Probabilistic Arc Consistency*

Student: Deepak Mehta Supervisor: M.R.C. van Dongen

Boole Centre for Research in Informatics/Cork Constraint Computation Centre

Abstract. The two most popular backtrack algorithms for solving Constraint Satisfaction Problems (CSPs) are Forward Checking (FC) and Maintaining Arc Consistency (MAC). MAC maintains full arc consistency while FC maintains a limited form of arc consistency during search. Previous work has shown that there is no single champion algorithm: MAC is more efficient on sparse problems which are tightly constrained but FC has an increasing advantage as problems become dense and constraints loose. Ideally a good search algorithm should find the right balance—for any problem—between visiting fewer nodes in the search tree and reducing the work that is required for detecting and removing inconsistent values. We propose to maintain a *probabilistic arc consistency* during search to achieve this. The idea is to assume that a support exists and skip the process of seeking a support if the probability of having some support for a value is at least equal to some, carefully chosen, stipulated bound. Experimental results show that the probabilistic approach performs well on both sparse and dense problems and in fact better than MAC and FC on the hardest problems in the phase transition.

1 Introduction

The two most popular backtrack algorithms for solving Constraint Satisfaction Problems (CSPs) are MAC [6] and FC [4]. MAC maintains full arc consistency during search. It ensures that each value in the domain of *each* variable is supported by at least one value in the domain of every variable by which it is constrained. FC maintains a limited form of arc consistency. It ensures that each value in the domain of each variable is FC *consistent*, i.e. supported by the value assigned to every past variable by which it is constrained. Previous work [3, 2] has shown that there is no single champion algorithm. MAC is more efficient than FC on sparse problems which are tightly constrained but FC has an increasing advantage as problems become dense and constraints become loose. For difficult problems the relationship between sparsity and tightness and between density and looseness roughly allows us to say that hard loose problems are better solved with FC, whereas hard tight problems are better solved with MAC. The reason why FC performs better than MAC for hard dense problems is that it exploits a common sense probabilistic argument: *the looser the constraints (the denser the hard problem), the higher the probability that* FC *consistency is tantamount to arc consistency*.

Ideally, a good search algorithm should find the right balance—for any problem — between visiting fewer nodes in the search tree and reducing the work that is required for detecting and removing inconsistent values. More specifically, for hard *dense* problems

* This work has received some support from Science Foundation Ireland under Grant No. 00/PI.1/C075.

a good search algorithm should keep the best features of MAC and FC by staying closer to MAC in terms of the number of visited nodes and closer to FC in terms of checks while for hard *sparse* problems it should behave like MAC. This is precisely what the probabilistic arc consistency discussed in this paper is able to do.

Arc consistency involves revisions of domains, which require support checks for identifying and deleting all unsupported values from the domain of a variable. In many revisions, some or all values find some support. For example, when RLFAP #11 is solved using MAC-3 or MAC-2001, 83% of the total revisions are ineffective and do not result in deleting any value. If we can predict the existence of a support with a high probability and avoid the process of seeking a support when the probability of having some support is at least equal to some, carefully chosen, stipulated threshold then a considerable amount of work can be saved.

In order to do so, first we show how to compute the probability of having some support for a value. Next, we introduce the notions of a *Probabilistic Support Condition* (PSC) and a *Probabilistic Revision Condition* (PRC). The PSC holds iff the probability of having some support for a value is above the threshold. If the PSC holds then we assume that a support exists and we will not seek it. The PRC holds iff for each value in a domain the probability of having some support is above the threshold. If the PRC holds then we assume that some support exists for each value and avoid the corresponding revision.

2 Background

For the purpose of this paper, before starting search all search algorithms transform the input CSP to its arc consistent equivalent. The *original* domain of a variable is the domain of that variable in this arc consistent equivalent. For the remainder of this paper for any variable x, we use $D(x)$ for the current domain of x and $D_o(x)$ for the original domain of x. The *directed constraint graph* of a given CSP is a directed graph having an arc (x, y) for each combination of two mutually constraining variables x and y. We will use G to denote the directed constraint graph of the input CSP.

The traditional approach to find if $a \in D(x)$ is supported by y is to *identify* some $b \in D(y)$ that supports a, which usually results in a sequence of support checks. Identifying the support is more than is needed to guarantee that a value is supportable: knowing that a support exists is enough. Most arc consistency algorithms proposed so far put a lot of effort in identifying a support. To reduce unnecessary checks and revisions to some extent, the notions of a *support condition* (SC) and a *revision condition* (RC) are introduced in [5] (see also [1]). SC guarantees that a value has *some* support while RC guarantees that *all* values have *some* support without identifying it. In the following paragraph we present a special version of SC and RC which facilitates the introduction of their probabilistic equivalents, which are to be presented in the following section.

Let C_{xy} be the constraint between x and y, let $a \in D(x)$, and let $R(y) = D_o(y) \setminus D(y)$ be the removed values from the original domain of y. The *support count* of (x, a) with respect to y, denoted *sc*(x, y, a), is the number of values in $D_o(y)$ supporting a. Note that $|R(y)|$ is an upper bound on the number of lost supports of (x, a) in $D(y)$. Therefore, if the following condition holds then (x, a) is supported by y:

$$sc(x, y, a) > |R(y)|. \tag{1}$$

Hence, there is no need to seek support for a in $D(y)$. The condition in Equation (1) is (a special version of) what is called a *Support Condition* (SC) in [5]. SCs help avoiding many (but not all) sequences of support checks eventually leading to a support.

For a given arc, (x, y), the *support count* of x with respect to y, denoted $sc(x, y)$, is defined by $sc(x, y) = \min(\{sc(x, y, a) : a \in D(x)\})$. If

$$sc(\, x, y\,) > |\, R(y)\,| \tag{2}$$

then *all* values in $D(x)$ are supported by y. The condition in Equation (2) is (a special version of) what is called a *Revision Condition* (RC) in [5]. RCs avoid many (but not all) unnecessary revisions and much queue maintenance overhead.

3 Probabilistic Approach

Even if the SC and RC are used they do not always make MAC solve more quickly than FC. We propose a probabilistic approach to achieve this. We generalise the notions of a support condition and a revision condition to the notions of a *probabilistic support condition* (PSC) and a *probabilistic revision condition* (PSC) respectively. The idea is to assume that a support exists if the probability of having some support is relatively high and to avoid the process of seeking a support. Similarly, if the probability of having some support is relatively high for *each* value then we avoid the corresponding revision.

3.1 Probabilistic Support Condition

Let $Ps_{(x,y,a)}$ be the probability that (x, a) has some support in $D(y)$. Then

$$Ps_{(x,y,a)} = 1 - \binom{|\, R(y)\,|}{sc(x, y, a)} / \binom{|\, D_o(y)\,|}{sc(x, y, a)}. \tag{3}$$

The justification for this equation is that its right hand side is equal to the probability that none of the $\binom{|\, R(y)\,|}{sc(x,y,a)}$ subsets of size $sc(\, x, y, a\,)$ of $R(y)$ contains all supports of (x, a). Note that if SC is satisfied, i.e. $sc(x, y, a) > |\, R(y)\,|$, then Equation (3) reduces to $Ps_{(x,y,a)} = 1$. Indeed, if fewer values have been removed than there were supports in the original domain then the probability that a support exists is equal to 1.

We now introduce a probabilistic version of the support condition. Let T be some desired threshold. If $Ps_{(x,y,a)} \geq T$ then (x, a) will have support with y with a probability of T or more. We call this condition a *Probabilistic Support Condition*. If it holds then we avoid seeking support for a.

3.2 Probabilistic Revision Condition

Remember that $sc(x, y)$ is the least support count of the values of $D(x)$ with respect to y. Similar to the definition of a probabilistic support condition, we now define a probabilistic revision condition. Let $Ps_{(x,y)}$ be the least probability that some value in $D(x)$ is supported by y. Note that for any value $a \in D(x)$, we immediately have

$$Ps_{(x,y)} = 1 - \binom{|\, R(y)\,|}{sc(x, y)} / \binom{|\, D_o(y)\,|}{sc(x, y)} \leq Ps_{(x,y,a)}. \tag{4}$$

```
Function PAC-3: Boolean;
begin
   Q := G
   Set threshold T such that 1 − 1/d_max < T ≤ 1.
   while Q not empty do begin
      select any v from { x : (x, y) ∈ Q }
      effective_revisions := 0
      for each y such that (x, y) ∈ Q do
         remove (x, y) from Q
         revise_p(x, y, change_x)
         if D(x) = ∅ then
            return False
         else if change_x then
            effective_revisions := effective_revisions + 1
            y'' := y;
      if effective_revisions = 1 then
         Q := Q ∪ { (y', x) ∈ G : y' ≠ y'', Ps_(y',x) < T }
      else if effective_revisions > 1 then
         Q := Q ∪ { (y', x) ∈ G : Ps_(y',x) < T }
   return True;
end;
```

Fig. 1. PAC-3

```
Function revise_p(x, y, var change_x)
begin
   change_x := False
   for each a ∈ D(x) do
      if Ps_(x,y,a) ≥ T then
         continue
      if ∄b ∈ D(y) such that b supports a then
         D(x) := D(x) \ { a }
         change_x := True
end;
```

Fig. 2. Algorithm $revise_p$

Let T be some threshold. If $Ps_{(x,y)} \geq T$ then, each value in $D(x)$ is supported by y with a probability of T or more. We call the condition $Ps_{(x,y)} \geq T$ a *Probabilistic Revision Condition* (PRC). If it holds then we skip the revision of $D(x)$ against $D(y)$. Note that when $sc(x,y) > |\, R(y)\,|$, then $\binom{|\, R(y)\,|}{sc(x,y)} = 0$ and, with a probability of 1, each values in $D(x)$ is supported by y.

4 Description of the new algorithm

Figure 1 depicts the result of incorporating PSC and PRC into AC-3 equipped with reverse variable based revision ordering heuristics [5]. We call this algorithm PAC-3 (Probabilistic AC-3). In Figure 1, arcs are only added if the PRC does not hold. The new revise function, which we call $revise_p$, uses PSC as shown in Figure 2. It seeks support for an arc-value pair *only* when $Ps_{(x,y,a)}$ falls below the threshold. In order to use PSC and PRC, the support count for each arc-value pair must be computed prior to search. The algorithm used to do so is mentioned in [5, Figure 2]. In order to ensure that PAC-3 does at least the amount of constraint propagation which is carried out by FC, $T > 1 - 1/d_{max}$, where d_{max} is the maximum domain size of the variables.

Since PAC-3 needs to be invoked at each node of the search tree, we call the new backtrack algorithm that maintains PAC-3 during search MPAC-3. The space complexity of using a support count for each arc-value pair is $\mathcal{O}(e\,d)$. The space complexity of storing the support count for each arc is $\mathcal{O}(e)$ but it may increase to $\mathcal{O}(e\,n)$ during search. The reason for this is that the support count of an arc may change as values are pruned during search and when backtrack occurs this count has to be restored. Therefore, the overall space complexity of MPAC-3 is $\mathcal{O}(e\,d + e\,n) = \mathcal{O}(e\,max(d,n))$.

5 Experimental Results

In this section, we shall present some results demonstrating the practical efficiency of MPAC-3 when compared to MAC-3 and FC. All algorithms were equipped with a

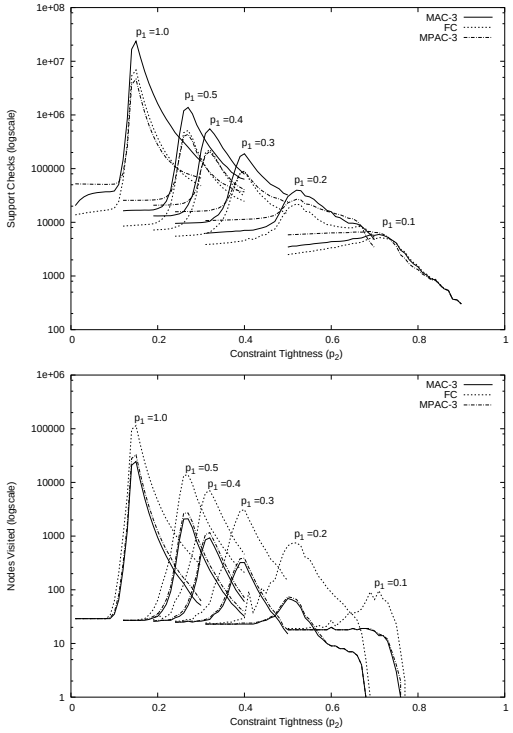

Fig. 3. Mean performance in terms of checks and nodes visited for $\langle\, 30, 10, p_1, p_2 \,\rangle$

dom/deg variable ordering with a lexicographical tie breaker. The experiments were carried out on a PC Pentium III having 256 MB of RAM running at 2.266 GHz processor with linux. All algorithms were written in C. We experimented with random problems which were generated by Frost *et al.*'s model B generator In this model a random CSP instance is typically represented as $\langle\, n, d, p_1, p_2 \,\rangle$ where n is the number of variables, d is the uniform domain size, p_1 is the average density, and p_2 is the uniform tightness. For each combination of $\langle n, d, p_1, p_2 \rangle$, 100 random problems were generated and their mean performance is reported.

Our main aim was to investigate how MPAC-3 behaves with sparse and dense problems. For the first set of experiments, n was kept at 30. The domain size d was kept at 10 which defines the range $(\,0.9, 1\,]$ for T, required by MPAC-3. The value of T was set to 0.95. We varied density p_1 in steps of 0.1 in the range $[\,0.1 \ldots 1\,]$ and p_2 in steps of 0.01 in the range as shown in Figure 3.

Figure 3 shows the mean performance of MPAC-3, MAC-3, and FC in terms of checks and the nodes visited. As expected MAC-3 always outperforms FC in terms of the visited nodes while FC outperforms MAC-3 in terms of the number of checks. Note that for dense problems when a peak occurs in the phase transition, MPAC-3 is more efficient compared to both MAC-3 and FC, when the effort is measured in terms of checks. The number of nodes visited by MPAC-3 is closer to MAC-3 than FC. The gap between

Table 1. Comparison between FC, MAC-3, and MPAC-3 on sparse problems.

$\langle n, d, p_1, p_2 \rangle$	Algorithm	Checks	Time (seconds)	Revisions	Visited nodes
$\langle 65, 20, 0.08, 0.65 \rangle$	FC	24,088,980	5.04	2,531,453	739,054
	MAC-3	15,138,669	1.44	974,903	10,545
	MPAC-3	8,271,149	1.55	948,283	12,918
$\langle 90, 20, 0.07, 0.59 \rangle$	FC	1,375,383,859	360.81	187,037,415	47,250,225
	MAC-3	867,722,685	105.86	67,002,984	617,760
	MPAC-3	507,456,096	113.43	72,798,693	892,139

MAC-3 and FC in terms of the visited nodes increases as the problems become sparse but it decreases between MAC-3 and MPAC-3. As the problems become sparse MAC-3 starts to perform better than FC. Although this may not be clear from Figure 3 but results shown in Table 1 confirm this. This time the value of threshold was set to 0.951, since T should be greater than $1 - 1/20$. Overall, unlike MAC-3 and FC, MPAC-3 performs well on average, both in terms of checks and the nodes visited.

6 Conclusions and Future Work

This paper presents a new search algorithm called MPAC-3 which maintains probabilistic arc consistency during search using *probabilistic support condition* and *probabilistic revision condition*. More specifically, it seeks for a support only when the probability of having some support falls below a stipulated bound. Unlike MAC and FC where the strength of constraint propagation is static and the behaviour is different on dense and sparse problems, maintaining probabilistic arc consistency allows to adjust the strength of constraint propagation dynamically during search and performs well on both dense and sparse problems. In future, we would like to investigate the use of probabilistic approach in solving real-world problems and academic problems. It seems relatively straightforward to generalise the notions of PSC and PRC to achieve probabilistic singleton consistencies and probabilistic hyper-arc consistency for non-binary CSPs.

References

1. F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Support inference for generic filtering. In *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming*, 2004.
2. A. Chmesis and L. Sais. Constraint satisfaction problems:backtrack search revisited. In *Proceedings of the Sixteenth IEEE International Conference on Tools with Artificial Intelligence*, pages 252–257, Boca Raton, FL, USA, 2004. IEEE Computer Society.
3. S.A. Grant and B.M. Smith. The phase transition behaviour of maintaining arc consistency. In W. Wahlster, editor, *Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI'96)*, pages 175–179, 1996.
4. R.M. Haralick and G.L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313, 1980.
5. D. Mehta and M.R.C. van Dongen. Reducing checks and revisions in coarse-grained MAC algorithms. Accepted for publication in the proceedings of *IJCAI'05*.
6. D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In A.G. Cohn, editor, *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI'94)*, pages 125–129. John Wiley and Sons, 1994.