Distributed Constraint Reasoning under Unreliable Communication

Pragnesh Jay Modi
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
pmodi@cs.cmu.edu

Syed Muhammad Ali
Computer Science Department
University of Southern California
Los Angeles, CA 90089
syedmuha@usc.edu

Abstract. The Distributed Constraint Optimization Problem (DCOP) is able to model many problems in multiagent systems but existing research has not considered the issue of unreliable communication which often arises in real world applications. Limited bandwidth, interference, loss of line-of-sight are some reasons why communication fails in the real world. In this paper we show that an existing asynchronous algorithm for DCOP can be made to operate effectively in the face of message loss through the introduction of a very simple timeout mechanism for selective communication. Despite its simplicity, this mechanism is shown to dramatically reduce communication overhead while preventing deadlocks that can occur when messages are lost. Results show that the optimal solution can be guaranteed even in the presence of message loss and that algorithm performance measured in terms of time to solution degrades gracefully as message loss probability increases.

1 Introduction

In recent years, several researchers have investigated Distributed Constraint Reasoning (DCR) as a framework for problem solving in Multiagent Systems [4] [6] [8] [13] [15]. One of the main features that make it attractive is its distributed nature in which a set of autonomous agents can each make local decisions but communicate in order to improve global solution quality. In terms of experimental domains, a distributed sensor network domain has been used as a key challenge problem for DCR with significant progress reported [3] [7][9][12].

Despite the many existing complete algorithms for DCR, such as ABT, AWC[15], AAS [13], DisDB[1] and APO[8], what is missing is a study of the performance of these algorithms when message transfer is unreliable. This paper takes the first steps toward evaluating DCR algorithm performance under message loss. We will assume a simple form of unreliable communication: the communication infrastructure has an unknown *loss probability* $r < 1$, where a message is dropped (not delivered) with probability r. For example, the radio-frequency communication used in the distributed sensor network mentioned above is susceptible to interference which can cause intermittent message loss. We not consider here permanent link failures which is also an important class of unreliability. We will focus on one particular asynchronous algorithm – the Adopt algorithm for Distributed Constraint Optimization Problems (DCOP). We focus on Adopt because it is currently one of the most efficient decentralized

constraint optimization algorithms that can provide strong guarantees on the global quality of its solutions[11]. While we will use Adopt for our investigation, our hope is that our techniques are valuable for other asynchronous DCR algorithms as well.

A common method for dealing with unreliable channels in communication networks is to implement an error correction layer in software that can ensure reliable message delivery even when the communication infrastructure itself is inherently unreliable. This is typically done through an acknowledgment protocol where ACK messages are used to verify that a message has been received. A number of such protocols have been developed in the field of computer networking, the most popular of which is TCP [14]. However, simple reliance on a lower layer error-correction mechanism to ensure reliable delivery is an inefficient approach for dealing with unreliable communication infrastructure in asynchronous DCR algorithms. First, it can significantly increase the number of messages that must be communicated since every message must be acknowledged. Second, to enforce in-order-delivery between a given pair of agents, a sender cannot deliver any messages to a given agent x_i until the ACK for a previously sent message is received from x_i. The time cost in waiting for ACKs can degrade performance and reduce the efficiency of the higher-level DCR algorithm. Third, this method is unable to take advantage of the fact that it may be okay for some messages to be lost without large negative effects on the DCR algorithm.

In this paper, we evaluate an alternative timeout-based asynchronous communication model for DCR algorithms. This approach does not require explicit acknowledgment messages for every message and allows agents to continue communicating asynchronously even when messages are lost. We show that this simple mechanism significantly reduces communication overhead, provides a method for trading off time-to-solution for communication overhead and provides robustness to message loss. We provide empirical results using Adopt in a real-world distributed sensor domain.

2 DCOP Definition

A Distributed Constraint Optimization Problem (DCOP) consists of n variables $V = \{x_1, x_2, ...x_n\}$, each assigned to an agent, where the values of the variables are taken from finite, discrete domains $D_1, D_2, ..., D_n$, respectively. The goal is for the agents to coordinate their choice of values so that a global objective function is optimized. The objective function is described as the summation over a set of *cost functions*. A cost function for a pair of variables x_i, x_j is defined as $f_{ij} : D_i \times D_j \rightarrow N$. The cost functions in DCOP are the analogue of constraints from DisCSP and are sometimes referred to as "valued" or "soft" constraints.

Figure 1.a shows an example DCOP with four agents where each has a single variable with domain $\{0, 1\}$. There are four constraints shown. Two agents x_i, x_j are *neighbors* if there is a constraint between their variables. In Figure 1, x_1 and x_3 are neighbors but x_1 and x_4 are not. The objective is to find an assignment $\mathcal{A}^*$ of values to variables such that the aggregate cost F is minimized. Stated formally, we wish to find $\mathcal{A}$ ($= \mathcal{A}^*$) such that $F(\mathcal{A})$ is minimized, where the objective function F is defined as

$$F(\mathcal{A}) = \sum_{x_i, x_j \in V} f_{ij}(d_i, d_j) \quad , where\ x_i \leftarrow d_i, \\ x_j \leftarrow d_j\ in\ \mathcal{A}$$

For example in Figure 1 if all variables are assigned the value 0 then $F = 4$. If all variables are assigned the value 1 then $F = 0$ which is the optimal solution.

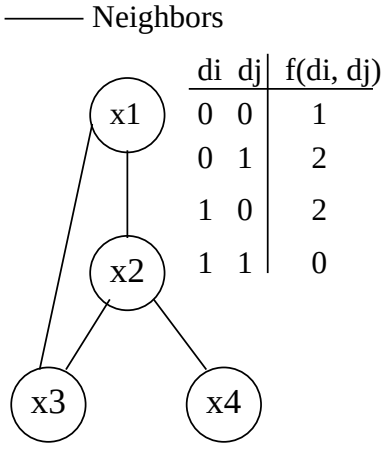


Figure 1: Example of a simple DCOP graph.

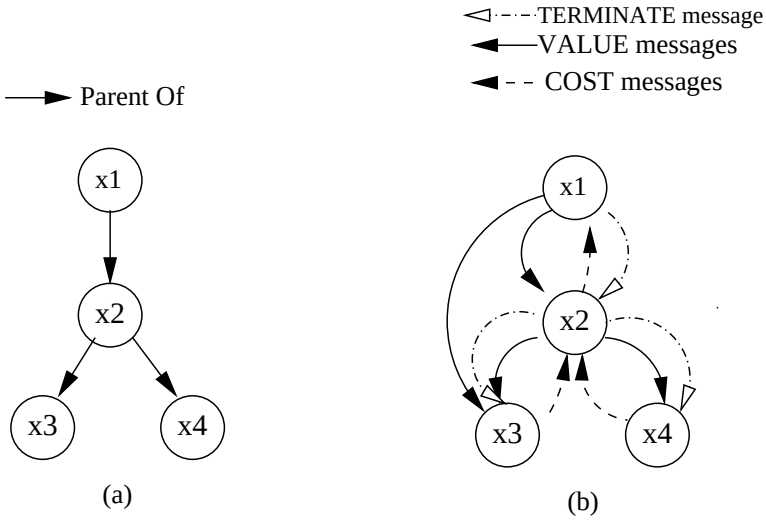


Figure 2: a) Example tree structure, b) Message Communication.

3 Adopt Algorithm for DCOP

Adopt 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 concurrent conditioning search using the communication of lower bounds to guide agents toward globally minimum cost value choices. It has been shown to be able to solve certain classes of benchmark problems efficiently due to its concurrency. We provide a brief overview of the Adopt algorithm and refer the reader to [11] for details.

As a preprocessing step before Adopt executes, the agents form a global tree structure over their variables in which each variable except the root variable has a single *parent*. Figure 2.a shows one possible tree formed from the constraint graph in 1. The tree provides an organizational structure that is useful for providing theoretical guarantees on termination and solution quality. Once the tree is formed, agents begin execution in which each agent asynchronously executes a processing loop in which it waits for incoming messages, processes them and sends outgoing messages. Agents pass messages up or down the tree. VALUE messages are sent down the tree along constraint edges informing lower agents of the value choices of higher

neighbors. Agents record the value choices of other agents in a "current context" or "agent view". A context is a partial solution representing an agent's view of the current assignments of other agents' variables. In response to VALUE messages, COST messages are sent from child to parent to provide feedback to higher agents on their value choices. The technique of *context attachment* is used to deal with asynchronous changes and so each COST message is attached with the sending agent's own current context. Finally, THRESHOLD messages are used to increase efficiency of the optimization process but are not necessary for correctness.

A key novelty of the Adopt algorithm is its built-in termination detection mechanism. Agents continue communicating messages until a local termination condition is true at the root agent. Once the termination condition is true at the root agent, the root sends a TERMINATE message to its children and terminates itself. After receiving a TERMINATE message from its parent, an agent knows that all of its higher neighbors have terminated and once its own local termination condition is true, it will send TERMINATE messages down to its own children and terminate itself. In this way, TERMINATE messages are recursively sent down the tree until all agents have terminated. While we have omitted many details for simplicity, the Adopt algorithm has been proved to terminate with the globally optimal solution [11], i.e., it is sound and complete assuming reliable communication.

4 Issues with Asynchronous Communication

The dominant asynchronous communication model in existing DCR algorithms like Adopt[11] and others, is for agents to initially send messages, and then execute a continuous loop in which they respond to received messages by sending more messages. In this model, an agent does not send any messages unless it receives a message and it must send a message every time a message is received in order to avoid possible deadlocks. This model has two problems. First, while this model has been successful in providing termination guarantees while allowing agents to execute asynchronously, message loss invalidates these guarantees due to deadlock problems. Second, even if there is no message loss, this model is not very efficient in terms of the number of messages exchanged because agents often communicate unnecessarily. We elaborate on each problem in turn.

When messages are dropped, one of the major difficulties that arises is the possibility of deadlock in which all agents are waiting for each other to communicate. For example, consider two agents x_1 and x_2 executing the Adopt algorithm. x_1 sends a VALUE message to x_2 and waits for an incoming message. x_2 receives the VALUE message and sends back to x_1 a COST response. Suppose this message is lost. At this point, x_1 is blocked and so is x_2 waiting for another message from x_1. Thus, the loss of one message has resulted in the agents getting deadlocked. The primary cause of this difficulty is the asynchronous communication model in which agents send messages only in response to receipt of messages. Indeed, this problem arises in any DCR algorithm that uses this asynchronous communication model.

To reduce communication overhead, the simplest approach is for an agent to only send messages if it has something new to say, i.e., a variable value or costs have changed. However, this approach by itself is insufficient to guarantee completeness because duplicate messages must be sent in some cases. Why must an agent send a message that is identical to a message it has just sent in the previous execution cycle? To see why, realize that DCR algorithms deal with asynchronous changes through the use of context attachment. When an agent x_{rec} receives a COST message attached with a context that does not match its current context (i.e.,

```
procedure sendMsgs
(1)   if timeSinceLastMsgSent > TIMEOUT or
(2)   valueChanged:
(3)     SEND (VALUE_MSG)
(4)       to each lower priority neighbor
(5)   if timeSinceLastMsgSent > TIMEOUT or
(6)   costChanged:
(7)     SEND (COST_MSG) to parent
(8)   if terminationCondition == true:
(9)     SEND (TERMINATE_MSG) to each child
```

Figure 3: Procedure for using TIMEOUT mechanism in Adopt

agent view), the message is deemed to be obsolete and is ignored. However, it is possible that in fact the agent's current context is obsolete and the received message has more up-to-date information. Thus, the sending agent, x_{send} must resend the message – even though no costs may not have changed.

In the next section, we present a simple solution that allows trade-offs between the two competing issues of deadlock avoidance and high communication overhead.

5 The Timeout Mechanism

We present a simple communication model which uses a timeout mechanism to significantly reduce communication overhead and provide robustness to message loss. Rather than communicating in every cycle to deal with asynchronous changes as has been done in previous algorithms or communicating only when information has changed which can lead to incompleteness, we propose a parameterized approach to communication in which agents communicate only if they have something new to say or they have not sent any messages in a given amount of time, as specified by an algorithmic parameter called TIMEOUT. The algorithm for communication in the Adopt algorithm is shown in Figure 3. To see that deadlocks will not occur, realize the timeout will ensure that an agent will not sit waiting forever for a message that may not come. Instead, it will continue sending messages to its children until a reply is received and its termination condition is eventually true.

The TIMEOUT mechanism dramatically reduces the amount of communication necessary to find the globally optimal solution, up to 80% in the experiments discussed later. The mechanism also overcomes deadlock problems caused by loss of messages. It can be ensured that the algorithm will eventually terminate with the optimal solution, regardless of the amount of network disturbance, so long as the probability of a message being lost is less than 1. Only the TERMINATE messages, which are essentially a distributed snapshot mechanism [2], require reliable communication. An acknowledgment protocol can be used to ensure TERMINATE messages are successfully communicated, but since they only need to be communicated once between parent and child, the overhead for this is not very severe, and is certainly less than requiring all types of messages to be communicated reliably.

The TIMEOUT mechanism is effective due to a key novelty of the Adopt algorithm: Adopt's built-in termination detection. Rather than relying on an external quiescence detec-

tion algorithm, the built-in termination condition allows an agent to locally determine whether to terminate. If no messages are received for a certain amount of time and an agent's termination condition is not true, then that agent can conclude that a deadlock may have occurred due to message loss. The agent can then resend messages to its neighbors in order to trigger the other agents and break the deadlock. In this way, the algorithm is able to intelligently determine that messages need to be resent by using the built-in termination condition as a guide.

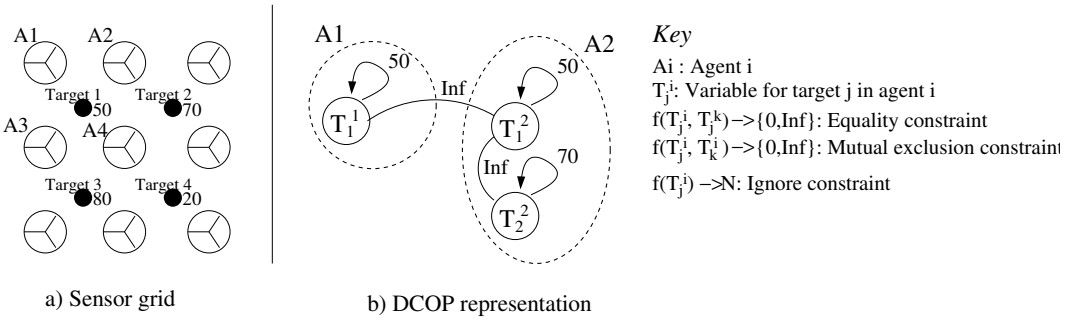
We also see that since Adopt is completely asynchronous, it is well suited for operating under unreliable communication infrastructure. In particular, agents are able to process any message no matter when it is received and are insensitive to the order in which messages are received. The method is also stateless in the sense that an agent does not need to remember the last message it sent in case a resend is needed. The agent can simply send out its current messages whenever a timeout occurs. This is in contrast to synchronous algorithms which require messages to be received and sent in a particular order. For example in Synchronous Branch and Bound [5], if a message is lost no progress can be made until that message is successfully retransmitted.

6 Experimental Domain: Sensor Network as DCOP

Our application is a distributed sensor network domain that has gained the attention of a number of multiagent researchers in recent years[3] [7] [12] [9]. The domain consists of multiple fixed directional sensors and multiple targets in their sensing range. The direction of each sensor is controlled by an autonomous on-board agent. Agents are able to send messages to each other using low-bandwidth radio-frequency communication however the communication is unreliable due to possible interference. Furthermore, sensors may become ineffective due to loss of power or damage. Figure 4.a shows 9 sensors in a grid configuration. Three sensors must be pointed at a target to track it and no sensor may track more than one target. Assuming each target is within the sensing range of only the four nearest sensors, we see that only two of the four targets can be feasibly tracked in Figure 4. The agents must coordinate in order to track the targets with highest weights as denoted by the number next to each target.

While there are many possible representations of this domain using distributed constraints, our representation of the sensor domain using DCOP is shown in Figure 4.b. Only the variables and constraints for agents A1 and A2 are shown. Intuitively, the DCOP requires assigning triples of agents to targets. If there are too many targets and not enough agents, lower-weighted targets are ignored. The goal then is to minimize the sum of the weights of the ignored targets. This mapping attempts to represents the domain in the sense that if agents find values for variables that minimize the sum cost over all the constraints, they will minimize the sum weight of ignored targets. The mapping is motivated by the desire to use only unary and binary constraints and avoid higher arity constraints.

The mapping proceeds as follows. We create a variable T_j^i for each target j and agent i who could possibly sense target j. The domain of variable T_j^i is the set of agent triples that could track target j. For example, the domain of variables T_1^1, T_1^2 in Figure 4.b (and T_1^3, T_1^4 not shown) is the set $\{A1A2A3, A1A2A4, A1A3A4, A2A3A4, Ignore\}$. The $Ignore$ value indicates that zero agents are allocated to the target. An equality constraint between two variables T_j^i and T_j^k belonging to agents i and k requires them to agree on which 3 agents will track the target j. A mutual exclusion constraint between two variables T_j^i and T_k^i within agent



a) Sensor grid

b) DCOP representation

Figure 4: Mapping of sensor network to DCOP

i requires that the agent not participate in tracking two targets. The cost of violating these two constraints is set to a very high value (Inf) essentially making them "hard" constraints. Finally, an Ignore constraint defined for each target gives the cost for choosing the $Ignore$ value for that target.

7 Experiments

Although Adopt allows agents to execute asynchronously, we perform our experiments in a simulation environment in which agents execute in synchronous cycles. One *cycle* is defined as all agents receiving all incoming messages and sending all outgoing messages simultaneously. The synchronous cycle metric allows repeatable experiments because it is not sensitive to differing computation speeds at different agents or fluctuations in message delivery time. Indeed, these factors are often unpredictable and we would like to control for them when performing systematic experiments. However, we note that a weakness of this metric is that it does not take into account the time required for local processing. So instead of only reporting the number of cycles, we also report a metric has called "concurrent constraint checks" [10] in which we record the maximum number of constraint checks by any agent in a given cycle and then sum over all cycles.

We experiment in the sensor network domain. Multiple variables per agent are handled using the virtual agent approach [16] in which multiple threads within one agent each run the Adopt algorithm independently for a single variable. We use two sensor configurations: GRID in which sensors are arranged as shown in Figure 4 and CHAIN in which sensors are arranged in two parallel rows. For each configuration, a specified number of targets are randomly placed and the four nearest sensors are assumed to be the only ones who could track it. Each target is given a random weight in [0,100]. Each datapoint is the average of 20 runs and in all cases, the globally optimal solution is obtained.

7.1 Reduction in Communication Overhead

We first present experiments evaluating the use of the TIMEOUT parameter on cycles, concurrent constraint checks, and number of messages. Figure 5 shows results in a GRID formation while Figure 6 shows results in a CHAIN formation. We show how performance varies for four TIMEOUT values: (1,10,100,1000). The results show that the number of messages communicated decreases dramatically as TIMEOUT is increased. On the other hand, if it is increased too high, we see that the number of cycles increases also. We conclude from these

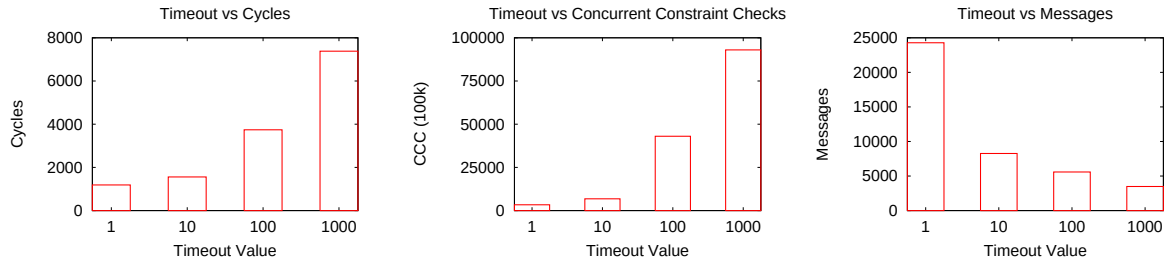


Figure 5: Grid Sensor Network, 16 sensors, 5 targets. Increasing TIMEOUT value increases cycles (left) and computation (middle), but reduces communication (right)

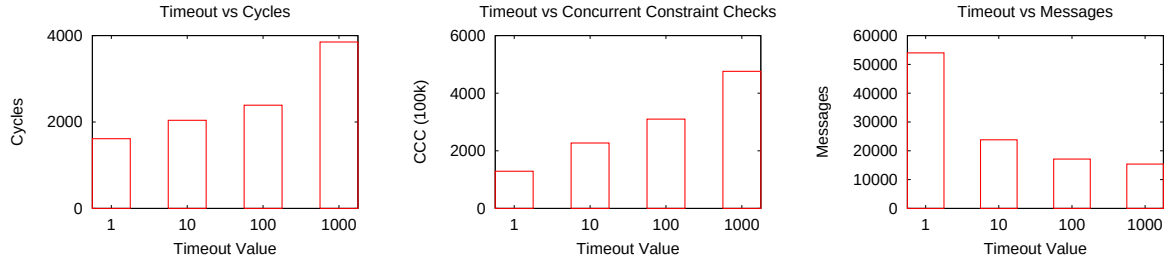


Figure 6: Chain Sensor Network, 24 sensors, 10 targets. Increasing TIMEOUT value increases cycles (left) and computation (middle), but reduces communication (right)

experiments that a TIMEOUT value of 10 cycles provides an appropriate trade off between these two competing aspects of performance, and reduces communication significantly, from 25000 (for TIMEOUT=1) to 10000 (for TIMEOUT=10) in the GRID experiments and similarly from 55000 to 25000 in the CHAIN experiments.

7.2 Robustness to Message Loss

We have also evaluated with the effect of message loss on the performance of Adopt. Figure 7 and 8 shows the change in performance on a GRID and CHAIN network respectively as message loss rates are increased from 0 to 20%. A timeout value of 10 cycles is used. We see that Adopt is fairly robust to low rates of message loss as performance according to all three metrics is not very much effected. At high loss rates of 10% and higher, we begin to see performance inevitably degrade although the optimal solution is still obtained in all cases.

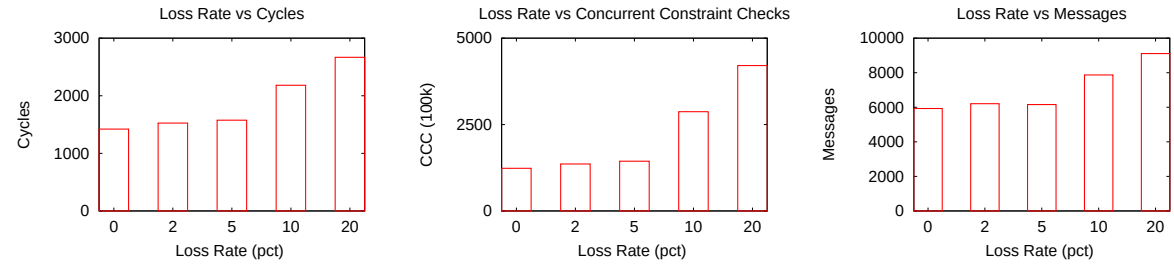


Figure 7: Grid with 9 sensors and 4 targets. Increasing message loss levels gracefully degrade performance in cycles (left), computation (middle) and communication (right).

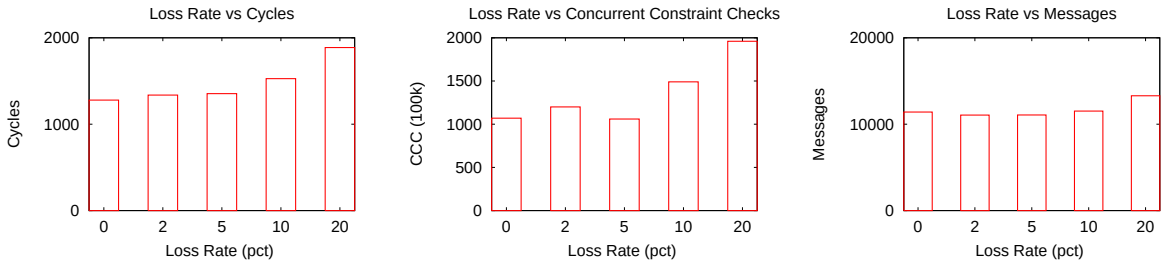


Figure 8: Chain with 24 sensors and 7 targets. Increasing message loss levels gracefully degrade performance in cycles (left), computation (middle) and communication (right).

8 Conclusions

We evaluated a simple timeout mechanism for both reducing communication overhead and preventing deadlocks when messages are lost in asynchronous distributed constraint optimization problem solving. We showed that this method allows the Adopt algorithm to tolerate message loss and still terminate with the globally optimal solution. Empirical results in a distributed sensor network domain showed that the mechanism dramatically reduced the amount of communication necessary to find the globally optimal solution, up to 80% in some experiments and algorithm performance decreases gracefully as message loss rate is increased.

References

[1] C. Bessire, A. Maestre, and P. Meseguer. Distributed dynamic backtracking. In *International Joint Conference on AI Workshop on Distributed Constraint Reasoning*, 2001.

[2] K.M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 1985.

[3] C. Fernandez, R. Bejar, B. Krishnamachari, C. Gomes, and B. Selman. Communication and computation in distributed csp algorithms. In *Distributed Sensor Networks: A Multiagent Perspective*, Series on Multiagent Systems. Kluwer academic publishers, 2003.

[4] M. Hannebauer and S. Mller. Distributed constraint optimization for medical appointment scheduling. In *Proceedings of the Fifth International Conference on Autonomous Agents*, 2001.

[5] K. Hirayama and M. Yokoo. Distributed partial constraint satisfaction problem. In G. Smolka, editor, *Principles and Practice of Constraint Programming*, pages 222–236. 1997.

[6] H. Jung, M. Tambe, A. Barrett, and B. Clement. Enabling efficient conflict resolution in multiple spacecraft missions via dcsp. In *Proceedings of the NASA workshop on planning and scheduling*, 2002.

[7] V. Lesser, C. Ortiz, and M Tambe, editors. *Distributed sensor nets: A multiagent perspective*. Kluwer academic publishers, 2003.

[8] R. Mailler and V. Lesser. A mediation based protocol for distributed constraint satisfaction. In *The Fourth International Workshop on Distributed Constraint Reasoning*, 2003.

[9] R. Mailler, V. Lesser, and B. Horling. Cooperative negotiation for soft real-time distributed resource allocation. In *Proceedings of Second International Joint Conference on Autonomous Agents and MultiAgent Systems*, 2003.

[10] Amnon Meisels, Eliezer Kaplansky, Igor Razgon, and Roie Zivan. Comparing Performance of Distributed Constraints Processing Algorithms. In *Proc. Workshop on Distributed Constraint Reasoning (AAMAS 2002)*, 2002.

[11] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *Proc of Autonomous Agents and Multi-Agent Systems*, 2003.

[12] P. Scerri, P.J. Modi, W-M. Shen, and M. Tambe. Applying constraint reasoning to real-world distributed task allocation. In *Proc. Workshop on Distributed Constraint Reasoning at AAMAS-2002*.

[13] M.C. Silaghi, D. Sam-Haroud, and Boi Faltings. Asynchronous search with aggregations. In *Proceedings of National Conference on Artificial Intelligence*, 2000.

[14] W. R. Stevens. *TCP/IP Illustrated, Volume 1*. Addison-Wesley, 1994.

[15] M. Yokoo. *Distributed Constraint Satisfaction:Foundation of Cooperation in Multi-agent Systems*. Springer, 2001.

[16] M. Yokoo and K. Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *Proceedings of International Conference on Multiagent Systems*, 1998.

Index