Revision ordering heuristics for the Constraint Satisfaction Problem

Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre

CRIL (Centre de Recherche en Informatique de Lens)
CNRS FRE 2499
rue de l'université, SP 16
62307 Lens cedex, France
{boussemart,hemery,lecoutre}@cril.univ-artois.fr

Abstract. For many years, arc consistency has been recognized as a basic property of constraint networks. Among all algorithms that have been proposed to establish arc consistency, AC3, and more precisely its recent extensions, still remains competitive. In this paper, we present three variants for AC3 based algorithms and we focus on the order in which revisions are applied by them. For the three variants, which respectively correspond to algorithms with an arc-oriented, variable-oriented and constraint-oriented propagation scheme, we propose some original revision ordering heuristics and adapt the ones defined in [21]. The experimentation which has been run both on binary and non-binary problems confirm that using such heuristics, when arc consistency is used as a preprocessing or/and when it is maintained during search, turns out to significantly reduce the number of constraint checks. Furthermore, we show that the variable-oriented variant is guaranteed to benefit from such heuristics in terms of cpu time.

1 Introduction

For many years, arc consistency has been recognized as a basic property of constraint networks. Arc consistency guarantees that any value of the domain of a variable can be found in, at least, a support of any constraint. This property can be established by an algorithm to make a constraint network arc consistent but it can also be maintained during the search of a solution.

Many algorithms have been proposed to establish arc consistency. One of the very first proposals is the algorithm AC3 [12]. This coarse grained algorithm involves applying successive revisions of arcs, i.e., of pairs (C, X) composed of a constraint C and of a variable X belonging to the set of variables of C. Later, other algorithms such as AC4 [14], AC6 [2] and AC7 [3] have been introduced. These fine grained algorithms involve applying successive revisions of "values", i.e., of triplets (C, X, a) composed of an arc (C, X) and of a value a belonging to the domain of X.

Even if, theoretically, AC3, unlike AC4, AC6 and AC7, has not an optimal worst case time complexity ($O(md^3)$ for AC3 and $O(md^2)$ for AC4, AC6 and

AC7 where m and d respectively denote the number of constraints and the size of the uniform domains) and if, in practice, AC3 is not always as fast as AC6 and AC7, AC3 has the great advantage to be easily implemented.

On the other hand, some recent extensions of AC3 have been developed which, while preserving the simplicity of AC3, turn out to be as competitive as fine grained algorithms (with, in particular, a worst case time complexity in $O(md^2)$ for most of them). These new algorithms are called AC2000 [5], AC2001/3.1 [5, 22], AC3$_d$ [18], AC3.2 [11] and AC3.3 [11]. It is also interesting to note that with respect to some desirable properties of arc consistency algorithms, it is possible to draw a parallel [5] between AC2001/3.1 and AC6 and another [11] between AC3.3 and AC7. Then, it clearly appears that AC3 based algorithms are up-to-date algorithms to establish arc consistency.

In this paper, we are interested in the order in which revisions are applied by AC3-based algorithms. First, we present three variants which respectively correspond to algorithms with an arc-oriented, variable-oriented and constraint-oriented propagation scheme. The first one is the most commonly presented, the second one corresponds to the algorithm proposed by [13, 6], and the third one is original. Even if, at first glance, all these variants seems to be equivalent, we shall emphasize a significant difference of their respective behaviour when considering different revision ordering heuristics. As far as we are aware, the only works which concern such heuristics are [21, 8, 18] which focus on the arc-oriented variant to solve binary problems, and [15] which focuses on fine-grained algorithms.

Then, our contribution is the study of various revision ordering heuristics with respect to the use of the three variants before and/or during the search of a solution for binary and non binary problems. In particular, we propose some original heuristics based on the proportion of removed values in the different domains and on the current domain size of the different constraints. Also, we adapt to the three proposed variants, the heuristic of [21] which is based on the current size of the domains. Experimental results show that using such heuristics before or/and during the search of a solution can be quite efficient in terms of constraint checks. Furthermore, we show that the variable-oriented variant is guaranteed to benefit from such heuristics in terms of cpu time.

This paper is organized as follows. Section 2 introduces some preliminaries. In Section 3, three variants of the basic arc consistency algorithm are described. Some revision ordering heuristics are presented in Section 4. Before concluding, Section 5 gives an experimental evaluation.

2 Preliminaries

Let us introduce some notations frequently used in the rest of the paper:

- $|S|$ denotes the cardinality of a set S, i.e. the number of elements of S,
- $\prod_{i=1}^{k} S_i$ denotes the Cartesian product over k sets $S_1, \ldots, S_k$, i.e. the set $\{(a_1, \ldots, a_k) \mid a_i \in S_i, 1 \leq i \leq k\}$,

Definition 1. *A constraint network 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* $vars(C_j)$ *involved in the constraint* C_j*.*

We shall say that a constraint C involves (or binds) a variable X if and only if X belongs to $vars(C)$. The arity of a constraint C is the number of variables involved in C, i.e., the number of variables in $vars(C)$. A binary constraint only involves 2 variables. The domain of a constraint C, denoted $\mathrm{dom}(C)$, is the Cartesian product $\prod_{j=1}^{k} dom(X_{i_j})$, where C is a k-ary constraint such that $\mathrm{vars}(C) = \{X_{i_1}, \ldots, X_{i_k}\}$. For any element $t = (a_{i_1}, \ldots, a_{i_k})$, called k-tuple, of $\mathrm{dom}(C)$, $t[X_{i_j}]$ denotes the value a_{i_j}. A k-tuple t is said to be a support of C iff $t \in rel(C)$ and is said to be a support of (X,a) in C iff t is a support of C such that $t[X] = a$. Determining if a tuple is allowed by a constraint C (i.e. is a support of C) is called a constraint check.

An instance of the Constraint Satisfaction Problem (CSP) is defined by a constraint network. A CSP instance is said to be satisfiable iff the constraint network to which it corresponds admits a solution, and unsatisfiable otherwise. Solving a CSP instance involves either finding one (or more) solution or determining its unsatisfiability. A solution is an assignment of values to all the variables such that all the constraints are satisfied.

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, which are based on some constraint network properties such as arc-consistency, remove some values which can not occur in any solution. Modifying the domains of a given constraint network in order to get it arc consistent involves performing constraint checks.

Definition 2. *Let* $P = (\mathscr{V}, \mathscr{C})$ *be a CSP,* $C \in \mathscr{C}$*,* $V \in vars(C)$ *and* $a \in dom(V)$*.* (V, a) *is said to be consistent wrt* C *iff there exists a support of* (V, a) *in* C*.* C *is said to be arc-consistent iff* $\forall V \in vars(C)$*,* $\forall a \in dom(V)$*,* (V, a) *is consistent wrt* C*.* P *is said to be arc consistent iff* $\forall C \in \mathscr{C}$*,* C *is arc-consistent.*

3 AC3 based algorithms

In this section, we present the basic coarse-grained algorithm to establish arc consistency, namely, AC3 [12]. Even if fine-grained algorithms such as AC4 [14], AC6 [2] and AC7 [3] have been introduced, the simplicity and relative efficiency (e.g., [20]) of AC3 have contributed to the fact that it is not still out of date. Further, some recent improvements, which are quite competitive with fine-grained algorithms, have been proposed. These new coarse-grained algorithms (or AC3 based algorithms) respectively correspond to AC2000 [5], AC2001/3.1 [5, 22],

Algorithm 1 arc-oriented AC3

1: $Q \leftarrow \{(C, X) \mid C \in \mathscr{C} \wedge X \in vars(C)\}$
2: **while** $Q \neq \emptyset$ **do**
3: pick (C, X) in Q
4: $nbRemovals \leftarrow revise(C, X)$
5: **if** $nbRemovals > 0$ **then**
6: **if** $dom(X) = \emptyset$ **then** return FAILURE
7: **else** $Q \leftarrow Q \cup \{(C', X') \mid X \in vars(C') \wedge X' \in vars(C') \wedge X \neq X' \wedge C \neq C'\}$
8: **end if**
9: **end while**
10: return SUCCESS

Algorithm 2 revise(C : Constraint, X : Variable) : integer

1: $nbRemovals \leftarrow 0$
2: **for** each $a \in dom(X)$ **do**
3: **if** $seekSupport(C, X, a) = false$ **then**
4: remove a from $dom(X)$
5: $nbRemovals \leftarrow nbRemovals + 1$
6: **end if**
7: **end for**
8: return $nbRemovals$

$AC3_d$ [18], AC3.2 [11] and AC3.3 [11]. Even if, for the sake of simplicity, from now on, we shall be mainly concerned with AC3, all what follows can be adapted to other coarse-grained algorithms.

As already stated, AC3 is a coarse-grained algorithm, that is to say, an algorithm whose principle is to apply successive revisions of pairs (C, X), called arcs, composed of a constraint C and of a variable X belonging to the set of variables of C. Each revision of an arc (C, X) aims at removing the values of $dom(X)$ without any support in C.

AC3 requires the management of a set Q recording the revisions to be still performed. Basically, Q corresponds to a set of arcs [12]. However, it is possible to consider Q as a set of variables [13, 6, 5, 22], and also, as a set of constraints. We present these three alternatives in the general context of non-binary constraint networks.

3.1 Arc-oriented AC3

First, we describe the variant of AC3 which uses an arc-oriented propagation scheme. This variant, which is simple, natural and the most commonly presented, is depicted in Algorithm 1. 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 removes values of dom(X) that have become inconsistent with respect to C. This function returns the number of removed values. On the other hand, the function *seekSupport*

Algorithm 3 variable-oriented AC3

```
1: Q ← {X | X ∈ 𝒳}
2: ∀C ∈ 𝒞, ∀X ∈ vars(C), ctr(C,X) ← 1
3: while Q ≠ ∅ do
4:    pick X in Q
5:    for each C | X ∈ vars(C) do
6:       if ctr(C,X) = 0 then continue
7:       for each Y ∈ vars(C) do
8:          if needsNotBeRevised(C,Y) then continue
9:          nbRemovals ← revise(C,Y)
10:         if nbRemovals > 0 then
11:            if dom(Y) = ∅ then return FAILURE
12:            Q ← Q ∪ {Y}
13:            for each C' | C' ≠ C ∧ Y ∈ vars(C') do
14:               ctr(C',Y) ← ctr(C',Y) + nbRemovals
15:            end for
16:         end if
17:      end for
18:      for each Y ∈ vars(C) do ctr(C,Y) ← 0
19:   end for
20: end while
21: return SUCCESS
```

Algorithm 4 needsNotBeRevised(C : Constraint, X : Variable) : boolean

```
1: return (ctr(C,X) > 0 and ∄Y ∈ vars(C) | Y ≠ X ∧ ctr(C,Y) > 0)
```

determines whether there exists a support of (X, a) in C. According to the the implementation of this function, we obtain the different AC3 based algorithms. The algorithm is stopped when the set Q becomes empty.

3.2 Variable-oriented AC3

The second variant of AC3, uses a variable-oriented propagation scheme as proposed by [13, 6]. The principle is to insert in Q all variables with reduced domains. Initially, all variables are inserted in Q. Then, iteratively, each variable X of Q is selected and each constraint C binding X is considered. Then, it is possible to perform the revision of all arcs (C, Y) with $Y \neq X$. When the revision of an arc (C, Y) involves the removal of some values in $dom(Y)$, the variable Y is added to Q.

The reader can notice that the description of Algorithm 3 differs slightly from the principle presented just above. Indeed, to avoid useless treatments, it is necessary to introduce some counters in order to determine whether a given revision is essential. For instance, let us assume a binary constraint $C_{i,j}$ binding the variables X_i and X_j. If the selection of the variable X_i involves an effective revision of $(C_{i,j}, X_j)$ (i.e., the removal of, at least, a value from the domain of X_j), and, if next, the selection of X_j involves an effective revision of $(C_{i,j}, X_i)$,

Algorithm 5 constraint-oriented AC3

```
1: Q ← {C | C ∈ 𝒞}
2: ∀C ∈ 𝒞, ∀X ∈ vars(C), ctr(C,X) ← 1
3: while Q ≠ ∅ do
4:    pick C in Q
5:    for each Y ∈ vars(C) do
6:       if needsNotBeRevised(C,Y) then continue
7:       nbRemovals ← revise(C,Y)
8:       if nbRemovals > 0 then
9:          if dom(Y) = ∅ then return FAILURE
10:         for each C' | C' ≠ C ∧ Y ∈ vars(C') do
11:            Q ← Q ∪ {C'}
12:            ctr(C',Y) ← ctr(C',Y) + nbRemovals
13:         end for
14:      end if
15:   end for
16:   for each Y ∈ vars(C) do ctr(C,Y) ← 0
17: end while
18: return SUCCESS
```

then there is no need to perform again the revision of $(C_{i,j}, X_j)$ if X_i is again selected and if the domain of X_i has not been modified elsewhere. As another illustration (as expressed in [5]), let us assume a ternary constraint $C_{i,j,k}$. If the selection of the variable X_i involves a revision of $(C_{i,j,k}, X_j)$ and of $(C_{i,j,k}, X_k)$ then there is no need to perform again the revision of $(C_{i,j,k}, X_k)$ if the variable X_j is selected and if the domains of X_i and of X_j have not been modified elsewhere.

By associating a counter $ctr(C, X)$ with any arc, it is possible to determine which revisions are relevant. The value of $ctr(C, X)$ denotes the number of removed values in $dom(X)$ since the last revision involving C. Initially, this value is arbitrarily fixed to 1 for all counters. Then, when a variable X is selected and when a constraint C binding X is considered, two situations can happen. If X is the only variable in $vars(C)$ such that $ctr(C, X) > 0$, then the revision of all arcs (C, Y) with $Y \neq X$ is performed. Otherwise (second situation), all arcs (C, Y), including $Y = X$, are revised. Indeed, it is also relevant to revise (C, X) since at least another variable of C has been modified elsewhere. This is the function *needsNotBeRevised*, described by Algorithm 4 which allows determining whether the revision of an arc is relevant. When taking into consideration the second situation and the fact that all counters related to C are reinitialized to 0 after C has been considered, the test of the line 8 of Algorithm 3 becomes meaningful: it allows avoiding useless revisions.

3.3 Constraint-oriented AC3

The third AC3 variant uses a constraint-oriented propagation scheme (as $AC3_d$ can be regarded in the binary case) and is depicted in Algorithm 5. The

principle is to insert in Q all constraints for which at least a revision is necessary. Initially, all constraints are inserted in Q. Then, iteratively, each constraint C of Q is selected and each variable X of $vars(C)$ is considered. Similarly as the variable-oriented variant, the introduction of some counters allows avoiding useless revisions.

4 Revision ordering heuristics

At this step, it is natural to wonder about the practical interest of the variable-oriented and constraint-oriented variants. Indeed, as the three variants seem to be equivalent, we could have just introduce the arc-oriented one since it is simpler and more natural. The answer is that the nature of the elements of the set Q can have important repercussions on the overall behaviour of the algorithms. From a certain perspective, the variable-oriented and constraint-oriented variants have a grain bigger than the arc-oriented one. Instead of being a drawback (as it seems to be at first sight), it can be in fact an advantage. This is what we are going to show by introducing so-called revision ordering heuristics, i.e., heuristics to order the revisions to be applied by AC3 (or its extensions)[1].

Some of the heuristics that we introduce here are original and some other are simply taken from or adaptations of previous works (a discussion about related work is proposed later in this section). But, first, we present the revision ordering heuristic $fifo$ that can be defined without any ambiguity whatever variant is chosen. This heuristic involves selecting the oldest element of Q (viewed as a queue).

On the other hand, from now on, we shall introduce the composition of heuristics whose operator is denoted $\circ$ (as in [19]). A composed heuristic $h_2 \circ h_1$ means that the heuristic h_1 is used first, and then, if necessary, the heuristic h_2 is used to break ties (a tie is a set of elements that are considered as equivalent by an heuristic). For all heuristics (including composed ones) presented below, when a tie is still to be broken, the oldest element of the tie is selected (using implicitly $fifo$). Note that other "final" tie-breakers could be studied.

4.1 Variable-oriented heuristics

Each of the variable-oriented heuristics, i.e., heuristics adapted to the variable-oriented variant, selects a variable X from Q with:

- dom^v: the smallest current domain size,
- rem^v: the greatest proportion of removed values in its domain.
- $ddeg$: the greatest current (also called dynamic) degree,

In some way, dom^v and rem^v are complementary. The former is based on the number of remaining values whereas the latter is based on the number (proportion) of removed values (since the last selection of the variable). Note that

[1] We think that the term of "revision ordering heuristics" is more appropriate than "ordering heuristics" [21], "constraint ordering heuristics" [8] or "arc heuristics" [18].

considering a raw number of removed values is in favour of large domains (since there are more opportunities to remove values) and, consequently, usually entails more constraint checks. This is the reason why we have preferred using proportions.

4.2 Constraint-oriented heuristics

Each of the constraint-oriented heuristics, i.e. heuristics adapted to the constraint-oriented variant, selects a constraint C from Q with:

- dom^c: the smallest current domain size,
- rem^c: the greatest proportion of removed values in its domain.

The heuristics dom^c and rem^c are similar to dom^v and rem^v. Remember that we call current domain of a constraint C the Cartesian product of the current domains associated with the variables in $vars(C)$. It must not be confused with the constraint size or satisfiability of [21]. Hence, the removed values from the domain of a constraint correspond to the removed tuples due to the modification of the domains of some variables. For instance, let us consider a binary constraint $C_{i,j}$ involving two variables X_i and X_j. Assume that, at last selection of $C_{i,j}$, the domains of X_i and X_j had 10 values. Then, if, at current selection, the domains respectively have 6 and 8 values, the size of the current domain $C_{i,j}$ is 48 and the proportion of removed values is 52/100.

4.3 Arc-oriented heuristics

Each of the arc-oriented heuristics, i.e., heuristics adapted to the arc-oriented variant, selects an arc (C, X) from Q with:

- dom^v: the variable which has the smallest current domain size,
- dom^c/dom^v: the smallest ratio between the current domain size of the constraint (i.e., the number of tuples in the Cartesian product built from the current domains attached to the variables involved in the constraint) and the current domain size of the variable,
- $ddeg \circ dom^v$: the variable which has the smallest current domain size, and in case of equivalence, the greatest current degree.

4.4 Related work

In this subsection, we present some works related to revision ordering heuristics. First, let us cite the seminal work of [21] which propose different revision ordering heuristics to be used with the arc-oriented variant of AC3. These heuristics devised for binary problems are based on three major features:

- the number of supports in each constraint (called satisfiability),
- the number of values in the domain of each variable,
- the degree of each variable.

The heuristic *sat up* is based on satisfiability and allows to obtain interesting results in terms of constraint checks. However, determining the number of supports in each constraint and maintaining this information is not a very practical approach. This observation has been stated by [8] which propose another heuristic κ_{ac} based on the number of supports in each constraint in order to minimizing the constrainedness of the resulting subproblem. Some experiments [8] show that κ_{ac}, which is time expensive, performs less constraint checks than *sat up* and *dom j up* at the phase transition of arc consistency.

The heuristic *dom j up* selects the arc $(C_{i,j}, X_i)$ such that the variable X_j, i.e. the variable relaxed against, has the smallest current domain size. With respect to our notation, this heuristic corresponds to dom^c/dom^v (when considering binary problems). There is also a correspondence between *dom j up* and the heuristic dom^v defined for the variable-oriented variant.

The heuristic *deg down* which corresponds to *ddeg* is particularly disappointing in the experiments of [21]. We have made the same observation.

On the other hand, [15] mentions the issue of ordering the removed values which are put in different queues (a queue per variable). However, this approach is specific to fine-grained algorithms. [10] propose an original approach by managing propagation events associated with variables. Each event entails the immediate propagation of some constraints binding the variable associated with this event. A layered propagation architecture schedules the propagation of constraints according to a compromise between the provided information and the computation cost.

Finally, let us mention the work of [18] which emphasizes the importance of revision ordering heuristics as well as so-called domain heuristics. In [19], a precise revision ordering heuristic, called *comp* is presented as being tuned for $AC3_d$ (an arc-oriented AC3-based variant). It is a heuristic composed of 6 basic criteria. Roughly speaking, it corresponds to (when only considering the two first criteria) $ddeg \circ dom^v$.

5 Experiments

To compare the efficiency of the heuristics introduced in this paper, we have implemented them in Java and performed some experiments (run on a PC Pentium IV 2,4GHz 512MB under Linux) with respect to random, academic and real-world problems. Performances have been measured in terms of the CPU time in seconds (time), the number of constraint checks (#ccks) and the number of times (#rohs) a revision ordering heuristic has to select an element in the propagation set Q. The arc consistency algorithm that has been used for our experimentation is AC3.2 [11].

5.1 Stand-alone arc consistency

First, we have considered stand alone arc consistency which involves making arc consistent a CSP instance (that is to say, no search is performed). The first

variant	heuristic	P1			P2		
		time	#ccks	#rohs	time	#ccks	#rohs
variable	$fifo$	0.064	94,012	150	0.130	326,752	67
variable	dom^v	0.065	94,012	150	0.030	63,540	55
variable	rem^v	0.065	94,012	150	0.037	85,152	26
variable	$ddeg$	0.066	94,012	150	0.049	102,379	36
arc	$fifo$	0.071	94,012	1000	0.177	441,859	927
arc	dom^v	0.094	94,012	1000	0.120	204,346	895
arc	dom^c/dom^v	0.195	94,012	1000	0.297	113,798	1,060
arc	$ddeg \circ dom^v$	0.151	94,014	1000	0.119	115,277	490
constraint	$fifo$	0.063	94,012	500	0.188	484,108	600
constraint	dom^c	0.096	94,012	500	0.079	42,884	463
constraint	rem^c	0.103	94,012	500	0.053	77,299	131

Table 1. Stand alone arc consistency on random instances

series of experiments that we have run corresponds to some random problems. In this paper, a class of random CSP instances will be characterized by a 4-tuple $<n, d, m, t>$ where n is the number of variables, d the uniform domain size, m the number of binary constraints and t the number of unallowed tuples.

We present the results, given in Table 1 and Table 2, about some random binary instances studied in [3, 5, 22]. More precisely, 4 classes, denoted here P1, P2, P3 and P4, have been experimented. P1 = $<150, 50, 500, 1250>$ and P2 = $<150, 50, 500, 2350>$ respectively correspond to classes of under-constrained and over-constrained instances. P3=$<150, 50, 500, 2296>$ and P4=$<50, 50, 1225, 2188>$ correspond to classes of instances at the phase transition of arc consistency for sparse problems and for dense problems, respectively. For each class, mean results are given for 50 generated instances using the generator of [7].

variant	heuristic	P3			P4		
		time	#ccks	#rohs	time	#ccks	#rohs
variable	$fifo$	0,244	546,900	701	0.446	911,748	203
variable	dom^v	0.219	478,135	708	0.426	857,789	225
variable	rem^v	0.233	519,207	525	0.433	888,890	173
variable	$ddeg$	0.253	500,287	669	0.473	911,748	203
arc	$fifo$	0.266	569,228	6,068	0.517	944,679	12,659
arc	dom^v	0.333	518,310	6,301	1.034	867,232	15,195
arc	dom^c/dom^v	0.903	506,318	7,456	4.423	882,329	16,926
arc	$ddeg \circ dom^v$	0.490	493,795	5,781	1.706	867,232	15,195
constraint	$fifo$	0.252	561,398	3,740	0.460	927,966	7,571
constraint	dom^c	0.539	459,739	5,652	2.098	823,478	13,051
constraint	rem^c	0.496	527,545	2,740	2.095	897,306	6,055

Table 2. Stand alone arc consistency on random instances

One can immediately notice that the number of heuristic solicitations (#rohs) is far less important for the variable-oriented variant. In fact, when a variable is

variant	heuristic	SCEN#05 time	SCEN#05 #ccks	SCEN#05 #rohs	SCEN#08 time	SCEN#08 #ccks	SCEN#08 #rohs
variable	$fifo$	0.276	899,793	1,184	0.243	830,824	212
variable	dom^v	0.159	294,882	625	0.095	39,795	69
variable	rem^v	0.228	585,194	815	0.119	150,047	64
variable	$ddeg$	0.286	652,228	943	0.214	365,830	125
arc	$fifo$	0.315	981,555	19,701	0.560	2,178,674	17,040
arc	dom^v	2.291	742,330	20,731	4.478	876,989	8,067
arc	dom^c/dom^v	7.921	251,064	8,567	4.662	30,674	816
arc	$ddeg \circ dom^v$	4.916	687,107	18,660	10.099	904,712	8,748
constraint	$fifo$	0.303	964,764	12,000	0.461	1,877,001	5,508
constraint	dom^c	4.277	261,892	9,010	2.043	25,028	781
constraint	rem^c	4.890	298,310	7,036	3.112	74,614	984

Table 3. Stand alone arc consistency on RLFAP instances

selected in the propagation set Q, it entails a number of revisions related to the degree of the variable. On the other hand, for the arc-oriented variant, when an arc is selected, it just entails one revision, and, for the constraint-oriented variant, when a k-ary constraint is selected, it entails at most k revisions. Hence, the variable-oriented variant has a bigger grain than the other variants: the number of constraints checks performed after each solicitation is far more important.

We also observe that the variable-oriented variant clearly appears to be the fastest one when using some revision ordering heuristics, and, more precisely, the heuristic dom^v. This behaviour can be explained as follows. The variable-oriented variant requires less solicitations, as stated above, and each solicitation is cheap. Indeed, the overhead of picking the best element in the propagation set is limited for the variable-oriented variant, unlike other ones, since there are less variables than constraints and arcs.

In terms of constraint checks, the best heuristics are the constraint-oriented heuristic dom^c and the variable-oriented dom^v whereas the worse heuristics are the three versions of $fifo$. However, the time performance of these "standard" heuristics is not too bad as, systematically, the first element of the propagation set is selected.

Next, we have tested real-world instances, taken from the FullRLFAP archive[2], which contains instances of radio link frequency assignment problems. Table 3 presents the results obtained for two instances, denoted SCEN#05 and SCEN#08, studied in [3, 18, 22], and confirms all remarks expressed above. Note that there is a gap between the standard heuristics $fifo$ and some other heuristics with respect to the number of constraints checks required for SCEN#08. A similar behaviour has been observed by [18].

5.2 Maintaining arc consistency during search

As it appears that one of the most efficient complete search algorithms is the algorithm which Maintains Arc Consistency during the search of a solution [16,

[2] We thank the Centre d'Electronique de l'Armement (France).

variant	heuristic	bqwh-15-106		Q1		Q2	
		time	#ccks	time	#ccks	time	#ccks
variable	$fifo$	4.68	1.601M	166.43	100.192M	44.21	15.321M
variable	dom^v	3.87	1.211M	122.77	66.270M	39.86	8.860M
variable	rem^v	4.11	1.270M	132.22	77.955M	45.59	11.292M
variable	$ddeg$	5.57	1.879M	194.13	102.730M	71.76	14.565M
arc	$fifo$	4.76	1.615M	173.26	103.099M	44.20	15.639M
arc	dom^v	6.28	0.921M	251.91	60.276M	69.39	9.329M
arc	dom^c/dom^v	26.81	1.193M	932.59	66.183M	124.54	8.858M
arc	$ddeg \circ dom^v$	28.91	1.015M	1,170.90	60.305M	222.01	9.317M
constraint	$fifo$	4.66	1.599M	169.65	100.915M	43.24	15.444M
constraint	dom^c	11.41	0.858M	491.94	52.753M	94.46	7.060M
constraint	rem^c	19.28	1.306M	809.09	80.343M	144.21	11.528M

Table 4. Maintaining arc consistency on random instances

4], we have implemented a MAC3.2 version which integrates the $dom/ddeg$ [4, 17] variable ordering heuristic, and the lexicographic value ordering heuristic.

First, to study the behaviour of the different heuristics wrt problems involving random generation, we have considered the following classes of instances :

- one class, denoted bqwh-15-106, of 100 satisfiable balanced Quasigroup With Holes (bQWH) instances [9] of order 15 with 106 holes,
- two classes Q1=<80, 10, 400, 35> and Q2=<900, 10, 1250, 70> of 100 random binary instances situated at the phase transition of search for relatively dense problems ($\approx 12\%$) with low tightness (35%) and sparse problems ($\approx 0.3\%$) with high tightness ($\approx 70\%$), respectively.

Table 4 presents the results obtained for all these classes. When considering the number of constraint checks, one can observe that all heuristics based on dom^c or dom^v are the best ones. More precisely, the constraint-oriented heuristic dom^c outperforms all other ones and saves about 50% of the constraint checks required by $fifo$ heuristics. We also note that the coarser grain of the variable-

variant	heuristic	SCEN#11			GRAPH#14		
		time	#ccks	#rohs	time	#ccks	#rohs
variable	$fifo$	88.950	36.379M	470,966	1.713	1.237M	5,700
variable	dom^v	80.547	22.238M	308,608	1.730	1.189M	4,428
variable	rem^v	82.461	22.329M	310,253	1.775	1.188M	4,428
variable	$ddeg$	98.475	34.808M	595,373	2.666	1.215M	4,978
arc	$fifo$	90.381	33.688M	7,548,869	1.711	1.237M	39,838
arc	dom^v	93.596	18.511M	4,781,551	4.271	1.218M	35,870
arc	dom^c/dom^v	214.370	21.169M	4,336,842	22.552	1.180M	30,517
arc	$ddeg \circ dom^v$	410.737	18.775M	4,833,847	24.614	1.218M	35,961
constraint	$fifo$	90.054	36.130M	5,586,648	1.708	1.237M	30,030
constraint	dom^c	111.995	17.318M	3,256,374	6.901	1.190M	24,032
constraint	rem^c	158.468	22.079M	4,136,456	7.847	1.188M	23,197

Table 5. Maintaining arc consistency on RLFAP instances

variant	heuristic	*cc*-7-2	*cc*-7-3	*gr*-34-9	*gr*-34-10	*qa*-5	*qa*-6
variable	$fifo$	9.159	238.630	47.853	1,736.913	62.104	5,943.426
variable	dom^v	8.688	213.966	31.238	1,174.526	56.138	4,331.342
variable	rem^v	8.531	222.600	35.666	1,292.841	57.310	4,636.216
variable	$ddeg$	8.955	291.663	52.885	1,907.120	69.987	6,435.984
arc	$fifo$	10.263	276.240	56.017	1,997.052	68.729	5,679.748
arc	dom^v	39.856	735.510	90.419	5,025.990	129.377	10,096.466
arc	dom^c/dom^v	129.977	2,636.612	251.786	13,871.150	457.250	49,364.050
arc	$ddeg \circ dom^v$	498.214	7,918.398	1,121.336	109,292.280	719.742	126,063.110
constraint	$fifo$	9.342	229.128	52.504	1,951.002	65.983	5,516.274
constraint	dom^c	19.926	736.137	98.216	6,747.980	205.111	18,382.519
constraint	rem^c	38.754	1,210.565	528.161	30,428.539	335.617	33,818.809

Table 6. cpu time when maintaining arc consistency on academic instances

variant	heuristic	*cc*-7-2	*cc*-7-3	*gr*-34-9	*gr*-34-10	*qa*-5	*qa*-6
variable	$fifo$	3.196M	116.585M	42.836M	1,609.906M	56.326M	3,584.318M
variable	dom^v	3.196M	111.633M	34.882M	1,347.494M	47.610M	2,765.204M
variable	rem^v	3.196M	111.652M	38.052M	1,434.013M	48.741M	2,876.704M
variable	$ddeg$	2.977M	121.794M	46.012M	1,660.198M	56.327M	3,584.320M
arc	$fifo$	3.155M	118.312M	45.175M	1,680.035M	58.098M	3,631.175M
arc	dom^v	2.261M	93.980M	26.979M	1,028.886M	42.862M	2,179.902M
arc	dom^c/dom^v	1.766M	92.250M	32.315M	1,213.946M	45.920M	2,677.797M
arc	$ddeg \circ dom^v$	2.503M	106.447M	27.121M	1,031.489M	42.861M	2,179.929M
constraint	$fifo$	3.196M	116.252M	44.733M	1,684.691M	56.366M	3,580.447M
constraint	dom^c	1.366M	80.645M	29.084M	1,115.535M	40.467M	2,125.472M
constraint	rem^c	2.843M	113.639M	40.145M	1,462.286M	48.367M	2,882.489M

Table 7. #ccks when maintaining arc consistency on academic instances

oriented heuristics has an impact on the number of constraint checks (dom^v entails more checks than dom^c). However, it is highly compensated by the small overhead of such heuristics as explained above.

Finally, we introduce three additional tables that confirm our previous results. Table 5 corresponds to the real-world instances SCEN#11 and GRAPH #14 of the RLFAP archive. Tables 6 and 7 respectively correspond to the cpu time and the number of constraint checks required to solve the following academic instances:

- two chessboard coloring instances [1], denoted *cc*-7-2 and *cc*-7-3, involving quaternary constraints,
- two Golomb ruler instances[3], denoted *gr*-44-9 and *gr*-44-10, involving binary and ternary constraints,
- two prime queen attacking instances[4], denoted *qa*-5 and *qa*-6, involving only binary constraints.

To summarize, the efficiency of all variable-oriented heuristics based on current domain sizes has been established both for stand alone arc consistency and

[3] See problem006 at `http://4c.ucc.ie/~tw/csplib/`
[4] See problem029 at `http://4c.ucc.ie/~tw/csplib/`

for MAC. On the other hand, the arc-oriented and constraint-oriented heuristics which allow saving some constraint checks, are quite costly. Nevertheless, there are at least three alternatives to improve all heuristics:

- restricting the search of the best element to a subset of Q,
- performing the search of the k best elements every k selections,
- improving the management of Q using a pigeonhole sort [21].

Such an optimization is proposed by [19] with the heuristic *comp* which belongs to the group of the best heuristics (when considering constraint checks). [19] uses an efficient representation for the queue which allows compensating the time spent on selection and maintenance.

6 Conclusion

Our first motivation, in this paper, was to clarify the situation about the different propagation schemes of AC3-based algorithms. Indeed, we can define three variants which respectively correspond to algorithms with an arc-oriented, variable-oriented and constraint-oriented propagation scheme. We have presented general versions of these algorithms so that they can be applied to non-binary problems whereas being careful about avoiding useless revisions (for variable-oriented and constraint-oriented variants). To determine which variant is the more appropriate to establish arc-consistency, we have studied the impact of introducing so-called revision ordering heuristics. Such heuristics have been proposed by [21] with respect to the arc-oriented variant and experimentations performed when arc consistency is used as a preprocessing of binary problems.

In this paper, we have extended our understanding of revision ordering heuristics by:

- introducing new heuristics and adapting heuristics of [21] with respect to the different variants of AC3-based algorithms,
- experimenting these heuristics when arc consistency is maintained during the search of a solution for binary and non binary problems.

Experimental results show that heuristics based on the number of removed values are disappointing, unlike heuristics based on the number of remaining values. The best heuristics save up to 50% of constraint checks when compared to the "standard" heuristic $fifo$. Hence, it confirms for MAC the observation of [21]. Also, the best variable-oriented heuristics can save about 25% of cpu-time when compared to $fifo$. However, it turns out that constraint-oriented and arc-oriented heuristics are penalized in terms of CPU time. The reason is that the constraint-oriented and arc-oriented variants need to record more elements in the propagation set Q than the variable-oriented variant. Hence, both variants are time-consuming when one heuristic iterates all recorded elements in Q.

To conclude, even if there exists some perspectives to optimize all these heuristics by avoiding systematic iterations of Q, we believe that a AC3-based variable-oriented variant associated with a revision ordering heuristic based on dom^v should preserve its advantage (as it could also benefit from such improvements).

References

1. M. Beresin, E. Levin, and J. Winn. A chessboard coloring problem. *The College Mathematics Journal*, 20(2):106–114, 1989.
2. C. Bessière. Arc consistency and arc consistency again. *Artificial Intelligence*, 65:179–190, 1994.
3. C. Bessière, E.C. Freuder, and J. Régin. Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, 107:125–148, 1999.
4. C. Bessière and J. Régin. MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems. In *Proceedings of CP'96*, pages 61–75, 1996.
5. C. Bessière and J. Régin. Refining the basic constraint propagation algorithm. In *Proceedings of IJCAI'01*, pages 309–315, 2001.
6. A. Chmeiss and P. Jégou. Efficient path-consistency propagation. *International Journal on Artificial Intelligence Tools*, 7(2):121–142, 1998.
7. D. Frost, R. Dechter, C. Bessière, and J.C. Régin. Random uniform CSP generators. http://www.lirmm.fr/~bessiere/generator.html, 1996.
8. I.P. Gent, E. MacIntyre, P. Prosser, P. Shaw, and T. Walsh. The constraindedness of arc consistency. In *Proceedings of CP'97*, pages 327–340, 1997.
9. C.P. Gomez and D. Shmoys. Completing quasigroups or latin squares: a structured graph coloring problem. In *Proceedings of Computational Symposium on Graph Coloring and Generalization*, 2002.
10. F. Laburthe and le projet OCRE. CHOCO : implémentation du noyau d'un système de contraintes. In *Actes de JNPC'00*, pages 151–165, 2000.
11. C. Lecoutre, F. Boussemart, and F. Hemery. Exploiting multidirectionality in coarse-grained arc consistency algorithms. In *Proceedings of CP'03*, pages 480–494, 2003.
12. A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:118–126, 1977.
13. J.J. McGregor. Relational consistency algorithms and their applications in finding subgraph and graph isomorphism. *Information Science*, 19:229–250, 1979.
14. R. Mohr and T.C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.
15. J.C. Régin. *Développement d'outils algorithmiques pour l'intelligance artificielle. Application à la chimie organique*. PhD thesis, Universit Montpellier II, 1995.
16. D. Sabin and E. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of the PPCPA'94*, Seattle WA, 1994.
17. B.M. Smith and S.A. Grant. Trying harder to fail first. In *Proceedings of ECAI'98*, pages 249–253, Brighton, UK, 1998.
18. 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.
19. M.R.C. van Dongen. Lightweight arc-consistency algorithms. Technical Report TR-01-2003, University college Cork, 2003.
20. R.J. Wallace. Why AC3 is almost always better than AC4 for establishing arc consistency in CSPs. In *Proceedings of IJCAI'93*, pages 239–245, 1993.
21. R.J. Wallace and E.C. Freuder. Ordering heuristics for arc consistency algorithms. In *Proceedings of NCCAI'92*, pages 163–169, 1992.
22. Y. Zhang and R.H.C. Yap. Making AC3 an optimal algorithm. In *Proceedings of IJCAI'01*, pages 316–321, Seattle WA, 2001.