Impact of Problem Centralization in Distributed Constraint Optimization Algorithms

John Davin and Pragnesh Jay Modi
Computer Science Department
Carnegie Mellon University
Pittsburgh PA 15213
{jdavin, pmodi}@cs.cmu.edu

Abstract

Recent progress in Distributed Constraint Optimization Problems (DCOP) has led to a range of algorithms now available which differ in their amount of problem centralization. Problem centralization can have a significant impact on the amount of computation required by an agent but unfortunately the dominant evaluation metric of "number of cycles" fails to account for this cost. We analyze the relative performance of two recent algorithms for DCOP: OptAPO, which performs partial centralization, and Adopt, which maintains distribution of the DCOP. Previous comparison of Adopt and OptAPO has found that OptAPO requires fewer cycles than Adopt. We extend the cycles metric to define "Cycle-Based Runtime (CBR)" to account for both the amount of computation required in each cycle and the communication latency between cycles. Using the CBR metric, we show that Adopt outperforms OptAPO under a range of communication latencies. We also ask: What level of centralization is most suitable for a given communication latency? We use CBR to create performance curves for three algorithms that vary in degree of centralization, namely Adopt, OptAPO, and centralized Branch and Bound search.

1. Introduction

The Distributed Constraint Optimization Problem (DCOP) is a general framework for distributed problem solving that has a wide range of applications in Multiagent Systems and has generated significant interest from researchers [5, 4, 2, 10, 6, 9]. A DCOP assumes that problem variables and constraints are distributed among a set of agents who must communicate to find an optimal assignment of values for the variables.

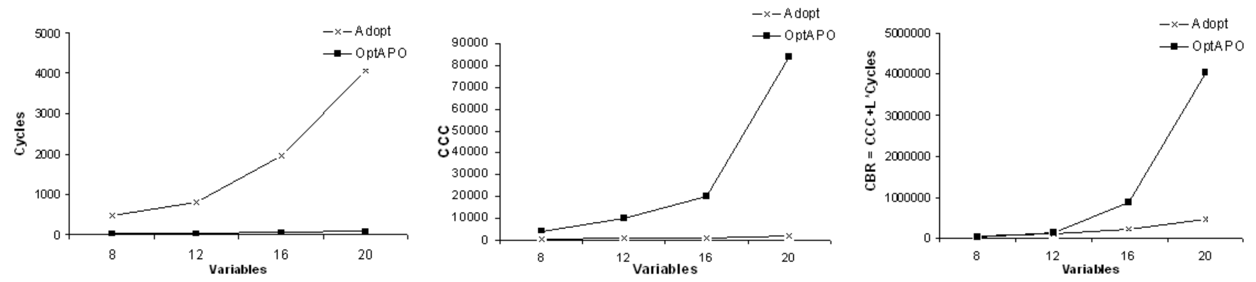
1 First author is a student.

Mailler and Lesser have recently proposed a complete, asynchronous algorithm for DCOP, named Optimal Asynchronous Partial Overlay (OptAPO) [6]. This algorithm uses a novel approach to DCOP in which variables and constraints are partially centralized during problem solving. A dynamically chosen agent who collects problem constraints is called a "mediator" and the general approach is termed *cooperative mediation*.

The cooperative mediation approach to DCOP is novel (in part) because it provides the first middle point on a spectrum that ranges from very centralized to more decentralized approaches to DCOP. In a very centralized approach, all agents communicate all their constraints to a single agent in the first step of the algorithm and a centralized optimization technique is applied, such as the classic Branch and Bound algorithm [3]. At the decentralized end of the spectrum, Modi et al. [9] have previously proposed an approach to DCOP, named ADOPT (Asynchronous Distributed OPTimization) that is more decentralized because agents do not explicitly communicate their constraints to others (although some indirect information about constraints can be leaked). OptAPO falls somewhere in the middle of this spectrum of centralization.

The degree of problem centralization, or equivalently, the amount of constraints that are communicated during an algorithm's execution, can have a significant impact on the amount of computation required at each agent. As an agent's subproblem grows and it has a greater number of constraints to process, it requires increased computational effort. Thus, in order to compare algorithms that fall on different points along our centralization spectrum, a metric that takes local computation effort into account is needed.

The dominant metric for evaluation of DCOP algorithms is number of synchronous *cycles* [11]. In cycle-based execution, all agents operate concurrently within a cycle, but do not move to the next cycle until all agents have completed their computations from the previous cycle. Any message sent in a cycle is not received until the next cycle. While not



(a) Number of Cycles (b) Concurrent Constraint Checks (c) Cycle-Based Runtime

Figure 1. (a) OptAPO requires fewer number of cycles than Adopt, as shown in previous research, (b) But requires an increased amount of computation as measured by constraint checks. (c) When both constraint checks and communication latency (with L=100) are accounted for, Adopt outperforms OptAPO.

perfect (see Meisels et. al. [8] and Brito et. al. [1] for a discussion), the cycles metric provides a convenient method to assess the performance of an asynchronous algorithm. True asynchronous execution of a DCOP algorithm is difficult to measure reliably because of the exponential number of possible execution paths that differ significantly in their runtimes. Thus for repeatable results, it is often most practical to execute a DCOP algorithm in a synchronous fashion on a single computer, and the "number of cycles" metric provides a useful way to measure this execution.

Although convenient, the cycles metric does not measure the amount of computation required by the algorithm, i.e., length of each cycle. As we have described, taking this factor into account is necessary when comparing algorithms that vary in their degree of centralization. To address this issue, we extend the cycles measurement to include the local computation costs of the algorithm. We use *concurrent constraint checks (ccc)* to measure the amount of computation within a cycle [8]. We define a new metric, called Cycle-Based Runtime (CBR), that takes into account two aspects of runtime – the computation time as measured by number of ccc and communication time as measured by the latency between cycles. The CBR metric still requires agents to execute in synchronous cycles, which we believe continues to be a useful method for analysis, but also incorporates computational cost, allowing us to more completely measure an algorithm's performance than with cycles alone.

We investigate two issues. First, using CBR, we compare the performance of algorithms that vary in their degree of centralization. Existing research [6] has found that OptAPO outperforms Adopt in terms of cycles. We reproduce those results. However, using CBR as a comparison, we show that Adopt performs better than OptAPO for a range of communication latencies. Second, because CBR takes into account communication latency, which is a property of the communication environment in which the algorithm operates, we can ask which algorithm is most appropriate for an environment with a given latency. We evaluate three algorithms on the spectrum of centralization: Adopt, OptAPO, and a fully centralized approach. By comparing all three algorithms using the CBR metric, we are able to provide a comparison of how differing levels of centralization perform under various communication latencies. This analysis is important because it provides assistance to researchers applying DCOP algorithms within new environments to determine the most appropriate level of centralization.

1.1. Key Result

We briefly summarize one of the key results of this paper. Previous comparisons of Adopt and OptAPO have used measurements of cycles to compare the algorithm performance. In our investigation of Adopt and OptAPO, which will be discussed later in this paper, we obtained cycle measurements in agreement with the existing research (Figure 1a). OptAPO solves graph 3-coloring problems in fewer cycles than Adopt. However, when constraint checks are measured to estimate the computational effort of the algorithms, we find that OptAPO uses more concurrent constraint checks than Adopt (Fig 1b). Using the CBR metric described in Section 3 which takes both constraint checks and communication latency into account, we see that Adopt performs better than OptAPO (Figure 1c). This graph shows results for a given communication latency, but as we will see our results hold across a range of latencies.

The rest of this paper provides background on Adopt and

OptAPO, explains the design and rationale of the methods we used to analyze these algorithms, and then presents analysis that shows results comparing Adopt with OptAPO.

2. Algorithms for DCOP

A Distributed Constraint Optimization Problem [9] (DCOP) is defined as:

- set of N *agents*, $A = \{A_1, A_2, \ldots, A_N\}$.
- set of n *variables*, $V = \{x_1,x_2, \ldots, x_n\}$.
- set of *domains* $D = \{D_1, D_2,\ldots, D_n\}$, where the value of x_i is taken from D_i. Each D_i is assumed finite and discrete.
- set of *cost functions* $f = \{f_1, \ldots, f_k\}$ where each f_i is a function $f_i : D_{i,1} \times \cdots \times D_{i,j} \rightarrow N \cup \infty$. Cost functions are also called *constraints*.
- a *distribution mapping* $Q : V \rightarrow A$ assigning each variable to an agent. $Q(x_i) = A_i$ means that A_i is responsible for choosing a value for x_i. A_i is given knowledge of x_i, D_i and all f_i involving x_i.
- an *objective function* F defined as an aggregation over the set of cost functions. Summation is most commonly used.

The goal for the agents is to choose values for variables such that F is minimized. Two agents whose variables share a constraint are called *neighbors*. Agents may send messages to any agent they know about and initially agents only know about their neighbors. When each agent is assigned a single variable, it is common to use the notation A_i and x_i interchangeably as we will in this paper.

2.1. Adopt and OptAPO

Adopt and OptAPO are two state of the art algorithms for DCOP. Both are *complete*, i.e., theoretically guaranteed to return the optimal solution, and *asynchronous*, i.e., they remain correct even when agents execute concurrently, potentially at different execution speeds. In both algorithms, agents interleave computation with communication. However, there are a number of qualitative differences in the algorithms which we describe below.

Adopt [9] is an algorithm for DCOP that is able to find globally optimal solutions while allowing agents to choose variable values in parallel. Adopt performs a distributed search using the communication of costs to guide agents toward globally optimal value choices. Agents communicate their current variable values to lower priority neighbors, who respond with messages containing lower bounds on F computed by conditioning on the value choices of higher priority agents. Higher priority agents respond by exploring new values. Lower bounds are communicated only to the lowest higher priority neighbor. As this process continues, lower bounds become progressively more accurate, until ultimately the lower bound of the minimum cost solution equals its upper bound, indicating the cost of the optimal solution has been found. Note that agents do not directly communicate their constraints to other agents and only send messages between neighbors.

OptAPO [6] is an alternative approach to DCOP that uses direct communication of constraints to partially centralize the problem within a mediator. Election of the mediator is done in an intelligent way using dynamic priorities determined during problem solving. The mediator uses a centralized optimization routine to find an optimal solution to its portion of the problem. The optimization routine used by Mailler and Lesser is the Branch and Bound algorithm of Freuder et. al. [3].

Agents in OptAPO use a novel cost justification technique to drive the communication of constraints. This technique avoids centralization when it is deemed unjustified based on problem structure. As an OptAPO agent receives constraints from other agents in the problem, it adds the other agents to a data structure called its *goodlist*. We will use the size of an agent's goodlist to measure amount of centralization in OptAPO. Finally, when constraints are communicated between two agents who are not neighbors, a linking procedure is used to establish a direct communication link.

2.2. Discussion of Qualitative Differences

Communication of Constraints: We see that a key difference between Adopt and OptAPO is that agents in OptAPO communicate their constraints to other agents which allows the agent who receives them to evaluate the constraint. The communication of constraints between agents has significant implications on load balancing and the amount of computation that each agent must perform during problem solving. This is because as the size of an agent's subproblem grows as constraints are gathered, more local computation (search) is required to find the optimal solution to the larger subproblem. Thus, when constraints are communicated between agents, the computation load at each agent may increase during problem solving. In OptAPO, we may expect that the computational load at some agents will grow as problem solving progresses and their sub-problems grow. On the other hand, in an algorithm which does not communicate constraints, such as Adopt, we may expect that the computational load at each agent will remain constant during problem solving.

Adding Links: Adopt and OptAPO seemingly make different assumptions about the communication links in the underlying application domain. OptAPO assumes that an agent has the ability to establish a direct communication link with any other agent. Adopt only requires that agents communicate with their neighbors. This may have a direct impact on algorithm performance in some domains. Wireless communication for example has limited range and so spatially distant agents who must communicate must do so via "multi-hop" messages which can substantially increase message delivery time. However, we do not investigate this issue in this paper.

3. An Evaluation Metric for Asynchronous Algorithms

Performance measurement and comparison of distributed algorithms is more complicated than for traditional centralized algorithms. Distributed algorithms have multiple agents that run concurrently and communicate asynchronously. This distribution of the algorithm creates several challenges for evaluation in a typical research lab environment. Running in a fully distributed manner across a cluster of many computers is often not practical. Alternatively, an asynchronous algorithm can be run on a single computer using multiple threads of execution, for example using a discrete-event network simulator. However, this is also problematic because there are an exponential number of execution paths for an asynchronous algorithm and there can be significant variation between runtimes depending on the path chosen by the underlying simulator. Evaluation over all possible execution paths is often not practical.

3.1. Number of Cycles

Because of the above difficulties, previous researchers have proposed evaluating asynchronous algorithms according to one standardized execution path, namely one in which agents synchronously interleave communication and computation. Specifically, algorithm execution is divided into a sequence of cycles [11] as defined below.

Definition: A *cycle* is defined as one unit of algorithm progress in which all agents, in parallel, process their incoming messages, perform any required computation, and send their outgoing messages. Importantly, a message sent in cycle i is not received until cycle i+1.

Cycles are a convenient standardized metric for estimating the performance of a DCOP algorithm that avoids the problems described earlier. However, a drawback of cycles is that it does not take into account the amount of computation required by the distributed agents. We wish to devise a metric that retains the desirable properties of the measurement using cycles but considers computation costs as well.

On initial consideration it might seem that the amount of computation performed by an algorithm could be accurately measured by the total runtime used by the process on a single computer. However, since the agents must take turns using a single processor and cannot execute in parallel as they would in a distributed system, the runtime may not accurately reflect the actual distributed performance. If the agents solving the problem do not share the computational burden relatively evenly, then they will not take advantage of the parallelism of distributed problem solving.

3.2. Cycle-Based Runtime

To more accurately measure the performance of DCOP algorithms, we desire a metric that approximates the total runtime of an algorithm whose execution has been measured using synchronous cycles. We begin with a simple definition of runtime:

$$total\ runtime\ of\ m\ cycles = \sum_{k=0}^{m} time\ for\ cycle\ k \quad (1)$$

Now, we need a definition for the time of a cycle. A cycle involves communication followed by computation. Let L denote the time required in a cycle to deliver all messages sent in the previous cycle. We call this the *latency* of the underlying communication environment. L is algorithm independent. So we have

$$time\ for\ cycle\ k = L + computation\ time\ in\ cycle\ k \quad (2)$$

In order to measure the computational cost in a cycle, we make use of a recent metric - concurrent constraint checks (ccc) [8]. A constraint check is the act of evaluating a constraint in the problem by comparing the value of one variable to another variable in the problem. Constraint checks are a well accepted measure of computation in traditional centralized constraint processing algorithms. Let $cc(x_i, k)$ be the number of constraint checks performed by agent x_i in cycle k. Then the computation time of cycle k is defined as:

$$computation\ time\ in\ cycle\ k = \max_{x_i \in V} cc(x_i, k) \times t \quad (3)$$

where t is the time required for one constraint check. t is a property of the underlying computing hardware and is algorithm independent. The max over all agents is used because the agents are conceptually executing in parallel. The length of a cycle is determined by how long the longest running agent took to complete. Substituting 3 into 2, we have

$$time\ for\ cycle\ k = L + \max_{x_i \in V} cc(x_i, k) \times t \quad (4)$$

Now substituting 4 in 1,

$$total\ runtime\ of\ m\ cycles = \sum_{k=0}^{m} (L + \max_{x_i \in V} cc(x_i, k) \times t) \quad (5)$$

Finally, the number of concurrent constraint checks (ccc) performed by an algorithm over n cycles is defined as:

$$ccc(m) = \sum_{k=0}^{m} \max_{x_i \in V} cc(x_i, k) \quad (6)$$

Substituting 6 in 5, we arrive at our final equation for the time of m cycles, called Cycle-Based Runtime (CBR):

$$CBR(m) = t \times ccc(m) + L \times m. \quad (7)$$

Note that the CBR metric is parameterized according to two environmental factors: the communication latency between cycles (L) and the speed of computation (t). Using this parameterized model, we can evaluate algorithm performance over a range of environments that vary in their relative speeds of communication and computation. Time required to transmit a message is usually greater than the time for a constraint check in most environments, so for simplicity we assume that a constraint check is the smallest atomic unit of time ($t = 1$), and assume L is given relative to t. We will explore four types of environments where communication costs are increasing by order of magnitude relative to computation, i.e., $L = t$, $L = 10t$, $L = 100t$, $L = 1000t$.

CBR does not take into account number of messages or the time required to process messages. In other words, we assume that message processing time per cycle is not a significant differentiating feature between algorithms under comparison. We believe this is true for the algorithms compared in this paper. While Adopt uses many more messages than OptAPO, this is explained by its higher cycle count, i.e, the number of messages communicated per cycle is about the same between the two algorithms. Also, we assume the time to process each message is similar for both algorithms.

4. Empirical Evaluation

We obtained the OptAPO code from its creators Roger Mailler and Victor Lesser, and the Adopt code from its creator Pragnesh Jay Modi. We used a simulator framework to measure ccc and cycles in both OptAPO and Adopt. Following previous work [9, 6], we then ran OptAPO and Adopt on a set of randomly generated 3-coloring problems. The problems were generated with problem sizes of n=8, 12, 16, or 20, and a link density of either 2n or 3n. Each problem size had 50 generated problems (a total of 8*50 = 400). The same set of randomly generated graphs was used for each algorithm.

4.1. Runtime as Measured By CBR

Constraint checks and cycle counts were logged and used to compute the value of CBR in Eqn 7 for four different values of L. We create a different graph for each value. As described in Section 3, L represents the time required by the communication environment to deliver messages between cycles specified relative to the time for a constraint check. For example if $L = 1$, we are assuming communication is very fast and on the same order of magnitude as a constraint check. If $L = 1000$, we are assuming communication takes three orders of magnitude longer than a constraint check.

Figures 2 and 3 show four graphs generated from a single set of experiments on problems of link density 2n and 3n respectively. Each datapoint represents the average of the 50 problems. In Figure 2, we see that when L is 1, 10, and 100, Adopt outperforms OptAPO. At $L = 1000$, Adopt performs slower than OptAPO on the problem sizes we tested. However, from the growth rates of the lines it appears that OptAPO may exceed Adopt on larger problem sizes. To investigate this, we were able to run a small number of experiments with problems containing 24 variables. We completed 20 problems for density 2 and 10 for density 3 (the lengthy runtimes on these large problem sizes prevented completion of more problems). The performance on these problems has been shown with a dotted line on the $L = 1000$ graph, and indicates that Adopt may outperform OptAPO on large problems even at $L = 1000$.

We observed that while Adopt requires more cycles than OptAPO, each OptAPO cycle takes significantly longer than each Adopt cycle. L provides a parameter to vary the relative cost between number of cycles and length of each cycle. We conclude that for a significant range of L, Adopt performs better than OptAPO, and as problem size grows this range increases.

4.2. Centralization of OptAPO

We have hypothesized that the degree of centralization is the reason that OptAPO's cycles take much longer than an Adopt cycle. To verify this, we recorded the amount of centralization that the OptAPO agents reached by termination, as represented by the size of the OptAPO *goodlist*, which contains the other agents whose constraints have been centralized to an agent.

We computed the average, minimum, and maximum goodlist sizes across the agents in a problem at termination. We obtained similar results to the centralization data reported in Mailler's thesis [7]. As seen in Figure 4, on low density problems OptAPO agents on average have centralized at least half of the problem by the time a solution is found. On highly dense graphs, which are more difficult

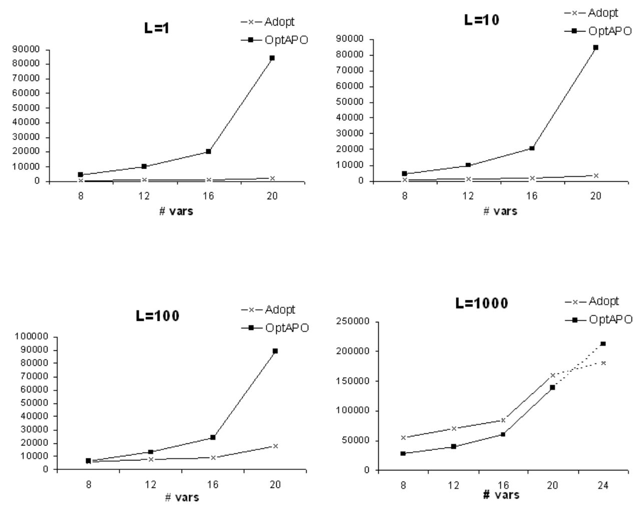


Figure 2. Comparison of Adopt and OptAPO using the CBR metric on graphs of low density. Each graph represents a different L value.

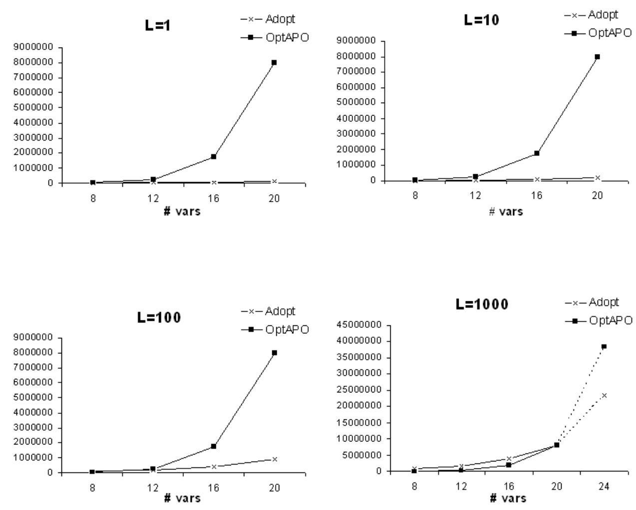


Figure 3. Comparison of Adopt and OptAPO using the CBR metric on graphs of high density. Each graph represents a different L value.

and time-consuming to solve, OptAPO on average centralizes nearly all of the problem.

The Max bars show that in high density graphs, almost all problems had at least one agent that fully centralized the problem. In low density problems, on average there was at least one agent who centralized about 75% of the problem.

4.3. Parallelization of Computation

So far we have found that OptAPO does more computation, based on our measurement of the maximum constraint checks performed across the agents during each cycle. However, we would also like to determine whether the higher maximum constraint checks is due to OptAPO simply doing more computation in *all* the agents during a cycle, or if it is due to uneven distribution of the computational load.

As discussed in Section 3, $cc(x_i, k)$ is the number of constraint checks performed by agent x_i in cycle k. Then, the distribution of computation within a cycle, which we will call $load(k)$, can be represented by the ratio of the maximum constraint checks to the total constraint checks in a cycle:

$$load(k) = \frac{\max_{x_i \in Agents} cc(x_i, k)}{\sum_{x_i \in Agents} cc(x_i, k)} \tag{8}$$

This equation represents the fraction of work that the maximum computing agent did during the cycle. A value of 1.0 means one agent did all of the computation in that cycle, and a lower value indicates the load was more balanced.

In Figure 5, the load ratio for OptAPO and Adopt is graphed for the execution of one representative graph coloring problem with 8 variables and a density of 2n. The x-axis is the cycle number of execution, and the y-axis is the load as defined in Eqn 8. The line for OptAPO shows spikes at cycles where an agent, the mediator, did a branch & bound search and accounted for most or all of the computation in that cycle. On the other hand, Adopt had very consistent distribution of computation, with most agents doing a fairly similar number of constraint checks for most of the algorithm's duration.

This chart illustrates that OptAPO finished in a fewer number of cycles than Adopt, but the computation during those cycles is less evenly distributed among the agents, which results in longer time per cycle.

4.4. Tradeoffs Between Communication Latency and Centralization

As our analysis has shown, a non-centralized algorithm like Adopt uses more communication cycles but has a lower computational cost per cycle. OptAPO, a partially central-

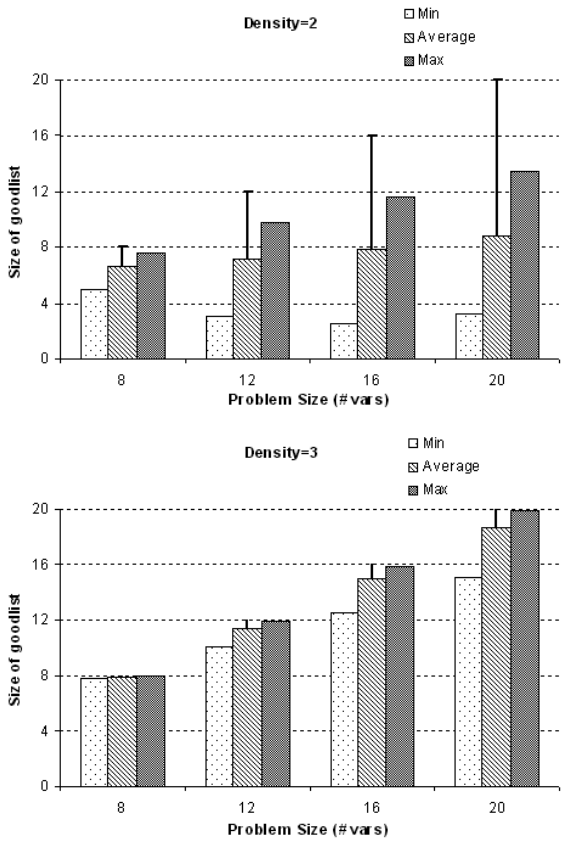


Figure 4. OptAPO centralization - Avg is the average centralization across the agents in a problem, Max is the highest centralization of all the agents in a problem, and Min is the lowest of the agents. The upper line above each bar marks n (# of variables), which is the maximum possible centralization at each problem size. Each measurement is the average of 50 problems.

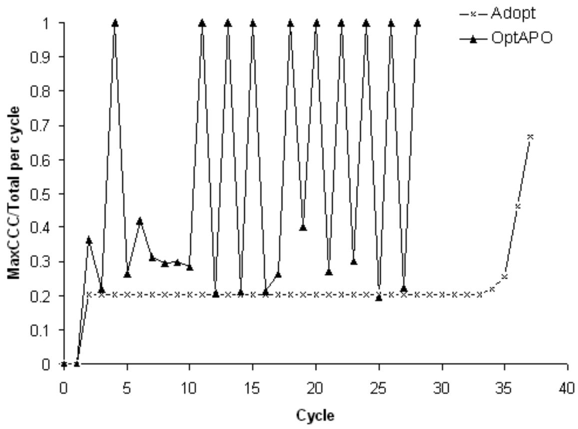


Figure 5. A measure of the distribution of computation in Adopt and OptAPO. The peaks on the OptAPO line indicate that in those cycles a single agent did most of the computation.

ized algorithm, has relatively low communication cycles but higher computational cost per cycle. We now ask how does a partially centralized approach like OptAPO and a decentralized approach like Adopt, compare with a completely centralized approach using CBR as an evaluation metric?

For the centralized approach, we assume one agent starts the algorithm with full knowledge of the problem, and simply invokes an optimization search procedure. We used OptAPO's implementation of centralized branch and bound search. We measured the number of constraint checks required to find the optimal solution. We ignored the overhead cost that would be required in a truly distributed setting of electing a centralizer and all agents communicating the problem information to it. In the worst case, this cost is only some small factor of the width of the communication graph.

Figure 6 shows the three algorithms at different L values. As expected, the centralized algorithm is insensitive to varying L values because no communication is required. For both graph densities, Adopt is the best performing algorithm at L values less than 100. The crossover point occurs between L=100 and L=1000. These crossover points are important because they tell us at what point communication becomes too expensive for Adopt to operate efficiently, and tell us which algorithm should be used for a given communication environment.

For density=2, the OptAPO performance curve outperforms its own centralized solver using the CBR metric. These results agree qualitatively with the results using a serial runtime metric reported by Mailler and Lesser [6]. On density=3, the fully centralized approach had a lower CBR than OptAPO, which we believe may be explained by the fact that OptAPO does repeated multiple branch and bound searches, which could become more costly on dense graphs. The OptAPO searches partially reuse past searches, but this partial reuse does not completely recover the cost of the previous searches. From our analysis, we conclude that on high density graphs OptAPO eventually centralizes most of the problem, but does so with a higher cost than doing a simple centralization in the first step of the algorithm.

Figure 6 is an key contribution of this paper because a researcher seeking to apply a DCOP algorithm to a new domain could use this chart to determine which algorithm

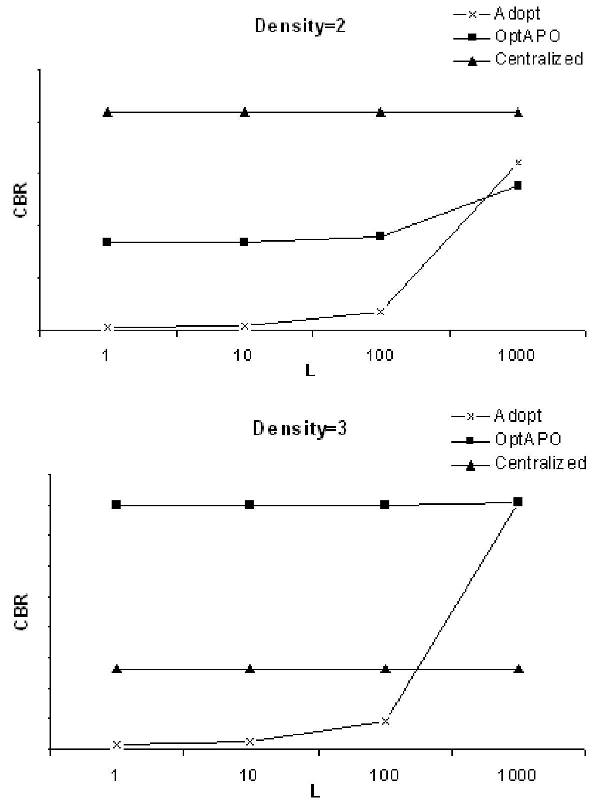


Figure 6. Adopt, OptAPO, and Centralized at 4 different L values. Each graph is based on 50 random problems of 20 variables.

would be most efficient given the communication model being used and the expected constraint density.

5. Conclusion

We have investigated two algorithms for DCOP, OptAPO and Adopt, that vary in the amount they centralize the problem in order to find the optimal solution. We developed a metric, CBR, for more accurately comparing these algorithms by taking into account communication latency between cycles and the length of each cycle. We have shown that while OptAPO requires fewer cycles than Adopt, OptAPO's cycles are longer because they require more computation. For domains with low communication latency compared to time to do a computation, Adopt outperforms OptAPO because in such domains agents are able to communicate efficiently and Adopt is able to take advantage of it by more evenly distributing the work of solving the DCOP. We have created graphs of the relative performance of Adopt, OptAPO, and centralized search under environments with varying communication latencies, providing the ability to choose the most effective level of centralization for each environment.

6. Acknowledgements

We thank Roger Mailler for generously providing us with his implementation of OptAPO, which made this investigation possible. We thank Manuela Veloso for productive discussions and her many insightful comments.

References

[1] I. Brito, F. Herrero, and P. Meseguer. On the Evaluation of DisCSP Algorithms. In *Proc. Workshop on Distributed Constraint Reasoning held at Constraint Programming 2004 (CP)*, 2004.

[2] B. Faltings and S. Macho-Gonzalez. Open constraint optimization. In *Principles and Practice of Constraint Programming - CP*, 2003.

[3] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artif. Intell.*, 58(1-3):21–70, 1992.

[4] K. Hirayama and M. Yokoo. An approach to over-constrained distributed constraint satisfaction problems: Distributed hierarchical constraint satisfaction. In *Proceedings of International Conference on Multiagent Systems*, 2000.

[5] J. Liu and K. Sycara. Exploiting problem structure for distributed constraint optimization. In *Proceedings of International Conference on Multi-Agent Systems*, 1995.

[6] R. Mailler and V. Lesser. Solving Distributed Constraint Optimization Problems Using Cooperative Mediation. In *Proceedings of Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 438–445. IEEE Computer Society, 2004.

[7] Roger Mailler. *A Mediation-Based Approach to Cooperative, Distributed Problem Solving*. PhD thesis, University of Massachussetts at Amherst, 2004.

[8] A. Meisels, E. Kaplansky, I. Razgon, and R. Zivan. Comparing Performance of Distributed Constraints Processing Algorithms. In *Proc. Workshop on Distributed Constraint Reasoning (AAMAS)*, 2002.

[9] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence Journal*, 2004.

[10] M.C. Silaghi and D. Mitra. Distributed constraint satisfaction and optimization with privacy enforcement. In *3rd IC on Intelligence Agent Technology*, 2004.

[11] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *Knowledge and Data Engineering*, 10(5):673–685, 1998.