2-C3: From Arc-Consistency to 2-Consistency

Marlene Arangú and Miguel A. Salido and Federico Barber
Instituto de Automática e Informática Industrial
Universidad Politécnica de Valencia.
Valencia, Spain

Abstract
Arc consistency algorithms are widely used to prune the search space of Constraint Satisfaction Problems (CSPs). Since many researchers associate arc consistency with binary normalized CSPs, there is a confusion between the notion of arc consistency and 2-consistency. 2-consistency guarantees that any instantiation of a value to a variable can be consistently extended to any second variable. Thus, 2-consistency can be stronger than arc-consistency in binary CSPs. In this paper, we present a new algorithm, called 2-C3, which achieves 2-consistency in binary and non-normalized CSPs. This algorithm is a reformulation of the well-known AC3 algorithm. The evaluation section shows that 2-C3 is able to prune more search space than AC3 and AC4.

Introduction
Constraint programming is a software technology for the description and effective solving of large and complex problems (in many areas of the real life), particularly combinatorial problems (Dechter 2003; Bartáková 1999). Many of these problems can be modeled as constraint satisfaction problems (CSPs) and can be solved using constraint programming techniques. The basic idea of CSP is to model the problem as a set of variables with finite domains (the values for the variables) and a set of constraints that impose a limitation on the values that a variable, or a combination of variables, may be assigned. The task is to find an assignment of values for the variables that satisfy all the constraints. In general, the tasks posed in the CSP paradigm are computationally intractable (NP-Complete).

Some of the algorithms used to manage CSP are systematic search, constraint propagation and consistency techniques. For reasons of brevity in this paper we only explain the consistency techniques.

The consistency-enforcing algorithm performs any partial solution of a small sub-network that is extensible to a surrounding network. The number of possible combinations can be huge, while only very few are consistent. By eliminating redundant values from the problem definition, the size of the solution space decreases. If any domain becomes empty as a result of reduction, then it is immediately known that the problem has no solution (Ruttkay 1998). There exist many levels of consistency depending on the number of variables involved: node-consistency involves only one variable; arc-consistency involves two variables; path-consistency involves three variables; and k-consistency involves k variables. More information can be seen in (Bartáková 2001; Dechter 2003).

Arc-consistency algorithms are a major component of many industrial and academic CSP solvers. Arc consistency algorithms are based on the notion of support. These algorithms ensure that each value in the domain of each variable is supported by one or more values in the domain of each variable by which it is constrained. Arc consistency algorithms which lie at the heart of a CSP solver, are very-time consuming. For this reason, arc consistency algorithms and their time complexity are areas that have been heavily researched (van Dongen, Dieker, and Saporzhnik 2008).

Proposing efficient algorithms to enforce arc-consistency has always been considered as a central question in the constraint reasoning community. Thus, there are many arc-consistency algorithms such as: AC1, AC2, and AC3 (Mackworth 1977); AC4 (Mohr and Henderson 1986); AC5 (Perlin 1992; Hentenryck, Deville, and Teng 1992); AC6 (Bessiere and Cordier 1993); AC7 (Bessiere, Freuder, and Régin 1999); AC8 (Chmeiss and Jegou 1998); AC2001, AC3.1 (Bessiere et al. 2005); and more. However, AC3 and AC4 are the ones most often used (Bartáková 2001).

Algorithms that perform arc-consistency have focused their improvements on time-complexity and space-complexity. The main improvements have been achieved by: changing the means of propagation from arcs to values, (i.e., changing the granularity from coarse-grained to fine-grained); appending new structures; performing bidirectional searches (AC7); changing the support search: searching for all supports (AC4) or searching for only the necessary supports (AC6, AC7, AC8 and AC2001); improving the propagation (i.e., AC7 and AC2001, which perform propagation only when necessary); etc.

The concept of consistency was generalized to k-consistency by (Freuder 1978). Thus, 2-consistency is related to constraints that involve two variables. Furthermore, many works on arc-consistency made the simplified assumptions that CSPs are binary (all constraints involve two variables) and normalized (two different constraints do not involve exactly the same variables), these notations are very
simple and the new concepts are easy to present. In their work (Rossi, Van Beek, and Walsh 2006) show a strange effect of associating arc-consistency with binary normalized CSPs: the confusion between the notions of arc-consistency and 2-consistency (2-consistency guarantees that any instantiation of a variable to a value can be consistently extended to any second variable). In binary CSPs, 2-consistency is at least as strong as arc-consistency. When the CSP is binary and normalized, arc-consistency and 2-consistency perform the same pruning. However, this is not true in general. For more details see (Rossi, Van Beek, and Walsh 2006).

Few works have been done to develop algorithms to achieve 2-consistency in binary CSPs. Therefore, in the following section, we provide the necessary definitions to understand the rest of the paper, and we present an example to clarify some important concepts. Later, we present our 2-C3 algorithm. This algorithm is a reformulation of AC3 to achieve 2-consistency in binary and non-normalized CSPs. An evaluation is presented to compare the behavior of the 2-C3 algorithm against AC3 and AC4 algorithm. Finally, some conclusions are presented.

Definitions

By following the standard notations and definitions in the literature (Bessiere 2006; Barták 2001; Dechter 2003), we have summarized the basic definitions that are used throughout the paper.

Definition 1. Constraint Satisfaction Problem (CSP) is a triple \( P = (X, D, R) \) where: \( X \) is the finite set of variables \( \{X_1, X_2, ..., X_n\} \); \( D \) is a set of domains \( D = D_1, D_2, ..., D_n \) such that for each variable \( X_i \in X \) there is a finite set of values that the variable can take; \( R \) is a finite set of constraints \( R = \{R_1, R_2, ..., R_m\} \) which restrict the values that the variables can take simultaneously.

Definition 2. Binary constraint. A constraint is binary if it involves only two variables. A CSP is binary iff all constraints are binary.

Definition 3. Block of constraints. A block of constraints \( C_{ij} \) is a set of binary constraints that involve the same variables \( X_i \) and \( X_j \).

Definition 4. Normalized CSP. A CSP is normalized iff two different constraints do not involve exactly the same variables.

Definition 5. Instantiation. It is a pair \( \langle X_i, a \rangle \), which represents an assignment of the value \( a \) to the variable \( X_i \), and \( a \) is in the domain of \( X_i \). We can use \( X_i = a \equiv \langle X_i, a \rangle \).

Definition 6. Constraint Satisfy. A constraint \( R_{ij} \) is satisfied if the instantiation of \( \langle X_i, a \rangle \) and \( \langle X_j, b \rangle \) is legal for this constraint \((\langle X_i, a \rangle, \langle X_j, b \rangle) \in R_{ij}\).

Definition 7. Value arc-consistency. A value \( a \in D_i \) is arc-consistent relative to \( X_j \) iff there exists a value \( b \in D_j \) such that \( \langle X_i, a \rangle \) and \( \langle X_j, b \rangle \) satisfy the constraint \( R_{ij} \).

Definition 8. Variable arc-consistency. A variable \( X_i \) is arc-consistent relative to \( X_j \) iff all values in \( D_i \) are arc-consistent.

Definition 9. CSP arc-consistency. A CSP is arc-consistent iff all the variables are arc-consistent, e.g., all the constraints \( R_{ij} \) and \( R_{ji} \) are arc-consistent. (Note: here we are talking about full arc-consistency).

Definition 10. Value 2-consistency. A value \( a \in D_i \) is 2-consistent relative to \( X_j \) iff there exists a value \( b \in D_j \) such that \( \langle X_i, a \rangle \) and \( \langle X_j, b \rangle \) satisfy all the constraints \( R^k_{ij} \), \( (\forall k) : (X_i = a, X_j = b) \in R^k_{ij} \).

Definition 11. Variable 2-consistency. A variable \( X_i \) is 2-consistent relative to \( X_j \) iff all values in \( D_i \) are 2-consistent.

Definition 12. CSP 2-consistency. A CSP is 2-consistent iff all the variables are 2-consistent, e.g., any instantiation of a value to a variable can be consistently extended to any second variable.

We will focus our attention on binary and non-normalized CSPs. Figure 1 left shows a binary CSP, which is presented in (Rossi, Van Beek, and Walsh 2006) with two variables \( X_1 \) and \( X_2 \), \( D_1 = \{1, 2, 3\} \) and two constraints \( R_{12} : X_1 \leq X_2 \), \( R_{12} : X_1 \neq X_2 \). It can be observed that this CSP is arc-consistent due to the fact that every value of every variable has a support for constraints \( R_{12} \) and \( R_{12} \). In this case, arc-consistency does not prune any value of the domain of variables \( X_1 \) and \( X_2 \). However, as (Rossi, Van Beek, and Walsh 2006) show, this CSP is not 2-consistent because the instantiation \( X_1 = 3 \) can not be extended to \( X_2 \) and the instantiation \( X_2 = 1 \) can not be extended to \( X_1 \).

The Cost of Translating a non-normalized CSP into a Normalized One

The only way to translate a non-normalized CSP into a normalized one is by means of the extensional representation of constraints. It is well known that a constraint can be represented intensionally (by an expression) or extensionally (by the set of allowed or disallowed tuples). The vast majority of constraints presented in real problems are modeled intensionally. Some of these constraints are then extensionally represented to be managed by CSP solvers, filtering techniques, etc. However, this is a very hard task, particularly if the domains are huge or impossible in continuous domains. Actually, from the mathematical point of view, the extensional and intensional representation are equal. From the computational perspective, however, the intensional one is a lot more compact and expressive than the extensional one.

For instance, for \( D_1 = D_2 = \{0, 1, 2\} \), a constraint \( C_{12} \) with the intensional meaning \( R_{12} : X_1 < X_2 \) could be defined extensionally by allowed tuples \( R_{12} = \{(0, 1), (0, 2), (1, 2)\} \) or disallowed tuples \( R_{12} = \{(0, 0), (1, 0), (1, 1), (2, 0), (2, 1), (2, 2)\} \). In CSPs with large domains, for instance, \( D_1 = D_2 = \{1, 2, ..., 1000\} \), a constraint \( C_{12} \) with the extensional representation generates one million of tuples \((D_1 \times D_2)\) to be labeled as allowed or disallowed tuples. This task has a high temporal and spatial complexity.

Once, all the constraints of the non-normalized CSP have been translated into an extensional representation (allowed
The allowed tuples for the constraint involve variable sectored tuples. For instance, if there exist two constraints that same variables must be grouped in order to select all inter- or disallowed tuple sets, then $C_{ij} = \{(a, b) | (a, b) \in S_{ij} \land (a, b) \in S'_{ij}\}$.

For instance, Figure 2 shows an example of non-normalized CSP due to the fact that variables $X_1$ and $X_2$ are restricted by two different constraints ($R_{12}$ and $R'_{12}$). The Cartesian Product of variable domains is $D_1 \times D_2 = \{(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)\}$. The allowed tuples for the constraint $R_{12}$ are $S_{12} = \{(0, 0), (0, 1), (0, 2), (1, 1), (1, 2), (2, 2)\}$, while the allowed tuples for the constraint $R'_{12}$ are $S'_{12} = \{(0, 1), (0, 2), (1, 0), (1, 2), (2, 0), (2, 1)\}$; so that $C_{12} = S_{12} \cap S'_{12} = \{(0, 1), (0, 2), (1, 2)\}$. Thus, by applying arc-consistency we can achieve the reduced domains $D_1 = \{0, 1\}$ and $D_2 = \{1, 2\}$. Although this translation technique is quite simple, its implementation consumes a lot of time and requires the generation of structures that consume memory.

In conclusion, the cost of translating a non-normalized CSP into a normalized one is a prohibitive task in problems with large domains. The main research carried out regarding filtering techniques for constraint satisfaction has focused attention on normalized CSPs or extensionally represented constraints. However, real life problems are usually represented as CSPs with non-normalized CSPs with intensionally represented constraints. Therefore, the development of filtering techniques to manage these problems is necessary.

**Algorithm 2-C3**

2-consistency guarantees that any instantiation of a value to a variable can be consistently extended to any second variable. While arc-consistency only checks the support hold in the constraint, 2-consistency checks the support hold in all constraints of the set. Therefore, 2-consistency can be stronger than arc-consistency in binary CSPs.

Following, we present a new algorithm called 2-C3 that achieves 2-consistency in binary and non-normalized CSPs. This algorithm is a reformulation of the well-known AC3 algorithm. The main algorithm is a simple loop (see Algorithm 2, lines 10-20) that selects and revises the block of constraints stored in a queue $Q$ (see Algorithm 1) until no change occurs ($Q$ is empty), or until the domain of a variable becomes empty. The first case ensures that all values of domains are consistent with all constraints, and the second case returns that the problem has no solution.

**Algorithm 1 Revise procedure**

**Input:** A CSP $P'$ defined by two variables $X = (X_1, X_2)$, domains $D_i$ and $D_j$, and constraint set $C_{ij}$.

**Output:** $D_i$, such that $X_i$ is 2-consistent relative $X_j$ and the boolean variable $change$.

1: $change \leftarrow false$
2: for each $a \in D_i$ do
3:  if $\not\exists b \in D_j$ such that $(X_i = a, X_j = b) \in C_{ij}$ then
4:    remove $a$ from $D_i$
5:  $change \leftarrow true$
6: end if
7: end for
8: return $change$

The Revise procedure of 2-C3 is very close to the Revise procedure of AC3. The only difference is that the instantiation $(X_i = a, X_j = b)$ must be checked with the block of constraints $C_{ij}$ instead of with only one constraint. This set of constraints $C_{ij}$ could also be ordered in order to avoid unnecessary checks. If we order this set from the tightest constraint to the loosest constraint, the constraint checking will find inconsistency constraints sooner, in which case no further constraint checks must be carried out.

For the example shown in Figure 2, $Q$ initially stores the constraints $Q = \{C_{02}, C'_{02}, C_{12}, C'_{21}\}$. This table also shows the corresponding constraints (Figure 2, right).

Table 1 shows how the 2-C3 procedure evaluates each constraint for this example. In loops 1 and 2, sets $C_{02}$ and $C'_{02}$ have only one constraint: $R_{02}$ and $R'_{02}$, respectively.
Figure 2: Example of Binary non-normalized CSP. The arc-consistency algorithms do not perform any pruning, unless normalization is performed on the restrictions.

Algorithm 2 2-C3 procedure

Input: A CSP, \( P = (X, D, R) \)

Output: true and \( P' \) (which is 2-consistent) or false and \( P' \) (which is 2-inconsistent because some domain remains empty)

1: for every \( i, j \) do
2: \( C_{ij} = \emptyset \)
3: end for
4: for every arc \( R_{ij} \in R \) do
5: \( C_{ij} \leftarrow C_{ij} \cup R_{ij} \)
6: end for
7: for every set \( C_{ij} \) do
8: \( Q \leftarrow Q \cup \{ C_{ij}, C_{ji} \} \)
9: end for
10: while \( Q \neq \emptyset \) do
11: select and delete \( C_{ij} \) from queue \( Q \)
12: if \( \text{Revise}(C_{ij}) = \text{true} \) then
13: if \( D_i \neq \emptyset \) then
14: \( Q \leftarrow Q \cup \{ (C_{ki} \mid k \neq i, k \neq j) \} \)
15: else
16: return false /*empty domain*/
17: end if
18: end if
19: end while
20: return true

Thus, their checks are equivalent to AC3, however, in loops 3 and 4, each set has two constraints, so 2-C3 checks that both values hold with all the constraints of the set. Thus, in loop 3, where \( C_{12} \) is processed, it is verified whether or not value 0 of \( D_1 \) has support with value 0 of \( D_2 \). This is true for \( R_{12} \); however, when the same values are checked with the next constraint of set \( (R_{12}') \), this constraint is not satisfied. For this reason, the next value in \( D_2 \) (value 1) is sought, and both constraints \( (R_{12}, R_{12}') \) are satisfied with values \( X_1 = 0 \) and \( X_2 = 1 \). We denote the set that must be re-evaluated \( (C_{12}) \) using dotted lines. This set constraint was added to the queue \( Q \) (See Table 1, loop 4).

In the above example to achieve 2-consistency, 2-C3 performs 3 prunes of domain values, carries out 37 constraint checkings (Cc), and carries out 1 propagation (Np) in \( Q \). AC3 and AC4 do not prune any value nor do they carry out any propagation. AC3 carries out 29 constraint checkings, and AC4 carries out 54 constraint checkings.

Table 1: Loops carried out by 2-C3 for the example shown in Figure 2.

<table>
<thead>
<tr>
<th>Loop</th>
<th>Set</th>
<th>( C_{ij} )</th>
<th>( D_i )</th>
<th>( R_{ij} )</th>
<th>hold</th>
<th>change</th>
<th>Frame ( X_i )</th>
<th>Add ( Q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( C_{02} )</td>
<td>0, 0, 0, 2</td>
<td>( R_{02} )</td>
<td>yes</td>
<td>false</td>
<td>( X_1 )</td>
<td>( X_1 )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( C_{20} )</td>
<td>0, 0, 0, 2</td>
<td>( R_{02} )</td>
<td>yes</td>
<td>false</td>
<td>( X_1 )</td>
<td>( X_1 )</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( C_{12} )</td>
<td>0, 0, 0, 2</td>
<td>( R_{02} )</td>
<td>yes</td>
<td>false</td>
<td>( X_1 )</td>
<td>( X_1 )</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( C_{21} )</td>
<td>0, 0, 0, 2</td>
<td>( R_{02} )</td>
<td>yes</td>
<td>false</td>
<td>( X_1 )</td>
<td>( X_1 )</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>( C_{02} )</td>
<td>0, 0, 0, 2</td>
<td>( R_{02} )</td>
<td>yes</td>
<td>false</td>
<td>( X_1 )</td>
<td>( X_1 )</td>
<td></td>
</tr>
</tbody>
</table>

Experimental Results

In this section, we compare the behavior of the arc-consistency algorithms AC3 and AC4 and our proposed 2-consistency algorithm (2-C3). Furthermore, we implemented the most efficient versions of AC3 and AC4 with the improvement shown in (Mackworth 1977; Dechter 2003; Bessiere 2006; Barták 2001), in order to remove ambiguity and improving efficiency.
The experiments were performed on random instances. A random CSP instance was characterized by the 4-tuple < n, d, m, c >, where n was the number of variables, d the domain size, m the total number of binary constraints and c the number of constraints in each block. The constraints were in the form X_i op X_j, where X_i, X_j ∈ X and op ∈ {<, ≤, =, ≠, ≥}. We randomly generated binary and non-normalized problems. All the variables maintained the same domain size. The problems were randomly generated by modifying these parameters. We evaluated 50 test cases for each type of problem.

Thus, Tables 2 and 3 fixed three of the parameters and varied the other one in order to evaluate the algorithm performance when this parameter was increased. Performance was measured in terms of number of values pruned. All algorithms were written in C. The experiments were conducted on a PC Pentium IV (3.0 GHz processor and 1 GB RAM).

Table 2 shows the number of constraint checks and prunes using AC3, AC4 and 2-C3: the number of variables was increased from 50 to 700; the number of constraints was set at 50, 20, and 2, respectively (< n, 20, 800, 2 >). As in the above evaluation, the results were similar. 2-C3 was able to carry out more pruning (> 50%) than AC3 and AC4. In this case, the number of constraint checks increased as the number of constraints increased. Since the random problems maintained the same number of variables but the number of constraints increased, the random problems were tight.

Table 3 presents the number of constraint checks, where the number of constraints was increased from 50 to 700 and the number of variables, the domain size, and block of constraints were set at 50, 20, and 2, respectively (< 50, 20, m, 2 >).
Conclusions
Filtering techniques are widely used to prune the search space of CSPs. AC3 is one of the best known arc consistency algorithms, and different versions have improved the efficiency of the original one. Since many researchers associate arc consistency with binary normalized CSPs, there is a confusion between the notion of arc consistency and 2-consistency. In this paper, we have presented a reformulation of AC3 to achieve 2-consistency in binary and non-normalized CSPs. The evaluation section shows that 2-C3 achieves 2-consistency and is therefore able to prune more search space than both AC3 and AC4. This filtering algorithm could be very appropriate in search tools to manage non-normalized problems.

References


