Preprocessing Techniques for Distributed Constraint Optimization

Syed Muhammad Ali*, Sven Koenig, and Milind Tambe

USC, CS Department, 941 W 37th Street, Los Angeles, CA 90089-0781, USA
{syedmuha,skoenig,tambe}@usc.edu

Abstract. Although algorithms for Distributed Constraint Optimization Problems (DCOPs) have emerged as a key technique for distributed reasoning, their application faces significant hurdles in many multiagent domains due to their inefficiency. Preprocessing techniques have been successfully used to speed up algorithms for centralized constraint satisfaction problems. This paper introduces a framework of very different preprocessing techniques that speed up ADOPT, an asynchronous optimal DCOP algorithm that significantly outperforms competing DCOP algorithms by more than one order of magnitude.

1 Introduction

Algorithms for Distributed Constraint Optimization Problems (DCOPs) [1, 2] have emerged as a key technique for distributed reasoning, given their ability to optimize over a set of distributed constraints. For example, DCOP algorithms have been used in distributed sensor networks [3] to optimize the allocation of sensors to targets so as to maximize the value of the tracked targets or the area covered by the sensors [3, 1]. Solving DCOPs optimally is known to be NP-hard, yet one often needs to develop DCOP algorithms that provide optimal solutions as efficiently as possible. For example, researchers have recently developed ADOPT, an asynchronous optimal DCOP algorithm that significantly outperforms competing optimal DCOP algorithms (that do not allow partial or complete centralization of value assignments) [1]. This paper introduces a framework of preprocessing techniques that make algorithms like ADOPT even more efficient. The idea of preprocessing is not new in the context of CSPs, where arc-consistency, path-consistency and general k-consistency algorithms can speed up CSP algorithms dramatically [4, 5]. However, preprocessing algorithms have not yet been investigated in the context of DCOPs. In this paper, we close this gap with preprocessing algorithms that are very different from preprocessing algorithms for CSPs. Our preprocessing algorithms demonstrate for the first time that preprocessing can indeed speed up DCOP algorithms. In fact, they can speed up ADOPT by more than one order of magnitude.

* This research was supported in part by a subcontract from NASA's Jet Propulsion Laboratory (JPL) and in part by 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.

2 DCOPs and ADOPT

DCOPs consist of a set of agents N. $D(n)$ denotes the set of possible values that agent $n \in N$ can assign to itself. The other agents cannot directly read this value, but the agent can communicate the value to its neighbors. $c(d(n), d(n'))$ denotes the cost of a soft binary constraint between agents $n \in N$ and $n' \in N$ if agent n is assigned value $d(n) \in D(n)$ and agent n' is assigned value $d(n') \in D(n')$. We refer to these cost functions simply as constraints. The objective is to assign a value to every agent so that the sum of the costs of the constraints is minimal.

In this paper, we build on ADOPT, the first optimal DCOP algorithm that uses only localized asynchronous communication and polynomial space for each agent [1]. ADOPT constructs a constraint tree, which is a tree of agents with the property that any two agents that are involved in some constraint are in a predecessor-successor (but not necessarily parent-child) relationship in the tree. ADOPT searches the constraint tree in a way that resembles uninformed and memory-bounded versions of A*, but does so in a distributed fashion, where every agent sends messages only to its parent or successors in the constraint tree. ADOPT begins by all agents choosing their values concurrently. Agents can send their values to those successors they are neighbors with. When an agent receives a value message, it computes and sends a cost message to its parent in the constraint tree. This cost message is an estimate of the total cost of the constraints for the best complete assignment of values to the message-sending-agent's successors (and to the agent itself) that is: (i) guaranteed to be a lower bound on the real cost, and (ii) guaranteed to be consistent with the current assignment of values to its predecessors. The agent calculates this lower bound estimate by adding the exact costs of all constraints that involve agents with known values (namely its predecessors with whom it has constraints) and a lower bound estimate of the smallest sum of the costs of all constraints in the subtree rooted at the agent (received from its children via cost messages). Initially, all agents use zero as lower bound estimates. The agents increase these lower bound estimates as they receive cost messages from their children. The agents set them back to zero whenever they receive a value message that indicates that one of their predecessors has changed its value.

3 Preprocessing Framework

Our preprocessing framework consists of a preprocessing phase followed by the main phase which just runs ADOPT. The preprocessing phase supplies ADOPT with non-zero lower bound estimates to focus its search. They are calculated by solving a relaxed version of the DCOP, which is why we refer to them in the following as heuristic values. The heuristic values can be calculated by using either ADOPT itself to solve the relaxed DCOP or specialized preprocessing algorithms. Our specialized preprocessing algorithms DP0, DP1 and DP2 are dynamic programming algorithms that assign heuristic values to the agents, starting at the leaves of the constraint tree and then proceeding from each agent to its parent. We use the following additional notation to describe them formally: $C(n) \in N$ denotes the set of children of agent $n \in N$. $A(n)$ denotes the set of predecessors of agent $n \in N$ with which the agent has constraints. Finally, the heuristic value $h(d(n))$ is a lower bound estimate of the smallest sum of the costs of all constraints in the subtree rooted at the agent $n \in N$, provided that agent n is assigned the value $d(n) \in D(n)$. DP0, DP1 and DP2 set $h(d(n)) := 0$ for all $d(n) \in D(n)$ and $n \in N$ with $C(n) = \emptyset$, that is, the heuristic values of all leaves to zero. They calculate the remaining heuristic values $h(d(n))$ for all $d(n) \in D(n)$ and $n \in N$ with $C(n) = \emptyset$ as follows:

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

DP0, DP1 and DP2 calculate different heuristic values and differ in their computation and communication overhead. Each heuristic value of DP2 is guaranteed to be at least as large as the corresponding heuristic value of both DP0 and DP1. Thus, the heuristic values of DP2 are at least as informed as the ones of DP0 and DP1.

4 Experimental Results

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. It is obvious that these versions solve DCOPs optimally since they use lower bound estimates as heuristic values. Since the processing times of DP0, DP1, and DP2 per iteration are polynomial and their number of iterations is polynomial as well, their runtimes are polynomial. Since solving DCOPs is NP-hard, the runtimes of the preprocessing algorithms do not contribute to the overall runtime in a major way. However, it is not immediately obvious whether the preprocessing algorithms are able to reduce the total runtime, that is, the sum of the runtimes of the preprocessing phase and the main phase. Hence, it is crucial to perform an experimental investigation of the different preprocessing algorithms.

Our test domains are three-coloring problems with a link density of two. The values of the agents correspond to the colors, and the costs of all pairs of values of any two neighboring agents are drawn from the integers from 1 to 100 with uniform probability. We follow other researchers and use cycles to measure the runtimes, where every agent is allowed to process all of its messages in each cycle. Figure 1 (left) shows the overall number of cycles of ADOPT and our three new versions of ADOPT as a function of the number of agents. When we report cycles, we penalize ADOPT1 and ADOPT2 for their larger messages in the preprocessing phase by increasing their cycle count in the preprocessing phase by a factor that equals the number of heuristic values they send per message, namely 3. ADOPT2 outperforms all other versions of ADOPT and its speedups increase with the number of agents. For example, ADOPT2 speeds up ADOPT by a factor of 9.8 for 12 agents.

To illustrate the advantage of our preprocessing algorithms, Figure 1 (right) compares the number of cycles taken by DP1 and the number of cycles that ADOPT would need if it were used in the preprocessing phase on the DCOP after all of its constraints that are not in a parent-child relationship in the constraint tree are deleted. These two

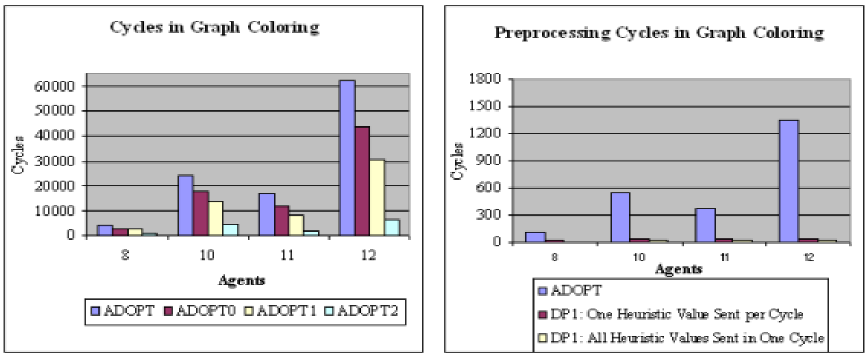

Fig. 1. Total Cycles (left) and Preprocessing Cycles (right)

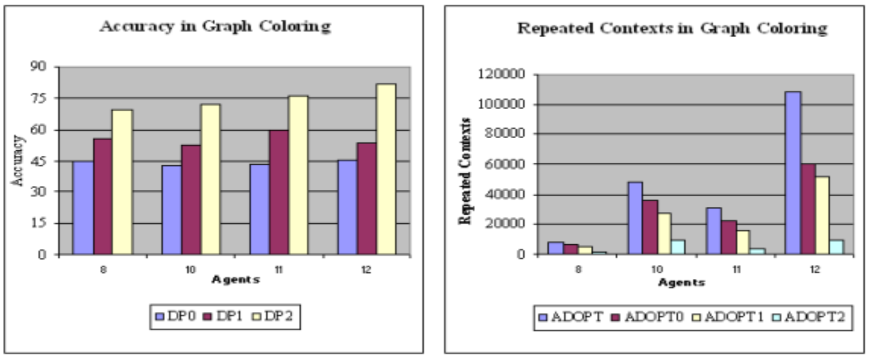

Fig. 2. Accuracy (left) and Regenerated Contexts (right)

preprocessing algorithms calculate the same heuristic values. However, the number of cycles of DP1 is smaller than the one of ADOPT by a factor of 52.5 for 12 agents. If we did not penalize DP1 for its larger messages by increasing its cycle count, its number of cycles would even be smaller than the one of ADOPT by a factor of 157.4.

To understand better why the speedups depend on the preprocessing algorithm, remember that the heuristic values computed by the preprocessing algorithms are used to seed lower bounds of ADOPT for the main phase. ADOPT can raise these lower bounds during its operation. We therefore computed the average ratio of the lower bounds provided by the preprocessing algorithms and the lower bounds after ADOPT terminated. We refer to this ratio as the accuracy. The larger the accuracy, the more informed the lower bounds are. An accuracy of 0 percent means that the lower bounds provided by the preprocessing phase were zero and thus no better than the lower bounds used by ADOPT itself. In this case, the preprocessing algorithm does not speed up ADOPT. On the other hand, an accuracy of 100 percent means that the lower bounds provided by the preprocessing algorithm were so good that ADOPT was not able to raise them. Figure 2 (left) shows the accuracies of DP0, DP1 and DP2. The accuracy of DP0 is 45.1 percent for 12 agents, the accuracy of DP1 is 53.4 percent, and the accuracy of DP2 is 81.6 percent. Figure 2 (left) shows that the accuracies are closely correlated with the number of cycles from Figure 1 (left). The overall number of cycles decreases as the accuracies and thus the informedness of the heuristic values increase.

ADOPT is a memory-bounded DCOP algorithm and thus has to regenerate partial solutions (contexts) when it backtracks to a previously explored part of the search space. We therefore measured the average number of regenerated contexts at each agent, where a context assigns a value to each predecessor of the agent in the constraint tree. Thus, a regenerated context is in effect a regenerated partial solution. Figure 2 (right) shows that the numbers of regenerated contexts are indeed closely correlated with the accuracies from Figure 2 (left) as well as the number of cycles from Figure 1 (left). It appears that more informed heuristic values reduce the amount of backtracking and thus the number of regenerated partial solutions, resulting in a smaller number of cycles. For example, ADOPT2 uses far more informed heuristic values than ADOPT and thus repeats far fewer contexts and, correspondingly, provides high speedups.

References

1. Modi, P., Shen, W., Tambe, M., Yokoo, M.: An asynchronous complete method for distributed constraint optimization. In: AAMAS. (2003) 161–168
2. Mailler, R., Lesser, V.: Solving distributed constraint optimization problems using cooperative mediation. In: AAMAS. (2004) (to appear)
3. Lesser, V., Ortiz, C., Tambe, M., eds.: Distributed sensor networks: A multiagent perspective. Kluwer (2003)
4. Dechter, R., Meiri, I.: Experimental evaluation of preprocessing techniques in constraint satisfaction problems. In: IJCAI. (1989) 271–277
5. Bistarelli, S., Gennari, R., Rossi, F.: Constraint propagation for soft constraints: generalization and termination conditions. In: CP. (2000) 83–97