Flexible and Dynamic Production Programming Tool Based on MAS Technology..................................01
Soledad Valero, Maria E. Garcia, Estefania Argente, Adriana Giret, Vicente Julian

Modeling and Validation of Mobile Agent Systems: A Case Study..................................................09
Nassima Benaissa, Bachir Djafri, Guillaume Hutzler, Hanna Klaudel

Distributed Prediction of Carbon Dioxide Exchange Using CBR-BDI Agents....................................16
Javier Bajo, Dante I. Tapia, Juan F. de Paz, Juan M. Corchado

Enabling Rapid Prototyping using Decoupling of Code Skeletons and Code Generation Process.......26
Jorge J. Gomez-Sanz, Juan Pavon, Ruben Fuentes

Agent-Based Simulator for Industry....................................................................................................35
Nancy Ruiz, Adriana Giret, Vicente Botti

Distributed Models for Solving CSPs.................................................................................................43
Montserrat Abril, Miguel A. Salido, Federico Barber

Tourism Services Using Agent Technology: A MultiAgent Approach.............................................51
Juan S. Lopez, Fabian A. Bustos, Vicente J. Inglada

A Study on Efficient Scheduling Algorithm for the Batch Wet Station.........................................58
Soong-Ho Ko, Kyung-Woo Kang, Hyun-Soo Park

Hybrid Neural Symbolic Systems for Dealing with Classification Tasks in Industrial Environments....63
Manuel G. Bedia, Juan M. Corchado, Luis F. Castillo
Distributed Models for solving CSPs

MONTSERAT ABRIL
MIGUEL A. SALIDO
FEDERICO BARBER

Universidad Politécnica de Valencia
DSIC - Departamento de Sistemas Informáticos y Computación
46022 - Valencia, Spain
(mabril, msalido, fbarber)@dsic.upv.es

Abstract. Nowadays, many real problems can be modelled as Constraint Satisfaction Problems (CSPs). Solving a general CSP is known to be NP-complete; so that closure and heuristic search are usually used. However, less effort has been focused on distributed CSPs, where the problem complexity can be reduced by dividing the problem into a set of subproblems. In this work, we present two techniques for partition a CSP. The first one is based on graph partitioning. Given a number of partitions n, the technique generates n subgraphs and number of edges connecting different graphs is minimized. The second technique is based on trees. This technique divide the problem into different subproblems in such a way, each subproblem is a tree. In this case, the number of partition is unknown in advance. The evaluation shows the behavior of both proposal. Both distributed techniques had better behavior than centralized techniques.

Keywords: Distributed Constraint Satisfaction Problems.

(Received June 14, 2006 / Accepted July 12, 2006)

1 Introduction

Many real problems in Artificial Intelligence (AI) as well as in other areas of computer science and engineering can be efficiently modelled as Constraint Satisfaction Problems (CSPs) and solved using constraint programming techniques. Some examples of such problems include: spatial and temporal planning, qualitative and symbolic reasoning, diagnosis, decision support, scheduling, hardware design and verification, real-time systems and robot planning.

Most of the work is focused on general methods for solving CSPs. They include backtracking-based search algorithms. While the worst-case complexity of backtrack search is exponential, several heuristics to reduce its average-case complexity have been proposed in the literature [3]. For instance, some algorithms incorporate features such as ordering heuristics, (variable ordering and value ordering) [8], and constraint ordering [12], [9]).

In recent years we have seen an increasing interest in Distributed Constraint Satisfaction Problem formulations to model combinatorial problems arising in distributed, multi-agent environments [1] [14], [15]. There are many real-world distributed applications, such as in the area of networked systems, planning and scheduling, for which the Distributed CSP paradigm is particularly useful. In such distributed applications, they deal with resource restrictions (such as limits on time and communication), privacy requirements, exploiting opportunities for cooperation, and designing conflict resolution strategies.

Furthermore, many researchers are working on graph partitioning [4], [6]. The main objective of graph parti-
tioning is to divide the graph into a set of regions such that each region has roughly the same number of nodes and the sum of all edges connecting different regions is minimized. Graph partitioning can be applied on telephone network design, sparse gaussian elimination, data mining, clustering and physical mapping of DNA. Fortunately, many heuristic may solve this problem efficiently. For instance, graphs with over 14000 nodes and 410000 edges can be partitioned in under 2 seconds [5]. This technique can be applied to divide a CSP into semi-independent sub-CSPs.

Nevertheless, an intuitive way of partition CSPs is by means of trees. A binary CSP that can be represent by a tree, it can be solved without backtracking.

Partition of a CSP can be performed in different ways. In this paper, we present two proposals: the first one is focused on distribute the problem by using graph partitioning techniques. In Figure 1 (2), we can observed that a CSP can be divided into several subCSPs so that constraints among variables of each sub-CSP are minimized. The second proposal is oriented to divide the problem by means of trees (see Figure 1 (3)). This means a easier manner to solve each sub-CSP. We have evaluated the behavior of these distributed proposal and we have compared the results with centralized techniques.

![Centralized Problem vs Distributed Problem](image)

**Figure 1:** Distributed Problem by graph partitioning and by trees.

In the following section, we summarize some definitions about distributed CSP. In section 2, we present our distributed model proposals. An evaluation among different models is carried out in section 3. Finally we summarizes the conclusions in section 4.

2 Distributed CSPs

This section presents numeric CSPs in a slightly non-standard form, which will be convenient for our purposes, and will unify works from constraint satisfaction communities. Furthermore, we present our distributed model.

2.1 Definitions

**Definition 1:** A CSP consists of: a set of variables \( X = \{x_1, x_2, \ldots, x_n\} \); each variable \( x_i \in X \) has a set \( D_i \) of possible values (its domain); a finite collection of constraints \( C = \{c_1, c_2, \ldots, c_p\} \) restricting the values that the variables can simultaneously take.

A solution to a CSP is an assignment of values to all the variables so that all constraints are satisfied; a problem with a solution is termed *satisfiable* or *consistent*.

**State:** one possible assignment of all variables.

**Partition:** A partition of a set \( C \) is a set of disjoint subsets of \( C \) whose union is \( C \). The subsets are called the blocks of the partition.

**Distributed CSP:** A distributed CSP (DCSP) is a CSP in which the variables and constraints are distributed among automated agents [15].

Each agent has some variables and attempts to determine their values. However, there are interagent constraints and the value assignment must satisfy these interagent constraints. In our model, there are \( k \) agents \( 1, 2, \ldots, k \). Each agent knows a set of constraints and the domains of variables involved in these constraints.

**Definition 2:** A block agent \( a_j \) is a virtual entity that essentially has the following properties: autonomy, social ability, reactivity and pro-activity [13].

*Block agents* are autonomous agents. They operate their subproblems without the direct intervention of any other agent or human. *Block agents* interact with each other by sending messages to communicate consistent partial states. They perceive their environment and changes in it, such as new partial consistent states, and react, if possible, with more complete consistent partial states.

**Definition 3:** A multi-agent system is a system that contains the following elements:

1. An environment in which the agents live (variables, domains, constraints and consistent partial states).
2. A set of reactive rules, governing the interaction between the agents and their environment (agent exchange rules, communication rules, etc).
3. A set of agents, \( A = \{a_1, a_2, \ldots, a_k\} \).

3 Distributed Model Proposal

Following, we present two different distributed models for solving CSPs. Depending on the problem topology, we can use the more appropriate model to take advantage of the distribution.
3.1 Distributed Model 1

In the specialized literature, there are many works about distributed CSPs. In [15], Yokoo et al. present a formalization and algorithms for solving distributed CSPs. These algorithms can be classified as either centralized methods, synchronous or asynchronous backtracking [15].

Our first model can be considered as a synchronous model. It is meant to be a framework for interacting agents to achieve a consistent state. The main idea of our multi-agent model is based on [10] but partitioning the problem in \( k \) subproblems as independent as possible, classifying the subproblem in the appropriate order and solving them concurrently.

In all our proposals, the problem is partitioned in \( k \) blocks or clusters in order to be studied by agents called block agents. Furthermore, a partition agent is committed to classify the subproblems in the appropriate order depending on the selected proposal. For instance, if Metis is selected to partition the problem, the partition agent must classify the subproblems such as the most interrelated problem is studied first.

Once the constraints are divided into \( k \) blocks by a preprocessing agent, a group of block agents concurrently manages each block of constraints. Each block agent is in charge of solving its own subproblem by means of a search algorithm. Each block agent is free to select any algorithm to find a consistent partial state. It can select a local search algorithm, a backtracking-based algorithm, or any other, depending on the problem topology. In any case, each block agent is committed to finding a solution in its particular subproblem. This subproblem is composed by its CSP subject to the variable assignment generated by the previous block agents. Thus, block agent 1 works on its group of constraints. If block agent 1 finds a solution to its subproblem, then it sends the consistent partial state to block agent 2, and both they work concurrently to solve their specific subproblems; block agent 1 tries to find other solution and block agent 2 tries to solve its subproblem knowing that its common variables have been assigned by block agent 1. Thus, block agent 3, with the variable assignments generated by the previous block agents, works concurrently with the previous block agents, and tries to find a more complete consistent state using a search algorithm. Finally, the last block agent \( k \), working concurrently with block agents 1, 2, \( \ldots \), \((k-1)\), tries to find a consistent state in order to find a problem solution.

Figure 2 shows the multi-agent model, in which the preprocessing agent carries out the network partition and the block agents (\( a_i \)) are committed to concurrently finding partial problem solutions (\( a_{ij} \)). Each block agent sends the partial problem solutions to the following block agent until a problem solution is found (by the last block agent). For example, state: \( a_{11} + a_{21} + \ldots + a_{k1} \) is a problem solution. The concurrence can be seen in Figure 2 in Time step 6 in which all block agents are concurrently working. Each block agent maintains the corresponding domains for its new variables. The block agent must assign values to its new variables so that the block of constraints is satisfied. When a block agent finds a value for each new variable, it then sends the consistent partial state to the next block agent. When the last block agent assigns values to its new variables satisfying its block of constraints, then a solution is found.

3.2 Distributed Model 2

Our second model can be considered as an asynchronous model. It is based on the topological properties of the constraint network. Specifically, it is centered on the graphical parameter known as induced width. The main idea of our multi-agent model is to divide the problem into a set of sub-problems where each sub-problem is a tree. In this case, each subproblem will be easily solved without backtracking because it has a width-1 ordering, then it has no cycles.

In our second proposal, the preprocessing agent divides the problem in an undefined number of blocks in order to be studied by block agents. The first block is obtained selecting a arbitrarily variable, the second selected variable is a random neighbor of the first selected variable, the next variable is a random neighbor of the previous selected variable which has not cycle with any of the previous selected variable in the block. Selection of the next variable continues recursively with the remaining graph until there is not any variable without cycle. The next blocks are obtained in the same way and taking into account that the selected variables are not included in the previous blocks.

Furthermore, the preprocessing agent is committed to arrange the subproblems in a Depth-First Search Tree (DFS Tree) where the root is the most constrained subproblem. DFS trees have already been investigated as a means to boost search [2]. Due to the relative independence of nodes lying in different branches of the DFS tree, it is possible to perform search in parallel on these independent branches. Once the variables are divided and arranged, a group of block agents manages each block of variables with their constraints. Each block agent is in charge of solving its own subproblem by means of the tree-solving algorithm defined in [2]. Each subproblem is composed by its CSP subject to the variable assignment generated by the ancestors block agents in the DFS tree.
Figure 2: Multi-agent model 1.

Figure 3: Multi-agent model 2.
Thus, root block agent works on its group of constraints. If root block agent finds a solution then it sends the consistent partial state to its children block agents in the DFS tree, and all children work concurrently to solve their specific subproblems knowing that its common variables have been assigned by root block agent. When a child block agent finds a solution it sends an OK message to its parent block agent. When all children answer with OK messages to the root block agent we will have the problem solution. When a child block agent does not find a solution, it sends a NOGOOD message to the root block agent. The NOGOOD message contains the variables which empty the variable domains of the child block agent. When the root block agent receives a NOGOOD message, it stops the children searches and it tries to find a new solution taking into account the NOGOOD information. If the root block agent does not find solution, then the problem has not solution. However, if it finds a new solution, it will start the same process again sending this new solution to its children. Each block agent works in the same way with its children in the DFS tree.

Figure 3 shows the multi-agent model 2, in which the preprocessing agent carries out the network partition and the subproblems arrangement. The root block agents (a1) starts the search process finding a partial solution. Then, it sends this partial solution to its children. The block agents which are brothers, are committed to concurrently finding the partial solutions of their subproblem. Each block agent sends the partial problem solutions to its children block agents. A problem solution is found when all leaf block agents find their partial solution. For example, (state s12 + s41) + (state s12 + s23 + s81) is a problem solution. The concurrence can be seen in Figure 3 in Time step 4 in which block agents a2 and a4 are concurrently working. Block agents a4 sends a NOGOOD message to its parent (block agents a1) in step 9 because it does not find a partial solution. Then, block agents a1 stops the search process of all its children and it finds a new partial solution which is sent to its children. Now, block agent a2 finds its partial solution, and block agent a0 works with its child, block agent a3, to find their partial problem solution. When block agent a3 finds its partial solution, the global problem will be found. It happens in Time step 25.

4 Evaluation

In this section, we carry out an evaluation between our distributed models and centralized models. To this end, we have used two well-known centralized CSP solver called Backtracking (BT) and Forward Checking (FC)

Evaluation of Distributed Model 1

In our first evaluation, each set of random CSPs was defined by the 3-tuple < n, a, p >, where n was the number of variables, a the arity of binary constraints and p the number of partitions. The problems were randomly generated by modifying these parameters.

In Tables 1 and 2, we compare the running time of the distributed model by METIS with the centralized problem. In Table 1, we fixed the arity of binary constraints and the size of the partition, and the number of variables was increased from 100 to 500. We can observe that the running time for small problems was worse by the distributed model than the centralized problem. However, when the number of variables increased, the behavior of the distributed problem was better. In Table 2, we fixed the number of variables and the arity of binary constraints and the size of the partition was increased from 3 to 20. We can observe that the size of the partition is important to distribute the problem. For small problems, the number of partition must be low. However, for large CSPs, the size of the partition must be higher. In this case, the appropriate number of partition was 7.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Distributed Model (s)</th>
<th>Centralized Model (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 50, 25, 10 &gt;</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>&lt; 100, 25, 10 &gt;</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>&lt; 150, 25, 10 &gt;</td>
<td>15</td>
<td>37</td>
</tr>
<tr>
<td>&lt; 200, 25, 10 &gt;</td>
<td>16</td>
<td>75</td>
</tr>
<tr>
<td>&lt; 250, 25, 10 &gt;</td>
<td>17</td>
<td>96</td>
</tr>
<tr>
<td>&lt; 300, 25, 10 &gt;</td>
<td>19</td>
<td>140</td>
</tr>
<tr>
<td>&lt; 350, 25, 10 &gt;</td>
<td>23</td>
<td>217</td>
</tr>
<tr>
<td>&lt; 400, 25, 10 &gt;</td>
<td>30</td>
<td>327</td>
</tr>
<tr>
<td>&lt; 450, 25, 10 &gt;</td>
<td>33</td>
<td>446</td>
</tr>
<tr>
<td>&lt; 500, 25, 10 &gt;</td>
<td>42</td>
<td>532</td>
</tr>
</tbody>
</table>

Evaluation of Distributed Model 2

In this evaluation, experiments were conducted on random networks of constraints defined by the 4-tuple < n, k, p, q >, where n was the number of variables, k the values in each domain, p was the constraints density (probability of a non-trivial edge) and q was the complementary of tightness (probability of an allowable pair in a constraint). These parameters are

1Forward Checking were obtained from CONFLEX. It can be found in: http://www�ia.inra.fr/T/CONFLEX/Logiciels/adressesConflex.html.
Table 2: Random instances <200, 25, p>, p: partition size.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Distributed Model (sec.)</th>
<th>Centralized Model (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;200, 25, 2&gt;</td>
<td>31</td>
<td>75</td>
</tr>
<tr>
<td>&lt;200, 25, 3&gt;</td>
<td>26</td>
<td>75</td>
</tr>
<tr>
<td>&lt;200, 25, 4&gt;</td>
<td>20</td>
<td>75</td>
</tr>
<tr>
<td>&lt;200, 25, 5&gt;</td>
<td>19</td>
<td>75</td>
</tr>
<tr>
<td>&lt;200, 25, 6&gt;</td>
<td>13</td>
<td>75</td>
</tr>
<tr>
<td>&lt;200, 25, 7&gt;</td>
<td>14</td>
<td>75</td>
</tr>
<tr>
<td>&lt;200, 25, 8&gt;</td>
<td>15</td>
<td>75</td>
</tr>
<tr>
<td>&lt;200, 25, 9&gt;</td>
<td>16</td>
<td>75</td>
</tr>
<tr>
<td>&lt;200, 25, 10&gt;</td>
<td>18</td>
<td>75</td>
</tr>
<tr>
<td>&lt;200, 25, 20&gt;</td>
<td>22</td>
<td>75</td>
</tr>
</tbody>
</table>

commonly used in experimental evaluations of CSP algorithms [11], [7]. The problems were randomly generated by modifying these parameters.

In Figures 4 and 5 we compare the running time of the distributed model 2 with two well-known centralized algorithm: Backtracking and Forward Checking. In Figure 4, we fixed the number of variables (40 variables) and the domain size was increased from 20 to 120. The running times are the average running times of 18 instance for each problem, where each problem had a probability of a not-allowable pair in a binary constraint between 10% and 30%. We can observe that the behavior of the distributed model was better than the other centralized algorithms. In Figure 5, we fixed the domain size to 60 and the number of variables was increased from 10 to 50. The average running time for problems with few variables was worse by the distributed model than the Forward Checking algorithm. However, the increase in the running time of the distributed model is the slowest. Therefore, the distributed model had a better behavior when the number of variables was large.

Figure 6 shows the average number of agents generated in each problem instance. Each problem instance is composed by the Cartesian Product of p and q (from 0.1 to 1). We can observe that the number of agents has a linear behavior with respect to the number of variables. Nevertheless, the number of agents is also directly related to the parameter p (constraints density). Figure 7 shows the number of agents generated in problems with 200 variables, when p and q increased from 0.1 to 1. The parameter q is not significative and it is not involved with the number of agents. However, the parameter p is very related with the number of agents. In Figure 7, we can observe that the number of agents for p = 1 is 100, that is, each agent is committed to only
two variables, because it is impossible to generate a tree with more than two variables.

5 Conclusions and further work

Distributed constraint satisfaction problems arise when pieces of information about variables, constraints or both are relevant to independent but communicating agents. They provide a promising framework to deal with the increasingly diverse range of distributed real world problems emerging from the evolution of computation and communication technologies. Furthermore, a general CSP can also distributed to be more efficiently solved. In this work, we present two techniques for distributing a CSP. The first one is based on graph partitioning, and the second one is based on trees. These techniques had a better behavior than centralized techniques.

Currently, we are working on a general architecture that integrate both models in order to select the more appropriate one, depending on the problem topology. This architecture will also be composed by different heuristics that increase efficiency without guaranteeing completeness.

References


