problem, presented in the sub-figure 14(g) is FDPIC. Unlike enforcing hard PIC, enforcing simple and full PIC supports can create new ternary functions, e.g., c_{ijk}, c_{ijl}. Whenever a binary cost need to be extended to a ternary cost function that does not exist, the ternary cost function will be created and initialized with an empty cost for every tuple.

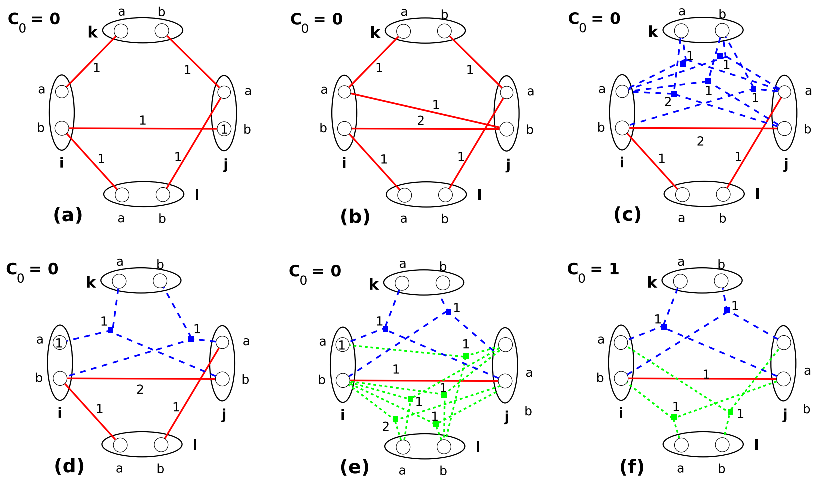


Fig. 14 Cost evolution in a WCSP during the enforcement of full PIC supports (a) original problem with 5 binary cost functions $c_{ij}, c_{ik}, c_{il}, c_{jk}, c_{jl}, i < j < k < l\}$. It is FDAC but not FDPIC because of variable i where (i, a) and (i, b) cannot fully extended on Δ_{ijk} and Δ_{ijl} respectively. (b) extending a cost of 1 from j_b on c_{ij} with $E_j[b] = 1$. (c) extending a cost of 1 from (i_a, j_b), (i_a, k_a) and (j_a, k_b) on c_{ijk} with $E_{ij}[a, b] = E_{ik}[a, a] = E_{jk}[a, b] = 1$. (d) projecting a cost of 1 from c_{ijk} on i_a with $P_i[a] = 1$. (e) extending a cost of 1 from $(i_b, j_b), (i_b, l_a)$ and (j_a, l_b) on c_{ijk} with $E_{ij}[b, b] = E_{il}[b, a] = E_{jl}[a, b] = 1$. (f) projecting a cost of 1 from c_{ijk} on i_b with $P_i[b] = 1$ and then enforcing NC by projecting a cost of 1 from c_i on $c_\varnothing$. The resulting problem is FDPIC.

5.1.2 Enforcing EDPIC

EDPIC is enforced by Procedure *enforceEDPIC* in Algorithm 4. The main idea is to simply enforce EPIC, DPIC and PIC simultaneously. Procedure *enforceEDPIC* consists of four inner-while loops and one for-loop to enforce respectively EPIC, DPIC, PIC and NC.

The first while-loop (line 5-7) aims to enforce EPIC. It firstly put in R all variables that need to be checked for EPIC based on the auxiliary queue S (line 4). EPIC supports of variables $i \in R$ are enforced by Procedure *findEPICSupport* (line 7). When enforcing the existential support for i, EPIC is only responsible for triangles on which i is not the smallest because DPIC takes care of the remaining ones. That's why the property of EPIC is only checked for such triangles (Algorithm 3, line 20). If i has no fully supported value (i.e., $\alpha > 0$) such a value can be created by enforcing full PIC supports for every value of i on every triangle in which i is not the smallest variable (Algorithm 3, line 23). The EPIC supports of variables neighbor to i

can also be destroyed (due to new values of non-zero cost made by the enforcement of full PIC supports on i) and thus are pushed back to R to be checked for EPIC later (Algorithm 3, line 23).

DPIC is enforced by the second while-loop at line 8. For a variable $j \in P$, only variables that are linked to j by a triangle (line 10) and are the smallest variable of the triangle (line 11, 11) are considered for checking for DPIC.

PIC is enforced by two while-loops at lines 12 and 16. For a variable $j \in Q$, every neighboring variable of i is checked for PIC. For each $c_{ij} \in T$, i, j and all variables connected to both i and j are checked for PIC. Simple PIC supports are enforced in the reverse direction of the DAC order, i.e. in triangles in which the considered variables are not the smallest (line 15 – 15, 19– 19).

From Algorithm 4 enforcing EPIC, we can obtain algorithms for enforcing other PICs by appropriately keeping the inner while-loops: the first loop (lines 4– 7) for EPIC, the second one at line 8 for DPIC, the third one at line 12 for PIC, and both the second and the third ones for FDPIC.

Algorithm 4: Algorithm enforcing EDPIC

```
1  Procedure enforceEDPIC
2      $S = P = Q = X$;  $T \leftarrow \varnothing$;
3      while $Q \neq \varnothing$ or $P \neq \varnothing$ or $S \neq \varnothing$ or $T \neq \varnothing$ do
4          $R \leftarrow S \cup \bigcup_{i \in S, \Delta_{ijk}} \{j, k\}$;
5          while $R \neq \varnothing$ do
6              $i \leftarrow R$.pop();
7              findEPICSupport($i$);
8          while $P \neq \varnothing$ do
9              $j \leftarrow P$.pop();
10             foreach $\Delta_{ijk}$ do
11                 if $isSmallest(i, \Delta_{ijk})$ then findFullPICSupport($i, \Delta_{ijk}$) if
                   $isSmallest(k, \Delta_{ijk})$ then findFullPICSupport($k, \Delta_{ijk}$)
12         while $Q \neq \varnothing$ do
13             $j \leftarrow Q$.pop();
14             foreach $\Delta_{ijk}$ do
15                 if $\neg isSmallest(i, \Delta_{ijk})$ then findPICSupport($i, \Delta_{ijk}$) if
                   $\neg isSmallest(k, \Delta_{ijk})$ then findPICSupport($k, \Delta_{ijk}$)
16         while $T \neq \varnothing$ do
17             $c_{ij} \leftarrow T$.pop();
18             foreach $\Delta_{ijk}$ do
19                 if $\neg isSmallest(i, \Delta_{ijk})$ then findPICSupport($i, \Delta_{ijk}$) if
                   $\neg isSmallest(j, \Delta_{ijk})$ then findPICSupport($j, \Delta_{ijk}$) if
                   $\neg isSmallest(k, \Delta_{ijk})$ then findPICSupport($k, \Delta_{ijk}$)
20         foreach $i \in X$ do PruneVar($i$)
```

5.2 Enforcing maxRPCs

In contrast to PICs that are enforced on triangles sharing a variable, maxRPCs are enforced on triangles sharing two variables of a binary cost function. The extensible arc support of a value (i, a) in a binary cost function c_{ij} is stored in maxRPCSupport(i, a, j) and the witness for this support on a variable k is stored in maxRPCWitness(i, a, j, k). In our algorithm enforcing EDmaxRPC, we use a parameter "level=0" to indicate the semi-fully extensible arc supports (used by FDmaxRPC) and "level=1" the fully extensible ones (used by EmaxRPC). We will use these functions below:

– $\Lambda_{ijk}(a, b, c) = c_{ik}(a, c) + c_{jk}(b, c) + c_{ijk}(a, b, c)$: denotes the incompletely combined cost of tuple (a, b, c), similar to $\Delta_{ijk}(a, b, c)$ but excludes $c_{ij}(a, b)$.

– $\curlywedge^{k}_{ij}(a, b)$ $= \min_{c \in D_k} \Lambda_{ijk}(a, b, c)$: denotes the minimum among the incompletely combined cost of tuples involving two values (i_a, j_b). This is the maximum cost that can be projected on the pair of values (i_a, j_b) from two sides c_{ik}, c_{jk} of the triangle Δ_{ijk}

– $\ddot{\curlywedge}^{k}_{ij}(a, b, \text{level}) = \begin{bmatrix} \min_{c \in D_k} \{\Lambda_{ijk}(a, b, c) + c_k(c)\} & (\text{level} = 1) \vee (i < k) \\ \min_{c \in D_k} \{\Lambda_{ijk}(a, b, c)\} & (\text{level} = 0) \end{bmatrix}$: is the same as $\curlywedge^{k}_{ij}(a, b)$ but takes into account the unary cost c_k of witnesses in the case of (1) fully extensible arc supports (level=1) or (2) semi-fully extensible arc supports on triangles w.r.t DAC order $(i < k)$

– $\curlywedge\!\!\curlywedge_{ij}(a, b) = \sum_k \{\curlywedge^{k}_{ij}(a, b)\}$: means the maximum sum of costs that can be projected on the pair of values (i_a, j_b) from all triangles Δ_{ijk} sharing i, j.

– $\ddot{\curlywedge\!\!\curlywedge}_{ij}(a, b, \text{level}) = \sum_k \{\ddot{\curlywedge}^{k}_{ij}(a, b, \text{level})\}$: is the same as $\curlywedge\!\!\curlywedge_{ij}(a, b)$ but takes into account the unary costs of witnesses c_k according to the condition ("level" and the order between i and k) indicated in the definition of $\ddot{\curlywedge}^{k}_{ij}$.

Associated to the above functions are functions arg$\curlywedge$, arg$\ddot{\curlywedge}$, arg$\curlywedge\!\!\curlywedge$ and arg$\ddot{\curlywedge\!\!\curlywedge}$ where the two first ones return the witness and the two last ones returns the array of witnesses on the third variables at which the given pair of values can be extended with a minimum extra cost.

5.2.1 Enforcing maxRPC supports and witnesses

Simple maxRPC support for a value (i, a) on c_{ij} is enforced by Procedure *findmaxRPCSupport* in Algorithm 5. The main idea to move costs from 2 sides c_{ik}, c_{jk} of all triangles Δ_{ijk} to c_{ij} via c_{ijk} (line 22 – 22) and finally project costs from c_{ij} to (i, a) (line 23) in such a way that there exists a value $b \in D_j$ and a value $c \in D_k$ for each triangle Δ_{ijk} such that the binary and ternary costs involved in the tuple (i_a, j_b, k_c) decrease to 0. The cost that pairs of values (i_a, j_b), $\forall b \in D_j$, can receive at most (computed by Procedure $\curlywedge\!\!\curlywedge_{ij}$) plus their available binary cost will define the maximum cost P_i that can be projected to (i, a) (line 12). This allows to compute the real cost $P_{ij}[a, b]$ that each triangle Δ_{ijk} provides to (i_a, j_b) for such a projection to be achieved on i_a. It is the minimum of what is needed for this pair of values $P_i - c_{ij}(a, b)$ and what can be provided for it by Δ_{ijk} (line 17). This condition guarantees that c_{ij} has enough costs to make a unary cost projection P_i on i_a without resulting in negative costs. Moreover, if more costs are projected on c_{ij}, this cannot lead to a unary cost projection greater than P_i. In order to project a cost of $P_{ij}[a,b]$ from c_{ijk} to (i_a, j_b) (line 22), each side (i_a, k_c) and (j_b, k_c) has to extend an amount of cost $E_{ik}[a,c]$ and $E_{jk}[b,c]$ to c_{ijk} (line 22, 22). These binary cost extensions $E_{ik}[a,c]$, $E_{jk}[b,c]$ are also the minimum of the available cost $c_{ik}(a,c), c_{jk}(b,c)$ that $(i_a, k_c), (j_b, k_c)$ have and the cost that they need to provide to c_{ijk} (line 19, 21).

Full maxRPC support at two levels (fully extensible for EmaxRPC and semi-fully extensible for FDmaxRPC) is enforced by $findFullmaxRPCSupport$ in Algorithm 5. The idea for enforcing a full maxRPC support for value (i,a) in a cost function c_{ij} is to extend unary costs from j to c_{ij} (line 5) and from third variables k to c_{ik} (line 9). Then, costs are moved in the same way as enforcing simple maxRPC support in Procedure $findmaxRPCSupport$ (line 10). The maximum cost P_i that can be projected on i_a is recomputed by taking into account the unary cost c_j of supporting values and the unary costs c_k of witnessing values via 从 (line 2). In order to archive this unary projection, each value j_b, k_c needs to extend respectively on c_{ij} and c_{ik} a amount of cost E_j, E_k (line 4 and 8). The order in which costs are moved when enforcing full maxRPC supports is described in Figure 15.

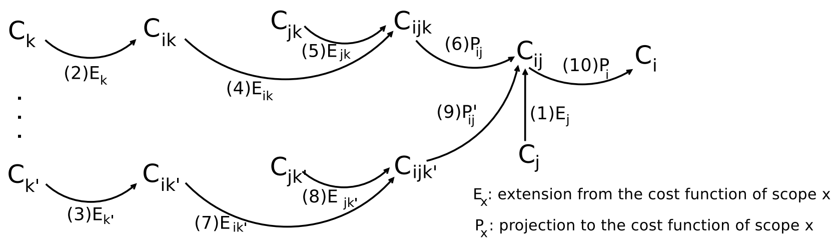


Fig. 15 The order of cost movements for enforcing simple full maxRPC supports where unary cost extensions are not included in the enforcement of simple PIC supports.

Example 3 Consider the WCSP(a) in Figure 16. It is FDPIC but not FD-maxRPC because i_a has no full AC support in c_{ij} which can be can be extended on both Δ_{ijk} and Δ_{ijl}: (i_a, j_a) can be extended on Δ_{ijl} but not on Δ_{ijk} while (i_a, j_c) can be extended on Δ_{ijk} but Δ_{ijl}. The positive projection/extension costs computed by the procedure $findfullmaxRPC(i,a,j)$ are: $P_i = 2, E_j[b] = 1$. The procedure extends a cost of 1 from j_b on c_{ij} and then calls $findmaxRPC(i,a,j)$ which computes the following positive projections/extension costs: $P_i = E_{jk}[a,b] = P_{ij}[a,a] = E_{jl}[c,b] = E_{jl}[c,b] = P_{ij}[a,a] = 2$. The final problem presented in the sub-figure (g) is FDmaxRPC.

Let j be a variable that has had a change in the domain D_j or in unary cost c_j (increasing from 0). The former case can break the witness for the simple or semi-full supports of variable i neighbor to j in some c_{ij}, while the

Algorithm 5: Algorithms enforcing maxRPC supports

```
1  Procedure findFullmaxRPCSupport($i, a, j$,level)
       // condition: $i < j$ or level $= 1$
2      $P_i \leftarrow \min_{b \in D_j}\{c_j[b] + c_{ij}(a,b) + \ddot{\curlywedge\!\!\curlywedge}_{ij}(a,b)\}$;
3      foreach $b \in D_j$ do
4          $E_j \leftarrow P_i - \ddot{\curlywedge\!\!\curlywedge}_{ij}(a,b) - c_{ij}(a,b)$;
5          Extend1To2($j, b, c_{ij}, E_j$);
6      foreach $\Delta_{ijk}$ s.t (level = 1 and $\neg$isSmallest($i, \Delta_{ijk}$)) or (level=0 and $i < k$) do
7          foreach $c \in D_k$ do
8              $E_k \leftarrow \min(c_k[c], \max_{b \in D_j}\{P_i - \Delta_{ijk}(a,b,c)\})$;
9              Extend1To2($k, c, c_{ik}, E_k$);
10     findmaxRPCSupport($i, a, j$);

11 Procedure findmaxRPCSupport($i, a, j$)
12     $P_i \leftarrow \min_{b \in D_j}\{c_{ij}(a,b) + \curlywedge\!\!\curlywedge_{ij}(a,b)\}$ ;
13     $b^* \leftarrow \mathrm{argmin}_{b \in D_j}\{c_{ij}(a,b) + \curlywedge\!\!\curlywedge_{ij}(a,b)\}$;
14     if $P_i = 0$ then return
15     foreach $\Delta_{ijk}$ do
16         foreach $b \in D_j$ do
17             $P_{ij}[a,b] \leftarrow \min\{P_i - c_{ij}(a,b),\ \curlywedge^k_{ij}(a,b)\}$ ;
18         foreach $c \in D_k$ do
19             $E_{ik}[a,c] \leftarrow \min\{c_{ik}(a,c),\ \max_{b \in D_j}\{P_i - c_{ijk}(a,b,c) - c_{ij}(a,b) - c_{jk}(,b,c)\}\}$ ;
20         foreach $b \in D_j, c \in D_k$ do
21             $E_{jk}[b,c] \leftarrow \min\{c_{jk}(b,c),\ P_i - c_{ijk}(a,b,c) - c_{ij}(a,b) - E_{ik}[a,c]\}$ ;
22         foreach $b \in D_j, c \in D_k$ do Extend2To3($j, b, k, c, c_{ijk}, E_{jk}[b,c]$) foreach
           $c \in D_k$ do Extend2To3($i, a, k, c, c_{ijk}, E_{ik}[a,c]$) foreach $b \in D_j$ do
           Project3To2($c_{ijk}, i, a, j, b, P_{ij}[a,b]$)
23     Project2To1($c_{ij}(a,b), i, a, P_i$);
24     ProjectUnary($i$);
25     maxRPCSupport$[i,a,j] \leftarrow b^*$;
26     foreach $\Delta_{ijk}$ do maxRPCWitness$[i,a,j,k] \leftarrow \mathrm{argmin}\ddot{\curlywedge\!\!\curlywedge}_{ij}(a,b^*)[k]$

27 Procedure FindEmaxRPCSupport(i)
28     $\alpha \leftarrow \min_{a \in D_i}\{c_i(a) + \sum_{c_{ij}} \min_{b \in D_j}\{c_{ij}(a,b) + \ddot{\curlywedge\!\!\curlywedge}_{ij}(a,b,\mathrm{fullLevel})\}\}$;
29     if $\alpha > 0$ then
30         foreach $c_{ij}$ do
31             foreach $a \in D_i$ do
32                 findFullmaxRPCSupport($i, a, j$, fullLevel);
33             $R \leftarrow R \cup \bigcup_{c_{ij}}\{j\}$;
34         UnaryProject($i, \alpha$) ;
```

last case can break the witness for the semi-full and full supports. The check and search for new witnesses is performed by Algorithm 6.

Procedure $findWitness\_remove(i, k, j)$ handles the case of domain reduction in D_j. For any value (i, a), it checks the availability of its current (simple or semi-full) support in c_{ik} (line 16). In the case that the current support is

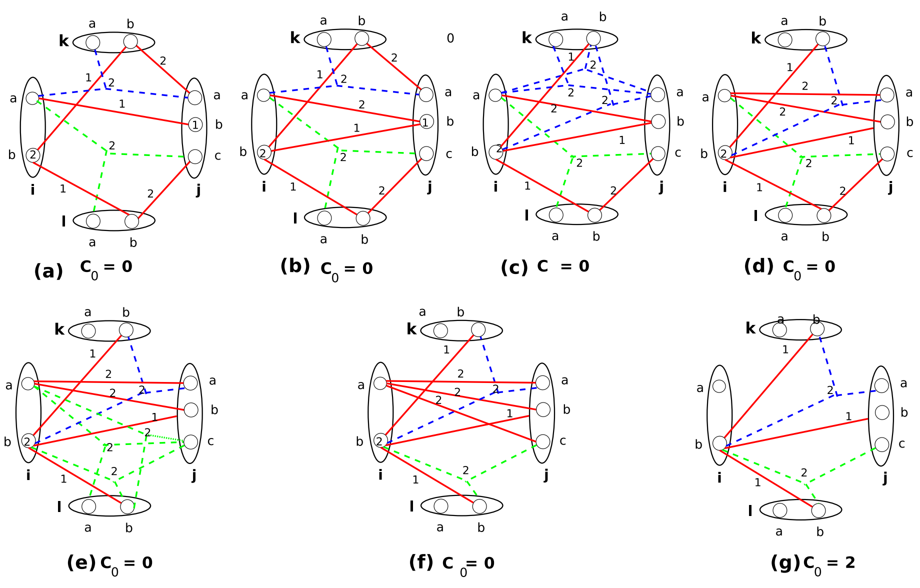


Fig. 16 Cost evolution during enforcing full maxRPC supports in a WCSP (a) original problem with 5 binary cost functions $c_{ij}, c_{ik}, c_{il}, c_{jk}, c_{jl}$ and 2 ternary functions c_{ijk}, c_{ijl}, $i < \{j, k, l\}$. It is FDPIC but not FDmaxRPC due to i_a (no full maxRPC support in c_{ij}) (b) extending a cost of 1 from j_b on c_{ij} with $E_j[b] = 1$ (c) extending a cost of 2 from (j_a, k_b) on c_{ijk} with $E_{jk}[a, b] = 2$ (d) projecting a cost of 2 from c_{ijk} on (i_a, j_a) with $P_{ij}[a, a] = 2$ (e) extending a cost of 2 from (j_c, l_b) on c_{ijk} with $E_{jl}[c, b] = 2$ (f) projecting a cost of 2 from c_{ijk} on (i_a, j_c) with $P_{ij}[a, a] = 2$ (g) projecting a cost of 2 from c_{ij} on i_a with $P_i = 2$ and then making NC by projecting a cost of 2 from i to $c_\varnothing$. The resulting problem is FDmaxRPC.

still available, if the current witness on D_k for this support is no longer available (line 18) and there does not exist any witness (line 19), another simple or full support needs to be searched for i_a according to $i > k$ (line 23) or $i < k$ respectively (line 23), This is the same as the case of loss of the current maxRPC support (line 7).

Procedure $findWitness\_project(i, k, j)$ handles the case that some unary costs c_j have increased from 0. This procedure is responsible for the semi-full supports because it is only called in the while-loop enforcing DmaxRPC at line 7 of Algorithm 7. It is different from $findWitness\_remove$ by the fact that unary costs are taken into account when checking the availability of the current supports and witnesses (line 5, 7) and the existance of another witness for replacing the current unavailable one ($\ddot{\curlywedge}$ instead of $\curlywedge$, line 8).

5.2.2 Enforcing EDmaxRPC

EDmaxRPC is enforced by Procedure $enforceEDmaxRPC$ in Algorithm 7. It consists of 4 inner-while loops that handle the same propagation queues S, P, Q, T used in the EDPIC enforcement algorithm.

Algorithm 6: Algorithms enforcing maxRPC witnesses

```
1  Procedure findWitness_project($i, k, j$)
       // evoked by a cost projection on a value of zero cost in $D_j$, used to
          search for a full witness in $D_j$ for the full supports of values of
          $i$ in $c_{ik}$ such that $i < j, i < k$
2      foreach $a \in D_i$ do
3          $s \leftarrow$ maxRPCsupport[$i, a, k$];
4          need $\leftarrow$ false; // need to search for a new full support from scratch
5          if $s \in D_k$ and $c_k(s) + c_{ik}(a, s) = 0$ then
6              $w \leftarrow$ maxRPCWitness[$i, a, k, j$];
7              if $w \notin D_j$ or $c_j(w) > 0$ or $\Delta_{ikj}(a, s, w) > 0$ then
8                  if $\ddot{\curlywedge}^j_{ik}(a, s, wit, semiLevel) = 0$ then
9                      maxRPCWitness[$i, a, k, j$] $\leftarrow$ wit;
10                 else need $\leftarrow$ true
11         else need $\leftarrow$ true if need=true then
           findFullmaxRPCSupport($i, a, k$,semiLevel)

12 Procedure findWitness_remove($i, k, j$)
       // evoked by the reduction of domain $D_j$, used to search for a witness
          in $D_j$ for the support of values of $i$ in $c_{ik}$ such that $i > j$ or $i > k$
13     foreach $a \in D_i$ do
14         $s \leftarrow$ maxRPCSupport[$i, a, k$];
15         need $\leftarrow$ true; // need to search for a new simple support from scratch
16         if $s \in D_k$ and $c_{ik}(a, s) = 0$ and ($i > k$ or $c_k(s) = 0$) then
17             $w \leftarrow$ maxRPCWitness[$i, a, k, j$];
18             if $w \notin D_j$ or $\Delta_{ikj}(a, s, w) > 0$ then
19                 if $\curlywedge^j_{ik}(a, s) = 0$ then
20                     maxRPCWitness[$i, a, k, j$] $\leftarrow$ arg $\curlywedge^j_{ik}(a, s)$;
21                     need $\leftarrow$ false;
22         if need=true then
23             if $i > k$ then findmaxRPCSupport($i, a, k$); else
               findFullmaxRPCSupport($i, a, k$,semiLevel);
```

The loop line 11 enforces maxRPC by propagating domain reductions of j stored in the queue Q. For any neighboring value (i, a) of j, the deleted values in D_j could have been: (1) the simple maxRPC support for (i, a) when $i > j$; (2) the simple witness for the simple maxRPC supports of i_a in some c_{ik} when $i > k$; and (3) the simple witness for semi-full maxRPC supports of i_a in c_{ik} (of course $i < k$) when $i > j$. The deleted values in D_j could not have been the full supports and witnesses because they must have a possitive cost (m) to be removed. The loop must to check and search for (1) a simple maxRPC (line 14-15) and (2) a simple witness (line 16-16).

The loop at line 7 enforces DmaxRPC by propagating the increase from 0 of unary cost c_j stored in P. The variables i neighbor to and smaller than j (line 9) can have lost full supports in c_{ij} and thus new full supports need to be searched for values of i (line 10). Moreover, the full supports in $c_{ik}, i < k$

(line 10) can have lost full witnesses on j if $i < j$ (line 9) and thus need to be searched for new witnesses (line 10).

The loop at line 4 enforces EmaxRPC by processing the propagation queue R containing variables that need to be checked for EmaxRPC. The construction of R from the auxiliary queue S (line 3) has been explained in the case of enforcing EDPIC. The search for a EmaxRPC support for a variable i is done by Procedure $findEmaxRPCSupport(i)$ in Algorithm 5. It first checks the EmaxRPC property at line 28. If there does not exist any EmaxRPC support (line 29), the procedure will search for a full maxRPC support for any value of i in any cost function c_{ij} by calling to $findFullmaxRPCSupport$ with the option level=fullLevel. It has to take care of the triangles Δ_{ijk} in which i is not the smallest variable, because DmaxRPC takes care of the remaining case (The condition at line 6 of Procedure $findFullmaxRPCSupport$).

The loop at line 17 enforces maxRPC by propagating the change in binary costs c_{ij} (caused by unary cost extensions from the greater variable between i and j on c_{ij}) stored in queue T. Let i^* and j^* be respectively the greater and the smaller variable between i and j. The modification c_{ij}:
- cannot break the full or semi-full maxRPC supports of the smaller variable j^* because its value have been supported by values of zero cost.
- can break the simple maxRPC supports for the values of the greater variable i^* and thus new supports need to be searched for such values (line 20).
- can break the witnesses for maxRPC supports in c_{ik} (line 22, 24) or in c_{jk} (line 23, 25).

6 Experimentation

We have built 3 usecases for high order consistencies (HOCs), denoted by:
- HOCs^1: uses HOCs for preprocessing and EDAC for search
- HOCs^2: uses resHOCs, a restriction of HOCs, for preprocessing and EDAC for search
- HOCs^3: uses resHOCs for both preprocessing and search.

The restricted resHOCs are defined by limiting the number of triangles to be checked for the consistencies. The purpose is to have consistencies weaker but cheaper than HOCs and still stronger than ACs. Let c, c^* and $c' = \min\{c, c^*\}$ respectively be the number of triangles that a WCSP have, can use at most and uses by HOC^r. If $c \leq c^*$, $c' = c$ and all triangles will be used by resHOCs. Otherwise, $c' = c^*$ and only the c^* strongest triangles are used by resHOCs, where triangles are evaluated and classified according to the sum of the mean cost of three involving binary cost functions. The mean cost of a binary cost function c_{ij} is computed by $(\sum_{a \in D_i, b \in D_j} c_{ij}(a,b))/(|D_i| \times |D_j|)$. In our experimentation, we have chosen $c^* = n(n-1)(n-2)/6.10^4$ by basing on the fact that HOCs seem to be overload when used for preprocessing the benchmarks having more than $n(n-1)(n-2)/6.10^4$ triangles, i.e. having a triangle density larger than 10^{-4}. The graph density of a problem is defined as

Algorithm 7: Algorithm enforcing EDmaxRPC

```
1  Procedure enforceEDmaxRPC
2      while $S \neq \varnothing$ or $P \neq \varnothing$ $Q \neq \varnothing$ or $T \neq \varnothing$ do
3          $R \leftarrow S \cup \bigcup_{i \in S, c_{ij}} \{j\}$;
4          while $R \neq \varnothing$ do
5              $j \leftarrow R$;
6              findEmaxRPCSupport($j$);
7          while $P \neq \varnothing$ do
8              $j \leftarrow P$;
9              foreach $c_{ij}, i < j$ do
10                 foreach $a \in D_i$ do findFullmaxRPCSupport($i, a, j$,semiLevel)
                   foreach $\Delta_{ikj}, i < k$ do findWitness_project($i, k, j$)
11         while $Q \neq \varnothing$ do
12             $j \leftarrow Q$;
13             foreach $c_{ij}$ do
14                 if $i > j$ then
15                     foreach $a \in D_i$ do findmaxRPCSupport($i, a, j$)
16                 foreach $\Delta_{ikj}$ s.t. $i > j$ or $i > k$ do findWitness_remove($i, k, j$)
17         while $T \neq \varnothing$ do
18             $c_{ij} \leftarrow T$; $i^* \leftarrow$max$\{i, j\}$; $j^* \leftarrow$min$\{i, j\}$;
19             foreach $a \in D_{i^*}$ do
20                 findmaxRPCSupport($i^*, a, j^*$);
21             foreach $\Delta_{ijk}$ do
22                 findWitness_remove($i, k, j$);
23                 findWitness_remove($k, i, j$);
24                 findWitness_remove($j, k, i$);
25                 findWitness_remove($k, j, i$);
```

the ratio of its number of triangles over the number of triangles in a complete graph. Notice that $c^* < 10$ is considered too small to make a difference between resHOCs and EDAC. In this case, resHOCs will be replaced by EDAC and c' is set to 0.

In order to evaluate the practical interest of establishing HOCs and their variants, we compared them to the default local consistency enforced in `toulbar2`: EDAC. Indeed, EDAC is still the state-of-the-art for WCSP solving (VAC being mostly useful for some very hard or specific problems). We use a large set of benchmarks, as described in Table 1, which has been used in the experimentation of EDAC in [2] for comparing the performance of the `toulbar2` solver with other solvers. This set consists of groups of benchmarks as follows:

– WCSP: contains cost function networks extracted from the Cost Function Library[1], including Combinatorial Auctions [14], Radio Link Frequency Assignment problems [5], Mendelian error correction problems on complex pedigree [15], Computational Protein Design problems [1] and SPOT5 satellite scheduling problems [3].

[1] https://mulcyber.toulouse.inra.fr/scm/viewvc.php/trunk/?root=costfunctionlib

Categories	#inst	$\overline{n}$	$\overline{d}$	$\overline{e}$	$\overline{r}$	$\overline{c}$	$\overline{c'}$	$\overline{\text{dens}}$
CVPR	1453							
ChineseChars	100	9147	2	276677	2	86557	86557	1,14E-06
ColorSeg	21	108910	9	474745	2	131805	32998	2,73E-09
GeomSurf-3	300	505	3	2140	3	8	8	4,46E-07
GeomSurf-7	300	505	7	2140	3	1366	1265	0,00018
InPainting	4	14400	4	57121	2	17732	17732	3,56E-08
Matching	4	19	19	166	2	701	0	0,679
MatchingSte	2	138407	18	414477	2	8	8	2,70E-14
ObjectSeg	5	68160	6	203947	2	31	31	5,91E-13
PhotoMont	2	469856	6	1408134	2	521	521	4,03E-14
SceneDecp	715	183	8	672	2	48	42	4,80E-05
MaxCSP	503							
BlackHole	37	114	27	657	2	5375	38	0,01
Coloring	22	120	4	1323	2	1227	277	0,024
Composed	80	58	10	517	2	791	0	0,079
EHI	200	306	7	4549	2	13604	475	0,0029
Geometric	100	50	20	471	2	1694	0	0,086
Langford	4	25	22	352	2	2722	0	0,736
QCP	60	159	7	1384	2	2671	108	0,0057
MaxSAT	427							
Haplotyping	100	150428	2	534105	483	61646	61646	2,39E-10
MaxClique	62	484	2	50093	2	1070886	2019	0,079
MIPLib	12	10523	2	45991	20	104	104	5,92E-07
PackupWei	99	9492	2	23731	61	9236	9236	6,87E-07
PlanWithPre	29	14991	2	111259	64	8026	8026	1,76E-06
TimeTabling	25	128243	2	785222	21	40052	40052	1,58E-09
Upgrad	100	18169	2	105097	77	1884	1884	1,88E-09
UAI	211							
Grid	21	3143	2	9379	2	2	2	3,74E-08
ImageAlign	10	191	70	1819	2	6218	37	0,0058
Linkage	22	917	5	1560	4	13	13	2,23E-07
ObjDetect	37	60	17	1830	2	34220	0	1
ProteinFold	21	486	267	2291	2	4698	273	0,52
Segment	100	229	12	851	2	315	185	0,00016
WCSP	226							
Auction	170	140	2	3593	2	47707	57	0,0869
CELAR	16	126	44	641	2	837	46	0,228
Pedigree	10	1758	11	3247	3	70	70	3,96E-06
ProteinDsn	10	13	123	97	2	311	0	0,966
SPOT5	20	385	4	6603	3	35976	2900	0,0055

Table 1 The set of benchmarks where each line corresponds to a category of benchmarks (#inst: number of instances, $\overline{n}$: mean number of variables, $\overline{d}$: mean domain size, $\overline{e}$: mean number of cost functions, $\overline{r}$: mean arity of cost functions, $\overline{c}$: mean number of triangles, $\overline{c'}$: mean number of triangles used by restricted HOCs at the root of the search tree, and $\overline{\text{dens}}$: mean triangle density.)

– MaxCSP: contains unsatisfiable binary CSP problems with constraints defined in extension, including BlackHole, Langford, Quasi-group completion problem,graph coloring, random composed, and random Geometric.

– UAI: consists of Markov Random Field problems that are collected from the Probabilistic Inference Challenge 2011[2] and Genetic Linkage Analysis problems[10].
– MaxSAT: contains Max-SAT problems that are collected from the Max-SAT Evaluation[3],
– CVPR: contains MRF instances from the Computer Vision and Pattern Recognition (CVPR) OpenGM2 benchmark[4]

Number of solved instances Table 2 reports the number of instances per category of benchmarks that are solved by each consistency (EDAC and HOCs implemented in three usecases). The block of 3 green lines shows that in general

– HOCs^2 (resHOCs used for preprocessing) have the best behavior: solve up to 1,02% more instances than EDAC, 4,69% than HOCs^1 and 4,02% HOCs^3.
– HOCs^3 (resHOCs used for both preprocessing and search) are better than HOCs^1 (HOCs used for preprocessing) but both solve less instances than EDAC by a factor up to 2,97% and 4,19% respectively.

For all usecases, HOCs^1 (especially EDmaxRPC when used for preprocessing) are better than EDAC on ChineseChars and GeomSurf7. (1) While EDAC cannot solve any ChineseChars instance (the same for all the other solvers reported in [2] as well as VAC) every HOC can solve a certain number of instances (at least 8 by PIC and at most 16 by EDmaxRPC). (2) Similarly to ChineseChars, HOCs^1 solve up to 5% instances more than EDAC on CVPR/GeomSurf-7. The advantage of HOCs on these problems are more clearly shown in Table 3: many instances cannot be solved in 1 hour by EDAC but can be solved by HOCs in less than 100s The restricted versions (HOCs^2,HOCs^3) slightly decrease the efficiency of HOCs (HOCs^1) on these problems.

However, HOCs^1 are worse than EDAC especially on Geometric, Max-Clique, PackupWei, ProteinFold, Auction and CELAR, on which the number of instances solved by HOCs^1 decreases by a factor up to 5,5%, 64%, 11,5%, 47%, 25% and 75%. Except for PackupWei, these problems are characterized by a very large mean triangle density, that is respectively 0.086, 0.079, 0.52, 0.0869 and 0.228. For these problems, the restricted versions can significantly improve the efficiency of HOCs and give results comparable to EDAC, thanks to the reduction in the number of instances . On the overall set of benchmarks, the usage of restricted HOCs for both preprcessing and search (HOCs^3) are less efficient than the usage for only preprocessing (HOCs^2).

[2] http://www.cs.huji.ac.il/project/PASCAL/realBoard.php
[3] http://maxsat.ia.udl.cat:81/13/benchmarks/
[4] http://hci.iwr.uni-heidelberg.de/opengm2

Table 2: The number of instances per category solved in less than 1200 seconds (1 hour for CVPR group). Each block of either one or three lines corresponds to a category of benchmarks whose the name and size are given in the two first columns. The number of instances per category solved by EDAC and HOCs are respectively given in the 3rd and the 8 last columns. Three lines of the combined blocks correspond respectively to three usecases: HOCs^1, HOCs^2 and HOCs^3. The blocks of a singleton line with name in bold mean that resHOCs are replaced by EDAC and their two usecases (HOCs^2, HOCs^3) give the same result as EDAC. The remaining blocks of a singleton line mean that HOCs^1, HOCs^2 and HOCs^3 give the same result as EDAC.

Problems	inst	EDAC	PIC	DPIC	FDPIC	EDPIC	maxRPC	DmaxRPC	FdmaxRPC	EdmaxRPC
summary	2820	2053	1972 2051 2013	1980 2055 2031	1979 2060 2030	1979 2058 2018	1967 2059 1993	1982 2069 2010	1980 2072 1992	1981 **2074** 1998
ChineseChars	100	0	8 9 9	8 7 9	10 9 10	10 10 10	10 14 11	9 10 10	10 13 12	**16** 15 11
GeomSurf-7	300	281	280 280 278	287 284 283	285 288 289	288 290 287	281 280 273	292 292 285	292 292 281	**295** 294 282
Coloring	22	17	**18** 17 17	**18** 17 17	17 17 16	17 17 16	**18** 17 17	**18** 17 16	**18** 17 16	**18** 17 16
Geometric	100	91	88 92 90	87 91 90	87 92 90	86 92 86	87 92 87	87 **93** 87	86 91 88	86 92 86
QCP	60	**14**	**14** **14** **14**	**14** **14** **14**	**14** **14** **14**	**14** **14** **14**	**14** **14** 13	**14** **14** 13	**14** **14** 13	**14** **14** **14**
Haplotyping	100	1	1 1 1	1 1 1	1 1 1	1 1 1	1 **2** 1	**2** **2** 1	**2** **2** 1	1 **2** 1
MaxClique	62	33	15 28 24	14 29 27	15 **30** 24	14 29 26	13 29 23	12 29 23	14 **30** 22	13 **30** 22
MIPLib	12	3	**3** **3** 2	**3** **3** 2	**3** **3** 2	**3** **3** 2	**3** **3** 2	**3** **3** 2	**3** **3** 2	**3** **3** 2
PackupWei	99	52	**48** **48** 41	47 46 39	47 **48** 42	47 47 40	47 **48** 41	46 47 39	47 47 40	47 47 40
Upgrad	100	**100**	**100** 96 92	**100** **100** **100**	**100** 96 94	98 92 92	**100** 97 89	**100** 99 96	99 98 93	98 97 91
ImageAlignment	10	**10**	7 **10** **10**	9 **10** **10**	7 **10** **10**	7 **10** **10**	6 **10** **10**	7 **10** **10**	5 **10** **10**	5 **10** **10**
Linkage	22	13	13 14 11	13 13 10	13 14 11	14 14 10	14 14 9	13 14 10	**15** **15** 10	**15** **15** 9
ProteinFold	21	19	10	10	10	10	10	10	10	10

Continued on next page

Table 2 – *Continued from previous page*

Problems	inst	EDAC	PIC	DPIC	FDPIC	EDPIC	maxRPC	DmaxRPC	FdmaxRPC	EdmaxRPC
			20 **20**	**20** **20**	**20** **20**	**20** **20**	**20** **20**	**20** **20**	**20** 19	**20** **20**
Segment	100	**100**	**100** 99 98	**100** **100** **100**	**100** **100** 99	**100** **100** 98	**100** 99 98	**100** 99 **100**	**100** **100** 98	**100** **100** 98
Auction	170	166	126 **167** 154	128 **167** 156	130 166 156	129 **167** 154	125 **167** 147	129 **167** 146	129 **167** 146	125 165 144
CELAR	16	**12**	4 **12** 11	4 **12** **12**	3 11 11	3 11 11	3 **12** 11	3 **12** 11	3 **12** 11	3 **12** 11
Matching	4	**4**	**4**	**4**	**4**	**4**	2	**4**	0	0
ProteinDsn	10	**9**	5	5	5	5	5	5	5	4
Composed	80	80	80	80	80	80	80	80	80	80
Langford	4	2	2	2	2	2	2	2	2	2
ObjDetect	37	0	0	0	0	0	0	0	0	0
ColorSeg	21	0	0	0	0	0	0	0	0	0
GeomSurf-3	300	300	300	300	300	300	300	300	300	300
InPainting	4	1	1	1	1	1	1	1	1	1
MatchingSte	2	0	0	0	0	0	0	0	0	0
ObjectSeg	5	0	0	0	0	0	0	0	0	0
PhotoMont	2	0	0	0	0	0	0	0	0	0
SceneDecp	715	715	715	715	715	715	715	715	715	715
BlackHole	37	10	10	10	10	10	10	10	10	10
EHI	200	0	0	0	0	0	0	0	0	0
PlanWithPre	29	6	6	6	6	6	6	6	6	6
TimeTabling	25	0	0	0	0	0	0	0	0	0
Grid	21	0	0	0	0	0	0	0	0	0
Pedigree	10	10	10	10	10	10	10	10	10	10
SPOT5	20	4	4	4	4	5	4	4	4	4

Number of backtracks Figure 17 presents the mean number of backtracks, computed on the overall set of benchmarks, that EDAC and HOCs implemented in 3 usescases need for the search. It shows that the number of backtracks is consistent with the strength of the consistencies. For 3 usescasess, HOCs use less backtracks than EDAC where HOCs^1 (HOCs for preprcessing) are the best. Compared to EDAC, (1) HOCs^1 reduce 35% in the number of backtracks (2) The restricted versions HOCs^2 (restrcited HOCs for preprcessing) only can slightly decrease the number of backtracks because of their reduced strength, where on many categories of benchmarks they are replaced by or become similar to EDAC with a significantly reduced number of triangles (3) The restricted versions HOCs^3 (restrcited HOCs for both preprcessing and search) use a number of backtracks smaller than HOCs^2 (thanks to the process of triangles of variables during search) but larger than HOCs^1 (because of their significant reduced strength).

Solving time Figure 18 presents the accumulated solving time of HOCs in three usecases. Each sub-figure corresponds to a consistency and each line corresponds to an usecase. We observe that (1) HOCs^2 are the fastest and HOCs^3 are the slowest. (2) The difference in solving time of 3 usecases are consistent with the strength of consistencies. Precisely, compared to HOCs^1:

- the total average time of HOCs^2 decreases by a factor going from 1.4 to 2.1 (for resPICs) or from from 1.7 to 3.1 (resMaxRPCs).
- the total average time of HOC^3 increases (except for PIC, DPIC, FD-PIC, FDmaxRPC which create very large speed-ups on hard instances ChineseChars).

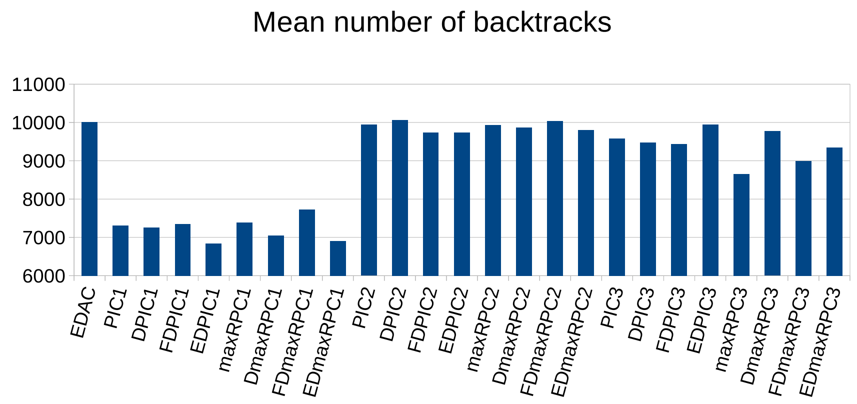


Fig. 17 The mean number of backtracks, computed on the overall set of benchmarks, that are used by EDAC and HOCs in three usecases

In summary, ChineseChars and GeomSurf7 are two favorables cases for the usage of HOCs. On these problems, (1) HOCs when used for preprcessing can solve more instances in less time than EDAC and their restricted versions slightly reduction the advantage of HOCs. However, on the problems having large triangle density, HOCs becomes significantly slower and thus solve less instances than EDAC. In this cases, the retricted versions when used for preprcessing allow to improve the results. We should not use the restricted HOCs for preprcessing and search that always behave worse than the usage for only preprcessing, on the overall set of benchmarks.

7 Conclusion

In this paper, we have proposed a group of high order consistencies that are an extension of hard RPC, PIC and maxRPC to WCSPs. The new consistencies are strictly stronger than EDAC in the sense that they provide lower bounds much better than EDAC. This improvement in lower bound is important for

Problem	EDAC	PIC	DPIC	FDPIC	EDPIC	maxRPC	DmaxRPC	FdmaxRPC	EdmaxRPC
ChineseChars									
TST_0012_88_103	-	195	575	34	41	**21**	119	27	44
TST_0020_96_94	-	-	2647	249	260	363	1709	**158**	**158**
TST_0024_88_126	-	-	-	-	-	-	-	-	**662**
TST_0027_88_109	-	-	-	-	-	-	-	-	**865**
TST_0041_88_96	-	-	-	3149	1569	269	-	760	**114**
TST_0047_112_121	-	215	83	23	49	**18**	26	28	64
TST_0052_96_107	-	1230	3564	1842	154	77	690	183	**46**
TST_0059_104_73	-	130	93	37	32	**11**	28	20	33
TST_0067_96_121	-	201	590	117	143	47	151	**41**	76
TST_0070_88_96	-	460	2194	392	158	56	210	99	73
TST_0084_120_115	-	-	-	-	-	-	-	-	**416**
TST_0087_88_124	-	-	-	-	-	-	-	-	**1910**
TST_0089_72_92	-	-	-	-	-	-	-	-	1148
TST_0099_72_105	-	502	2012	112	101	30	347	**65**	75
TST_0100_80_102	-	1199	-	591	532	227	1577	402	**78**
GeomSurf-7									
gm113	1487	-	349	254	486	678	166	**95**	138
gm125	-	-	1877	2938	-	-	344	**274**	2516
gm126	-	-	2135	-	-	-	1196	**119**	201
gm144	-	-	-	-	2806	-	2770	**1461**	1481
gm157	-	-	1914	2173	-	-	403	438	**262**
gm169	-	-	-	-	3146	-	2108	2971	**962**
gm179	-	1431	366	806	137	-	74	72	**67**
gm186	-	-	-	-	1674	-	951	842	**223**
gm187	-	-	2206	365	281	-	685	600	**182**
gm189	-	-	-	-	-	-	-	-	**2961**
gm223	-	-	2383	-	922	-	477	**426**	1473
gm246	-	-	-	-	-	-	-	-	**1744**
gm256	-	-	-	-	-	-	-	-	**2291**
gm25	1490	-	1387	-	653	2880	279	**171**	395
gm269	-	-	-	1656	**452**	-	1046	2948	1182
gm275	-	-	-	-	-	-	1180	2664	**568**

Table 3 The solving time (in seconds) for a subset of benchmarks. This subset contains only ChineseChars and GeomSurf-7 instances which respectively can and cannot be solved by one of consistencies in 1 hour. "–" means that the problem cannot be solved. Best results are in bold.

accelerating the search, that is shown by the experimental results. High order consistencies, especially EDmaxRPC, are efficient on grid graphs such as ChineseChars, GeomSurf-7 in the sense that they solve more instances in less time than EDAC. However, high order consistencies have not a good behavior on graphs having a large triangle density.

We also have proposed a restrcited version of high order consistencies. The best approach for solving WCSPs seems to be to use this restricted version for pre-processing, not for search. By limiting the number of triangles to be processed, the restricted versions still have a good behavior on favorable prob-

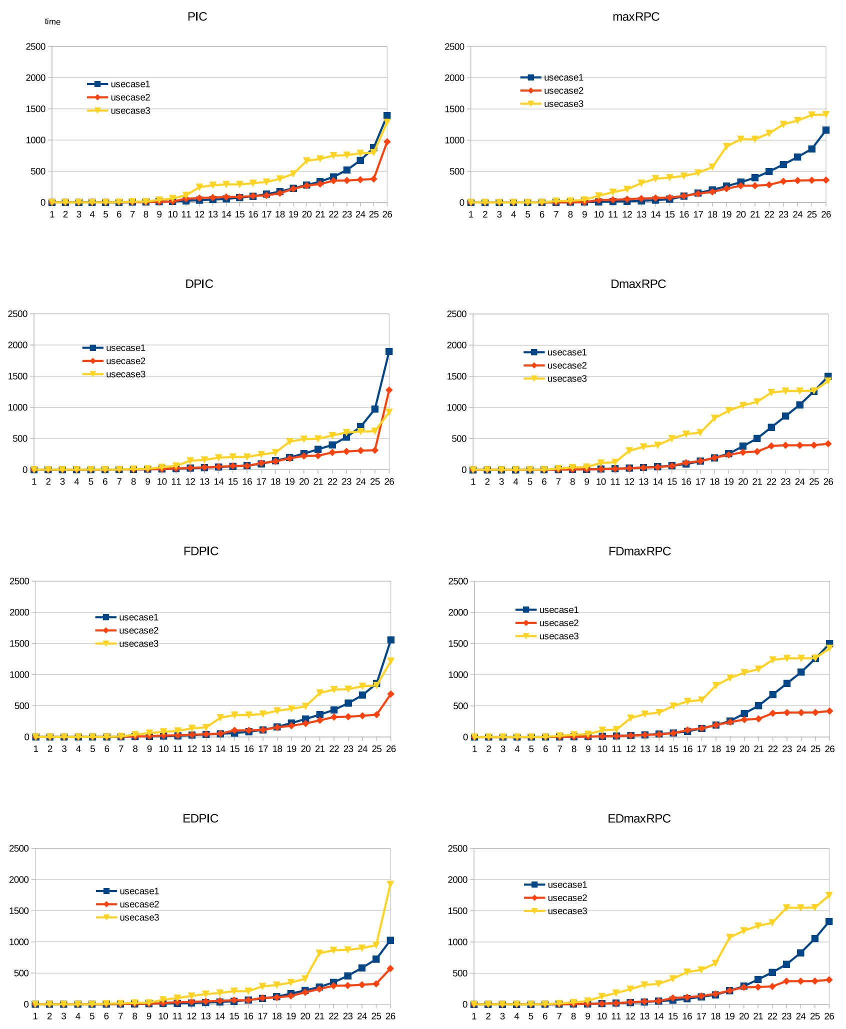


Fig. 18 Accumulated mean solving time taken by HOCs in three usecases. Each sub-figure corresponds to a high order consistency and the three lines in it correspond to three usecases: HOCs^1 (usecase1), HOCs^1(usecase2) and HOCs^1(usecase3). Axis X represents the categories and Axis Y represents the solving time (in seconds) that is accumulated on each category of benchmarks. For each consistency (i.e., each sub-figure), categories of benchmarks are arranged w.r.t the solving time of usecase1.

lems while providing a result better than the original ones and comparable to EDAC on unfavorable cases.

References

1. Allouche, D., Bessiere, C., .Boizumault, P., Givry, S., Gutierrez, P., Loudni, S., Metivier, J., Schiex, T.: Decomposing global cost functions. In: Proc. of AAAI (2012)
2. Allouche, D., de Givry, S., Hurley, B., Katsirelos, G., O'Sullivan, B., Schiex, T.: Une comparaison de logiciels d'optimisation sur une large collection de modles graphiques. In: Proc. of JFPC-14 (2014)
3. Bensana, E., Lemaître, M., Verfaillie, G.: Earth observation satellite management. Constraints **4**(3), 293–299 (1999)
4. Berlandier, P.: Improving domain filtering using restricted path consistency. In: Proceedings IEEE Conference on Artificial Intelligenece and Applications (CAIA'95) (1995)
5. Cabon, B., de Givry, S., Lobjois, L., Schiex, T., Warners, J.: Radio link frequency assignment. Constraints Journal **4**, 79–89 (1999)
6. Cooper, M.C.: High-order consistency in Valued Constraint Satisfaction. Constraints **10**, 283–305 (2005)
7. Cooper, M.C., Schiex, T.: Arc consistency for soft constraints **154**(1-2), 199–227 (2004)
8. Debruyne, R., Bessière, C.: From restricted path consistency to max-restricted path consistency. In: Proc. of CP'97, no. 1330 in LNCS, pp. 312–326. Springer-Verlag, Linz, Austria (1997)
9. Dehani, D., Lecoutre, C., Roussel, O.: Extension des cohrences wcsps aux tuples. In: Proc. of JFPC-13 (2013)
10. Favier, A., de Givry, S., Legarra, A., Schiex, T.: Pairwise decomposition for combinatorial optimization in graphical models. In: Proc. of IJCAI'11. Barcelona, Spain (2011)
11. Freuder, E.C.: A sufficient condition for backtrack-bounded search. Journal of the ACM **32**(14), 755–761 (1985)
12. Larrosa, J.: On arc and node consistency in weighted CSP. In: Proc. AAAI'02, pp. 48–53. Edmondton, (CA) (2002)
13. Larrosa, J., de Givry, S., Heras, F., Zytnicki, M.: Existential arc consistency: getting closer to full arc consistency in weighted CSPs. pp. 84–89 (2005)
14. Larrosa, J., Heras, F., de Givry, S.: A logical approach to efficient max-sat solving. Artif. Intell. **172**(2-3), 204–233 (2008)
15. Sánchez, M., de Givry, S., Schiex, T.: Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. Constraints **13**(1-2), 130–154 (2008)
16. Schiex, T.: Arc consistency for soft constraints. In: Principles and Practice of Constraint Programming - CP 2000, *LNCS*, vol. 1894, pp. 411–424. Singapore (2000)
17. Schiex, T., Fargier, H., Verfaillie, G.: Valued constraint satisfaction problems: hard and easy problems. pp. 631–637 (1995)