Breaking Symmetries in Distributed Constraint Programming Problems

Xavier Olive and Hiroshi Nakashima
Graduate School of Informatics, Kyoto University

Abstract. Though various preprocessing techniques have been studied for improving the performance of distributed constraint satisfaction problems, no approach for detecting and breaking symmetries has been studied in depth. In this paper, we describe a method for detecting some symmetries of a given distributed problem and for exploiting them. Then, we validate it as a preprocessing method for ADOPT and DPOP algorithms for some instances of the SensorDCSP problem, to find our symmetry breaker improve their performance up to 1.7 and 1.8 times respectively.

Keywords. distributed constraint optimisation, symmetry breaking

1 Introduction

Distributed constraint optimisation was first formalised in 1998 by Yokoo [1]. In many real-life multi-agent situations, individual agents must share their knowledge and attempt to solve some problems involving a set of distributed constraints. This decentralised model is particularly relevant when the limited computational and communication power or privacy concerns make centralised constraint optimisation difficult.

In the last few years, numerous studies focused on developing efficient algorithms, which mainly fall into two categories: ADOPT [2], which is an improved version of Yokoo’s Asynchronous Backtracking [3], and DPOP [4], which is a decentralised version of a Bucket Elimination Algorithm. In contrast, few studies have been made concerning preprocessing techniques, though some approaches came out for improving ADOPT [5] or for using symmetries with Asynchronous Backtracking [6].

In this paper, we focus on symmetries which are recognised to be fundamentally important in Constraint Satisfaction Problems, and study how we can detect and exploit them in a distributed context, using ADOPT and DPOP. Then, we measure the relevance of the algorithm we propose on SensorDCSP [7] a benchmark of sensors tracking moving mobiles over a map. We implemented the proposed algorithm and attached it to Frodo [8] framework for our simulations.

2 Distributed Constraint Satisfaction Problems

A Distributed Constraint Satisfaction Problem (DCSP) is a Constraint Satisfaction Problem (CSP) where variables are distributed over different agents.
Constraints fall into two categories: local (private) and global (distributed) constraints. Formally speaking, a distributed constraint satisfaction problem is defined as a finite set of agents $a_1 \cdots a_n$, a set of local CSPs for each agent $a_i$, and a global CSP defined among variables that belong to different agents.

We can also define a DCSP as a finite set of variables $x_1 \cdots x_n$, a set of domains $d_1 \cdots d_n$, a list of agents $a_1 \cdots a_n$ (not necessarily all different), and a set of constraints $c_1 \cdots c_t$. Each constraint has a scope of variables, thus a scope of agents. Two agents are neighbours if they share at least one constraint.

A local assignment is a selection of values for each variable owned by one agent. A global assignment is a selection of values for each variable. A solution to a DCSP is a global assignment violating none of the constraints. The resolution process involves communication between agents to check their local assignments against the constraint, since no agent has a global view on the problem. Asynchronous Backtracking [3] is one of the first proposed algorithms, though poorly efficient. Recently, the more popular algorithms are based on DFS (Depth-First Search) structures grouping neighbour variables together. ADOPT [2] algorithms proceed an improved backtracking method, whereas DPOP [4] algorithms are based on bucket elimination. DPOP is more efficient timewise than ADOPT, but its memory requirements are exponential in the induced width of the DFS pseudo-tree used.

SensorDCSP [7] is a benchmark for DCSP algorithms: given multiple sensors $s_1 \cdots s_m$ and multiple mobiles $m_1 \cdots m_n$ to be tracked by the sensors, the goal is to allocate three sensors to track each mobile node. The allocation is subject to visibility and compatibility constraints. The distribution of the sensors makes the problem naturally distributed. For our simulations, each variable is a boolean associated to a sensor/mobile couple, and owned by the corresponding sensor. Each constraint over a mobile is distributed over all the sensors detecting it. The benchmarks have been executed over a fixed detection range and a fixed repartition of sensors, with a variable number of mobiles.

### 3 Detecting Symmetry over a Distributed CSP

Symmetries in CSP are widely acknowledged as an approach to avoid revisiting equivalent states. They can be defined as mappings that permute the variables of the problem by pair, while leaving the constraints unchanged [9, 10]. For example, if $(x, y, z, t)$ are all defined on $\{0, 1\}$ and constrained by $x + 2t = z$ and $y + 2t = z$, swapping $x \leftrightarrow y$ would leave the set of constraints globally unchanged.

Symmetries are widely exploited during search [6, 11], though they can also be used to enhance the representation of CSP, or to derive implied constraints, hence reduce significantly the search effort. As DPOP performance is related to the induced width of the generated DFS tree, the problem reformulation approach is the only acceptable way.

Detecting symmetries locally is a problem which has already been studied in detail with the help of group theory [12]. However, in a distributed context, no agent knows the entire problem: the symmetries the agent can detect locally
are not necessarily symmetries of the problem over all the agents. Likewise, a
symmetry on the global problem does not necessarily leave the constraints owned
by a given agent unchanged. We propose here a method that will detect all the
symmetries leaving invariant each local set of constraints on each agent.

We set a priority order on all the agents (e.g. alphabetic order). Each agent
first executes a symmetry detection on its local problem and keeps in \( p_i \) the
set of correlated variable permutations leaving \( a_i \)'s constraints unchanged, and
involving variables owned only by \( a_i \) itself or agents of lower priority. We name
the set of these lower priority agents \( A_i \). Then, each \( a_i \) sends concurrently \((A_i, p_i)\)
to the first agent in \( A_i \).

When an agent \( a_j \) receives \((A_i, p_i)\), it builds \( p_{i,j} \) by taking out of \( p_i \) the
symmetries which are not leaving the set of local constraints of \( a_j \) invariant.
Then, if \( p_{i,j} \) is not empty, \( a_j \) sends it to the next agent in \( A_i \), otherwise it stops
the process by alerting \( a_i \). If there is no next agent left in \( A_i \), we can conclude
that \( s_i = p_{A_i} \) is a set of symmetries for all the agents: we send \( s_i \) back the way
the message came, so that all the agents involved can record it.

Since all the agents affected by a local symmetry \( p \in p_i \) are in \( A_i \), they all
check whether \( p \) leaves their own constraints unchanged. Therefore, the agree-
ment by all the agents in \( A_i \) ensures that \( p \) is a global symmetry. On the other
hand, if \( p \) is a global symmetry which is also a local symmetry for some agents,
the detection of the symmetry \( p \) by all the involved agents is ensured, and only
the agent of higher priority \( a_i \) will keep that symmetry \( p \) in \( p_i \).

**Fig. 1.** Message flow initiated by two agents in symmetry detection algorithm

Figure 1 shows two instances of the detection process. In this example, some
of the symmetries detected by \( a_1 \) are also symmetries for \( a_3, a_4, \) and \( a_5 \). However
the symmetries detected by \( a_2 \) are not symmetries for \( a_4 \) and \( a_5 \). The reception
of either a no or a symmetry message triggers a message done to the controller.
When the controller gathers done messages from all the agents, the algorithm
is over. Then each agent can reformulate its local definition of the problem as
soon as it gets the signal to process the symmetries.
## 4 Evaluation

Our symmetry detection algorithm requires a preprocessing which initiate a number of messages in $O(n \cdot k)$ for $n$ agents having $k$ neighbours at most. This number is insignificant compared to the number of messages we avoid sending thanks to the symmetry breaking. Thus, the total execution time should be cut down. To confirm these expectations, we evaluated the performance of two instances of SensorDCSP problem with our implementation we attached to Frodo framework on a Core2Duo based Linux PC with 2GB memory.

Table 1 shows the performance evaluation of the proposed detection algorithm on a SensorDCSP problem with 25 sensors and $n$ moving agents. The symmetry detection process lets us reformulate a big problem into an equivalent smaller problem, and consequently enhance the performances of both ADOPT and DPOP. For the biggest problem, DPOP (resp. ADOPT) sees its execution time improved by about 30% (resp. 40%). As for the number of sent messages, it is cut down by some 50%.

With 81 sensors (Table 2), the execution time is also reduced by up to 40% and 30% respectively. Although the difference in the number of sent messages is not as distinct, the method allows a reformulation into a smaller problem, thereby making the local resolution process much faster.

<table>
<thead>
<tr>
<th>$n$</th>
<th>DPOP</th>
<th>Adopt</th>
<th>DPOP</th>
<th>Adopt</th>
<th>Sym</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>13.35</td>
<td>15.18</td>
<td>18.41</td>
<td>23.47</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>45.89</td>
<td>34.79</td>
<td>55.46</td>
<td>49.90</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>58.36</td>
<td>50.66</td>
<td>89.19</td>
<td>61.92</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>70.66</td>
<td>58.78</td>
<td>107.91</td>
<td>87.59</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>88.10</td>
<td>59.51</td>
<td>126.25</td>
<td>94.21</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>103.98</td>
<td>70.28</td>
<td>142.51</td>
<td>85.03</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$n$</th>
<th>DPOP</th>
<th>Adopt</th>
<th>DPOP</th>
<th>Adopt</th>
<th>Sym</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>52.96</td>
<td>40.79</td>
<td>84.10</td>
<td>71.45</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>227.7</td>
<td>82.68</td>
<td>380.0</td>
<td>251.5</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>319.4</td>
<td>236.9</td>
<td>830.2</td>
<td>559.4</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>582.6</td>
<td>313.0</td>
<td>1238</td>
<td>956.4</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>620.9</td>
<td>408.9</td>
<td>1445</td>
<td>1038</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>752.1</td>
<td>426.7</td>
<td>1815</td>
<td>1242</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. SensorDCSP with 25 sensors and $n$ mobiles (average on 100 executions)

<table>
<thead>
<tr>
<th>$n$</th>
<th>DPOP</th>
<th>Adopt</th>
<th>DPOP</th>
<th>Adopt</th>
<th>Sym</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1031</td>
<td>1076</td>
<td>18.82</td>
<td>15.53</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>6204</td>
<td>5150</td>
<td>107.3</td>
<td>87.25</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>12522</td>
<td>12207</td>
<td>257.8</td>
<td>208.4</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>24054</td>
<td>24095</td>
<td>418.0</td>
<td>382.4</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>30162</td>
<td>28281</td>
<td>507.8</td>
<td>405.7</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>37930</td>
<td>33647</td>
<td>632.2</td>
<td>485.3</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. SensorDCSP with 81 sensors and $n$ mobiles (average on 100 executions)
5 Conclusion and Future Work

In this paper, we proposed a method for detecting DCSP global symmetries that are also local symmetries. We validated this method on DPOP and ADOPT algorithms for some instances of the SensorDCSP problem to find their performance is improved by up to 1.8 and 1.7 times respectively.

One of the major drawbacks of symmetry detection is that even if the simplified version of the problem is solved faster, the time spared would not compensate for the effort spent in detecting the symmetries. Especially in distributed CSP where communication cost is expensive, we might waste time trying to detect DCSP global symmetries that would not exist, or would not be detectable.

Our urgent future work is to evaluate our method with other types including larger scale problems. Another issue to be investigated in future work is to estimate the effectiveness of symmetry breaking quickly instead of blindly applying it potentially wasting time for problems having few or no symmetries.

Acknowledgements

The authors would like to thank the EPFL–LIA team for placing at our disposal their Frodo framework, and for their precious advice.

References