Conservative Dual Consistency

Christophe Lecoutre and **Stéphane Cardon** and **Julien Vion**
CRIL−CNRS FRE 2499
Université d'Artois
62307 Lens cedex, France
$\{lecoutre, cardon, vion\}$@cril.univ-artois.fr

Abstract

Consistencies are properties of Constraint Networks (CNs) that can be exploited in order to make inferences. When a significant amount of such inferences can be performed, CNs are much easier to solve. In this paper, we interest ourselves in relation filtering consistencies for binary constraints, i.e. consistencies that allow to identify inconsistent pairs of values. We propose a new consistency called Dual Consistency (DC) and relate it to Path Consistency (PC). We show that Conservative DC (CDC, i.e. DC with only relations associated with the constraints of the network considered) is more powerful, in terms of filtering, than Conservative PC (CPC). Following the approach of Mac Gregor, we introduce an algorithm to establish (strong) CDC with a very low worst-case space complexity. Even if the relative efficiency of the algorithm introduced to establish (strong) CDC partly depends on the density of the constraint graph, the experiments we have conducted show that, on many series of CSP instances, CDC is largely faster than CPC (up to more than one order of magnitude). Besides, we have observed that enforcing CDC in a preprocessing stage can significantly speed up the resolution of hard structured instances.

Introduction

Consistencies are properties of Constraint Networks (CNs) that can be exploited (enforced) to make inferences. By simplifying the problem, this permits to reduce the search space so that the CNs are much easier to solve (i.e. find a solution or prove that none exists). Most of the current Constraint Programming solvers interleave inference and search.

Currently, the most successful consistencies are *domain filtering consistencies* (Debruyne & Bessiere 2001), which allow to identify inconsistent values that can be removed from the domains of variables. For binary constraints, one can cite Arc Consistency (AC) and Singleton Arc Consistency (SAC) (Bessiere & Debruyne 2005). Generalized Arc Consistency (GAC) and Pairwise Inverse Consistency (PWIC) (Stergiou & Walsh 2006) hold on non-binary constraints. Another class of consistencies are those which allow to identify inconsistent pairs of values. They can be called *relation filtering consistencies* – but should not be confused with relational consistencies (Dechter & van Beek 1997). The most famous one is Path Consistency (PC). A pair of values for a pair of variables is path-consistent iff it can be extended to a consistent instantiation of any third variable. Many algorithms have been proposed to enforce PC on a given CN: PC4 (Mohr & Henderson 1986), PC8 (Chmeiss & Jégou 1998), PC2001 (Bessiere *et al.* 2005), etc.

Path consistency, and more generally relation filtering consistencies, are somewhat neglected by developers of Constraint Programming systems. The main reason is that exploiting such consistencies involves modifying the relations associated with the constraints, and more importantly, modifying the structure of the constraint graph. Indeed, when a pair of values is identified as being path-inconsistent, it has to be removed from the network. When no constraint binding the two involved variables exists in the CN, a new one has to be inserted (consequently, changing the constraint graph). For example, the CN that represents the instance $scen$-11 of the Radio-Frequency Assignment Problem (RLFAP) involves 680 variables and $4,103$ constraints. In the worst-case, enforcing PC on this network will entail the creation of $C^2_{680} - 4,103 = 226,757$ new constraints, which can really be counter-productive both in time and space.

However, it is possible to avoid the main drawback of PC by adopting a *conservative* approach. It simply means that we can limit our attention to existing constraints when looking for inconsistent pairs of values. This is called Conservative Path Consistency (CPC) (Debruyne 1999). We define a new relation filtering consistency, called Dual Consistency (DC), which exploits the outcome of AC enforcement. We show that PC is stronger than Conservative DC (CDC) which itself is stronger than CPC – CDC can filter out conservatively more inconsistent pairs of values than CPC. We propose an algorithm that establishes (strong) CDC, which can be seen as an adaptation of the one proposed in (McGregor 1979). It requires no specific data structure (except for those of the underlying AC algorithm) and is expected to quickly converge towards a unique fix-point.

Constraint Networks and Consistencies

A Constraint Network (CN) P is a pair $(\mathscr{X}, \mathscr{C})$ where $\mathscr{X}$ is a finite set of n variables and $\mathscr{C}$ a finite set of e constraints. Each variable $X \in \mathscr{X}$ has an associated domain, denoted $dom^P(X)$, which represents the set of values allowed for

X. Each constraint $C \in \mathscr{C}$ involves a subset of variables of $\mathscr{X}$, called scope and denoted $scp(C)$, and has an associated relation denoted $rel^P(C)$, which represents the set of tuples allowed for the variables of its scope. When possible, we will write $dom(X)$ and $rel(C)$ instead of $dom^P(X)$ and $rel^P(C)$. X_a denotes a pair (X, a) with $X \in \mathscr{X}$ and $a \in dom(X)$ and we will say that X_a is a value of P. λ denotes the number of allowed tuples over all constraints of P (with $\lambda = \sum_{C \in \mathscr{C}} |rel(C)|$), and K denotes the number of 3-cliques in P. If P and Q are two CNs defined on the same sets of variables $\mathscr{X}$ and constraints $\mathscr{C}$, then we will write $P \leq Q$ iff $\forall X \in \mathscr{X}$, $dom^P(X) \subseteq dom^Q(X)$ and $\forall C \in \mathscr{C}$ $rel^P(C) \subseteq rel^Q(C)$. $P < Q$ iff $P \leq Q$ and $\exists X \in \mathscr{X} \mid dom^P(X) \subset dom^Q(X)$ or $\exists C \in \mathscr{C} \mid rel^P(C) \subset rel^Q(C)$. A binary constraint is a constraint which only involves two variables. In the following, we will restrict our attention to binary networks, i.e., networks that only involve binary constraints. Further, without any loss of generality, we will consider that the same scope cannot be shared by two distinct constraints. The density D of a binary CN is then defined as the ratio e/C_n^2.

A solution to a constraint network is an assignment of values to all the variables such that all the constraints are satisfied. A constraint network is said to be satisfiable iff it admits at least one solution. The Constraint Satisfaction Problem (CSP) is the NP-complete task of determining whether a given constraint network is satisfiable. Constraint networks can be characterized by some properties called consistencies. The usual way to exploit them is to enforce them on a CN while preserving the set of solutions. It then consists in identifying and removing some inconsistent values (e.g. with arc consistency), some inconsistent pairs of values (e.g. with path consistency), etc. Here, "inconsistent" means that the identified values, pairs of values, etc. correspond to nogoods, i.e. cannot participate to any solution. We start introducing the consistencies we are interested in at the level of pairs of values. From now on, we will consider a binary constraint network $P = (\mathscr{X}, \mathscr{C})$.

Definition 1. A pair (X_a, Y_b) of values of P s.t. $X \neq Y$ is:

- arc-consistent (AC) iff either $\nexists C \in \mathscr{C} \mid scp(C) = \{X, Y\}$ or $(X_a, Y_b) \in rel(C)$.
- path-consistent (PC) iff (X_a, Y_b) is AC and $\forall Z \in \mathscr{X} \mid Z \neq X \wedge Z \neq Y$, $\exists c \in dom(Z)$ such that (X_a, Z_c) is AC and (Y_b, Z_c) is AC.
- conservative path-consistent (CPC) iff either $\nexists C \in \mathscr{C} \mid scp(C) = \{X, Y\}$ or (X_a, Y_b) is PC.

We can now introduce Arc Consistency (AC), Path Consistency (PC) and Conservative Path Consistency (CPC (Debruyne 1999)) w.r.t. a CN.

Definition 2. A value X_a of P is AC iff $\forall Y (\neq X) \in \mathscr{X}$, $\exists b \in dom(Y) \mid (X_a, Y_b)$ is AC. P is AC iff $\forall X \in \mathscr{X}$, $dom(X) \neq \emptyset$ and $\forall a \in dom(X)$, X_a is AC.

Definition 3. A pair (X, Y) of distinct variables of $\mathscr{X}$, is PC (resp. CPC) iff $\forall a \in dom(X)$, $\forall b \in dom(Y)$, (X_a, Y_b) is PC (resp. CPC). P is PC (resp. CPC) iff any pair of distinct variables of $\mathscr{X}$ is PC (resp. CPC).

All consistencies ϕ considered in this paper admit the following property: for any network P, there exists a greatest subnetwork of P which is ϕ-consistent, denoted by $\phi(P)$, and it is possible to compute it in polynomial time. For example, $AC(P)$ will denote the constraint network obtained after enforcing AC on P. $AC(P)$ is such that all values of P that are not arc-consistent have been removed. Remark that if any variable in $AC(P)$ has an empty domain, P is unsatisfiable. We will denote this by $AC(P) = \bot$. For any value X_a, we will write $X_a \in AC(P)$ iff $a \in dom^{AC(P)}(X)$ (note that $X_a \notin \bot$). Finally, $P|_{X=a}$ represents the network obtained from P by restricting the domain of X to the singleton $\{a\}$. We can now introduce Singleton Arc Consistency (SAC) and a new consistency, called Dual Consistency (DC).

Definition 4. A value X_a of P is SAC iff $AC(P|_{X=a}) \neq \bot$. P is SAC iff $\forall X \in \mathscr{X}$, $dom(X) \neq \emptyset$ and $\forall a \in dom(X)$, X_a is SAC.

Definition 5. A pair (X_a, Y_b) of values of P s.t. $X \neq Y$ is:

- dual-consistent (DC) iff $Y_b \in AC(P_{|X=a})$ and $X_a \in AC(P_{|Y=b})$.
- conservative dual-consistent (CDC) iff either $\nexists C \in \mathscr{C} \mid scp(C) = \{X, Y\}$ or (X_a, Y_b) is DC.

Definition 6. A pair (X, Y) of distinct variables of $\mathscr{X}$, is DC (resp. CDC) iff $\forall a \in dom(X)$, $\forall b \in dom(Y)$, (X_a, Y_b) is DC (resp. CDC). P is DC (resp. CDC) iff any pair of distinct variables of $\mathscr{X}$ is DC (resp. CDC).

From any relation filtering consistency, it is possible to obtain a new consistency by additionally considering Arc Consistency. Classically, a network is strong path-consistent, denoted sPC, iff it is both arc-consistent and path-consistent. Similarly, we can define sCPC, sDC and sCDC.

Qualitative Study

In order to compare the pruning capability of the different consistencies that have been presented above, we need to introduce a pre-order relation as in (Debruyne & Bessiere 2001). A local consistency ϕ is stronger than another local consistency ψ, denoted $\phi \succeq \psi$, iff whenever ϕ holds on a CN P, ψ also holds on P. ϕ is strictly stronger than ψ, denoted $\phi \succ \psi$, iff $\phi \succeq \psi$ and there exists at least one CN P such that ϕ holds on P but not ψ. For any pair (ϕ, ψ) of consistencies mentioned in this paper, we have $\phi \succeq \psi$ iff for any CN P, $\phi(P) \geq \psi(P)$ and $\phi \succ \psi$ iff $\phi \succeq \psi$ and there exists a CN P such that $\phi(P) > \psi(P)$. Also, any consistency ϕ mentioned in this paper is monotone: it means that for any pair (P, Q) of CNs such that $P \geq Q$, we have $\phi(P) \geq \phi(Q)$. Finally, we define $\phi \circ \psi(P)$ as being $\phi(\psi(P))$ and $(\phi \circ \psi)^{n+1}(P)$ as being $\phi \circ \psi \circ (\phi \circ \psi)^n(P)$.

Surprisingly, DC appears to be equivalent to PC although it could be predicted since McGregor had already proposed an AC-based algorithm to establish sPC (McGregor 1979). We provide a direct proof in our context:

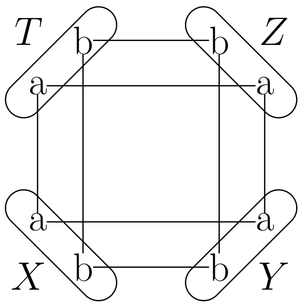

Figure 1: A network (no constraint binds X with Z and Y with T) that is CDC and sCDC but neither sPC nor PC. For example, (X_a, Z_b) is not PC.

Proposition 1. *DC = PC*

Proof. We will show that for any CN P, $DC(P) = PC(P)$. Let (X_a, Y_b) be a pair of values of P.

- If (X_a, Y_b) is not PC, either (X_a, Y_b) is not AC and then (X_a, Y_b) is not DC, or $\exists Z \in \mathscr{X} | \forall c \in dom(Z)$, (X_a, Z_c) is not AC or (Y_b, Z_c) is not AC. In this case, we know that $Y_b \notin AC(P_{|X=a})$ since after enforcing AC on $P_{|X=a}$, all values c of $dom(Z)$ are such that (X_a, Z_c) is AC. Necessarily, by hypothesis, all these values are incompatible with Y_b, which entails that b is removed from $dom(Y)$ when enforcing AC. Then, (X_a, Y_b) is not DC.
- If (X_a, Y_b) is not DC, it means that $Y_b \notin AC(P_{|X=a})$ (or $X_a \notin AC(P_{|Y=b})$). Let H(n) be the following recurrence hypothesis: if the number of revisions (i.e. the number of steps in an algorithm such as AC2001) to remove b from $dom(Y)$ when enforcing AC on $P_{|X=a}$ is less than or equal to n then (X_a, Y_b) does not belong to $PC(P)$. H(1) holds since, in this case, it means that (X_a, Y_b) is not AC. Let us suppose that H(n) is true and let us show that H($n + 1$) holds. If b is removed from $dom(Y)$ after $n + 1$ revisions while enforcing AC on $P_{|X=a}$, then it means that the last revision involves a constraint binding Y and another variable Z. Any value c of $dom(Z)$ that was supporting Y_b has been removed after at most n revisions. By hypothesis, it means that for all such values c, (X_a, Z_c) does not belong to $PC(P)$. We can then deduce at this step that (X_a, Y_b) is not PC. □

In the remaining of this section, we will study the relationships of the conservative variant of DC, namely CDC, with PC and CPC.

Proposition 2. *PC $\succ$ CDC.*

Proof. Clearly, DC $\succeq$ CDC. So, by Proposition 1, we obtain PC $\succeq$ CDC. Moreover, Figure 1 depicts a network (edges represent allowed tuples) which is CDC but not PC. □

Proposition 3. *CDC $\succ$ CPC.*

Proof. Similarly, as in the first part of the proof of Proposition 1, we can show that CDC $\succeq$ CPC. Moreover, Figure 2 depicts a network which is CPC but not CDC. Indeed, as there is no 3-clique, the network is trivially CPC. □

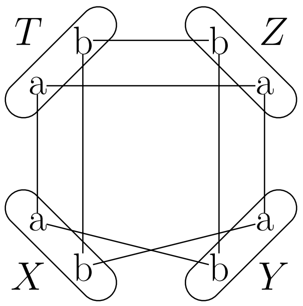

Figure 2: A network (no constraint binds X with Z and Y with T) that is CPC and sCPC but neither sCDC nor CDC. For example, (X_a, T_a) is not CDC as $AC(P_{|X=a}) = \bot$.

Before studying the relationships between sPC, sCDC and sCPC, let us remark that that enforcing AC (only once) on a PC network is sufficient to obtain an sPC network. This well-known fact is also true for CDC. Actually, all arc-inconsistent values are completely isolated, as shown by the following proposition (whose proof is omitted).

Proposition 4. *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN which is PC (resp. CDC). A value X_a of P is not AC iff $\forall Y \in \mathscr{X}$ (resp. s.t. $\exists C \in \mathscr{C} \mid scp(C) = \{X, Y\}$), $\forall b \in dom(Y)$, (X_a, Y_b) is not PC (resp. CDC).*

Corollary 1. *Let P be a CN. $AC \circ PC(P) = sPC(P)$ and $AC \circ CDC(P) = sCDC(P)$.*

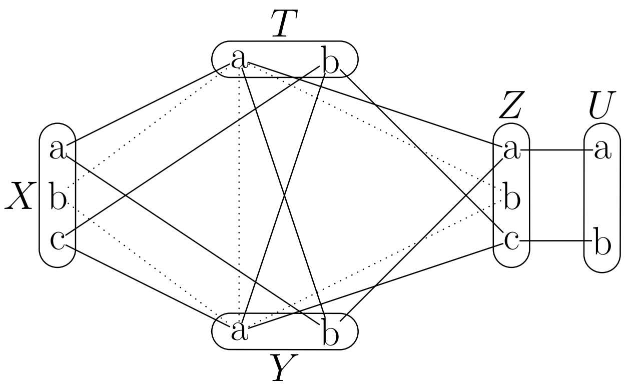

Figure 3: Pattern to show that $AC \circ CPC \neq sCPC$. No constraint binds X with Z.

Interestingly enough, the previous proposition does not hold for CPC. Consider the example depicted in Figure 3. Enforcing AC implies the removal of Z_b. If we enforce CPC, all tuples corresponding to dotted edges are removed. Next, if we enforce AC, the value X_b is removed. We can imagine the same pattern (X,Y,Z,T) occurring on the left of the graph starting from X. So, for any integer n, we can build a network P such that $(AC \circ CPC)^{n+1}(P) = sCPC(P)$ while $(AC \circ CPC)^n(P) \neq sCPC(P)$ (and such that the size of P grows polynomially with n).

Proposition 5. *sPC $\succ$ sCDC.*

Proof. For any CN P, $PC(P) \geq CDC(P)$ by Prop. 2. By monotony, we obtain $AC \circ PC(P) \geq AC \circ CDC(P)$. By Corollary 1, we obtain $sPC(P) \geq sCDC(P)$. Finally, Figure 1 shows a network that is sCDC but not sPC. □

Proposition 6. *sCDC $\succ$ sCPC.*

Proof. Let us first show the following recurrence hypothesis $H(n)$: for any network P, $AC \circ CDC(P) \geq (AC \circ CPC)^n(P)$. For $n = 1$, this is immediate since $CDC \succ CPC$. Now, let us suppose $H(n)$ and show that $H(n+1)$ holds. We have, by hypothesis, $AC \circ CDC(P) \geq (AC \circ CPC)^n(P)$. So, we obtain $CPC \circ AC \circ CDC(P) \geq CPC \circ (AC \circ CPC)^n(P)$ by monotony. As $AC \circ CDC(P) = sCDC(P)$, this network is CDC and then CPC. Hence, we have $CPC \circ AC \circ CDC(P) = AC \circ CDC(P) \geq CPC \circ (AC \circ CPC)^n(P)$. By monotony, we have $AC \circ AC \circ CDC(P) \geq AC \circ CPC \circ (AC \circ CPC)^n(P)$. We deduce $AC \circ CDC(P) \geq (AC \circ CPC)^{n+1}(P)$. As $sCPC = (AC \circ CPC)^n(P)$ for a given finite integer n, we have shown that $sCDC \succeq sCPC$. Finally, Figure 2 shows a network that is sCPC but not sCDC. □

The following proposition is immediate since any singleton arc-inconsistent value is removed by an algorithm that enforces sCDC.

Proposition 7. *sCDC $\succ$ SAC.*

Algorithm sCDC-1

The algorithm that we propose to establish sCDC is called sCDC-1. It performs successive singleton checks until a fix-point is reached. The description is given in the context of using an underlying coarse-grained AC algorithm, such as AC3 (Mackworth 1977), AC2001/3.1 (Bessiere *et al.* 2005) or AC3^{rm} (Lecoutre & Hemery 2007), with a variable-oriented propagation scheme. If $P = (\mathscr{X}, \mathscr{C})$, then $AC(P, Q)$ with $Q \subseteq \mathscr{X}$ means enforcing arc consistency on P from the given propagation set Q. For a description of AC, see, for instance, the function $propagateAC$ in (Bessiere & Debruyne 2005).

To enforce sCDC on a given network P, the function $sCDC$-1 is called (see Algorithm 1). AC is enforced first (line 1), and then, at each turn of the main loop, a different variable is considered: assuming here that $\mathscr{X}$ is ordered, $first(\mathscr{X})$ returns the first variable of $\mathscr{X}$ and $next\text{-}modulo(X, \mathscr{X})$ returns the variable which follows X in $\mathscr{X}$, or $first(\mathscr{X})$ if X is the last one. The call to $check$ at line 5 then tries to make some inferences from X. When any inference is performed, $check$ returns true and arc consistency is re-established (line 6). To manage termination, a marker initialized with a special value (line 3) and updated whenever there are some inferences w.r.t. the current variable X (line 7) is used.

Performing all sCDC inferences w.r.t. a variable X is achieved by calling the function $check$ (Algorithm 2). For each value a in the domain of X, AC is enforced on $P|_{X=a}$. If a is singleton arc-inconsistent, then a is removed from the domain of X (line 5). Otherwise (lines 8 to 12), for any

Algorithm 1: sCDC-1($P = (\mathscr{X}, \mathscr{C})$: CN)

```
1 $P \leftarrow AC(P, \mathscr{X})$
2 $X \leftarrow first(\mathscr{X})$
3 $marker \leftarrow X$
4 repeat
5     if $check(P, X)$ then
6         $P \leftarrow AC(P, \{X\})$
7         $marker \leftarrow X$
8     $X \leftarrow next\text{-}modulo(\mathscr{X}, X)$
9 until $X = marker$
```

Algorithm 2: check(P : CN, X : Variable) : Boolean

```
1 $modified \leftarrow false$
2 foreach $a \in dom^P(X)$ do
3     $P' \leftarrow AC(P|_{X=a}, \{X\})$
4     if $P' = \bot$ then
5         remove $a$ from $dom^P(X)$
6         $modified \leftarrow true$
7     else
8         foreach $C \in \mathscr{C} | X \in scp(C)$ do
9             let $Y$ be the second variable in $scp(C)$
10            foreach $b \in dom^P(Y) | b \notin dom^{P'}(Y)$ do
11                remove $(X_a, Y_b)$ from $rel^P(C)$
12                $modified \leftarrow true$
13 return $modified$
```

value Y_b present in P and removed in P' such that there exists a constraint binding X and Y, the tuple (X_a, Y_b) is removed from $rel(C)$.

There is a strong connection between the algorithm sCDC-1 and the algorithm proposed in (McGregor 1979) to establish sPC. Our algorithm can be seen as a refinement of Mac Gregor's since we can deal with CDC and have integrated two important modifications. First, AC is maintained during execution in order to start singleton checks with a propagation set composed of only one variable (it allows to avoid a lot of unnecessary revisions, in particular on sparse constraint graphs). Second, we handle termination through an enhanced mechanism. It is possible to reason about termination this way because for any variable X and any two values a and b in $dom(X)$, any inference concerning X_a (the removal of X_a or the removal of a tuple linking X_a to another value) has no impact on $P_{X=b}$, and vice-versa.

Proposition 8. *The algorithm sCDC-1 enforces sCDC.*

Proof. First, it is immediate that any inference performed by sCDC-1 is correct. Completeness is guaranteed by the following invariant: when $P' = AC(P|_{X=a}, \{X\})$ is performed at line 3 of Algorithm 2, we have $P' = AC(P|_{X=a}, \mathscr{X})$. The reason is that the network is maintained arc-consistent whenever a modification is performed (line 6 of Algorithm 1) and that any inference performed w.r.t. a value X_a has no impact on $P|_{X=b}$, where b is any other value in the domain of the variable X. □

Proposition 9. *The worst-case time complexity of sCDC-1 is $O(\lambda end^3)$ and its worst-case space complexity is $O(ed^2)$.*

Proof. In the worst case, the function *check* can be called λ times for a given variable, since between two successive calls, at least one tuple must be removed from a relation. An optimal $O(ed^2)$ AC algorithm such as AC2001 may be used on line 3, and removing inconsistent tuples (lines 8 to 12) is $O(nd)$. As $n < e$ is assumed, a call to *check* is thus in $O(d(ed^2 + nd)) = O(ed^3)$. Thus, we obtain $O(\lambda end^3)$ for sCDC-1. In terms of space, one needs to store domains and relations: this is in $O(nd + ed^2) = O(ed^2)$ (but only $O(nd + e)$ if constraints are initially given in intention). The only data structures used by sCDC-1 are those used by the underlying AC algorithm. For AC2001 or AC3^{rm}, it is $O(ed)$. The overall worst-case space complexity is then $O(ed^2)$. □

As λ is bounded by $O(ed^2)$, sCDC-1 may be up to $O(e^2nd^5)$. This seems to be rather high, but our opinion is that sCDC-1 quickly reaches a fix-point (i.e. the number of times the function *check* is called for a given variable is very small) because inferences about inconsistent values and, especially pairs of values, can be immediately taken into account. The following corollary also indicates that the time wasted to apply sCDC-1 on a network which is already sCDC is quite limited provided that the size of the domains is not too high.

Corollary 2. *Applied to a sCDC network, the worst-case time complexity of sCDC-1 is $O(end^3)$.*

It is also interesting to compare sCDC-1 with algorithms enforcing sCPC. Due to lack of space, we only give a few words about the algorithms sCPC8 and sCPC2001 that can be directly derived from PC8 (Chmeiss & Jégou 1998) and PC2001 (Bessiere *et al.* 2005). First, the worst-case space complexity of these two algorithms is $O(ed^2)$ and $O((e + K)d^2)$, respectively. The initialization phase (we will not discuss about the propagation phase) of these two algorithms is the same: for each 3-clique of the network, check that any allowed tuple of a relation of this clique is PC. This is done in $O(Kd^3)$. From these observations, it appears that, the less dense the network is, the less the number of 3-cliques is and the less the space and time required by these algorithms are. On the other hand, if we consider dense networks (e tending to C_n^2), we can adjust the worst-case time complexity of the initialization phase to $O(n^3d^3)$. This is the same as the worst-case time complexity of a single pass (calling *check* for all variables only once) of sCDC-1. For sparse networks, sCDC-1 should then be slower than sCPC8 or sCPC2001, but for dense (or highly structured) networks, we do believe that sCDC-1, due to its capability to make inferences quickly, should be faster.

Our algorithm is also related to algorithms that establish singleton arc consistency like SAC-OPT and SAC-SDS (Bessiere & Debruyne 2005). SAC-OPT admits an optimal worst-case time complexity of $O(end^3)$ but a high worst-case space complexity of $O(end^2)$. SAC-SDS relaxes time optimality to save space: its worst-case time complexity is in $O(end^4)$ and its worst-case space complexity is in $O(n^2d^2)$.

	AC3^{rm}	SAC-SDS	sCPC8 / sCPC2001	sCDC-1
Langford (4 instances)				
cpu	0.22	0.46	4.02 / 4.94	0.52
λ	$105,854$	$105,769$	$75,727$	$75,727$
blackhole-4-13 (7 instances) ($K = 92,769$; $D = 20\%$)				
cpu	1.26	19.39	140.54 / $-$	46.91
λ	$8,206,320$	$8,206,320$	$8,206,320$	$7,702,906$
$\langle 40, 180, 84, 0.9\rangle$ (20 instances) ($K = 12$; $D = 10\%$)				
cpu	0.71	10.57	2.28 / 2.02	17.42
λ	$272,253$	$244,887$	$244,272$	$210,874$
$\langle 40, 8, 753, 0.1\rangle$ (20 instances) ($K = 8,860$; $D = 96\%$)				
cpu	0.16	0.21	0.62 / 0.69	0.20
λ	$43,320$	$43,320$	$43,318$	$43,318$
job-shop enddr1 (10 instances) ($K = 600$; $D = 21\%$)				
cpu	1.58	4.06	7.91 / 10.54	4.67
λ	$2,937,697$	$2,937,697$	$2,937,697$	$2,930,391$
RLFAP scens (11 instances)				
cpu	0.86	$-$	25.96 / $-$	3.47
λ	$1,674,286$	$-$	$1,471,132$	$1,469,286$

Table 1: Experimental results (*cpu* in seconds)

sCDC-1 has the advantage to prune more efficiently the search space since $sCDC \succ SAC$ while limiting the worst-case space complexity to $O(ed^2)$.

Experiments

In order to show the practical interest of the approach described in this paper, we have conducted an extensive experimentation on a i686 2.4GHz processor equipped with 1024 MiB RAM. We have compared the CPU time and the filtering level (the λ value of the network obtained after having applied the algorithm) of various algorithms used stand-alone (i.e. without search). These algorithms are (an optimized version for binary constraints of) AC3^{rm} (Lecoutre & Hemery 2007), SAC-SDS, sCPC8, sCPC2001 and sCDC-1. More precisely, we used AC $\circ$ CPC8 and AC $\circ$ CPC2001 as an approximation to establish sCPC as we observed that one pass was sufficient to reach sCPC most of the time.

We have first tested the different algorithms against different series of problems[1]. Constraints defined in intention (i.e. by a predicate) in some instances were converted in extension (this only had a significant impact on cpu time for large instances of *fapp* series). As expected (see Table 1), sCDC-1 filters more than the other algorithms: the smaller λ is, the more reduced the search space is. Except for the series $\langle 40, 180, 84, 0.9\rangle$, sCDC-1 is from two to eight times faster than sCPC8 and sCPC2001. Besides, sCDC-1 is almost as fast as SAC-SDS which in addition runs out of memory on some series (symbolized by $-$). The unstructured $\langle 40, 180, 84, 0.9\rangle$ binary random instances present only 12 3-cliques on average, which explains why it is cheap here to enforce sCPC.

Table 2 provides some other representative results, instance per instance. Once again, it appears that, except for the instance *fapp*-01-200-4, sCDC-1 largely outperforms the algorithms enforcing sCPC. There is a difference by one order of magnitude on instance *haystack*-40 and by almost two orders of magnitude on instance *knights*-50-5. The relative bad performance of sCDC-1, in terms of cpu time, on instance *fapp*-01-200-4 can once again be explained by the very small density (D is only 0.5%) and the small number

[1]`http://cpai.ucc.ie/06/Competition.html`

	AC3^{rm}	SAC-SDS	sCPC8 / sCPC2001	sCDC-1
driverlogw-09 ($K = 233,834$; $D = 8\%$)				
cpu	1.60	48.42	33.84 / 36.52	10.83
mem	14	87	59 / 155	23
λ	369,736	147,115	306,573	18,958
haystack-40 ($K = 395,200$; $D = 2\%$)				
cpu	9.64	–	580.48 / –	55.91
mem	19	–	209 / –	107
λ	48,670,518	–	48,670,518	48,670,518
knights-50-5 ($K = 10$; $D = 100\%$)				
cpu	12.38	34.43	1759 / –	21.49
mem	5	163	29 / –	19
λ	31,331,580	0	0	0
pigeons-50 ($K = 19,600$; $D = 100\%$)				
cpu	1.38	2.85	33.82 / 44.52	2.7
mem	2	12	9 / 636	5
λ	2,881,200	2,881,200	2,881,200	2,881,200
qcp-25-264-0 ($K = 43,670$; $D = 5\%$)				
cpu	2.28	6.08	8.15 / 10.49	2.08
mem	8	210	29 / 215	21
λ	77,234	77,234	76,937	76,937
qwh-25-235-0 ($K = 35,700$; $D = 4.5\%$)				
cpu	1.87	5.62	7.09 / 9.05	2.56
mem	7	183	26 / 173	19
λ	56,721	56,721	56,380	56,380
fapp01-200-4 ($K = 247$; $D = 0.5\%$)				
cpu	10.73	–	16.05 / 18.63	104.05
mem	15	–	22 / 254	17
λ	3,612,163	–	3,317,135	2,117,575
scen-11 ($K = 13,775$; $D = 1.7\%$)				
cpu	2.87	–	85.82 / 78.49	9.78
mem	5	–	22 / 426	16
λ	5,434,107	–	4,829,442	4,828,650

Table 2: Experimental results (mem in MiB)

$Instance$		MAC	$sCDC$+MAC
scen11-f8	cpu	8.08	14.31
	$nodes$	14,068	4,946
scen11-f5	cpu	259	225
	$nodes$	$1,327K$	$680K$
scen11-f3	cpu	2,338	1,725
	$nodes$	$12M$	$5,863K$
scen11-f2	cpu	7,521	5,872
	$nodes$	$37M$	$21M$
scen11-f1	cpu	17,409	13,136
	$nodes$	$93M$	$55M$

Table 3: Impact of sCDC at preprocessing on MAC

of 3-cliques. However, the improvement in terms of filtering is quite significant. Moreover, sCDC-1 is far less space-consuming than SAC and PC algorithms. The difference existing with AC3^{rm} comes from the fact that more relations can be shared by constraints when they are not modified.

Finally, we have compared the performance of the state-of-the-art generic algorithm MAC with and without sCDC enforcement at preprocessing on some difficult real-world instances of the Radio Link Frequency Assignment Problem (RLFAP). Even if some techniques (restarts, nogood recording, etc.) allow to solve more efficiently these instances, we do not employ them in order to observe the real impact (on search) of enforcing sCDC at preprocessing. Table 3 shows that for the hardest instances, sCDC at preprocessing pays off: MAC alone is about 40% slower than sCDC+MAC and visits almost twice more nodes.

Conclusion

General-purpose solvers in constraint satisfaction are usually built on top of a systematic or local search algorithm. Making as much inferences as possible during preprocessing can dramatically improve their efficiency: for instance, enforcing singleton arc consistency on structured networks before solving them often pays off. Unfortunately, current algorithms are not able to take into account large amounts of available information. Indeed, if a value Y_b is removed after having performed the assignment $X \leftarrow a$ and enforced arc consistency, we just learned that $X = a$ and $Y = b$ are not compatible.

In this paper, we have proposed to take such inferences into account in a conservative way, i.e. without adding new constraints. We have introduced a new consistency called Dual Consistency (DC) and have focused on its conservative variant CDC. It has been shown in particular that CDC is a relation filtering consistency which is stronger than conservative PC (CPC), and enforcing strong CDC (i.e. enforcing both CDC and AC) can be done in a quite natural way. The experimental results obtained from a wide range of problems clearly show the practical interest of CPC, in particular on hard dense problems.

Acknowledgments

This paper has been supported by the CNRS and the ANR “Planevo” project n^{o}JC05_41940.

References

Bessiere, C., and Debruyne, R. 2004. Theoretical analysis of singleton arc consistency. In *Proceedings of ECAI’04 workshop on modelling and solving problems with constraints*, 20–29.

Bessiere, C., and Debruyne, R. 2005. Optimal and suboptimal singleton arc consistency algorithms. In *Proceedings of IJCAI’05*, 54–59.

Bessiere, C.; Régin, J.; Yap, R.; and Zhang, Y. 2005. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence* 165(2):165–185.

Chmeiss, A., and Jégou, P. 1998. Efficient path consistency propagation. *International Journal on Artificial Intelligence Tools* 7(2):121–142.

Debruyne, R., and Bessiere, C. 1997. Some practical filtering techniques for the constraint satisfaction problem. In *Proceedings of IJCAI’97*, 412–417.

Debruyne, R., and Bessiere, C. 2001. Domain filtering consistencies. *Journal of Artificial Intelligence Research* 14:205–230.

Debruyne, R. 1999. A strong local consistency for constraint satisfaction. In *Proceedings of ICTAI’99*, 202–209.

Dechter, R., and van Beek, P. 1997. Local and global relational consistency. *Theoretical Computer Science* 173(1):283–308.

Lecoutre, C., and Hemery, F. 2007. A study of residual supports in arc consistency. In *Proc. of IJCAI’07*, 125–130.

Mackworth, A. 1977. Consistency in networks of relations. *Artificial Intelligence* 8(1):99–118.

McGregor, J. 1979. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Sciences* 19:229–250.

Mohr, R., and Henderson, T. 1986. Arc and path consistency revisited. *Artificial Intelligence* 28:225–233.

Stergiou, K., and Walsh, T. 2006. Inverse consistencies for non-binary constraints. In *Proc. of ECAI’06*, 153–157.