Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Cycle-Based Singleton Local Consistencies

Robert J. Woodward,[1,2] **Berthe Y. Choueiry**[1]
[1]Constraint Systems Laboratory*
University of Nebraska-Lincoln, USA
{rwoodwar|choueiry}@cse.unl.edu

Christian Bessiere[2]
[2]LIRMM-CNRS
University of Montpellier, France
bessiere@lirmm.fr

Abstract

We propose to exploit cycles in the constraint network of a Constraint Satisfaction Problem (CSP) to vehicle constraint propagation and improve the effectiveness of local consistency algorithms. We focus our attention on the consistency property Partition-One Arc-Consistency (POAC), which is a stronger variant of Singleton Arc-Consistency (SAC). We modify the algorithm for enforcing POAC to operate on a minimum cycle basis (MCB) of the incidence graph of the CSP. We empirically show that our approach improves the performance of problem solving and constitutes a novel and effective localization of consistency algorithms. Although this paper focuses on POAC, we believe that exploiting cycles, such as MCBs, is applicable to other consistency algorithms and that our study opens a new direction in the design of consistency algorithms. This research is documented in a technical report (Woodward, Choueiry, and Bessiere 2016).[1]

Introduction

Local consistency properties are enforced on a CSP to filter values from variables' domains, reducing the size of the search space. These properties can be enforced either before (i.e., pre-processing) or during (i.e., lookahead) search. One such consistency property is Singleton Arc-Consistency (SAC) (Debruyne and Bessière 1997). SAC performs a series of *singleton tests* by assigning a value to a variable and enforcing arc consistency (AC) on the entire CSP. While SAC is a strong form of consistency, enforcing it on a CSP during search is prohibitively costly in practice. To reduce the cost of SAC, Wallace (2015) proposed NSAC, restricting AC to the neighborhood of the variable. Bennaceur and Affane (2001) proposed Partition-One Arc-Consistency (POAC), a strictly stronger property than SAC. The algorithm for POAC operates similarly to SAC and can *both* reduce the cost of SAC and increase its filtering. Balafrej et al. (2014) further reduced the cost of POAC by interrupting the POAC algorithm as soon as filtering has subsided and before a fixpoint is reached. Their approach results in an adaptive version of POAC, APOAC.

*Supported by NSF Grants No. RI-111795 and RI-1619344. Experiments conducted at the Holland Computing Center of UNL.

[1]http://consystlab.unl.edu/our_work/StudentReports/TR-UNL-CSE-2016-0004.pdf

We explore further reducing the cost of POAC by localizing the variables on which the singleton test is propagated. As a first step, we mimic the approach of Wallace (2015) for SAC, restricting POAC to the direct neighborhood of each variable. Then, we explore a completely new direction: we restrict POAC to cycles in which a variable participates, the rationale being that *cycles channel propagation and are thus likely to expose inconsistencies*. We propose to use the cycles in a *Minimum Cycle Basis* (MCB) of a graph, which is a graph-theoretic concept and can be efficiently computed (Horton 1987). We discuss the resulting consistency properties and algorithms and empirically validate our approach.

Background

A Constraint Satisfaction Problem (CSP) is defined by $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$. $\mathcal{X}$ is a set of variables, and each variable $x_i \in \mathcal{X}$ has a finite domain $dom(x_i) \in \mathcal{D}$, where $(x_i, v_i) \in \mathcal{P}$ if $v_i \in dom(x_i)$. $\mathcal{C}$ is a set of constraints defined as relations over the domains of the variables. The question is to determine whether or not there is an assignment to the variables that satisfies all the constraints. The *incidence graph* of a CSP is a bipartite graph where one set of vertices contains the variables of the CSP and the other set the constraints. An edge connects a variable and a constraint if and only if the variable appears in the scope of the constraint. The incidence graph is the same graph used in the hidden-variable encoding (Rossi, Petrie, and Dhar 1990). We say that $(x_i, v_i) \in \mathcal{P}$ is SAC iff AC($\mathcal{P} \cup \{x_i \leftarrow v_j\}$) is consistent (the singleton test), where $AC(\mathcal{P} \cup \{x_i \leftarrow v_i\})$ is the CSP after assigning $x_i \leftarrow v_i$ and enforcing Arc Consistency (AC).

Definition 1 *(Bennaceur and Affane 2001) A constraint network $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is* Partition-One Arc-Consistent (POAC) *iff $\mathcal{P}$ is SAC and for all $x_i \in \mathcal{X}$, for all $v_i \in dom(x_i)$, for all $x_j \in X$, there exists $v_j \in dom(x_j)$ such that $(x_i, v_i) \in AC(\mathcal{P} \cup \{x_j \leftarrow v_j\})$.*

A *cycle basis* of a graph is a maximal set of linearly independent cycles: a cycle in the basis, represented as a set of edges, cannot be obtained by taking the symmetric difference (exclusive-or) of the edges of other cycles in the basis (Horton 1987). In unweighted graphs, a *minimum cycle basis* is one with the minimum total length of the cycles. An MCB is efficiently computed but is not unique. We propose to generate MCBs on the incidence graph of the CSP.

Localizing POAC

We introduce two localizations of POAC: Neighborhood POAC (NPOAC) and Union-Cycle POAC ($\cup_{cyc}$POAC). The definitions of these properties restrict the singleton test in Definition 1 to the neighborhood of a variable and to the union of the MCB cycles in which the variable appears.

We derive our algorithms from POAC-1, which enforces POAC (Balafrej et al. 2014). The algorithm POAC-1 runs a singleton test on *each* variable-value pair of the CSP. In each test, it enforces arc consistency on the *entire* CSP. Whenever the domain of *any* variable is updated, the entire process is repeated: POAC-1 re-runs the singleton test on *all* the variables. Our algorithm for enforcing NPOAC (respectively, $\cup_{cyc}$POAC) modifies POAC-1 in two ways. First, we restrict the singleton test to the neighborhood of the variable (respectively, to the union of the MCB cycles in which it appears). Second, we use a dynamic queue for propagation. That is, when a variable's domain is updated, we add only the variables in its neighborhood (respectively, only those that appear in its MCB cycles) to the queue. The resulting algorithms are NPOACQ and $\cup_{cyc}$POACQ.

Experimental Evaluation

We evaluate our approach in the context of finding the first solution of a CSP using backtrack search, real-full lookahead (RFL) (Haralick and Elliott 1980), and dom/wdeg dynamic variable-ordering. Comparing NPOACQ and $\cup_{cyc}$POACQ showed that the latter largely outperforms the former. Thus, we discuss only $\cup_{cyc}$POACQ below.

We compare the performance of the following consistency algorithms used for RFL: **GAC** (GAC2001 (Bessière et al. 2005)), **POAC** (POAC-1), **APOAC**,[2] **$\cup_{cyc}$POACQ**, and **A$\cup_{cyc}$POACQ** (an adaptive version of our new algorithm). We conduct our experiments on the set of benchmarks used by Balafrej et al. (2014). We set a time limit of four hours per instance with 8GB of RAM. The time includes setting up the instance and generating an MCB, which for most problems, is not costly. Table 1 reports the total number of instances for each benchmark (# inst). For each algorithm, we first report the number of instances solved and then the sum of the CPU time in seconds computed over the instances where at least one algorithm terminated. When an algorithm does not terminate within four hours, we add 14,400 seconds to the CPU time and indicate with a $>$ sign that the time reported is a lower bound. The best value in a row of the table is bold faced. The table separates the benchmarks on which adaptive versions of POAC perform best (top), the non-adaptive versions of POAC perform best (middle), and GAC performs best (bottom). Indeed, as mentioned by Balafrej et al. (2014), GAC is sometimes sufficient.

The top two categories show that whenever POAC outperforms GAC, cycle-based methods are the best (right versus left column), except for k-insertions (graph coloring). The bottom category shows that using cycles improves POAC-based algorithms and cut the distance to GAC.

[2]Adaptive POAC, using last drop with $\beta = 5\%$ and the initial $maxK = n$ (Balafrej et al. 2014).

Table 1: Lookahead with POAC-based algorithms

Benchmark	GAC	POAC	$\cup_{cyc}$POACQ	APOAC	A$\cup_{cyc}$POACQ
Adaptive POAC the best					
TSP-25	**15**	14	**15**	**15**	**15**
(# inst 15)	4,303.12	>41,382.27	32,654.67	6,152.91	**2,418.41**
cril	6	7	7	**8**	**8**
(# inst 8)	>30,458.10	>16,282.45	>16,651.04	2,321.96	**1,831.60**
QWH-20	**10**	**10**	**10**	**10**	**10**
(# inst 10)	2,256.61	6,154.43	3,007.98	2,236.32	**2,061.63**
k-ins.	17	17	**18**	**18**	**18**
(# inst 32)	>17,034.30	>21,639.31	11,814.83	**6,129.92**	8,940.59
Non-adaptive POAC the best					
mug	6	6	**8**	6	6
(# inst 8)	>54,724.38	>29,385.02	**13,655.87**	>34,207.98	>41,583.97
GAC the best					
TSP-20	**15**	**15**	**15**	**15**	**15**
(# inst 15)	**302.21**	2,750.90	3,096.07	593.04	384.13
renault	**50**	**50**	**50**	**50**	**50**
(# inst 50)	**55.87**	277.74	176.28	196.04	155.88
myciel	**13**	12	12	**13**	**13**
(# inst 16)	**1,711.93**	>21,564.06	>26,196.15	3,118.86	2,555.54

Future Work & Conclusions

In this paper, we advocate the use of cycles to improve the performance of POAC algorithms and provide empirical evidence of the benefit of our approach. Future work should extend our approach to other consistency algorithms.

References

Balafrej, A.; Bessiere, C.; Bouyakhf, E.; and Trombettoni, G. 2014. Adaptive Singleton-Based Consistencies. In *AAAI 14*, 2601–2607.

Bennaceur, H., and Affane, M.-S. 2001. Partition-k-AC: An Efficient Filtering Technique Combining Domain Partition and Arc Consistency. In *CP 01*, 560–564.

Bessière, C.; Régin, J.-C.; Yap, R. H.; and Zhang, Y. 2005. An Optimal Coarse-Grained Arc Consistency Algorithm. *Artificial Intelligence* 165(2):165–185.

Debruyne, R., and Bessière, C. 1997. Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. In *IJCAI 97*, 412–417.

Haralick, R. M., and Elliott, G. L. 1980. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence* 14:263–313.

Horton, J. D. 1987. A Polynomial-Time Algorithm to Find the Shortest Cycle Basis of a Graph. *SIAM J. on Computing* 16(2):358–366.

Rossi, F.; Petrie, C.; and Dhar, V. 1990. On the Equivalence of Constraint Satisfaction Problems. In *ECAI 90*, 550–556.

Wallace, R. J. 2015. SAC and Neighbourhood SAC. *AI Communications* 28(2):345–364.

Woodward, R. J.; Choueiry, B. Y.; and Bessiere, C. 2016. Cycle-Based Singleton Local Consistencies. Technical Report TR-UNL-CSE-2016-0004, CSE, University of Nebraska-Lincoln.