Preprocessing Techniques for Accelerating the DCOP Algorithm ADOPT*

Syed Ali Sven Koenig Milind Tambe
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781
{syedmuha,skoenig,tambe}@usc.edu

ABSTRACT

Methods for solving Distributed Constraint Optimization Problems (DCOP) have emerged as key techniques for distributed reasoning. Yet, their application faces significant hurdles in many multiagent domains due to their inefficiency. Preprocessing techniques have successfully been used to speed up algorithms for centralized constraint satisfaction problems. This paper introduces a framework of different preprocessing techniques that are based on dynamic programming and speed up ADOPT, an asynchronous complete and optimal DCOP algorithm. We investigate when preprocessing is useful and which factors influence the resulting speedups in two DCOP domains, namely graph coloring and distributed sensor networks. Our experimental results demonstrate that our preprocessing techniques are fast and can speed up ADOPT by an order of magnitude.

Categories and Subject Descriptors

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

General Terms

Algorithms

Keywords

Distributed Constraint Optimization

*We thank Rahul Iyer and Sumit Borar for their help in generating the experimental results reported in this paper. This research was supported in part by a subcontract from NASA's Jet Propulsion Laboratory (JPL) and an NSF award under contract IIS-0350584. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations or the U.S. government.

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

1. INTRODUCTION

Methods for solving Distributed Constraint Optimization Problems (DCOP) [13, 10] have emerged as key techniques for distributed reasoning in multiagent domains, given their ability to optimize over a set of distributed constraints. For example, DCOPs are able to model the task of scheduling meetings in large organizations, where privacy needs make centralized constraint optimization difficult [9]. DCOPs are also able to model the task of allocating sensor nodes to targets in sensor networks [8, 13, 15], where the limited communication and computation power of individual sensor nodes makes centralized constraint optimization difficult. Finally, DCOPs are able to model the task of coordinating teams of unmanned air vehicles [17], where the need for rapid local responses makes centralized constraint optimization difficult.

Unfortunately, the application of DCOP algorithms faces significant hurdles in many multiagent domains due to their inefficiency. Solving DCOPs optimally is known to be NP-hard, yet one often needs to find optimal DCOP solutions quickly. In this context, researchers have recently developed ADOPT, an asynchronous complete and optimal DCOP algorithm that has been compared against competing DCOP algorithms [3] and shown to significantly outperform complete and optimal DCOP algorithms that do not allow partial or complete centralization of value assignments [13]. In this paper, we introduce a framework of preprocessing techniques that make ADOPT even more efficient. We focus on ADOPT since it provides an efficient baseline and has been used to solve DCOPs in domains where one needs to find optimal DCOP solutions quickly, namely sensor networks [15] and meeting scheduling for teams of personal assistant agents [9].

Preprocessing techniques have been studied before in the context of CSPs. For example, arc-consistency, path-consistency and general k-consistency algorithms can speed up CSP algorithms dramatically [4]. Recent work has applied similar preprocessing techniques to both distributed CSPs [7, 18] and centralized COPs [1, 16]. However, preprocessing techniques have not yet been investigated in the context of DCOPs, which is not surprising since efficient complete and optimal DCOP algorithms have been developed only recently. In this paper, we close this gap. The purpose of our preprocessing techniques is to focus the subsequent search rather than reduce its search space. ADOPT is an uninformed search method and our preprocessing techniques speed it up by supplying it with heuristic values that focus its search, an idea that has successfully been used to

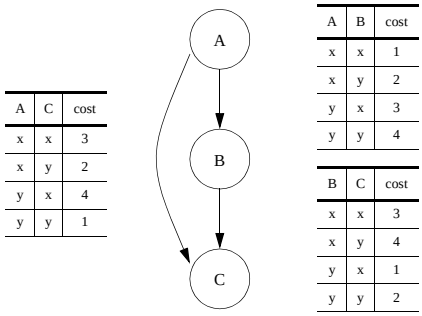

Figure 1: Example DCOP

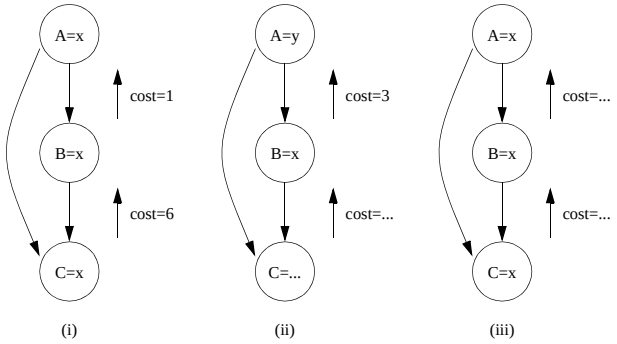

Figure 2: Possible Execution Trace of ADOPT

speed up centralized branch-and-bound search methods [19]. Our framework consists of a preprocessing phase followed by the main phase which just runs ADOPT. The preprocessing phase solves a relaxed version of the DCOP to calculate the heuristic values, using either ADOPT itself or specialized preprocessing techniques. We show how one can systematically construct preprocessing techniques of polynomial runtime, some of which are more computation or communication intensive than others and thus tend to calculate more informed heuristic values, thus trading off effort in the preprocessing phase and main phase. We investigate when preprocessing decreases the total effort and which factors influence the resulting speedups in two DCOP domains, namely graph coloring and distributed sensor networks. Our experimental results are very encouraging. For example, our new versions of ADOPT can solve a distributed graph coloring problem with 12 nodes about 10 times faster than ADOPT.

2. DCOP

A DCOP consists of a set of nodes (= agents) N. $D(n)$ denotes the set of possible values of node $n \in N$. $c(d(n), d(n'))$ denotes the cost of a soft binary constraint between nodes $n \in N$ and $n' \in N$ if node n is assigned value $d(n) \in D(n)$ and node n' is assigned value $d(n') \in D(n')$. The objective is to assign a value to every node so that the sum of the costs of the constraints is minimal.

Figure 1 shows an example DCOP with three nodes (A, B and C). All nodes can be assigned either the value x or the value y. There are constraints between A and B, B and C, and A and C. The DCOP has two cost-minimal solutions, namely (A=x, B=y, C=x) and (A=x, B=y, C=y).

3. ADOPT

ADOPT is an asynchronous complete and optimal DCOP algorithm that significantly outperforms competing complete and optimal DCOP algorithms that do not allow partial or complete centralization of value assignments [13, 14]. It was the first optimal DCOP algorithm that used only localized asynchronous communication and polynomial space for each node. Communication is local in the sense that a node does not send messages to every other node. Rather, ADOPT constructs a constraint tree, which is a tree of nodes with the property that any two nodes that are involved in some constraint are in an ancestor-successor (but not necessarily parent-child) relationship in the tree. For instance, the DCOP in Figure 1 is organized as a tree where A is the root, B is the child of A, and C is the child of B. In this case, the constraint tree is a chain since every node has at most one child. ADOPT searches the constraint tree in a way that resembles uninformed and memory-bounded versions of A*, except that it does so in a distributed way where every node sends messages only to its parent or successors in the constraint tree: Each node asynchronously executes a processing loop in which it waits for incoming messages, processes them and sends outgoing messages. VALUE messages are sent from a node to its successors in the constraint tree, informing them of the values of their ancestors. The successors then record these values in a "current context." In response to VALUE messages, nodes send COST messages to their parents to provide them with feedback about the costs of the best complete assignment of values to nodes that is consistent with the current context of the node. To this end, a node adds the exact costs of all constraints that involve nodes with known values (= its ancestors) and a lower bound cost estimate of the smallest sum of the costs of all constraints in the subtree rooted at the node (received from its children via COST messages) for its current context. Thus, COST messages contain estimates of the cost of the constraints for the best complete assignment of values to nodes that is both consistent with the current context of the node and a lower bound on the actual cost. Nodes initially use zero as cost estimates, and update these cost estimates when they receive COST messages from their children. Nodes reset their cost estimates to zero when their current context changes.

Figure 2 illustrates the execution of ADOPT for the DCOP from Figure 1, with an emphasis on aspects that illustrate the benefits of our modifications of ADOPT. The figure shows three snapshots in the progression of a possible execution path of ADOPT. Initially, the cost estimate of choosing value x and the one of choosing value y are zero for every node, and either value can thus be chosen. In Figure 2(i), nodes A, B and C initially each choose value x. Node A now sends VALUE messages to inform its successors B and C about its choice of value x, and node B sends a VALUE message to inform its successor C about its choice of value x, as indicated by the downward arrows. The current context of node B now records that node A has chosen value x, and computes its cost estimate for the best complete assignment of values to nodes that is consistent with node A having chosen value x. This cost estimate is one: If node B chooses value x (value y) then the constraint cost between

nodes A and B is one (two, respectively), and the constraint cost between nodes that involve node C is estimated to be zero since node B has not yet received a COST message from node C. Thus, node B sends a COST message to inform node A of an estimated cost of one. The cost estimate of choosing value x is now one for node A while the cost estimate of choosing value y is still zero. In Figure 2(ii), node A now chooses value y (the value with the smallest cost estimate) and sends VALUE messages to inform its successors B and C about its choice of value y. Node B then sends a COST message to inform node A of an estimated cost of three. The cost estimate of choosing value x is now one for node A while the cost estimate of choosing value y is three. Thus, in Figure 2(iii), node A now switches back to value x and thus backtracks in its search space, and the execution of ADOPT continues. ADOPT is described in detail in [14], including some optimizations that are not relevant to this paper and that we did not describe here. Our key point is that node A switched its value from x to y and back to x based on the cost estimates of its values. While such context switching is appropriate to avoid blocking in an asynchronous execution environment, it causes successors to reconstruct their solution, and thus we could potentially improve the performance of ADOPT if we were able to reduce such context switching by supplying it with better cost estimates. For example, if the cost estimate of choosing value y had been three for node A, then one would have avoided the context switch in Figure 2(ii).

Our new versions of ADOPT are motivated by the need to avoid or reduce such unnecessary context switches. These new versions of ADOPT are identical to ADOPT except that they initialize ADOPT with non-zero cost estimates, called heuristic values. They solve DCOPs optimally if we guarantee that the heuristic values are indeed lower bound cost estimates, which is the case since they use preprocessing techniques that calculate heuristic values by solving a relaxed version of the DCOP (= the DCOP with some constraints deleted or their costs reduced) in a preprocessing phase before they run ADOPT in the main phase. The main question of this paper then is whether the total runtime of the new versions of ADOPT is smaller than the one of ADOPT itself. The answer is not obvious since it takes time to compute the heuristic values. It is known that running an uninformed version of A* on a relaxed version of a search problem to obtain heuristic values that are then used to focus the search of an informed version of A* on the original version of the search problem cannot result in smaller total runtimes than just using the uninformed version of A* on the original version of the search problem [5]. However, the scheme may potentially work for ADOPT because ADOPT does not resemble A* but memory-bounded versions of A*.

4. PREPROCESSING FRAMEWORK

The heuristic values can be calculated by using either ADOPT on a relaxed version of the given DCOP or specialized preprocessing techniques on the given DCOP directly. In the following, we describe three preprocessing techniques (DP0, DP1 and DP2) that trade-off between how long it takes to calculate the heuristic values and how informed they are. We use the following additional notation to describe them formally: $C(n) \in N$ denotes the set of children of node $n \in N$. $A(n)$ denotes the set of those ancestors of

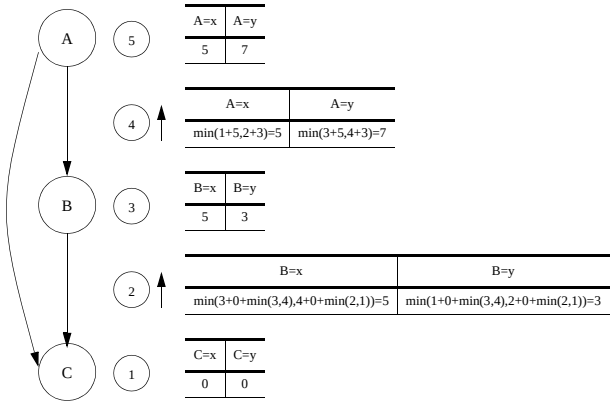

Figure 3: DP2 Example

node $n \in N$ with which the node has constraints. Finally, the heuristic value $h(d(n))$ is a lower bound cost estimate of the smallest sum of the costs of the constraints between two nodes, at least one of which is a successor of node $n \in N$ in the constraint tree if node n is assigned value $d(n) \in D(n)$.

DP0, DP1 and DP2 are dynamic programming algorithms that assign heuristic values to the nodes, starting at the leaves of the constraint tree and then proceeding from each node to its parent. They set the heuristic values of all leaves to zero, that is, they set $h(d(n)) := 0$ for all $d(n) \in D(n)$ and $n \in N$ with $C(n) = \emptyset$. Table 1 shows how they calculate the remaining heuristic values $h(d(n))$ for all $d(n) \in D(n)$ and $n \in N$ with $C(n) \neq \emptyset$. The heuristic values are lower bounds because some constraints do not get taken into account or their costs get underestimated (which is evident from the minimizations in the formulas).

It is straightforward to implement DP0, DP1 and DP2 in a decentralized way where nodes send messages to their parents. Basically, the leaves in the constraint tree calculate the heuristic values for each possible value of their parents and then send them in a message to their parents. All other nodes wait until they have received such messages from each of their children, then set the heuristic value of each of their possible values to the sum of the heuristic values reported by their children for this value, and then proceed in the same way as the leaves. For example, Figure 3 describes the operation of DP2 on the DCOP example from Figure 1. In Step 1 of the preprocessing phase, C initializes the heuristic values for its values x and y to 0. In Step 2, C calculates the heuristic values for the values x and y of B. The heuristic value for the value x of B is calculated as follows: If C is assigned the value x then it is cost-minimal to assume that A is assigned the value x. In this case, the cost of the constraint between A and C is 3 and the cost of the constraint between B and C is 3, resulting in an overall cost estimate of 6. On the other hand, if C is assigned the value y then it is cost-minimal to assume that A is assigned the value y as well. In this case, the cost of the constraint between A and C is 1 and the cost of the constraint between B and C is 4, resulting in an overall cost estimate of 5. The heuristic value for the value x of B is the minimum of the two cost estimates and thus 5. Similarly, C calculates the heuristic value for the value y of B. It then sends these heuristic values to B. In Step 3, B updates its heuristic values and, in Step 4,

DP0	$h(d(n)) := \sum_{n' \in C(n)} \sum_{n'' \in A(n')} \min_{d(n') \in D(n')} \min_{d(n'') \in D(n'')} c(d(n'), d(n''))$
DP1	$h(d(n)) := \sum_{n' \in C(n)} \min_{d(n') \in D(n')} (h(d(n')) + c(d(n'), d(n)))$
DP2	$h(d(n)) := \sum_{n' \in C(n)} (\min_{d(n') \in D(n')} (h(d(n')) + c(d(n'), d(n)) + \sum_{n'' \in A(n') \setminus \{n\}} \min_{d(n'') \in D(n'')} c(d(n'), d(n''))))$

Table 1: Calculation of DP0, DP1 and DP2

calculates the heuristic values of the values x and y of A. It then sends these heuristic values to A, and finally, in Step 5, A updates its heuristic values, which ends the preprocessing phase. In the main phase, node A initially chooses value x and switches to value y only when the cost estimate of choosing value x exceeds seven (= the initial cost estimate of choosing value y) which avoids the initial context switch in Figure 2(ii).

DP0, DP1 and DP2 can differ in both the heuristic values they calculate and in their computation and communication overhead. Each heuristic value of DP2 is guaranteed to be at least as large (= at least as informed) as the corresponding heuristic value of either DP0 or DP1. The following table contains the heuristic values for our example, where the last row contains the largest lower-bound cost estimates that satisfy our definition of the heuristic values:

	A=x	A=y	B=x	B=y	C=x	C=y
DP0	1	1	2	2	0	0
DP1	3	5	3	1	0	0
DP2	5	7	5	3	0	0
optimal	6	7	5	3	0	0

We can now examine the overhead of DP0, DP1 and DP2. Unfortunately, it is nontrivial to measure the runtime of the preprocessing techniques since nodes can operate in parallel but are often simulated in different threads on a single-processor machine. We follow other researchers and measure the runtime using cycles, where every node is allowed to process all of its messages in each cycle. However, cycles typically measure only the communication but not the computation overhead. While this is appropriate in those situations where the communication overhead dominates the computation overhead, we also investigate the computation overhead to ensure that it is not excessive.

- **Computation Overhead:** The computation overhead is affected by how many constraint costs a node must access. DP1 needs to access only the costs of the constraints that a node has with its parent while DP0 and DP2 also need to access the costs of the constraints that the node has with its other ancestors.

- **Communication Overhead:** The communication overhead is measured in cycles. DP0 needs only one cycle because it does not propagate heuristic values up the constraint tree while DP1 and DP2 need a number of cycles that equals the depth of the constraint tree (plus one). For example, Steps 1 and 2 constitute one cycle in the DP2 example from Figure 3, Steps 3 and 4 constitute another cycle, and Step 5 constitutes the third and final cycle. Another key difference between DP0 and the other two preprocessing techniques is that DP0 sends only one heuristic value from a node to its parent (because the heuristic values are identical for all possible values of the parent) while DP1 and DP2 send one heuristic value for each possible value of the parent (because they can be different). For example, every node sends two heuristic values to its parent in the DP2 example from Figure 3. However, since DP1 and DP2 communicate both of these heuristic values to the parent node at the same time, they can bundle them within a single message, and the number of messages per cycle then is the same for all three preprocessing techniques. While the number of messages, rather than their size, is a suitable measure of the communication overhead in many domains, a more detailed investigation on real hardware may be needed to fully understand the impact of the increase in the message size. In the following, we discuss the impact of such an increase in the message size, but our experimental results count all cycles the same, independent of the size of the messages sent.

Based on these two axes of computation and communication overhead, we identify two key design choices. They provide the rationale for our choices of DP0, DP1 and DP2. In the following, we always list the choice first that results in more informed heuristic values.

- **Property a (= Computation Overhead):** A preprocessing technique can either take all constraints into account (1) or only the constraints between nodes and their parents (2), in which case the constraints form a tree. (2) corresponds to relaxing the DCOP by deleting all constraints that are between any two nodes that are not in a parent-child relationship in the constraint tree, which is basically exactly what DP1 does. Instead of using DP1 on a given DCOP, one can therefore also use ADOPT itself on the relaxed DCOP to calculate similar heuristic values, which needs more cycles than DP1 but makes the preprocessing step easier to implement and might still result in substantial speedups. (We also experimented with other ways of deleting constraints. For example, randomly deleting a given percentage of constraints turned out not to be advantageous.)

- **Property b (= Communication Overhead):** A preprocessing technique can either take the heuristic values of a node into account (1) or ignore them (2) when calculating the heuristic values of the parent. (1) needs a number of cycles that equals the depth of the constraint tree (plus one) to propagate the heuristic values up the constraint tree, while (2) can be computed in only one cycle.

The following table categorizes DP0, DP1 and DP2 according to these two properties:

	Property a	Property b
DP0	(1)	(2)
DP1	(2)	(1)
DP2	(1)	(1)

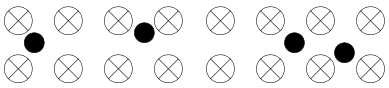

Figure 4: Sensor Network Chain

The following table shows the runtimes of DP0, DP1, and DP2 per cycle as a function of the two properties, where $v = \max_{n \in N} |D(n)|$ is the largest cardinality of the set of possible values of any node, $k = \max_{n \in N} |A(n)|$ is the largest cardinality of the set of those ancestors with which any node has constraints, c denotes the runtime of the preprocessing technique measured in cycles, and m denotes the size of its messages:

	Preprocessing Cost per Cycle				
		low (c=1, m=1)		high (c=tree depth, m=v)	
Graph Structure	tree		$O(v^2)$	DP1	$O(v^2)$
	full graph	DP0	$O(kv^2)$	DP2	$O(kv^2)$

There are v^2 constraint costs for each constraint. Each preprocessing technique might have to process all v^2 constraint costs for each of the at most k ancestors of a node with which it has constraints. If the constraints form a tree (upper row of the table), then the number of ancestors is one ($k = 1$). When the constraints do not form a tree (lower row of the table), each node must examine its input and thus v^2 constraint costs for each of its k ancestors, and thus the kv^2 cost is mandatory for both DP0 and DP2. The cost for DP2 needs further explanation since given a node n, it iterates over all k ancestors of all children of node n, and would thus appear to require an additional cost of iterating over all such children. However, in a decentralized implementation, each child node only computes the heuristic values relevant to itself and sends the values to the parent node n, which sums the inputs from the children. Thus, each child incurs the cost of kv^2 per cycle. This explains the table. Since the runtimes of DP0, DP1, and DP2 are polynomial per cycle and their number of cycles is polynomial as well, their runtimes are polynomial. This means that their runtimes are small in the worst case compared to the runtime of the main phase since solving DCOPs is NP-hard.

5. EXPERIMENTAL RESULTS

It is not immediately obvious whether the runtime of the preprocessing techniques is sufficiently overcome by the speedups achieved in the main phase and, if so, which preprocessing technique results in the smallest total runtime, that is, sum of the runtimes of the preprocessing phase and main phase. We conducted experiments in two different DCOP domains to answer these questions:

- **Graph Coloring:** Our first domain is a three-coloring problem with a link density (= number of constraints over the number of nodes) of two, which are relatively easy to solve according to [2]. The values of the nodes correspond to the colors, and all constraint costs are drawn with uniform probability from the integers between 1 and 100.
- **Distributed Sensor Network (DSN):** Our second domain is a distributed sensor network problem where 54 sensors, arranged in a chain, have to track 10, 15, 20 or 25 targets that are randomly positioned between four sensors each [12]. At most one target is between any four sensors. Figure 4 shows an example, where hollow circles with Xs represent sensors and solid circles without Xs represent targets. Each sensor can track at most one target, which needs to be in its immediate vicinity. Each target is either tracked by exactly three sensors or one incurs a cost that is drawn with uniform probability from the integers between 60 and 660. Details are given in [12, 14]. Basically, one creates the nodes TA1, TB1, TC1, and TD1 if the sensors A, B, C, and D are able to track target 1. Thus, there is one node for each combination of a sensor and one of its possible targets. (The total number of nodes is thus equal to four times the number of targets.) The possible values of these three nodes are all combinations of three sensors that are able to track the target (ABC, ABD, ACD, BCD), and the value IGNORE that represents that no sensor will track the target. There are equality constraints between any two nodes with the same target. For example, there is an equality constraint between TA1 and TB1 that requires sensors A and B to agree on the set of sensors that track target 1. Similarly, there are mutual exclusion constraints between any two nodes with the same sensor. For example, there is a mutual exclusion constraint that enforces that sensor A cannot track targets 1 and 2 at the same time. The costs are zero if these mutual exclusion constraints are satisfied and very high (= 1,000,000) if they are not satisfied, making them hard constraints. If a node is assigned the value IGNORE, then it incurs a cost for ignoring that target.

The following table gives details on the number of nodes and the number of their possible values for the two DCOP domains. We varied the sizes of the domains by varying the number of their nodes. We report averages over 15 problem instances for each domain and size. The code for our experiments and the results for the different sizes of the two domains can be found at `teamcore.usc.edu/preAdopt.htm`:

Domain	Nodes	Values per Node
Graph Coloring	9, 10, 11, 12	3
DSN	40, 60, 80, 100	5

5.1 Discussion of Cycle Count

In the following, we refer to ADOPT0, ADOPT1 and ADOPT2 as the combination of DP0, DP1 and DP2, respectively, in the preprocessing phase and ADOPT in the main phase. Figure 5 shows the total number of cycles of ADOPT and the three new versions of ADOPT as a function of the number of nodes (= agents). ADOPT2 outperforms all other versions of ADOPT in graph coloring and its speedups increase with the number of nodes. For example, ADOPT2 speeds up ADOPT by a factor of 10.59 in graph coloring with 12 nodes. ADOPT0 does not speed up ADOPT in DSN, and ADOPT1 and ADOPT2 speed it up by the same amount. ADOPT2 speeds up ADOPT by a factor of 2.79 in DSN with 100 nodes. To summarize, ADOPT2 has the smallest total number of cycles in graph coloring. Both ADOPT1 and ADOPT2 have the smallest

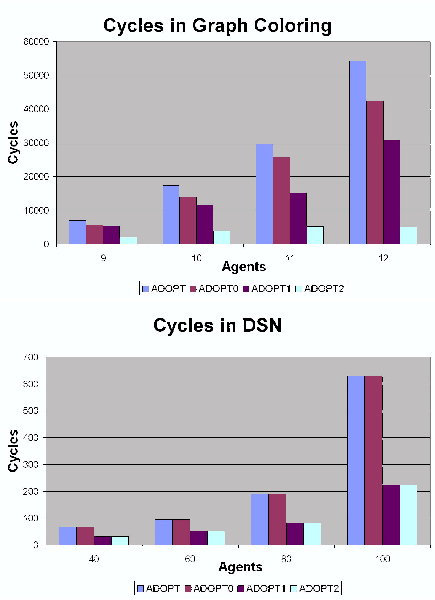

Figure 5: Cycles

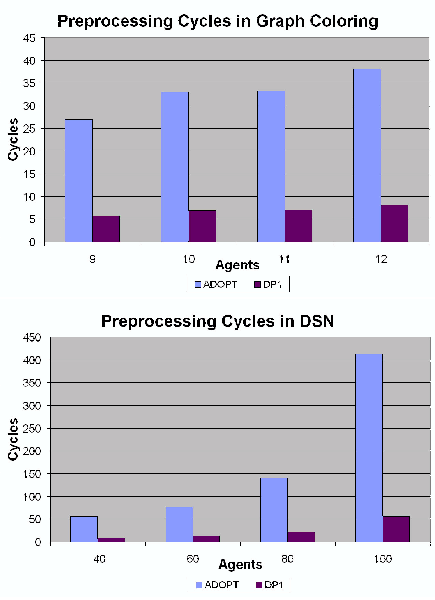

Figure 6: Preprocessing Cycles

total number of cycles in DSN, which means that ADOPT1 should be preferred over ADOPT2 in this domain since the computation overhead of DP1 is smaller than the one of DP2. On the other hand, ADOPT0 is not the method of choice in either domain despite the small computation and communication overhead of DP0 over DP1 and DP2.

Recall that one can use both DP1 on a given DCOP or ADOPT on a relaxed version of the DCOP to calculate the similar heuristic values in the preprocessing phase. Thus, the number of cycles in the main phase will be similar in both cases and one should choose the preprocessing technique that results in the smallest number of cycles in the preprocessing phase. Figure 6 shows that the number of cycles of DP1 in the preprocessing phase is smaller than the

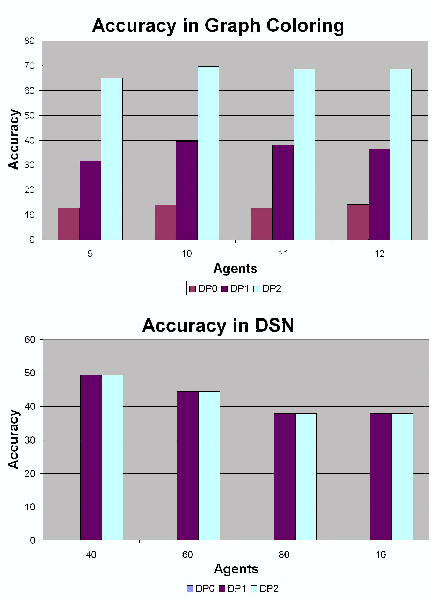

Figure 7: Accuracy

one of ADOPT by a factor of 4.68 in graph coloring with 12 nodes and by a factor of 7.35 in DSN with 100 nodes. To summarize, there is an advantage to using specialized preprocessing techniques in the preprocessing phase rather than the more general ADOPT itself.

5.2 Discussion of Accuracy

To understand better why the speedups depend on the preprocessing technique, recall that the heuristic values computed by the preprocessing techniques are used to seed the cost estimates of ADOPT in the main phase. ADOPT can raise these cost estimates during its operation. We therefore computed the ratio of each cost estimate computed by the preprocessing technique and the one after the termination of ADOPT, averaged over all cost estimates that are still available when ADOPT terminates (except for the cost estimates in leaf nodes of the constraint graph, which are zero). We refer to this ratio as the accuracy. The larger the accuracy, the more informed the heuristic values are. An accuracy of 0 percent means that the heuristic values are no more informed than the initial cost estimates of ADOPT itself. In this case, the preprocessing technique does not speed up ADOPT. On the other hand, an accuracy of 100 percent means that the heuristic values computed by the preprocessing technique were so good that ADOPT was not able to raise them. Figure 7 shows the accuracies of DP0, DP1 and DP2. The accuracy of DP0 is 14.4 percent, the accuracy of DP1 is 35.8 percent, and the accuracy of DP2 is 69.0 percent in graph coloring with 12 nodes. On the other hand, the accuracy of DP0 is zero percent (and hence the bar does not appear in the figure) in DSN with 100 nodes since the heuristic values calculated by DP0 are all zero. This is so since every constraint has at least one constraint cost that is zero. Thus, ADOPT and the main phase of ADOPT0 are equally fast in this case. The accuracies of DP1 and DP2 are larger than zero percent but, for a similar reason, identical at 37.9 percent. Thus, the main phases of ADOPT1 and ADOPT2 are equally fast in this case. Figure 7 shows

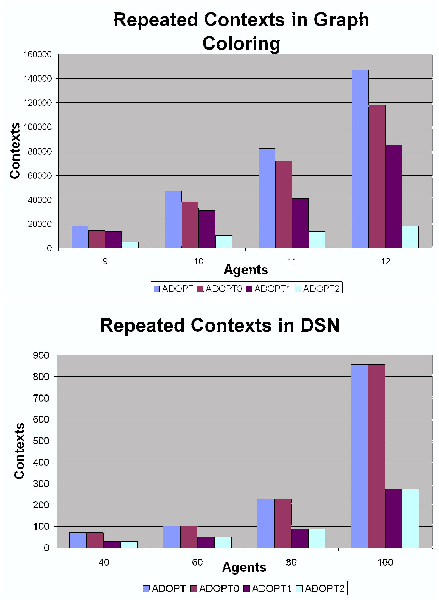

Figure 8: Repeated Contexts

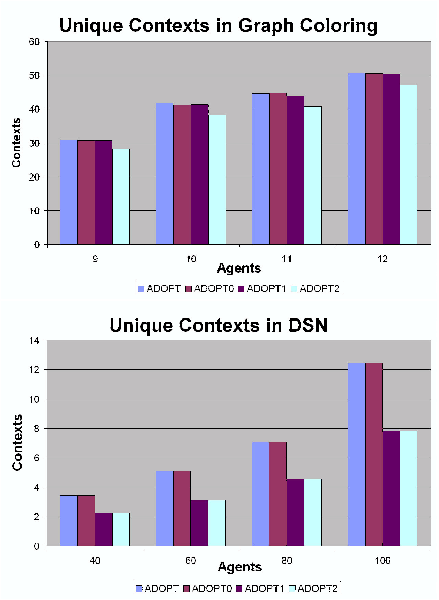

Figure 9: Unique Contexts

that the total number of cycles from Figure 5 are closely correlated with the accuracies of the heuristic values. The more accurate the heuristic values, the more they speed up ADOPT on the same DCOP.

To examine this relationship further in graph coloring with 11 nodes, we first ran ADOPT without preprocessing and obtained the cost estimates after its termination. We then ran ADOPT again but now simulated a preprocessing phase that produces heuristic values that are equal to the product of the corresponding cost estimates after the termination of ADOPT and the same constant factor (no larger than one), which represents the desired accuracy of the heuristic values. Figure 10 shows that the total number of cycles is closely correlated with the factors. Similar to the previous experiment, the larger the factors and thus the more accurate the heuristic values, the more they speed up ADOPT.

5.3 Discussion of Number of Contexts

There are two reasons why the informedness of the heuristic values can have a large effect on the resulting speedups. We explore both reasons, illustrating that the speedups are a combination of both reasons. For the explanation, we use the number of unique and regenerated (= repeated) current contexts at each node. Whenever a node receives a VALUE message from one of its ancestors, it calculates its new current context. If it has not seen this current context before, it counts the current context as a unique current context, otherwise it counts the current context as a regenerated current context:

- The first reason why the informedness of the heuristic values can have a large effect on the resulting speedups is that ADOPT, as a memory-bounded DCOP algorithm, has to regenerate partial solutions (in the form of current contexts) when it backtracks to a previously explored part of the search space. More informed heuristic values reduce the amount of backtracking of ADOPT and thus the number of regenerated current contexts, resulting in a smaller number of cycles in the main phase. To verify our hypothesis, we measured the number of regenerated current contexts at each node, averaged over all nodes in the constraint graph. Figure 8 shows that the number of regenerated current contexts is indeed closely correlated with the total number of cycles from Figure 5 and the accuracies from Figure 7. The more accurate the heuristic values, the fewer current contexts are repeated in the main phase, and the more the heuristic values speed up ADOPT.

- The second reason why the informedness of the heuristic values can have a large effect on the resulting speedups is that more informed heuristic values reduce the part of the search space explored by ADOPT and thus also the number of unique current contexts, resulting in a smaller number of cycles in the main phase. To verify our hypothesis, we measured the number of unique current contexts at each node, averaged over all nodes in the constraint graph. Figure 9 shows that the number of unique current contexts, surprisingly, changes very little in graph coloring. The number of unique current contexts decreases with the accuracy of the heuristic values in DSN. The more accurate the heuristic values in this case, the fewer unique current contexts are generated in the main phase, and the more the heuristic values speed up ADOPT.

5.4 Related Work

Precursor work to the systematic investigation of preprocessing in this paper appeared in [9], where a single preprocessing technique was reported, namely a hybrid of our DP0 and DP1 techniques. Figure 11 shows that the total number of cycles of ADOPT2 is smaller than the one of this precursor technique by a factor of 5.7 in graph coloring with 12 nodes.

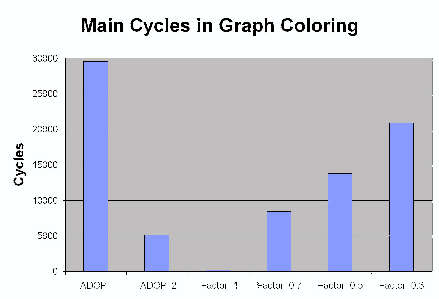

Figure 10: Impact of Accuracy in Graph Coloring

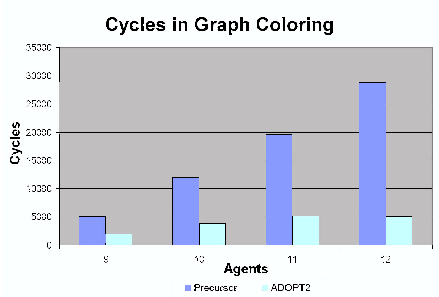

Figure 11: Previous Speedups in Graph Coloring

6. CONCLUSIONS

In this paper, we developed a framework of preprocessing techniques that speed up ADOPT, an asynchronous complete and optimal DCOP algorithm. Our preprocessing techniques use dynamic programming to calculate informed lower bound cost estimates for ADOPT. Our empirical results in two DCOP domains, namely graph coloring and distributed sensor networks, demonstrated that our preprocessing techniques are fast and can speed up ADOPT by an order of magnitude, at a relatively low preprocessing cost. We showed that the key reason for the speedup is the informedness of the heuristic values, which in turn determines how many partial solutions ADOPT generates and how many of these it revisits. The results also demonstrated that the preprocessing techniques are significantly more efficient than using ADOPT itself in the preprocessing phase. As outlined in [13], it is essential to use lower bound cost estimates in DCOP algorithms. Since our preprocessing techniques focus on computing such lower bound cost estimates, the ideas behind them might also apply to DCOP algorithms other than ADOPT such as Synchronous Branch and Bound [6], Synchronous Iterative Deepening [14] and Optimal Asynchronous Partial Overlap [11]. It is future work to explore their applicability to these and other DCOP algorithms as well as to develop even more sophisticated preprocessing techniques.

7. REFERENCES

[1] S. Bistarelli, R. Gennari, and F. Rossi. Constraint propagation for soft constraints: generalization and termination conditions. In *CP*, pages 83–97, 2000.

[2] J. Culberson and I. Gent. Frozen development in graph coloring. *Theoretical Computer Science*, 265(1–2):227–264, 2001.

[3] J. Davin and P. Modi. Impact of problem centralization in distributed constraint optimization algorithms. In *AAMAS*, 2005.

[4] R. Dechter and I. Meiri. Experimental evaluation of preprocessing techniques in constraint satisfaction problems. In *IJCAI*, pages 271–277, 1989.

[5] O. Hansson, A. Mayer, and M. Valtorta. A new result on the complexity of heuristic estimates for the A* algorithm. *Artificial Intelligence*, 55(1):129–143, 1992.

[6] K. Hirayama and M. Yokoo. Distributed partial constraint satisfaction problem. In *CP*, pages 222–236, 1997.

[7] H. Jung and M. Tambe. Performance models for large scale multiagent systems using POMDP building blocks. In *AAMAS*, pages 297–304, 2003.

[8] V. Lesser, C. Ortiz, and M. Tambe, editors. *Distributed sensor networks: A multiagent perspective*. Kluwer, 2003.

[9] R. Maheswaran, M. Tambe, E. Bowring, J. Pearce, and P. Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed event scheduling. In *AAMAS*, pages 310–317, 2004.

[10] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS*, pages 438–445, 2004.

[11] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS*, pages 438–446, 2004.

[12] P. Modi, H. Jung, M. Tambe, W. Shen, and S. Kulkarni. A dynamic distributed constraint satisfaction approach to resource allocation. In *CP*, pages 685–700, 2001.

[13] P. Modi, W. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *AAMAS*, pages 161–168, 2003.

[14] P. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161:149–180, 2005.

[15] P. Scerri, J. Modi, M. Tambe, and W. Shen. Are multiagent algorithms relevant for real hardware? A case study of distributed constraint algorithms. In *ACM Symposium on Applied Computing*, pages 38–44, 2003.

[16] T. Schiex. Arc consistency for soft constraints. In *CP*, pages 411–424, 2000.

[17] N. Schurr, S. Okamoto, R. Maheswaran, P. Scerri, and M. Tambe. Evolution of a teamwork model. In R. Sun, editor, *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation*, page (to appear). Cambridge University Press, 2005.

[18] M. Silaghi, D. Sam-Haroud, and B. Faltings. Consistency maintenance for ABT. In *CP*, pages 271–285, 2001.

[19] R. Wallace. Enhancements of branch and bound methods for the maximal constraint satisfaction problem. In *AAAI*, pages 188–195, 1996.