

Hierarchical Variable Ordering for Distributed Constraint Optimization

John Davin
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
jdavin@cs.cmu.edu

Pragnesh Jay Modi
Department of Computer Science
Drexel University
Philadelphia, PA 19104
pmodi@cs.drexel.edu

ABSTRACT

The Multiagent Agreement Problem (MAP) is a special form of Distributed Constraint Optimization (DCOP) that requires agents to choose values for variables to satisfy not only their own constraints, but also equality constraints with other agents. We introduce the AdoptMVA algorithm, an extension of the existing Adopt algorithm, designed to take advantage of MAP domains where agents often control multiple variables. We also propose an approach to agent ordering which leverages known ordering techniques from the centralized and distributed constraint satisfaction literature and applies them to MAPs. By combining ordering at the agent level with orderings at the variable level, we hope to obtain efficient global orderings. While the contributions discussed in this paper are applicable to general DCOPs, we focus our evaluation on MAPs because we feel it is a significant problem class worthy of specific attention.

Categories and Subject Descriptors

I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—*Multiagent Systems*

General Terms

Algorithms

Keywords

Distributed Constraint Satisfaction/Optimization

1. INTRODUCTION

The Multiagent Agreement Problem (MAP) is an instance of the more general Distributed Constraint Optimization Problem (DCOP) and is a convenient framework for representing coordination problems where multiple agents must come to an agreement for example on the time of a meeting or a joint action [7][5]. We propose AdoptMVA as an extension of the existing Adopt algorithm to operate more efficiently on MAPs where agents own multiple variables. The key idea is to use centralized search procedures to solve each

agents' subproblem, in conjunction with the distributed Adopt algorithm to solve the broader problem. We show that when variable ordering is controlled for, AdoptMVA completes in fewer cycles than original Adopt on MAPs.

It is well known that the choice of variable ordering can have a large effect on the performance of a constraint-based search algorithm. Indeed, the choice of variable ordering becomes an even more important issue when we have multiple variables per agent. We define a set of inter-agent ordering heuristics for ordering at the macro-agent level and evaluate their comparative performance on MAPs using AdoptMVA. We present a general technique for converting existing effective variable orderings into effective agent orderings. In summary, by combining inter-agent ordering and intra-agent ordering heuristics and inventing AdoptMVA to exploit them, we enable a hierarchical approach to variable ordering in DCOP.

2. MULTIAGENT AGREEMENT PROBLEM

The Multiagent Agreement Problem (MAP) [7] is a special class of DCOP [6] in which a variable can be shared among multiple agents. The model can be made equivalent to a DCOP by giving copies of the variable to each agent and using inter-agent equality constraints to ensure agreement. We will use MAP definitions to aid in explaining experiments described in this paper. The components of MAP are defined as follows:

- $\mathcal{A} = \{A_1, A_2, \dots, A_n\}$ is a set of *agents*.
- $\mathcal{V} = \{V_1, V_2, \dots, V_m\}$ is a set of *variables*.
- $\mathcal{D} = \{d_1, d_2, \dots, d_k\}$ is a set of *values*. Each value can be assigned to any variable.
- $participants(V_i) \subseteq \mathcal{A}$ is the set of agents assigned the variable V_i . The participants are responsible for choosing the value of V_i .
- $vars(A_i) \subseteq \mathcal{V}$ is the set of variables assigned to agent A_i .
- For each agent A_i , C_i is an *intra-agent* constraint that must be defined *only* over the variables in $vars(A_i)$. C_i can be used to model an agent's local preferences for certain values.¹
- For each variable V_i , an *inter-agent* "agreement" constraint is satisfied if and only if the same value from \mathcal{D} is assigned to V_i by all the agents in $participants(V_i)$.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.
Copyright 2006 ACM 1-59593-303-4/06/0005 ...\$5.00.

¹This is a generalization of the model introduced in [7] where C_i was required to evaluate to either true or false.

3. ADOPTMVA

The existing Adopt algorithm can be trivially adapted to handle problems in which an agent owns multiple variables by treating variables within an agent as virtual agents. However, this previous approach has the drawback that it artificially segments problem information within an agent and fails to take advantage of the full information sharing that is possible. We aim to eliminate this inefficiency by allowing full sharing of information within an agent. Our approach is to create a single process for each agent and allow it to control all of the variables owned by the agent. Then, a centralized search procedure is used to find the optimal solution for the agent's subproblem. We call this **AdoptMVA**, for Multiple Variables per Agent.

In the Adopt formalism, a value assignment to a set of variables is called a context and is defined as follows [6].

- **Definition:** A *context* is a partial solution of the form $\{(x_j, d_j), (x_k, d_k), \dots\}$. A variable can not appear in a context more than once. Two contexts are *compatible* if they do not disagree on any variable assignment. *CurrentContext* is a context which holds an agent's current knowledge of variable values for higher priority neighbors.

As defined in MAP, we let $\text{vars}(A_i) \subseteq \mathcal{V}$ denote the variables owned by agent A_i . We then denote a context $s \in S$ where S is the set of all possible assignments to variables in $\text{vars}(A_i)$ and s is a particular one of those assignments.

Whereas previously an agent owned only a single variable and so its cost was determined only by its constraints with other agents, we now must define a function that also includes the cost of constraints between variables *within* an agent. We define the *local cost* δ for a particular value assignment s made by agent A_i for the variables it owns as:

$$\delta(s) = \sum_{(x_j, d_j) \in s} \sum_{(x_i, d_i) \in s} f_{ij}(d_i, d_j) + \sum_{(x_j, d_j) \in \text{CurrentContext}} \sum_{(x_i, d_i) \in s} f_{ij}(d_i, d_j) \quad (1)$$

The first half of the definition sums the constraints between every possible pairing of variables within an agent, and the second half evaluates the constraints between each of the agent's variables and all of the external variables in its *CurrentContext*.

An agent computes a *lower bound for a solution* s using lower bound costs received from its children, denoted as $lb(s, x_l)$.

- **Definition:** $LB(s) = \delta(s) + \sum_{x_l \in \text{Children}} lb(s, x_l)$ is a *lower bound* for the subtree rooted at A_i when A_i chooses solution $s \in S$.

Similarly for an *upper bound on solution* s :

- **Definition:** $UB(s) = \delta(s) + \sum_{x_l \in \text{Children}} ub(s, x_l)$ is an *upper bound* for the subtree rooted at A_i when A_i chooses solution $s \in S$.

The overall lower and upper bounds for A_i are the minimums over the bounds for all possible solutions in S :

- **Definition:** $LB = \min_{s \in S} LB(s)$ is a *lower bound* for the subtree rooted at A_i .
- **Definition:** $UB = \min_{s \in S} UB(s)$ is an *upper bound* for the subtree rooted at A_i .

Algorithm 1: AdoptMVA Branch & Bound search

```

depth ← 0
cost ← 0
bestCost ← ∞

procedure search(  $s \in S$ , depth, cost ):
if depth == s.size() then
    bestSolution ← s
    bestCost ← cost
    return;
end
 $V_i \leftarrow$  variable at depth (order determined by variable
ordering)
Domain ←  $D_i$ 
Reorder Domain with best-first heuristic (move current value
of  $V_i$  in best known solution to be first)
for  $d_i \in \text{Domain}$  do
     $s\{V_i\} \leftarrow d_i$ 
    cost+ =  $\sum_{(x_j, d_j) \in s} f_{ij}(d_i, d_j) +$ 
 $\sum_{(x_j, d_j) \in \text{CurrentContext}} f_{ij}(d_i, d_j)$ 
    if depth == s.size() - 1 then
        cost ← cost +  $\sum_{x_l \in \text{Children}} lb(s, x_l)$ 
    end
    if cost < bestCost then
        search( s, depth+1, cost )
    end
     $s\{V_i\} \leftarrow \text{null}$ 
end

```

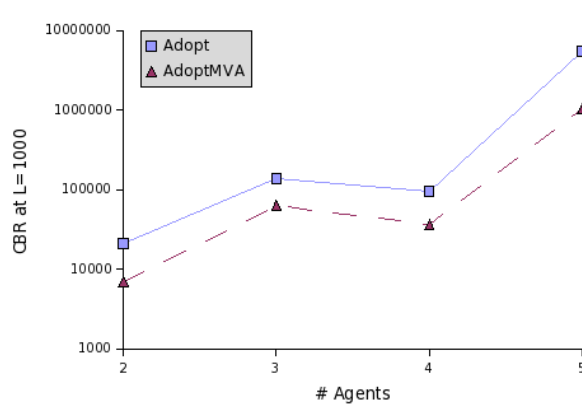
The above definitions are based on the original Adopt definitions [6] with the main change being that lower and upper bounds are now calculated using a local solution context s rather than a single variable x_i . Also, the most important change to the local cost δ is that it now includes the cost of intra-agent constraints.

In order to properly communicate the values of an agent's local variables to the other agents in the problem, we also have to modify Adopt's VALUE messages and its handling of COST messages. In particular, VALUE messages must now include the values of all variables owned by the agent, rather than just a single variable value. Therefore, we extend VALUE messages to include a solution context:

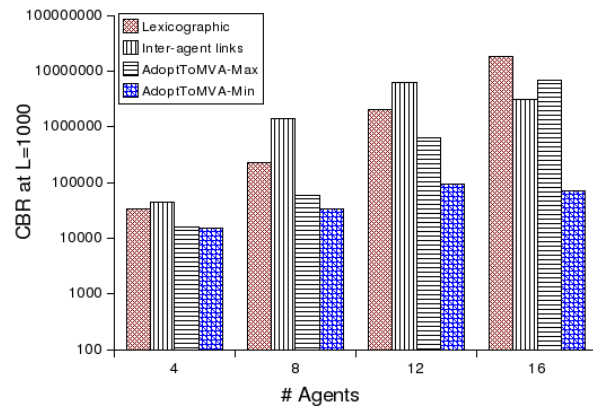
- **Definition:** $\text{VALUE}(s_i = \{(x_j, d_j), (x_k, d_k), \dots\})$ is the form of the new VALUE messages sent to neighbors lower in the tree.

Finally, each agent uses a centralized Branch & Bound search [4] to find the optimal solution to its local variables, given the values of its ancestors in the Adopt hierarchy, and knowledge of the costs of its children. The search is used to calculate both the lower and upper bounds (LB and UB) on the agent's variables. Algorithm 1 shows pseudo code for the lower bound search (upper bound search is identical except $lb(s, x_l)$ is replaced with $ub(s, x_l)$). The value ordering is a best-first heuristic that puts the best domain value first in the search order for a given variable, where the best value is taken from the current best solution (if known).

Also note that the cost calculation in each step of the recursive search is not exactly the same as the δ function in Equation 1. Rather, the true cost of δ is reached at the bottom of the recursion because we add in the lower or upper bounds from our children at the end. This is so we don't redundantly calculate this cost at each recursion. If a bound for a child at the given solution s is not



(a) AdoptMVA versus Adopt



(b) AdoptMVA agent ordering heuristics

Figure 1: Experimental results on MAPs in the meeting scheduling domain.

known, the lower bound defaults to 0 and the upper bound to ∞ .

4. ORDERING HEURISTICS

We define four inter-agent orderings, which are macro-level variable orderings used to determine the parent/child relationships in AdoptMVA.

- **Lexicographic** - Agents are ordered alpha-numerically by agent name. This is an uninformed heuristic.
- **Inter-agent constraints heuristic** - Agents are ordered recursively by the number of inter-agent constraints to already ordered agents. Ties are broken by the number of inter-agent constraints to currently unordered agents, and then lexicographically. This is a similar to the existing Brelaz variable ordering heuristic [1] except rather than counting all constraints, we only count inter-agent constraints.

The next two orderings were created based on the theory that many effective variable ordering heuristics exist and so converting them to an agent ordering might produce similar results. In particular, we can convert any given global variable ordering to an agent ordering by assigning agent priorities based on the priorities of variables within that agent. Given a prioritization of variables, a general technique for computing agent A_i 's agent priority is as follows:

$$\text{priority}(A_i) = \max_{V_j \in \text{vars}(A_i)} \text{priority}(V_j).$$

Below are two heuristics that instantiate this theme.

- **AdoptToMVA-Max** - We first compute a global variable ordering using Brelaz [1], and then convert it into an agent ordering for AdoptMVA using the above conversion.
- **AdoptToMVA-Min** - This is the same as AdoptToMVA-Max except that agent priorities correspond to the *minimum* priority variable within each agent.

We will present experimental results for the four agent ordering heuristics. Although not presented here, we also investigated a set of heuristics suitable for MAP for ordering variables *within an agent*, i.e., intra-agent variable orderings. We evaluated their comparative performance and are able to identify an intra-agent variable ordering heuristic for MAP that is the most efficient of the ones tested.

5. RESULTS

We conducted experiments on MAPs used to model a meeting scheduling domain. More results than those presented here are available [2]. We present results in terms of the Cycle-Based Runtime (CBR) metric [3] which we believe is a more informative and fairer metric because it more closely approximates true distributed run-time than the previously used synchronous cycles metric. The formula for CBR is $CBR = \text{Cycles} * L + CCC$, where cycles is the number of synchronous cycles consumed by the algorithm, L is a factor indicating the relative communication speed, and CCC is the total number of concurrent constraint checks.

First, we compare the performance of AdoptMVA versus Adopt. To ensure a level playing field, we control for agent ordering using a static lexicographic agent ordering for both algorithms. We found that when agent ordering is controlled for, AdoptMVA has a lower CBR than Adopt (see Figure 1a). The results show that AdoptMVA would perform better than Adopt in systems with a high communication cost ($L=1000$).

Next, we compare four agent ordering heuristics on MAPs. Figure 1b shows our results. AdoptToMVA-Min was the best performing heuristic on the problems tested. We also saw that the agent ordering can make a tremendous difference. For some of the problems in Figure 1b, the difference between two heuristics is one or two orders of magnitude in size.

6. REFERENCES

- [1] D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.
- [2] J. Davin. Algorithmic and domain centralization in distributed constraint optimization problems. Master's thesis, Carnegie Mellon University, 2005.
- [3] J. Davin and P. J. Modi. Impact of problem centralization in distributed constraint optimization algorithms. In *Proc of AAMAS*, 2005.
- [4] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artif. Intell.*, 58(1-3):21–70, 1992.
- [5] R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham. Taking DCOP to the real world. In *Proc of AAMAS*, 2004.
- [6] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence Journal*, 2005.
- [7] P. J. Modi and M. Veloso. Bumping strategies for the multiagent agreement problem. In *Proc of AAMAS*, 2005.