# Bound Consistencies for the discrete CSP

Christophe Lecoutre and Julien Vion

CRIL-CNRS FRE 2499,
Université d'Artois
Lens, France
$\{lecoutre, vion\}$@cril.univ-artois.fr

**Abstract.** Many works in the area of Constraint Programming have focused on inference, and more precisely, on filtering methods based on properties of constraint networks. Such properties are called domain filtering consistencies when they allow removing some inconsistent values from the domains of variables, and bound consistencies when they focus on bounds of domains. In this paper, we study the relationship between consistencies introduced with respect to discrete and continuous constraint networks, and experiment the effectiveness of exploiting bound consistencies on discrete instances.

## 1 Introduction

Many problems arising in Artificial Intelligence and Computer Science involve constraint satisfaction as an essential component. Such problems occur in numerous domains such as scheduling, planning, molecular biology and circuit design. The methods that have been developed for processing constraints can be classified into inference and search [13]. Inference is used to transform a problem into an equivalent form which is simpler than the original one while search is used to traverse the search space of the problem in order to find a solution. Problems involving constraints are usually represented by so-called constraint networks.

A constraint network is simply composed of a set of variables and of a set of constraints. Finding a solution to a constraint network involves assigning a value to each variable such that all constraints are satisfied. The Constraint Satisfaction Problem (CSP) is the task to determine whether or not a given constraint network, also called CSP instance, is satisfiable. It comes in two forms. The first one, called discrete or finite CSP, corresponds to constraint networks such that each variable takes its values in an associated discrete domain while the second one, called continuous or numeric CSP, corresponds to networks such that each variable takes its values in an associated continuous domain.

Many works in the area of Constraint Programming have focused on inference, and more precisely, on filtering methods based on properties of constraint networks. The idea is to exploit such properties in order to identify some no-goods where a no-good corresponds to a set of variable assignments that can not lead to any solution. Properties that allow identifying no-goods of size 1, which correspond to inconsistent values, are called domain filtering consistencies [12]. In this paper, we focus on domain filtering consistencies that have been introduced with respect to discrete and continuous CSP instances.

On the one hand, when dealing with discrete CSP instances, a usual approach to solve them is to use the MAC algorithm, i.e. the algorithm which maintains arc consistency during search. Arc consistency (AC) means that any value occurring in the associated domain of a variable $X$ admits at least a support in each constraint involving $X$. Recent works have shown that there exist promising alternatives to AC, namely, max-restricted path consistency (Max-RPC) [10] and singleton arc consistency (SAC) [11]. Max-RPC and SAC are stronger consistencies than AC, that is to say, they allow identifying more inconsistent values than AC does. Max-RPC holds when all values have at least one path consistent support on each constraint whereas SAC holds when the constraint network can be made arc consistent after any variable assignment. It can be useful to establish Max-RPC or SAC at pre-processing time (i.e. before search) [22, 12], but it seems that maintaining such a strong consistency during search does require some control about the effort performed at each step. In fact, it remains an open issue although recent advances [1, 5, 18] show it is a direction of future research.

On the other hand, when dealing with continuous CSP instances, one has to reason about intervals. For instance, it is possible to represent a domain by a finite set of (disjoint) continuous intervals and to propose some adaptations [16, 14] of the arc consistency enforcing algorithm which, otherwise, is subject to early quiescence and infinite iterations. However, it is more usual that domains are considered as convex, i.e. represented by a single interval. By restricting arc consistency with respect to the bounds of each (convex) domain, new consistencies can be introduced. The consistency that is based on an approximation (in order to maintain domains convex) of projection functions for the narrowing of domains is called 2B-consistency (2B) by [19] and hull-consistency by [2]. However, it requires, for each pair $(C,X)$ composed of a constraint $C$ and a variable $X$, the existence of two functions computing the min bound and the max bound of the set of values given by the projection over $X$ of the set of supports of $C$. When such functions can not be exhibited, it is necessary to perform some decomposition of the constraint system. Another consistency, called box-consistency [2], exploits interval arithmetic and does not require any constraint decomposition. It is known [9] that 2B and box-consistency match when no variable occurs several times in the expression of a constraint. It is also possible to define stronger consistencies than 2B or box-consistency by assuming that each variable is assigned, in turn, with the two bounds of its domain and by checking consistency when establishing 2B or box-consistency. Such consistencies are called 3B-consistency (3B) [19] and bound-consistency, respectively. Finally, it is possible to introduce a recursive definition of kB-consistencies [20] with $k \geq 2$.

In the following, we will define a consistency restricted to the bounds of the domains as a bound consistency. For example, 2B corresponds to bound AC while 3B is a relaxation of bound SAC. The aim of this paper is to study the practical effectiveness of exploiting bound consistencies with respect to discrete CSP instances, as even if 2B has been integrated into some constraint logic programming solvers [8, 15], we are not aware of any experimental comparison involving different bound consistencies wrt finite domains.

## 2 Domain Filtering Consistencies

In this section, we introduce some consistencies that allow removing some inconsistent values from the domains of a constraint network (CN). Such consistencies, called domain filtering consistencies in [12], share the nice property of not modifying the structure of the network.

**Definition 1.** *A Constraint Network $P$ is a pair $(\mathscr{X},\mathscr{C})$ where:*

- *$\mathscr{X} = \{X_1, \ldots, X_n\}$ is a finite set of $n$ variables such that each variable $X_i$ has an associated domain $dom(X_i)$ denoting the set of values allowed for $X_i$,*
- *$\mathscr{C} = \{C_1, \ldots, C_m\}$ is a finite set of $m$ constraints such that each constraint $C_j$ has an associated relation $rel(C_j)$ denoting the set of tuples allowed for the variables $vars(C_j) \subseteq \mathscr{X}$ involved in the constraint $C_j$.*

For any variable $X$, $\min(X)$ and $\max(X)$ will respectively denote the smallest and greatest values in $\text{dom}(X)$. Note that a value will usually refer to a pair $(X,a)$ with $X \in \mathscr{X}$ and $a \in \text{dom}(X)$. We will note $(X,a) \in P$ (respectively, $(X,a) \notin P$) iff $X \in \mathscr{X}$ and $a \in \text{dom}(X)$ (respectively, $a \notin \text{dom}(X)$).

A constraint network is said to be satisfiable iff it admits at least a solution. The Constraint Satisfaction Problem (CSP) is the NP-complete task of determining whether a given constraint network, also called CSP instance, is satisfiable. To solve a CSP instance, a depth-first search algorithm with backtracking can be applied, where at each step of the search, a variable assignment is performed followed by a filtering process called constraint propagation. Usually, constraint propagation algorithms are based on domain filtering consistencies, among which the most widely studied ones are called arc consistency, max-restricted path consistency and singleton arc consistency.

Arc Consistency (AC) is the basic property of constraint networks. It guarantees that each value occurs in at least a support of each constraint. Algorithms to establish AC entails removing all arc inconsistent values and can be classified into coarse-grained and fine-grained algorithms. Optimal worst-case time complexity to establish AC is $O(md^2)$ where $d$ is the size of the largest domain.

**Definition 2.** *Let $P = (\mathscr{X},\mathscr{C})$ be a CN, $X \in \mathscr{X}$ and $a \in dom(X)$. $(X,a)$ is arc consistent iff $\forall C \in \mathscr{C} | X \in vars(C)$, there exists a support of $(X,a)$ in $C$, i.e., a tuple $t \in rel(C)$ such that $t[X] = a$. $P$ is arc consistent iff $\forall X \in \mathscr{X}$, $dom(X) \neq \emptyset$ and $\forall a \in dom(X)$, $(X,a)$ is arc consistent.*

Max-Restricted Path Consistency (Max-RPC) can be seen as a generalization of restricted path consistency [3] and k-restricted path consistency [10] and also as a restriction of path consistency [21]. It is defined with respect to binary constraint networks, i.e. networks that only involves binary constraints. Max-RPC guarantees that each value can be found a path in each 3-clique of the network. Optimal worst-case time complexity to establish Max-RPC is $O(mn + md^2 + cd^3)$ where $c$ denotes the number of 3-cliques in the constraint network.

**Definition 3.** *Let $P = (\mathscr{X}, \mathscr{C})$ be a binary CN, $X_i \in \mathscr{X}$ and $a \in dom(X_i)$. $(X_i, a)$ is max-restricted path consistent iff $\forall C_{ij} \in \mathscr{C}$, $\exists b \in dom(X_j)$ s.t. $(a, b) \in rel(C_{ij})$ and $\forall X_k \in \mathscr{X} | C_{ik} \in \mathscr{C} \wedge C_{jk} \in \mathscr{C}$, $\exists c \in dom(X_k)$ s.t. $(a, c) \in rel(C_{ik}) \wedge (b, c) \in rel(C_{jk})$. $P$ is max-restricted path consistent iff $\forall X_i \in \mathscr{X}$, $dom(X_i) \neq \emptyset$ and $\forall a \in dom(X_i)$,$(X_i, a)$ is max-restricted path consistent.*

Singleton Arc Consistency (SAC) is a stronger consistency than Max-RPC which is itself stronger than AC. It means that SAC can identify more inconsistent values than Max-RPC can, and subsequently more than AC can. SAC guarantees that enforcing arc consistency after performing any variable assignment does not show unsatisfiability, i.e., does not entail a domain wipe-out. Optimal worst-case time complexity to establish SAC is O($mnd^3$) [5].

To give a formal definition of SAC, we need to introduce some notations. AC($P$) denotes the constraint network obtained after enforcing arc consistency on a given constraint network $P$. AC($P$) is such that all values of $P$ that are not arc consistent have been removed. If there is a variable with an empty domain in AC($P$), denoted AC($P$) $= \bot$, then $P$ is clearly unsatisfiable. $P|_{X=a}$ is the constraint network obtained from $P$ by restricting the domain of $X$ to $\{a\}$.

**Definition 4.** *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN, $X \in \mathscr{X}$ and $a \in dom(X)$. $(X, a)$ is singleton arc consistent iff $AC(P|_{X=a}) \neq \bot$. $P$ is singleton arc consistent iff $\forall X \in \mathscr{X}$, $dom(X) \neq \emptyset$ and $\forall a \in dom(X)$, $(X, a)$ is singleton arc consistent.*

Finally, [4] have proposed an extension of SAC that is called Singleton Propagated Arc Consistency (SPAC). It is based on the following observation. If $(Y, b) \notin AC(P|_{X=a})$ then it corresponds to the detection of the nogood $\neg(X = a \wedge Y = b)$ and we can deduce that $(X, a) \notin AC(P|_{Y=b})$. We can exploit this inference when checking the singleton arc consistency of $(Y, b)$ as it gives more chances to detect an inconsistency.

**Definition 5.** *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN, $X \in \mathscr{X}$ and $a \in dom(X)$. $(X, a)$ is singleton propagated arc consistent iff $\tilde{P}|_{X=a} \neq \bot$ where $\tilde{P}|_{X=a}$ is the constraint network obtained from $P$ by removing any value $(Y,b)$ of $P$ (i.e. $b$ from $dom(Y)$) such that $(X, a) \notin AC(P|_{Y=b})$. $P$ is singleton propagated arc consistent iff $\forall X \in \mathscr{X}$, $dom(X) \neq \emptyset$ and $\forall a \in dom(X)$, $(X, a)$ is singleton propagated arc consistent.*

It is possible to define a bound version for any domain filtering consistency $\Phi$ as follows.

**Definition 6.** *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN. $P$ is bound $\Phi$-consistent iff $\forall X \in \mathscr{X}$, $dom(X) \neq \emptyset$ and both $min(X)$ and $max(X)$ are $\Phi$-consistent.*

On the other hand, 2B and 3B [19] are consistencies that have been introduced wrt continuous constraint networks. 2B($P$) denotes the constraint network obtained after enforcing 2B on a given constraint network $P$ and 2B($P$) $= \bot$ indicates that there is a variable with an empty domain in 2B($P$).

**Definition 7.** *$P = (\mathscr{X}, \mathscr{C})$ is 2B-consistent iff $\forall X \in \mathscr{X}$, $dom(X) \neq \emptyset$ and both $min(X)$ and $max(X)$ are arc consistent. $P$ is 3B-consistent iff $\forall X \in \mathscr{X}$, $dom(X) \neq \emptyset$ and both $2B(P|_{X=min(X)}) \neq \bot$ and $2B(P|_{X=max(X)}) \neq \bot$.*

**Algorithm 1** seekSupportArc(C : Constraint, X : Variable, a : Value) : boolean

```
1: $t \leftarrow \bot$
2: while $t \neq \top \wedge C(t) = false$ do
3:     $t \leftarrow setNextTuple(C, X, a, t)$
4: return $t \neq \top$
```

**Algorithm 2** revise(C : Constraint, X : Variable) : boolean

```
1: $domainSize \leftarrow |dom(X)|$
2: while $|dom(X)| > 0 \wedge \neg\ seekSupportArc(C, X, min(X))$ do
3:     remove $min(X)$ from dom($X$)
4: while $|dom(X)| > 1 \wedge \neg\ seekSupportArc(C, X, max(X))$ do
5:     remove $max(X)$ from dom($X$)
6: return $domainSize \neq |dom(X)|$
```

Clearly, 2B-consistency corresponds to bound AC while 3B-consistency is a relaxation of bound SAC since for each pair ($X$,$a$) with $a$ = min($X$) or $a$ = max($X$), 3B-consistency requires that 2B($P|_{X=a}$) $\neq \bot$ whereas bound SAC requires that AC($P|_{X=a}$) $\neq \bot$. We can also observe (see next sections) that a consistency and its bound version admit the same optimal worst-case time complexity. For example, establishing AC or 2B is O($md^2$) while establishing SAC, bound SAC or even 3B is O($mnd^3$). This statement seems to be in contradiction with the optimality of 2B-consistency and 3B-consistency algorithms which is O($md$) [19] and O($mnd^2$) [7], respectively. However, it is then assumed that all constraints are basic, that is to say, that for each constraint $C$, it is possible to find two functions that compute in bounded time the min bound and the max bound of the domain of any variable involved in $C$.

One nice advantage of exploiting bound consistencies is that space complexity can be very affordable. Indeed, it is possible to reduce the space required by some algorithms by a factor $d$ or even $d^2$ as we can just generate data structures wrt two bounds. Further, if convex domains are considered, i.e. domains are such that all values between the min and the max bounds belong to the domain, then a constraint network can be represented in O($n + m$). It can be very useful for networks involving variables with large domains as for some scheduling instances.

Finally, remark that we have ignored in this paper the adaptation of (numeric) consistencies such as box-consistency and bound-consistency wrt discrete CSP instances.

## 3 2B (Bound arc consistency)

Arc consistency (AC) is the most studied and used local consistency. Algorithm 4 is the bound adaptation of the coarse-grained arc consistency algorithm AC3 [21]. It just calls Algorithm 3 with the set of variables of the given constraint network as a second parameter. This second algorithm allows establishing bound arc consistency of the given constraint network by initializing a set $Q$ with some

**Algorithm 3** 2B ($P = (\mathscr{X},\mathscr{C})$ : CN, $S$ : set of Variables)

```
1: $Q \leftarrow \{(C,X) \mid C \in \mathscr{C} \wedge X \in vars(C) \wedge \exists Y \in S \cap vars(C) | Y \neq X\}$
2: while $Q \neq \emptyset$ do
3:    pick and delete $(C,X)$ in $Q$
4:    if revise($C$,$X$) then
5:        $Q \leftarrow Q \cup \{(C',X') \mid X \in vars(C') \wedge X' \in vars(C') \wedge C \neq C'\}$
6: end while
```

**Algorithm 4** 2B ($P = (\mathscr{X},\mathscr{C})$ : CN)

```
1: 2B($P$,$\mathscr{X}$)
```

arcs and then performing successive revisions until a fix-point is reached. Algorithm 3 has been introduced as it is useful later in the paper. But, assuming that no unary constraint is allowed, one should observe that the call $2B(P,\mathscr{X})$ (line 1 of Algorithm 4) involves the following standard initialization of the set $Q$ (line 1 of Algorithm 3):

$$Q \leftarrow \{(C,X) \mid C \in \mathscr{C} \wedge X \in vars(C)\}$$

Hence, initially, all arcs $(C,X)$ are put in a set $Q$. Then, each arc is revised in turn, and when a revision is effective (at least one value has been removed), the set $Q$ has to be updated. A revision is performed by a call to the function *revise*$(C,X)$, depicted in Algorithm 2 and entails removing values at bounds of dom$(X)$ that have become inconsistent with respect to $C$. The algorithm is stopped when the set $Q$ becomes empty. Remark that when a revision of an arc $(C,X)$ is effective, it is necessary to take into account the arcs of the form $(C',X)$ (with $C \neq C'$) since the consistency of the new bound(s) of dom$(X)$ is not guaranteed wrt $C'$. The function *seekSupportArc*, depicted in Algorithm 1, determines from scratch whether or not there exists a support of $(X,a)$ in $C$. It iteratively calls the function *setNextTuple* which returns either the smallest valid tuple $t'$ in $C$ such that $t \prec t'$ and $t'[X] = a$ or $\top$ if it does not exist. Note that $C(t)$ must be understood as a constraint check and that $C(\bot)$ returns false.

Finally, Algorithm 4 can also be seen as an adaptation of the procedure IP_1 proposed in [19] where it is assumed that constraints are basic. Also, a variant with a constraint-oriented propagation scheme can be found in [7].

**Proposition 1.** *Applied to binary constraint networks, Algorithm 4 admits a worst-case time and space complexity in O($md^2$) and O($m$), respectively.*

*Proof.* Each arc $(C,X)$ may enter $d$ times in $Q$ to be revised [21, 7, 6]. When a revision entails no removal, at most $2 \times d$ constraint checks are performed. When some removals occur, there are at most $d$ additional constraint checks per value removed. For each arc, we then obtain $2 \times d \times d + d \times d$ as an upper bound of the global number of constraint checks. As there are $2 \times m$ different arcs, we obtain a worst-case time complexity in O($md^2$). On the other hand, the only structure used by the algorithm is the queue $Q$ which is O($m$). □

**Algorithm 5** seekSupportPath(C$_{ij}$ : Constraint, X$_i$ : Variable, a : Value) : bool

```
1: for each value $b \in \text{dom}(X_j)$ s.t. $C_{ij}(a,b)$ do
2:   for each variable $X_k$ s.t. $(X_i,X_j,X_k)$ forms a 3-clique do
3:     for each value $c \in \text{dom}(X_k)$ do
4:       if $C_{ik}(a,c) \wedge C_{jk}(b,c)$ then
5:         continue loop 2:
6:     end for
7:     continue loop 1:
8:   end for
9:   return true
10: end for
11: return false
```

It is interesting to note that even if last supports are recorded as with an underlying optimal arc consistency technique such as AC2001/3.1, the worst-case time complexity remains O($md^2$) although one could have expected a better complexity as bound consistencies only consider the min and the max values.

## 4 2B+ (Bound max-restricted path consistency)

Max-Restricted Path Consistency (Max-RPC) [10] is one of the most promising local consistencies. Max-RPC is stronger than arc consistency, restricted path consistency [3] and k-restricted path consistency [10] but weaker than singleton arc consistency. In this section, we propose a bound adaptation of Max-RPC in the context of a coarse-grained algorithm. Actually, as this adaptation, denoted 2B+, does not guarantee that each bound has a path consistent support with respect to each constraint, it should be viewed as an opportunistic algorithm that is simple to define and implement.

The algorithm 2B+ is obtained from 2B by simply replacing, in function *revise*, calls to function *seekSupportArc* by calls to function *seekSupportPath* which is described by Algorithm 5. The function *seekSupportPath* returns true (line 10) iff the given value has a path consistent support on the given constraint. In order to guarantee that the resulting constraint network is (at least) arc consistent, we have to replace $C \neq C'$ by $(C \neq C' \vee X \neq X')$ in line 5 of Algorithm 3. It means that, when the revision of an arc $(C_{ij}, X_i)$ is effective, it is necessary to take into account the arc $(C_{ij}, X_j)$. Indeed, let us suppose that $(C_{ij}, X_j)$ has been revised and that $a = \min(X_i)$ has been found as a path consistent support for $b = \min(X_j)$ requiring a value $c$ for a variable $X_k$. Next, some revision is performed that entails the removal of ($X_k$,$c$) and $(C_{ij}, X_i)$ is revised. Imagine that $a = \min(X_i)$ has no more path consistent support in dom($X_j$) ($b = \min(X_j)$ was one such support but it required $c$ that has been removed) then $a$ is removed. If the arc $(C_{ij}, X_j)$ is not added to $Q$, then it is possible that propagation finishes although ($X_j$,$b$) is not supported by $X_i$.

**Algorithm 6** 3B-X($P = (\mathscr{X},\mathscr{C})$ : CN)

```
1: $P \leftarrow$ 2B($P$,$\mathscr{X}$)
2: repeat
3:   $P_{before} \leftarrow P$
4:   for each $X \in \mathscr{X}$ do
5:     $domainSize \leftarrow |dom(X)|$
6:     while $|dom(X)| > 0 \wedge \neg$ check2B-X($P$,$X$,$min(X)$) do
7:       remove $min(X)$ from dom($X$)
8:     while $|dom(X)| > 1 \wedge \neg$ check2B-X($P$,$X$,$max(X)$) do
9:       remove $max(X)$ from dom($X$)
10:    if $|dom(X)| < domainSize$ then
11:      $P \leftarrow$ 2B($P$,{$X$})
12:  end for
13: until $P = P_{before}$
```

## 5 3B (Bound singleton arc consistency)

There is a recent attraction about singleton consistencies, and more particularly about SAC (Singleton Arc Consistency), as illustrated by recent works of [11, 22, 1, 4, 5, 18]. Even if it is possible to propose an algorithm to establish bound SAC, it does not seem quite appropriate when dealing with large domains as AC requires to represent domains in extension (and not by simple intervals). We propose here two algorithms to establish 3B which can be seen as a relaxation of bound SAC.

### 5.1 3B-X

Algorithm 6 is the bound adaptation, denoted 3B-X, of a basic singleton arc consistency algorithm. 3B-X starts by enforcing bound arc consistency (2B) on the given network (line 1). Then, each bound of the domain of each variable is checked to be 2B-consistent by calling the function $check2B - X$ (lines 6 and 8). Two variants, denoted $check2B - 1$ and $check2B - 2$, of this function are given in the subsequent subsections. Bounds that are not consistent are then removed (lines 7 and 9). When the domain of a variable is modified, bound arc consistency is maintained (lines 10 and 11). The process continues until a fix-point is reached.

### 5.2 3B-1

3B-1 corresponds to the algorithm 6 that uses the function $check2B-1$ depicted by Algorithm 7. Roughly speaking, 3B-1 is the bound adaptation of the singleton arc consistency algorithm SAC-1 [11].

**Proposition 2.** *Applied to binary constraint networks, Algorithm 3B-1 admits a worst-case time and space complexity in $O(mn^2d^3)$ and $O(m)$, respectively.*

**Algorithm 7** check2B-1($P = (\mathscr{X},\mathscr{C})$ : CN, $X$ : Variable, $a$ : Value) : boolean

```
1: return 2B($P|_{X=a}$, $\{X\}$) $\neq \bot$
```

*Proof.* The number of turns of the main loop of Algorithm 3B-X is at most $nd$, one element being removed at each turn. The number of calls to check2B-X is $2 * n$ at each turn. As a call to check2B-1 is equivalent to a call to 2B which is O($md^2$), we obtain an overall worst-case time complexity in O($mn^2d^3$). As Algorithm 3B-1 does not require any additional data structure, its space complexity is the same as check2B-1, namely O($m$). □

## 5.3 3B-2

3B-2 corresponds to the algorithm 6 that uses the function $check2B-2$ depicted by Algorithm 10. The idea is to improve the performance of the basic algorithm by recording and exploiting some information. For instance, when the consistency of a value must be checked again, it is inefficient to restart checking from scratch [7, 5]. Hence, we introduce three data structures:

- *initialized* is an array that indicates for any pair ($X$,$a$) whether the 2B-consistency of ($X$,$a$) has been checked at least one time,
- $minInferences$ is a three-dimensional array that allows recording for any triplet ($X$,$a$,$Y$) the value min($Y$) in $2B(P|_{X=a})$,
- $maxInferences$ is a three-dimensional array that allows recording for any triplet ($X$,$a$,$Y$) the value max($Y$) in $2B(P|_{X=a})$.

We will assume that *initialized* is an array whose elements are initially set to false (it does not appear in the given algorithm). Inferences with respect to a pair ($X$,$a$) are relevant only when *initialized*$[X, a]$ is equal to true. For instance, imagine that after achieving $2B(P|_{X=a})$, we obtain a network such that min($Y$) = $c$ and max($Y$) = $d$ (hence, dom($Y$) = $\{c, \ldots, d\}$). Then, we set *initialized*$[X, a]$ to true, $minInferences[X, a, Y]$ to $c$ and $maxInferences[X, a, Y]$ to $d$.

When running $check2B - 2$ (Algorithm 10), recorded information is first exploited (line 2) by a call to $exploitInferences$. After exploitation of recorded

**Algorithm 8** exploitInferences($X$ : Variable, $a$ : Value) : Set of variables

```
1: if $\neg$initialized[$X, a$] then
2:    return $\{X\}$
3: $S \leftarrow \emptyset$
4: for each $Y \in \mathscr{X}$ do
5:    min($Y$) $\leftarrow$ max(min($Y$),minInferences[X,a,Y])
6:    max($Y$) $\leftarrow$ min(max($Y$),maxInferences[X,a,Y])
7:    if min($Y$) > minInferences[X,a,Y] or max($Y$) < maxInferences[X,a,Y] then
8:       add Y to $S$
9: end for
10: return $S$
```

**Algorithm 9** recordInferences($X$ : Variable, $a$ : Value)

```
1: initialized[X,a] ← true
2: for each $Y \in \mathscr{X}$ do
3:    minInferences[X,a,Y] ← min(Y)
4:    maxInferences[X,a,Y] ← max(Y)
5: end for
```

**Algorithm 10** check2B-2($P = (\mathscr{X}, \mathscr{C})$ : CN, $X$ : Variable, $a$ : Value) : boolean

```
1: $P_{store} \leftarrow P$
2: $S \leftarrow$ exploitInferences($X$,a)
3: if $S = \emptyset$ then
4:    consistent ← true
5: else
6:    consistent ← 2B($P|_{X=a}, S$) $\neq \bot$
7:    if consistent then
8:       recordInferences($X$,a) then
9: end if
10: $P \leftarrow P_{store}$
11: return consistent
```

inferences, either the 2B-consistency of ($X$,$a$) still holds (line 4) as the empty set is returned by *exploitInferences* (since no value in $2B(P|_{X=a})$ has been removed), or we have to check the 2B-consistency of ($X$,$a$) from the set $S$ of variables whose domain has been reduced (line 6). Inferences are updated (line 8) by a call to the function *recordInferences*.

The function *exploitInferences* (Algorithm 8) returns either the singleton $\{X\}$ (line 2) if the 2B-consistency of ($X$,$a$) has never been checked or the set of variables whose domain has lost a value which does not belong to (the last achievement of) $2B(P|_{X=a})$ (line 8). The function *recordInferences* (Algorithm 9) just updates data structures.

The algorithm described here can be seen as a bound adaptation of the SAC-OPT algorithm proposed in [5] and also as a variant of the optimized 3B algorithm described in [7].

**Proposition 3.** *Applied to binary constraint networks, Algorithm 3B-2 admits a worst-case time and space complexity in O($mnd^3$) and O($n^2$), respectively.*

*Proof.* By storing information and avoiding unnecessary computation [7], Algorithm 3B-2 exploits the incrementality of arc consistency [5]. It means that the $2*n$ potential successive calls to check2B-2 wrt a value ($X$,$a$) is in O($md^2$). Functions *exploitInferences* and *recordInferences* are in O($n$) and then can be ignored. As there are $nd$ values, the overall worst-case time complexity is O($mnd^3$). It is possible to modify the data structures in order to keep only storage wrt two bounds per variable. With this slight modification (not proposed here due to lack of space), we obtain $2*n$ values to be recorded for $2*n$ current bounds. Hence, we obtain O($n^2$). □

## 6 3B+ (Bound singleton propagated arc consistency)

Finally, we propose an improvement of the algorithm 3B-2 presented above. Indeed, it is possible to benefit from some inferences when exploiting recorded information. For example, we know that if $\min(X) > a$ in 2B($P|_{Y=min(Y)}$) then we can infer that $Y > \min(Y)$ in 2B($P|_{X=a}$). By inserting the following instructions between lines 6 and 7 of Algorithm 8:

```
if $initialized[Y, min(Y)] \wedge minInferences[Y, min(Y), X] > a$ then
    $\min(Y) \leftarrow \min(Y) + 1$
if $initialized[Y, max(Y)] \wedge maxInferences[Y, max(Y), X] < a$ then
    $\max(Y) \leftarrow \max(Y)$ - 1
```

we obtain an algorithm which corresponds to an extension of the 3B-consistency and which can be seen as a coarse relaxation of bound SPAC.

## 7 Experiments

To prove the practical interest of the properties introduced in this paper, we have implemented the different algorithms described in the previous sections and conducted an experimentation with respect to some scheduling and frequency assignment instances. Algorithms 2B, 2B+, 3B-1, 3B-2, 3B-2+ correspond to the descriptions given in previous sections while 3B-1d is a dichotomic version (not described here) of 3B-1 related to the procedure *ref_filtering* in [19].

First, we have searched to establish a comparison between all algorithms wrt a set of 20 job-shop scheduling instances generated using the model of Taillard [23] by fixing 8 jobs and 8 machines. For each instance, a lower bound $LB$ as well as the optimal makespan $OPT$ have been computed. We have considered different sets of unsatisfiable CSP instances by setting different time windows between $LB$ and $OPT$ (only point where instances are in fact satisfiable). Figure 1 shows the proportion of instances that have been proved to be unsatisfiable at each variation $x$ of the time window when establishing different consistencies. For example, at $x = 0$, the time window considered is (from 0 to) $LB$ while at $x = 1$, it is (from 0 to) $OPT$. One can observe that 3B (and 3B+) allow a level of filtering which is sufficient to detect the inconsistency of most of the instances, unlike 2B+ and, especially unlike 2B and AC which behave similarly. Figure 2 shows the effort, in terms of number of constraint checks, required to establish the different consistencies (similar results are obtained when considering cpu times). Quite naturally, the more filtering a consistency is, the more costly it is. One can notice that the dichotomic variant of 3B outperforms the other ones.

Next, we have considered the real-world instances of the fullRLFAP archive. Table 1 shows the results obtained on some selected instances. Here, it clearly appears that AC is the best approach wrt these instances both in terms of cpu and of filtering (#rmvs denote the number of detected inconsistent values). It can be explained by the fact that frequency assignment instances are less favorable to bound consistencies than scheduling ones (which involve many precedence constraints). One can also notice that 2B+ allows a slight improvement of filtering wrt 2B (see *graph*4 and *graph*10) unlike 3B-2+ wrt 3B-2.

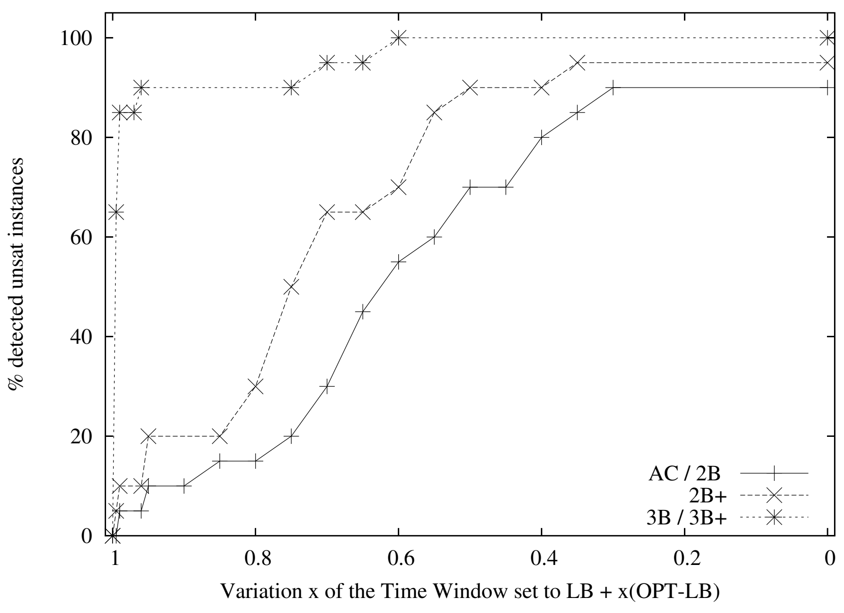

**Fig. 1.** Proportion of detected unsat instances when establishing consistencies

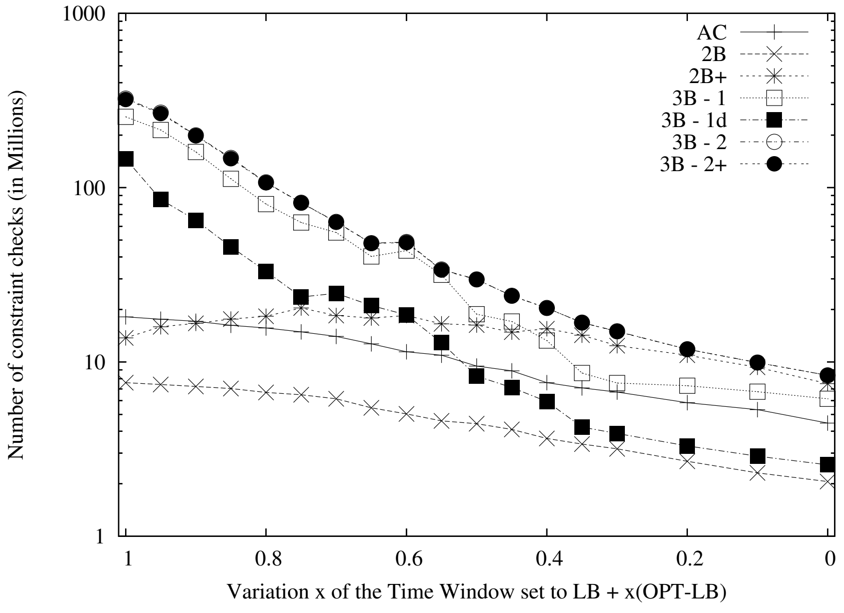

**Fig. 2.** Number of performed constraint checks when establishing consistencies

| | | AC | 2B | 2B+ | 3B - 1 | 3B - 1d | 3B - 2 | 3B - 2+ |
|---|---|---|---|---|---|---|---|---|
| *graph*4 | cpu | 0.47 | 0.1 | 2.23 | 8.29 | 10.25 | 34.26 | 44.4 |
| | #rmvs | 776 | 0 | 187 | 411 | 411 | 411 | 411 |
| *graph*10 | cpu | 0.86 | 0.15 | 4.15 | 11.87 | 13.42 | 32.75 | 42.24 |
| | #rmvs | 386 | 0 | 46 | 122 | 122 | 122 | 122 |
| *graph*14-*f*27 | cpu | 0.43 | 0.16 | 1.93 | 3.25 | 2.48 | 3.32 | 5.76 |
| | #rmvs | 2,314 | 0 | 0 | 0 | 0 | 0 | 0 |
| *graph*14-*f*28 | cpu | 0.43 | 0.16 | 2.1 | 5.81 | 4.72 | 4.56 | 6.73 |
| | #rmvs | 3,230 | 0 | 0 | 2 | 2 | 2 | 2 |
| *scen*02-*f*25 | cpu | 0.14 | 0.07 | 0.55 | 0.58 | 0.52 | 0.58 | 0.75 |
| | #rmvs | 106 | 0 | 0 | 0 | 0 | 0 | 0 |
| *scen*11-*f*8 | cpu | 0.55 | 0.13 | 2.7 | 3.27 | 2.95 | 3.33 | 4.16 |
| | #rmvs | 4,992 | 0 | 0 | 0 | 0 | 0 | 0 |
| *scen*11-*f*10 | cpu | 0.51 | 0.21 | 4.11 | 2.98 | 2.66 | 3.12 | 4.24 |
| | #rmvs | 6,324 | 3,024 | 3,024 | 3,024 | 3,024 | 3,024 | 3,024 |

**Table 1.** Establishing consistencies on RLFAP instances

In a second stage, we have searched to maintain all consistencies during the search of a solution. We have first studied the 10 open-shop scheduling instances with 7 jobs and 7 machines described in [23]. For each instance, we have searched to reach the optimal makespan in less than 300 seconds by using a branch and bound approach while exploiting constraint propagation. Table 2 gives the average relative distance, as well as the standard deviation, between the optimal makespan and the makespan found by the solver for different filtering algorithms. The results clearly show that 2B is the worst approach while 3B is the best one. In particular, 3B-1d and 3B-2+ have the best behaviour. We have also again considered the 20 job-shop scheduling instances already described above. Table 3 presents the average time (cpu) required to reach the optimal makespan and the proportion of instances that have been detected as unsatisfiable in less than 300 seconds with a time window fixed to $OPT-1$. Once again, Maintaining 3B is the best approach.

Table 4 shows the results (time-out has been set to 900 seconds) obtained when solving the selected RLFAP instances mentioned above. On some difficult instances, it is interesting to note that maintaining 2B is the quickest approach while maintaining a stronger consistency is always penalizing.

Finally, we must remember that all algorithms are based on AC3. We believe that using a more sophisticated foundation such as AC2001/3.1 [6] or AC3.2/3.3 [17] to establish 2B or 3B will not change the results a lot (but using $AC3_d$

| | AC | 2B | 3B - 1 | 3B - 1d | 3B - 2 | 3B - 2+ |
|---|---|---|---|---|---|---|
| Average Distance | 4.79% | 28.6% | 4.28% | 3.00% | 3.32% | 3.30% |
| Standard Deviation | 3.5% | 8.2% | 2.4% | 2.2% | 2.4% | 1.8% |

**Table 2.** Maintaining consistencies on Taillard's 7x7 open-shop instances

| | AC | 2B | 2B+ | 3B - 1 | 3B - 1d | 3B - 2 | 3B - 2+ |
|---|---|---|---|---|---|---|---|
| Average cpu (TW = OPT) | 212 | 216 | 282 | 88 | 73 | 117 | 117 |
| % detected unsat (TW = OPT-1) | 45% | 55% | 30% | 85% | 85% | 85% | 85% |

**Table 3.** Maintaining consistencies on 8x8 job-shop instances

| | | AC | 2B | 2B+ | 3B - 1 | 3B - 1d | 3B - 2 | 3B - 2+ |
|---|---|---|---|---|---|---|---|---|
| $graph04$ | cpu | 2.11 | *timeout* | *timeout* | 260 | 314 | 391 | 678 |
| $graph10$ | cpu | 8.18 | *timeout* | 14.27 | *timeout* | *timeout* | *timeout* | *timeout* |
| $graph14$-$f27$ | cpu | 6.34 | *timeout* | *timeout* | *timeout* | 847 | *timeout* | *timeout* |
| $graph14$-$f28$ | cpu | 40.73 | 8.11 | 20.48 | 347.57 | 435.91 | *timeout* | *timeout* |
| $scen02$-$f25$ | cpu | 4.12 | 2.86 | 43.46 | 47.84 | 43.61 | 49.95 | 159.1 |
| $scen11$-$f8$ | cpu | 115.26 | 74.62 | 98.86 | *timeout* | *timeout* | *timeout* | *timeout* |
| $scen11$-$f10$ | cpu | 6.69 | 6.4 | 15.59 | 388.28 | 372.71 | 652.25 | *timeout* |

**Table 4.** Maintaining consistencies on RLFAP instances

[24] could be worthwhile). Indeed, we know for example that establishing 2B remains O($md^2$) even if it is based on an optimal arc consistency algorithm. Further, our preliminary tests have confirmed this prediction. Nevertheless, 2B+ is one consistency that could benefit from such sophistication since many path consistency checks could be avoided.

## 8 Conclusion

The modest contribution of this paper is to establish a, hopefully clearer, connection between domain filtering consistencies, taken from the discrete CSP model, and bound consistencies, taken from the continuous CSP model. In particular, we have studied bound versions of well-known domain filtering consistencies.

The great advantage of using bound consistencies is that space requirement can be very limited, especially when domains are convex. For some discrete CSP instances with very large domains, it can be the only realistic approach. On the other hand, when space saving is not mandatory, worst-case time complexities of establishing bound consistencies wrt discrete instances (for which, no constraint semantics is available) are rather disappointing. For instance, in this context, the worst-case time complexity of establishing 2B (i.e. bound AC) is similar to the one of establishing AC. From a practical point of view, using bound consistencies is a good approach when dealing with problems which involve "bound-oriented" constraints such as precedence constraints. But, in this case, it is often possible to adopt a specific filtering by exploiting constraint semantics and also obtain a better complexity. In a less favorable context, our experimental results from some frequency assignment problems does not show an overall real advantage of using bound consistencies wrt arc consistency. However, we believe that bound consistencies could play a role in the development of methods for controlling the effort required to maintain a strong consistency during search.

## References

1. R. Bartak and R. Erben. A new algorithm for singleton arc consistency. In *Proceedings of FLAIRS'04*, 2004.
2. F. Benhamou, D. MacAllester, and P. Van Hentenryck. CLP(Intervals) revisited. In *Proceedings of ILPS'94*, pages 124–138, 1994.
3. P. Berlandier. Improving domain filtering using restricted path consistency. In *Proceedings of IEEE-CAIA'95*, 1995.
4. C. Bessière and R. Debruyne. Theoretical analysis of singleton arc consistency. In *Proceedings of ECAI'04 workshop on modelling and solving problems with constraints*, pages 20–29, 2004.
5. C. Bessière and R. Debruyne. Optimal and suboptimal singleton arc consistency algorithms. In *Proceedings of IJCAI'05*, pages 54–59, 2005.
6. C. Bessiere, J.C. Régin, R.H.C. Yap, and Y. Zhang. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165(2):165–185, 2005.
7. L. Bordeaux, E. Monfroy, and F. Benhamou. Improved bounds on the complexity of kB-consistency. In *Proceedings of IJCAI'01*, pages 303–308, 2001.
8. P. Codognet and D. Diaz. Compiling constraints in clp(FD). *Journal of Logic Programming*, 27(3):185–226, 1996.
9. H. Collavizza, F. Delobel, and M. Rueher. Comparing partial consistencies. *Reliable computing*, 5:213–228, 1999.
10. R. Debruyne and C. Bessière. From restricted path consistency to max-restricted path consistency. In *Proceedings of CP'97*, pages 312–326, 1997.
11. R. Debruyne and C. Bessière. Some practical filtering techniques for the constraint satisfaction problem. In *Proceedings of IJCAI'97*, pages 412–417, 1997.
12. R. Debruyne and C. Bessière. Domain filtering consistencies. *Journal of Artificial Intelligence Research*, 14:205–230, 2001.
13. R. Dechter. *Constraint processing*. Morgan Kaufmann, 2003.
14. B. Faltings. Arc consistency for continuous variables. *Artificial Intelligence*, 65:363–376, 1994.
15. P. Van Hentenryck, V. Saraswat, and Y. Deville. Design, implementation and evaluation of the constraint language cc(FD). *Journal of Logic Programming*, 37(1-3):139–164, 1998.
16. E. Hyvonen. Constraint reasoning based on interval arithmetic: the tolerance approach. *Artificial Intelligence*, 58:71–112, 1992.
17. C. Lecoutre, F. Boussemart, and F. Hemery. Exploiting multidirectionality in coarse-grained arc consistency algorithms. In *Proc. of CP'03*, pages 480–494, 2003.
18. C. Lecoutre and S. Cardon. A greedy approach to establish singleton arc consistency. In *Proceedings of IJCAI'05*, pages 199–204, 2005.
19. O. Lhomme. Consistency techniques for numeric csps. In *Proceedings of IJCAI'93*, pages 232–238, 1993.
20. O. Lhomme. *Contribution à la résolution de contraintes sur les réels par propagation d'intervalles.* PhD thesis, Université de Nice-Sophia Antipolis, 1994.
21. A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.
22. P. Prosser, K. Stergiou, and T. Walsh. Singleton consistencies. In *Proceedings of CP'00*, pages 353–368, 2000.
23. E. Taillard. Benchmarks for basic scheduling problems. *European journal of operations research*, 64:278–295, 1993.
24. M.R.C. van Dongen. AC3$_d$ an efficient arc consistency algorithm with a low space complexity. In *Proceedings of CP'02*, pages 755–760, 2002.