Cycle-Based Singleton Local Consistencies
Robert J. Woodward, Berthe Y. Choueiry, Christian Bessière

To cite this version:

HAL Id: lirmm-01711585
https://hal-lirmm.ccsd.cnrs.fr/lirmm-01711585
Submitted on 18 Oct 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Cycle-Based Singleton Local Consistencies

Robert J. Woodward,1,2 Berthe Y. Choueiry1
1 Constraint Systems Laboratory∗
University of Nebraska-Lincoln, USA
{rwoodwar|choueiry}@cse.unl.edu

Christian Bessiere2
2 LIRMM-CNRS
University of Montpellier, France
bessiere@lirmm.fr

Abstract
We propose to explore 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).∗

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. Balafrey 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 a more adaptive version of POAC, APOAC.

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 \( \text{dom}(x_i) \in \mathcal{D} \), where \( (x_i, v_i) \in \mathcal{P} \) if \( v_i \in \text{dom}(x_i) \), \( \mathcal{C} \) 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 \( AC(\mathcal{P} \cup \{x_i \leftarrow v_i\}) \) 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 \( v_i \in \text{dom}(x_i) \), for all \( x_j \in \mathcal{X} \), there exists \( v_j \in \text{dom}(x_j) \) 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.

∗Supported by NSF Grants No. RI-111795 and RI-1619344. Experiments conducted at the Holland Computing Center of UNL. Copyright © 2017, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.
1http://consystlab.unl.edu/our_work/StudentReports/TR-UNL-CSE-2016-0004.pdf
Localizing POAC

We introduce two localizations of POAC: Neighborhood POAC (NPOAC) and Union-Cycle POAC ($\cup_{cyc} \text{POAC}$). 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, $\cup_{cyc} \text{POAC}$) 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 NPOAC and $\cup_{cyc} \text{POAC}$. 

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 NPOAC and $\cup_{cyc} \text{POAC}$ showed that the latter largely outperforms the former. Thus, we discuss only $\cup_{cyc} \text{POAC}$ below.

We compare the performance of the following consistency algorithms used for RFL: GAC (GAC2001 (Bessi`ere et al. 2005)), POAC (POAC-1), APOAC, $\cup_{cyc} \text{POAC}$, and $\cup_{\text{cyc}} \text{POAC}Q$ (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.

Table 1: Lookahead with POAC-based algorithms

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>GAC</th>
<th>POAC</th>
<th>$\cup_{cyc} \text{POAC}$</th>
<th>APOAC</th>
<th>$\cup_{\text{cyc}} \text{POAC}Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSP-25</td>
<td>15</td>
<td>14</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>(# inst 15)</td>
<td>4,303.12</td>
<td>&gt;41,382.27</td>
<td>32,654.67</td>
<td>6,152.91</td>
<td>2,418.41</td>
</tr>
<tr>
<td>ciril</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>(# inst 8)</td>
<td>&gt;30,458.10</td>
<td>&gt;16,282.45</td>
<td>&gt;16,651.04</td>
<td>2,312.96</td>
<td>1,831.60</td>
</tr>
<tr>
<td>QWH-20</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>(# inst 10)</td>
<td>2,256.61</td>
<td>6,154.43</td>
<td>3,007.98</td>
<td>2,236.32</td>
<td>2,061.63</td>
</tr>
<tr>
<td>k-sm</td>
<td>17</td>
<td>17</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>(# inst 32)</td>
<td>&gt;17,034.30</td>
<td>&gt;21,639.31</td>
<td>11,814.83</td>
<td>6,129.92</td>
<td>8,940.59</td>
</tr>
<tr>
<td>non-Adaptive POAC the best</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TSP-20</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>(# inst 15)</td>
<td>302.21</td>
<td>2,750.90</td>
<td>3,096.07</td>
<td>593.04</td>
<td>384.13</td>
</tr>
<tr>
<td>renault</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>(# inst 50)</td>
<td>55.87</td>
<td>277.74</td>
<td>176.28</td>
<td>196.04</td>
<td>155.88</td>
</tr>
<tr>
<td>mycie</td>
<td>13</td>
<td>12</td>
<td>12</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>(# inst 16)</td>
<td>1,711.93</td>
<td>&gt;21,564.06</td>
<td>&gt;26,196.15</td>
<td>3,118.86</td>
<td>2,555.54</td>
</tr>
<tr>
<td>GAC the best</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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


