Dynamic Agent Ordering in Distributed Constraint Satisfaction Problems *

Lingzhong Zhou, John Thornton, and Abdul Sattar

School of Information Technology,
Griffith University Gold Coast, Southport, Qld, Australia, 4215
{l.zhou, j.thornton, a.sattar}@griffith.edu.au

Abstract. The distributed constraint satisfaction problem (CSP) is a general formalisation used to represent problems in distributed multi-agent systems. To deal with realistic problems, multiple local variables may be required within each autonomous agent. A number of heuristics have been developed for solving such multiple local variable problems. However, these approaches do not always guarantee agent independence and the size of problem that can be solved is fairly limited.
In this paper, we are interested in increasing search efficiency for distributed CSPs. To this end we present a new algorithm using unsatisfied constraint densities to dynamically determine agent ordering during the search. The independence of agents is guaranteed and agents without neighbouring relationships can run concurrently and asynchronously. As a result of using a backtracking technique to solve the local problem, we have been able to reduce the number of nogoods stored during the search, leading to further efficiency gains. In an empirical study, we show our new approach outperforms an equivalent static ordering algorithm and a current state-of-the-art technique both in terms of execution time and memory usage.

1 Introduction

The constraint satisfaction paradigm is a well recognised and challenging field of research in artificial intelligence, with many practical and important applications. A constraint satisfaction problem (CSP) is a problem with a finite number of variables, each of which has a finite and discrete set of possible values, and a set of constraints over the variables. A solution of a CSP is an instantiation of all variables for which all the constraints are satisfied.

When the variables and constraints of a CSP are distributed among a set of autonomous and communicating agents, this can be formulated as a distributed constraint satisfaction problem (distributed CSP), where agents autonomously and collaboratively work together to get a solution. A number of heuristics have been developed for solving distributed CSPs, such as synchronous backtracking, asynchronous backtracking (ABT) [4], asynchronous weak-commitment search

* The authors gratefully acknowledge the financial support of the Australian Research Council, grant A00000118, in the conduct of this research.

(AWC) [4] and the distributed breakout algorithm (DB) [5]. However, these algorithms can only handle one variable per agent. In [1], Armstrong and Durfee use dynamic prioritisation to allow agents with multiple local variables in distributed CSPs. Here, each agent tries to find a local solution, consistent with the local solutions of higher priority agents. If no local solution exists, backtracking or modification of the prioritisation occurs. The approach uses a centralised controller, where one agent controls the starting and ending of the algorithm, and a nogood processor which records all nogood information. However, these centralised mechanisms are often not appropriate for realistic distributed CSPs. In [6], Yokoo and Hirayama extended AWC search to deal with multiple local variables in distributed CSPs. However, their approach requires a large space to store nogoods during the search.

In this paper, we propose a new *Dynamic Agent Ordering* (DAO) algorithm, which uses constraint density measures to locally compute a *degree of unsatisfaction* for each agent. These values are used to dynamically set the order in which agents are allowed to change their particular variable instantiations. In effect, the agents' orders are decided naturally by their unsatisfied constraint densities during the search. As each local computation is independent from other agents, the benefits of parallelism are retained, resulting in an approach that is suitable for agent oriented design and efficient in terms of memory cost.

In the rest of the paper, we formalise the definition of a distributed CSP. Then, we describe the new algorithm and investigate its performance in an empirical study. Finally, we discuss the possibility of using the new algorithm to solve other variants of distributed CSPs.

2 Distributed Constraint Satisfaction Problems

A distributed constraint satisfaction problem is defined as a CSP, in which variables and constraints are distributed among multiple autonomous and communicating agents. The agents may be distributed in different locations or in the same location but among different processes. Each agent contains a subset of the variables and tries to instantiate their values. Constraints may exist between the variables of one agent or between different agents. The instantiations of the variables must satisfy all constraints. In this paper, we consider that all constraints are binary.

2.1 Formalisation

In a distributed constraint satisfaction problem:

1. There exists an agent set A:
$$A = \{A_1, A_2, ..., A_n\},\ n \in Z^+;$$
2. Each agent has a variable set X_i and domain set D_i,
$$X_i = \{X_{i1}, X_{i2}, ..., X_{ip_i}\};$$
$$D_i = \{D_{i1}, D_{i2}, ..., D_{ip_i}\},\ \forall i \in [1,n],\ p_i \in Z^+;$$

3. There are two kinds of constraints over the variables among agents:
   (a) Intra-agent constraints, which are between variables of same agent.
   (b) Inter-agent constraints, which are between variables of different agents.

   Agent A_i knows all constraints related to its variables. A variable may involve both intra-agent and inter-agent constraints.
4. A solution S, is an instantiation for all variables that satisfies all intra-agent and inter-agent constraints.

Since agents are distributed in different locations or in different processes, each agent only knows the partial problem associated with those constraints in which it has variables. A global solution then consists of a complete set of the overlapping partial solutions for each agent. Communication among agents is necessary and important in distributed CSPs, since each agent only knows its variables, variable domains and related intra-agent and inter-agent constraints. Hence, to evaluate a search algorithm, we not only need to measure the search speed but also to consider the communication cost.

Example:

Consider the following distributed CSP, shown in Figure 1:

1. Agents: A_1 and A_2;
2. Variable sets: $\{X_1, Y_1, Z_1\}$ in agent $A_1$and $\{X_2, Y_2, Z_2\}$ in agent A_2;
3. Domain sets: $\{D_{X_1} = \{1, 3\}, D_{Y_1} = \{1, 2, 3\}, D_{Z_1} = \{2, 4\}\}$ in agent $A_1$and $\{D_{X_2} = \{1, 2, \}, D_{Y_2} = \{2, 3\}, D_{Z_2} = \{2, 3\}\}$ in agent A_2;
4. Intra-agent constraints: $\{X_1 \neq Y_1, Y_1 = Z_1\}$ and $\{X_2 = Y_2, Y_2 \neq Z_2\}$;
5. Inter-agent constraints: $\{X_1 \neq X_2, Y_1 \neq Z_2\}$;
6. Solution: $S = \{X_1 = 1, Y_1 = 2, Z_1 = 2, X_2 = 2, Y_2 = 2, Z_2 = 3\}$

In this example, each arc represents one constraint. If we reduce the number of the agents to one, a distributed CSP would become a local CSP. So we may consider a distributed CSP as a combination of several local CSPs. Compared with local CSPs, a distributed CSP has to deal with communication costs and delay, privacy, cooperation, extra computation, consistency, asynchronous changes, infinite processing loops, and all the basic problems of distributed computing.

3 The Dynamic Agent Ordering Algorithm

3.1 Motivation

In a CSP, the order in which values and variables are processed significantly affects the running time of an algorithm. Generally, we instantiate variables that maximally constrain the rest of the search space. For instance, selecting the variable with the least number of values in its domain tends to minimise the size of the search tree. When ordering values, we try to instantiate a value that maximises the number of options available for future instantiations.

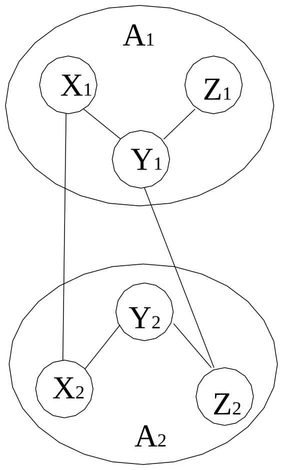

Fig. 1. An example of a distributed CSP

The efficiency of algorithms for distributed CSPs is similarly affected by the order of value and variable selection. In the case where agents control multiple variables, the order in which agents are allowed to instantiate shared variables also becomes important. Agent communication and external computation (instantiating variables to be consistent with inter-agent constraints) is more costly than local computation (instantiating variables to be consistent with intra-agent constraints), and wrong or redundant computation can occur as a result of inappropriate agent ordering. It is therefore worth investigating agent orderings in order to develop more efficient algorithms.

The task of ordering agents is more complex than ordering variables, as more factors are involved, i.e. not only constraints and domains but also the structure of neighbouring agents. Deciding on agent ordering is analogous to granting a priority to each agent, where the priority order represents a hierarchy of agent authority. When the priority order is static, the order of agents is determined before starting the search process, and the efficiency of the algorithm is highly dependent on the selection of variable values. If the priority order is dynamic, this can be used to control decision making for each agent and the algorithm is more able to flexibly exploit to the current search conditions.

We propose a new algorithm which uses constraint density (related to both intra-agent and inter-agent constraints) to order agents in a distributed CSP. When a search becomes stuck (i.e. an inconsistency is found), the algorithm calculates the unsatisfied constraint densities and the degree of unsatisfaction for each agent, and the agent that is most unsatisfied is reassigned. As a backtracking search is used for each local computation, agents can still run asynchronously and concurrently. The algorithm also reduces the size of the nogood store (in comparison to AWC), and so allows larger problems to be solved.

3.2 Agent Ordering

To develop a dynamic agent ordering algorithm requires the specification of those features of the search space that should determine the ordering. In this study we develop a measure of the *degree of unsatisfaction* for each agent, such that the agent with the highest degree of unsatisfaction has the highest priority. In a standard CSP, the degree of unsatisfaction can simply be measured as the number of constraints unsatisfied divided by the total number of constraints. However, in a distributed CSP, we have the additional consideration of the relative importance of intra- versus inter-agent constraints. As inter-agent constraints affect variables in more than one agent, and these variables in turn can affect the intra-agent problems, we decided to develop separate measures for the intra- and inter-agent problems, such that the inter-agent constraints are given greater importance. To do this we looked at two problem features: (i) the degree of interconnectedness between constraints (or constraint density) and (ii) the degree of interconnectedness between inter-agent constraints and the intra-agent local problem.

To measure constraint density, we firstly divided the problem for a particular agent into an intra-agent constraint problem and an inter-agent constraint problem:

Intra-Agent Constraint Density: For the intra-agent problem, the maximum constraint density is simply defined as the ratio of the number of constraints over the number of variables, i.e. for agent i:

$$intraDensity_i = \frac{|intraC_i|}{|intraV_i|}$$

where $intraC_i$ is the set of intra-agent constraints for agent i and $intraV_i$ is the set of variables constrained by $intraC_i$. Assuming that each constraint has the same *tightness*[1], then we would expect a larger density to indicate a more constrained and hence more difficult problem.

Inter-Agent Constraint Density: The constraint density measure for the inter-agent problem contains two additional features which increase the relative importance of the inter-agent measure in comparison to the intra-agent measure. Firstly, for agent i, instead of dividing by the total number of variables constrained by i's inter-agent constraints $interC_i$, we divide only by the number of variables that are constrained by $interC_i$ *and* controlled by i, i.e. $|interV_i|$.

In addition, when counting agent i's jth inter-agent constraint, $c_{i,j}$, we also count the number of *intra*-agent constraints $m_{i,j}$ that share a variable with $c_{i,j}$. This means the more interconnected $c_{i,j}$ is with the intra-agent problem, the larger the value of $m_{i,j}$ and the greater the effect of $c_{i,j}$ on the overall inter-agent constraint density, given by:

[1] i.e. the ratio of the number unsatisfying assignments over the total number of possible assignments.

$$interDensity_i = \frac{|interC_i| + \sum_{j=1}^{|interC_i|} m_{i,j}}{|interV_i|}$$

The sum $staticDensity_i = intraDensity_i + interDensity_i$ now provides a general measure of the overall density of the problem for a particular agent. The greater this measure, the more constrained or difficult we would consider the problem to be.

Dynamic Constraint Density: The dynamic constraint density for a particular agent is based on the static density measure, except that only unsatisfied constraints are counted in the numerator. In this way the density of a current level of constraint violation during a search can be measured. Using the functions $intraUnsat(i, j)$, which returns one if the jth intra-agent constraint for agent i is unsatisfied, zero otherwise, and $interUnsat(i, j)$, which returns one if the jth inter-agent constraint for agent i is unsatisfied, zero otherwise, we define the following measures:

$$intraUnsat_i = \frac{\sum_{j=1}^{|intraC_i|} intraUnsat(i, j)}{|intraV_i|}$$

and

$$interUnsat_i = \frac{\sum_{j=1}^{|interC_i|} (interUnsat(i, j) \times (m_{i,j} + 1))}{|interV_i|}$$

These measures then range from a value of zero, if all constraints are satisfied, to $intraUnsat_i = intraDensity_i$ and $interUnsat_i = interDensity_i$ if all constraints are unsatisfied. Combining these measures, we define:

$$dynamicDensity_i = intraUnsat_i + interUnsat_i$$

and

$$degreeUnsat_i = \frac{dynamicDensity_i}{staticDensity_i}$$

$degreeUnsat_i$ now ranges from a value of zero, if all constraints for agent i are satisfied, to one, if all constraints are unsatisfied, while embodying the increased importance of inter-agent constraints in the overall evaluation. It is this measure we use to dynamically decide agent priority in our proposed algorithm.

Example: To further clarify the details of these measures, we use a distributed 3-colouring problem shown in Figure 2. The goal of the problem is to assign colours to each node so that nodes connected by the same arc have different colours[2]. In Figure 2 (a), Agent 1 has three inter-agent constraints ($interC_1 = \{C_{25}, C_{24}, C_{34}\}$) and two intra-agent constraints ($intraC_1 = \{C_{12}, C_{23}\}$). When Agent 1 attempts to satisfy the inter-agent constraint C_{25} by changing variable V_2, its instantiation affects two other intra-agent constraints C_{12} and C_{23}. Using definition of $interDensity_i$, this equates to $m_{1,1} = 2$. Similarly, Agent 1's second inter-agent constraint C_{24} is also connected to both of Agent 1's intra-agent constraints (C_{12} and C_{23}), making $m_{1,2} = 2$, and finally Agent 1's third inter-agent constraint C_{34} is connected to a single intra-agent constraint C_{23}, making $m_{1,3} = 1$, and giving:

$$\sum_{j=1}^{3} m_{1,j} = 2 + 2 + 1 = 5$$

As Agent 1 has three inter-agent constraints ($|interC_1| = 3$), two intra-agent constraints ($|intraC_1| = 2$), three intra-agent variables ($intraV_1 = \{V_1, V_2, V_3\}$, $|intraV_1| = 3$) and two inter-agent variables ($interV_1 = \{V_2, V_3\}$, $|interV_1| = 2$), Agent 1's intra- and inter-constraint densities are given by the following:

$$intraDensity_1 = \frac{|intraC_1|}{|intraV_1|} = \frac{2}{3}$$

and

$$interDensity_1 = \frac{|interC_1| + \sum_{j=1}^{3} m_{1,j}}{|interV_1|} = \frac{3+5}{2} = 4$$

Now considering Figure 2 (b), we can see that all Agent 1's intra-agent constraints are satisfied (i.e. each pair of colours on each arc is different) and all inter-agent constraints are satisfied except C_{24} where $V_2 = V_4 = \text{Y}$. This means the expression $\sum_{j=1}^{2} intraUnsat(1,j)$ evaluates to zero, (i.e. $intraUnsat(1,1) = 0$ as C_{12} is satisfied and $intraUnsat(1,2) = 0$ as C_{23} is satisfied). Similarly each term in $\sum_{j=1}^{3} interUnsat(1,j)$ will evaluate to zero, except $interUnsat(1,2)$, corresponding to the unsatisfied inter-agent constraint C_{24}. In this case $m_{1,2} + 1 = 3$ as C_{24}'s variable V_2 is connected to two intra-agent constraints. From this it follows:

$$intraUnsat_1 = \frac{\sum_{j=1}^{2} intraUnsat(1,j)}{|intraV_1|} = \frac{0}{3} = 0$$

and

$$interUnsat_1 = \frac{\sum_{j=1}^{3} interUnsat(1,j) \times (m_{1,j} + 1)}{|interV_1|} = \frac{3}{2}$$

Putting these measures together we can now determine the degree of unsatisfaction for Agent 1:

[2] B = blue, R= red, Y = yellow.

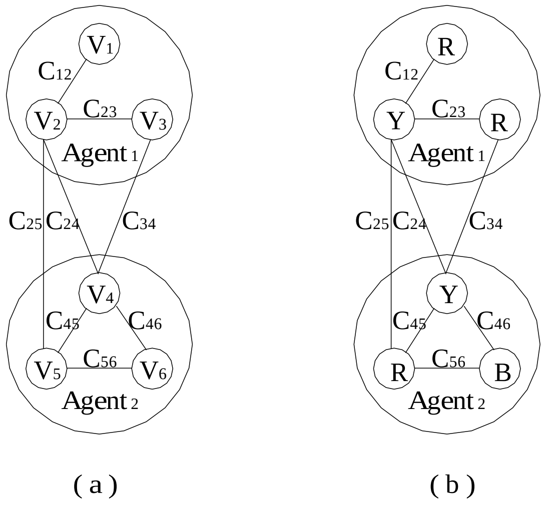

Fig. 2. A Distributed 3-colouring Problem

$$degreeUnsat_1 = \frac{intraUnsat_1 + interUnsat_1}{intraDensity_1 + interDensity_1} = \frac{0 + \frac{3}{2}}{\frac{2}{3} + 4} = \frac{9}{28}$$

Performing the same series of calculations for Agent 2, we obtain a degree of unsatisfaction of:

$$degreeUnsat_2 = \frac{intraUnsat_2 + interUnsat_2}{intraDensity_2 + interDensity_2} = \frac{0 + \frac{3}{2}}{\frac{3}{3} + \frac{9}{2}} = \frac{3}{11}$$

As $\frac{9}{28} > \frac{3}{11}$, it follows that $degreeUnsat_1 > degreeUnsat_2$, giving Agent 1 priority over Agent 2, and hence the authority to perform the next instantiation. In this case that would mean setting V_2 to B and hence finding a global solution to the problem. Note that the alternative of allowing Agent 2 to move would have resulted in several further instantiations before a global solution could be found. The reason for preferring Agent 1 in this situation can be expressed as follows: both Agent 1 and 2 have a single inter-agent constraint unsatisfied (C_{24}) for which their *dynamicDensity* measures are equal ($\frac{3}{2}$). However, Agent 1's overall static density measure is less than Agent 2 ($4\frac{2}{3}$ versus $5\frac{1}{2}$), because Agent 1's intra-agent problem is easier. Consequently the *dynamicDensity* value of $\frac{2}{3}$ has a greater effect on Agent 1, giving it priority. In effect, Agent 1 was preferred because its sub-problem was less dense (i.e. less constrained), meaning it would have the greater probability of finding a satisfying instantiation.

Overall, the *degreeUnsat* measure has the ability to show the complexity of the problem and the constraint strength in each agent. This further allows autonomous agents make decisions about who should change their variable values, without relying on a centralised evaluation mechanism.

3.3 Algorithm Implementation

The Dynamic Agent Ordering (DAO) algorithm was implemented as follows:

1. In the initial state, each agent concurrently instantiates their variables to construct a local solution, while checking consistency to guarantee that all intra-agent constraints are satisfied. Each agent then sends its local solution to its neighbouring agents (i.e. those with which it shares at least one inter-agent constraint);
2. Each agent then starts to construct a local solution which attempts to satisfy both intra- and inter-agent constraints. Assuming the overall problem is satisfiable, if an agent is unable to satisfy its partial solution, an inter-agent constraint must be involved. In this case, the two agents that share the constraint compare their *degreeUnsat* values, and the agent with greater value has priority and is allowed to reassign its variable. If the values are same[3], we randomly assign priority; if an agent's *degreeUnsat* is less than its neighbours and its local problem is satisfied, then it simply waits for messages. If there is no suitable value for a local variable, the local agent tries to satisfy as many constraints as possible. This state will then be recorded as a nogood and sent to the related agents for the completeness of the algorithm;
3. After assigning its own variables, an agent sends messages to neighbouring agents. These messages contain the *degreeUnsat* value and the local instantiation for the agent.
4. The search will stop when each agent detects its and all its neighbouring agents' *degreeUnsat* values are equal to zero. No extra consistency detection method is needed.

The DAO algorithm is shown in more detail in Figure 3. Here, an *optimal_local_partial_solution* is a partial solution for local variables that satisfy a maximal number of intra-agent and inter-agent constraints. If all constraints are satisfied, the optimal local partial solution is equal to the local solution. The *culprit_variables* are variables in unsatisfied constraints that prevent a partial solution from being expanded further. These *culprit_variables* may be in different agents.

3.4 Experimental Evaluation

We evaluated the Dynamic Agent Ordering algorithm on a benchmark set of 3-colouring problems and against two other algorithms. The first, Asynchronous Weak-commitment (AWC) search, is recognised as the state-of-the-art for distributed CSPs, where each agent has control over multiple variables [6,2]. We implemented the latest version of AWC which uses nogood learning and obtained

[3] Normally, this rarely happens in a distributed CSPs, since the density value not only depends on the number of unsatisfied constraints and variables in a local agent, but also on the structure of neighbours, the distribution of intra-agent and inter-agent constraints, and so on.

```
when received($Sender\_id$, $variable\_values$, $degreeUnsat$) do
    if all neighbouring $degreeUnsats = 0$
    then search is terminated;
    else add ($Sender\_id$, $variable\_value$, $degreeUnsat$) to agent_view;
        calculate $local\_degreeUnsat$;
        if $local\_degreeUnsat > degreeUnsat$
        then assign_local_variables;
            calculate $degreeUnsat$;
            send($Sender\_id$, $variable\_values$, $degreeUnsat$) to
            neighbouring agents;
        endif
    endif
enddo
Procedure assign_local_variables
  sequentially assign variables using chronological backtracking to construct
  $optimal\_local\_partial\_solution$, which satisfies all related
  intra-agent and inter-agent constraints within lower
  $degreeUnsat$ agents, and against local $nogood\_set$;
  if $optimal\_local\_partial\_solution$ is not the $local\_solution$
  then assign remaining variables to satisfy as many constraints as possible;
      add the $culprit\_variables$ with their $values$, $degreeUnsat$ and
      $agent\_ids$ to the $nogood\_set$;
      if $nogood\_set$ is new
      then record the new nogood;
          send $nogood\_set$ messages to the related agents;
      endif
  endif
```

Fig. 3. The Dynamic Agent Ordering Algorithm

comparable results to those reported in [2]. In addition, we implemented a version of DAO with the dynamic variable ordering switched off, called Static Agent Ordering (SAO). In this case, agent priority is determined statically before the search is commenced using the *staticDensity* measure defined in Section 3.2.

To simulate an autonomous agent environment we used an agent oriented design, implementing threads in FreeBSD that allow agents to run asynchronously and concurrently. All experiments were run on a Dell OptiPlex GX240 with a 1.6GHz P4 CPU and 256MB of PC133 DRAM. We used the same 3-colouring problem generator described in [3] and improved in [6] to evaluate the performance of our algorithms. We chose this domain as the 3-colouring problem has been used in many other studies, and this type of problem is often used in connection with scheduling and resource allocation problems. To build the problem set, we randomly generated 100×50 variable and 40×100 variable problems in the hard region of 3-colouring with a constraint to variable ratio of 2.7, assigning 50% of constraints as inter-agent and 50% as intra-agent constraints (within each problem). Each agent was also constrained to have at least one inter-agent constraint.

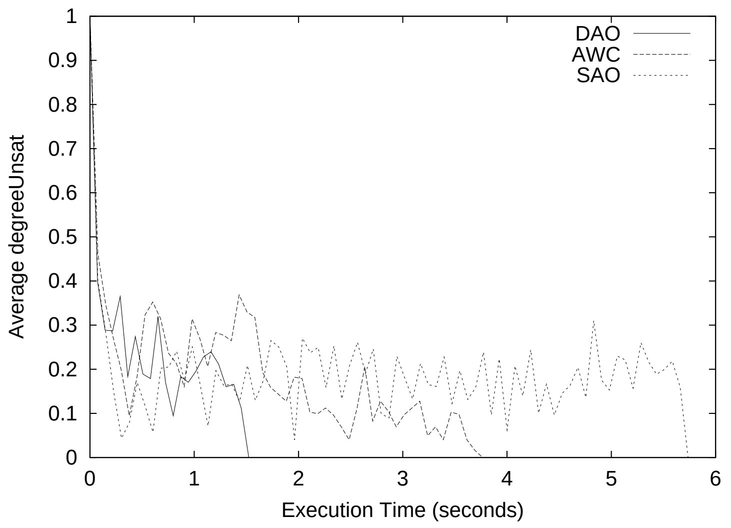

Fig. 4. The average *degreeUnsat* plotted against time

Figure 4 shows the average *degreeUnsat* for each of the three algorithms over the 50 variable problem set, and Table 1 shows the the number of average checks, the number of total nogoods produced, the number of total local instantiations broadcasted and the total running time for all agents over the complete problem set. From these results it is clear that the new algorithm is considerably more efficient than AWC and SAO in terms of execution time. Although DAO produces more local instantiations, the size of its nogood store is significantly smaller. Also, unlike AWC, an optimal local solution can be sent by each agent during the search. As a result, a local agent has more options to instantiate its variables. The main disadvantage of DAO is that more communication costs are incurred, nevertheless, these costs are more than compensated for by the smaller number of nogoods recorded and the faster search times.

Problem	Method	Checks	Nogoods	Local Instantiations	Time (seconds)
50 variables 100 runs	DAO	1,860.3	87.6	138.8	1.7254
	AWC	2,129.5	224.3	98.3	3.9624
	SAO	3,928.2	175.8	349.5	6.2387
100 variables 40 runs	DAO	17,357.4	734.8	1,277.0	33.4567
	AWC	23,617.6	1,927.2	825.4	56.9256
	SAO	44,998.2	1,256.0	2,774.8	245.4677

Table 1. Results for distributed 3-colouring problems with 10 agents

4 Conclusion and Future Work

We have demonstrated a new algorithm that uses constraint density to dynamically order agents and increase the search speed in distributed CSPs. We argue that our algorithm is more feasible and offers greater agent independence than the existing algorithms for distributed CSPs, especially for situations with multiple local variables in each agent.

Since agent independence is guaranteed, DAO can be used to solve dynamic distributed CSPs and distributed over-constrained CSPs. Dynamic distributed CSPs are common in realistic problems, where agents may be lost or added over time. By using our algorithm, real-time calculations are able to build new relations among agents, and constraints and/or variables in one agent will not affect other agents' local computations. In fact, it is not necessary to modify the algorithm to handle dynamic distributed CSPs. When a distributed CSP has no solutions, it is over-constrained. To deal with this kind of problem, we can setup a gate value (between 0 and 1) for the $degreeUnsat$ values. After all $degreeUnsat$ values reach the gate value, the problem is solved.

Finally, for problems where individual constraints have varying degrees of tightness, we can amend our constraint density measures to consider tightness directly. Currently we *count* the number of intra- and inter-agent constraints for each agent when calculating density. Alternatively, we can sum the tightness of these constraints, where tightness is defined as the number of possible unsatisfying assignments for a constraint divided by the total number of possible assignments.

References

1. Aaron Armstrong and Edmund Durfee. Dynamic prioritization of complex agents in distributed constraint satisfaction problems. In *The Fifteenth International Joint Conference on Artificial Intelligence*, pages 620–625, 1997.
2. Katsutoshi Hirayama and Makoto Yokoo. The effect of nogood learning in distributed constraint satisfaction. *The 20th International Conference on Distributed Computing Systems (ICDCS 2000)*, April 2000.
3. S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, pages 161–205, 1992.
4. Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transaction on Knowledge and Data Engineering*, 10(5):673–685, 1998.
5. Makoto Yokoo and Katsutoshi Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction problems. *Proceedings of the Second International Conference on Multiagent Systems (ICMAS-96)*, pages 401–408, 1996.
6. Makoto Yokoo and Katsutoshi Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *the Third International Conference on Multiagent Systems (ICMAS-98)*, pages 372–379, 1998.