AC-3_d an Efficient Arc-Consistency Algorithm with a Low Space-Complexity

M.R.C. van Dongen

CS Department UCC/Cork Constraint Computation Centre, Western Road, Cork, Ireland,
dongen@cs.ucc.ie

Abstract. Arc-consistency algorithms prune the search-space of Constraint Satisfaction Problems (CSPs). They use *support-checks* to find out about the properties of CSPs. Their *arc-heuristics* select the constraint and their *domain-heuristics* select the values for the next support-check. We shall combine AC-3 and DEE and equip the resulting hybrid with a *double-support* domain-heuristic. The resulting hybrid AC-3_d is easy to implement and requires the same data structures as AC-3 thereby improving on AC-7's space-complexity. We shall present experimental results which indicate that AC-3_d can compete with AC-7.

1 Introduction

Arc-consistency algorithms are widely used to prune the search-space of Constraint Satisfaction Problems (CSPs). They use *support-checks* to find out about the properties of CSPs. Their *arc-heuristics* select the constraint and their *domain-heuristics* select the values for the next support-check. We shall integrate AC-3 and DEE and equip the resulting hybrid with a *double-support* domain heuristic thereby creating an arc-consistency algorithm called AC-3_d, which can compete with AC-7 in time and which has a space-complexity which improves on that of AC-7.

One reason for the increased performance of AC-3_d is that it uses a double-support heuristic and not the most commonly used *lexicographical* domain-heuristic.

We shall present experimental results which indicate that AC-3_d can compete with AC-7 both in time on the wall and in the number of support-checks.

2 Constraint Satisfaction

A *Constraint Satisfaction Problem* (or CSP) comprises a set of n variables, a function D that maps each of these variables to its domain, and a collection of e constraints.

Let α and β be two variables, let $D(\alpha) = \{1, \ldots, a\} \neq \emptyset$, and let $D(\beta) = \{1, \ldots, b\} \neq \emptyset$. In this paper binary constraints are matrices. The set containing all a by b zero-one matrices is denoted $\mathbb{M}^{ab}$. Let $M \in \mathbb{M}^{ab}$ be a constraint between α and β. A value $i \in D(\alpha)$ is *supported* by $j \in D(\beta)$ if $M_{ij} = 1$. Similarly, $j \in D(\beta)$ is supported by $i \in D(\alpha)$ if $M_{ij} = 1$. Matrices, rows and columns are *non-zero* if they contain more than zero ones, and *zero* otherwise. M is *arc-consistent* if for each $i \in D(\alpha)$ the i-th row of M is non-zero and for each $j \in D(\beta)$ the j-th column of M is non-zero. A CSP is *arc-consistent* if its domains are non-empty and its constraints

are arc-consistent. A variable is a *neighbour* of another variable if there is a binary constraint between them. The *degree* $\deg(\alpha)$ of α is the number of neighbours of α. The *density* of a (connected) CSP is defined as $2e/(n^2-n)$. The *tightness* of $M \in \mathbb{M}^{ab}$ is defined as $1 - \frac{1}{ab}\sum_{i=1}^{a}\sum_{j=1}^{b} M_{ij}$.

The *row-support* (*column-support*) of a matrix is the set containing the indices of its non-zero rows (columns). The *support-check* $M^?_{ij}$ is a test to find the value M_{ij}. We shall write $\mathrm{checks}_{\mathcal{A}}(M)$ for the number of support-checks required by arc-consistency algorithm $\mathcal{A}$ to compute the row-support and the column-support of M.

$M^?_{ij}$ *succeeds* if $M_{ij} = 1$. $M^?_{ij}$ is a *single-support check* if, just before it was carried out, the row-support status of i was known and the column-support status of j was unknown, or vice versa. $M^?_{ij}$ is a *double-support check* if, just before the check was carried out, both the row-support status of i and the column-support status of j were unknown. A domain-heuristic is a *double-support heuristic* if it prefers double-support checks. The potential payoff of a double-support check is twice as large as that of a single-support check. This is an indication that arc-consistency algorithms should prefer double-support checks. Another indication is that to minimise the total number of support-checks one has to maximise the number of successful double-support [6].

3 Related Literature

Mackworth presented the AC-3 arc-consistency algorithm [4]. With Freuder he presented a lower bound of $\mathbf{\Omega}\left(ed^2\right)$ and an upper bound of $\mathbf{O}\left(ed^3\right)$ for its worst-case time-complexity [5]. As usual, d is the maximum domain size. AC-3 has a $\mathbf{O}\left(e+nd\right)$ space-complexity. Experimental results indicate that arc-heuristics influence the average performance of AC-3 [8].

Bessière, Freuder and Régin present an arc-consistency algorithm called AC-7 [1, 2]. AC-7 has an optimal upper bound of $\mathbf{O}\left(ed^2\right)$ for its worst-case time-complexity and has been reported to behave well on average. AC-7's space-complexity is $\mathbf{O}\left(ed\right)$.

Results from an experimental comparison between the support-checks required by AC-7 and AC-3_b are presented in [6]. AC-3_d is a cross-breed between AC-3 and DEE [3, 4, 6]. Both AC-7 and AC-3_b were equipped with a lexicographical arc-heuristic. AC-3_b used a double-support and AC-7 a lexicographical domain-heuristic. AC-3_b was more efficient than AC-7 for the majority of the $30,420$ random problems. Also AC-3_b was more efficient on average. These are surprising results because AC-3_b, unlike AC-7, repeats support-checks. The results are also interesting because AC-3_b has a space-complexity of $\mathbf{O}\left(e+nd\right)$ which is better than that of AC-7. These results were the first indication that domain-heuristics can improve arc-consistency algorithms.

4 The AC-3_d Algorithm

In this section we shall study AC-3_d and its domain-heuristic $\mathcal{D}$. Space constraints led to a minimal presentation. The reader is referred to [6, 7] for proof and further details.

AC-3_d is inspired by AC-3 and DEE [3, 4]. AC-3_d uses a queue of arcs just like AC-3. If AC-3_d's arc-heuristics select the arc (α, β) from the queue and if (β, α) is

not in the queue then AC-3_d proceeds like AC-3 by *revising* $D(\alpha)$ using the constraint M between α and β. Here, to revise a domain using constraint M, means to remove its unsupported values using the constraint M. AC-3_d uses Mackworth's revise to revise $D(\alpha)$ with M [7]. If $D(\alpha)$ was changed due to the revision then for each neighbour $\gamma \neq \beta$ of α the arc (γ, α) is added to the queue if it was not in the queue. The difference between AC-3 and AC-3_d becomes apparent if (β, α) is also in the queue. If this is this case then AC-3_d also removes (β, α) from the queue and uses $\mathcal{D}$ to *simultaneously* revise $D(\alpha)$ and $D(\beta)$. Arcs are added to the queue in a similar way as described before. AC-3_d inherits its space-complexity and worst-case time-complexity from AC-3.

$\mathcal{D}$ does not repeat support-checks. It will first find its row-support in the lexicographical order on its rows. When it tries to find support for row r it will first use double-support checks and then single-support checks until the support-status of r is known. Finally, $\mathcal{D}$ will use single-support checks for the unsupported columns.

For sufficiently large domain sizes a and b the average *time-complexity* of $\mathcal{D}$ is less than $2\max(a, b) + 2$. This is almost optimal and it is about twice as efficient as the average time-complexity of a lexicographical heuristic [6].

5 Experimental Results

We have taken results from Bessière, Freuder and Régin as published in [2] and compared them against our own results. We divided their times by 5 because their algorithms were run on a machine which was 5 times slower [2, 7].

		$\langle 150, 50, 0.045, 0.500 \rangle$ underconstrained		$\langle 150, 50, 0.045, 0.940 \rangle$ overconstrained	
		checks	time	checks	time
AC-3 BFR		100,010	0.016	514,973	0.074
AC-7 BFR		94,030	0.038	205,070	0.058
AC-3		99,959	0.022	135,966	0.013
AC-3_d		50,862	0.019	69,742	0.007
		$\langle 150, 50, 0.045, 0.918 \rangle$ phase-transition/sparse		$\langle 50, 50, 1.000, 0.875 \rangle$ phase-transition/dense	
		checks	time	checks	time
AC-3 BFR	AC	2,353,669	0.338	2,932,326	0.382
	IC	4,865,777	0.734	8,574,903	1.092
AC-7 BFR	AC	481,878	0.154	820,814	0.247
	IC	535,095	0.184	912,795	0.320
AC-3	AC	2,254,058	0.162	4,025,746	0.302
	IC	2,602,318	0.196	6,407,079	0.491
AC-3_d	AC	1,734,362	0.140	2,592,579	0.245
	IC	2,010,055	0.171	4,287,835	0.394

Table 1. Average Results for Random Problems

The problem set consists of Radio Link Frequency Assignment Problems (RLFAPs) and random problems. The RLFAP problems were obtained from `ftp://ftp.cs.unh.edu/pub/csp/archive/code/benchmarks` The objective for each CSP is that it be made arc-consistent or to decide that this is not possible. To generate the random problems, we used Frost, Dechter, Bessière and Régin's random constraint generator, which is available from `http://www.lirmm.fr/~bessiere/generator.html`. The generator was run with seed 0.

The random CSPs consist of four groups. Each group contains 50 random CSPs and is uniquely determined by a tuple $\langle\, n, d, p_1, p_2 \,\rangle$. Here, n is the number of variables, d is the (uniform) size of the domains, p_1 is the density of the constraint-graph, and p_2 is the (uniform) tightness of the constraints. The groups are $\langle\, 150, 50, 0.045, 0.500 \,\rangle$ under-constrained CSPs (easy), $\langle\, 150, 50, 0.045, 0.940 \,\rangle$ over-constrained CSP (easy), $\langle\, 150, 50, 0.045, 0.918 \,\rangle$ low density CSPs at the phase-transition (difficult), and finally $\langle\, 50, 50, 1.000, 0.875 \,\rangle$ high density CSPs at the phase-transition (very difficult).

The algorithms that were compared are AC-7 (called AC-7 BFR from here on) as presented in [2], AC-3 (called AC-3 BFR from here on) as presented in [2], our implementation of AC-3, and our implementation of AC-3_d. AC-3 was equipped with a lexicographical domain-heuristic. The arc-heuristic that was used for AC-3 and AC-3_d prefers arc $(\,\alpha, \beta\,)$ to $(\,\alpha', \beta'\,)$ if $s_\alpha < s_{\alpha'}$, if $s_\alpha = s_{\alpha'} \wedge d_\alpha < d_{\alpha'}$, if $s_\alpha = s_{\alpha'} \wedge d_\alpha = d_{\alpha'} \wedge s_\beta < s_{\beta'}$, or if $s_\alpha = s_{\alpha'} \wedge d_\alpha = d_{\alpha'} \wedge s_\beta = s_{\beta'} \wedge d_\beta \leq d_{\beta'}$, where $S_x = |D(x)|$ and $d_x = \deg(x)$. This very expensive heuristic is better for AC-3_d than a lexicographical heuristic with which it almost "degenerates" to AC-3.

The results for the random problems are listed in Table 1. The columns "checks" and "time" list the average number of support-checks and the average time. For the phase-transition we separated results for problems that could be made arc-consistent (marked by "AC") and problems that could not (marked by "IC").

It is difficult to explain the differences between AC-3 BFR and AC-3. Sometimes AC-3 BFR is better and sometimes AC-3. We don't know anything about AC-3 BFR's implementation, but we believe that the differences are solely caused by arc-heuristics.

AC-3_d is better than both AC-3 BFR and AC-3. Only for underconstrained problems does it require slightly more time than AC-3 BFR. This is consistent with the literature [7]. It is interesting to notice that AC-3_d is a lot better than AC-3 BFR and AC-3 for the overconstrained problems.

Outside the phase-transition region AC-3_d outperforms AC-7 BFR in time and checks. AC-3_d is much better than AC-7 BFR for the overconstrained problems. In the phase-transition region AC-3_d requires more checks than AC-7 BFR. For the sparse problems in the phase-transition region AC-3_d saves time. AC-7 should be preferred for dense problems in the phase-transition region.

The results for the RLFAP Problems are presented in Table 2. AC-3_d does better in checks than AC-3 BFR and AC-3. AC-3 BFR performs better in time than AC-3_d for Problems 3, 5, and 11. This is consistent with our findings for the random problems because these problems are relatively easy [7]. AC-3_d does significantly better than AC-3 BFR for RLFAP#8 both in time and checks. This is also consistent with our findings for the overconstrained problems because RLFAP#8 cannot be made arc-consistent and is relatively easy.

	AC-3 BFR		AC-7 BFR		AC-3		AC-3_d	
	checks	time	checks	time	checks	time	checks	time
RLFAP#3	615,371	0.050	412,594	0.138	615,371	0.124	267,532	0.092
RLFAP#5	1,735,239	0.126	848,438	0.232	833,282	0.252	250,797	0.136
RLFAP#8	2,473,269	0.168	654,086	0.168	1,170,748	0.420	25,930	0.040
RLFAP#11	971,893	0.072	638,932	0.212	971,893	0.268	406,247	0.186

Table 2. Average Results for RLFAP Problems

AC-3_d performs better in time and checks than AC-7 BFR for all problems. Again, the results for RLFAP#8 are consistent with our findings for the overconstrained problems. The results for the other problems are also consistent with the other results because the RLFAP Problems are not in the phase-transition region and are relatively easy.

6 Conclusions and Recommendations

In this paper we have presented a general purpose arc-consistency algorithm called AC-3_d which can compete with AC-7 in time and whose $\mathbf{O}(e + nd)$ space-complexity improves on AC-7's $\mathbf{O}(ed)$ space-complexity. We have presented experimental results of a comparison between AC-7 and AC-3_d. For the problems under consideration AC-3_d performs better in time on the wall and in the number of support-checks outside the phase-transition region. In the phase-transition region AC-7 always requires fewer checks. Only for dense problems in the phase-transition region does it require less time.

One reason for the performance of AC-3_d is its double-support heuristic. We should like to extend our comparison with AC-3_d to include other arc-consistency algorithms.

References

1. C. Bessière, E.C. Freuder, and J.-C. Régin. Using inference to reduce arc consistency computation. In C.S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, volume 1, pages 592–598, Montréal, Québec, Canada, 1995. Morgan Kaufmann Publishers, Inc., San Mateo, California, USA.
2. C. Bessière, E.G. Freuder, and J.-C. Régin. Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, 107(1):125–148, 1999.
3. J. Gaschnig. Experimental case studies of backtrack vs. Waltz-type vs. new algorithms for satisficing assignment problems. In *Proceeding of the Second Biennial Conference, Canadian Society for the Computational Studies of Intelligence*, pages 268–277, 1978.
4. A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.
5. A.K. Mackworth and E.C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25(1):65–73, 1985.
6. M.R.C. van Dongen. *Constraints, Varieties, and Algorithms*. PhD thesis, Department of Computer Science, University College, Cork, Ireland, 2002.
7. M.R.C. van Dongen. AC-3_d an efficient arc-consistency algorithm with a low space-complexity. Technical Report TR-01-2002, Cork Constraint Computation Centre, 2002.
8. R.J. Wallace and E.C. Freuder. Ordering heuristics for arc consistency algorithms. In *AI/GI/VI '92*, pages 163–169, Vancouver, British Columbia, Canada, 1992.