On Communication in Solving Distributed Constraint Satisfaction Problems

Hyuckchul Jung[1] and Milind Tambe[2]

[1] Florida Institute for Human and Machine Cognition, USA
[2] Department of Computer Science, University of Southern California, USA

Abstract. Distributed Constraint Satisfaction Problems (DCSP) is a general framework for multi-agent coordination and conflict resolution. In most DCSP algorithms, inter-agent communication is restricted to only exchanging values of variables, since any additional information-exchange is assumed to lead to significant communication overheads and to a breach of privacy. This paper provides a detailed experimental investigation of the impact of inter-agent exchange of additional legal values among agents, within a collaborative setting. We provide a new run-time model that takes into account the overhead of the additional communication in various computing and networking environments. Our investigation of more than 300 problem settings with the new run-time model (i) shows that DCSP strategies with additional information-exchange can lead to big speedups in a significant range of settings; and (ii) provides categorization of problem settings with big speedups by the DCSP strategies based on extra communication, enabling us to selectively apply the strategies to a given domain. This paper not only provides a useful method for performance measurement to the DCSP community, but also shows the utility of additional communication in DCSP.

1 Introduction

Distributed, collaborative agents play an important role in large-scale multiagent applications such as sensor networks [6]. Collaborative agents in such applications must coordinate their plans, resolving conflicts, if any, among their action or resource choices. Distributed Constraint Satisfaction Problems (DCSP) is a major technique in multiagent coordination and conflict resolution in collaborative settings [10]. DCSP provides rich foundation for the representation of multiagent coordination and conflict resolution, and there exist highly efficient baseline algorithms [3, 7, 9, 10].

In most DCSP algorithms, inter-agent communication is restricted to only exchanging values of variables, since any additional information-exchange is assumed to lead to significant communication overheads, a breach of privacy, and knowledge transformation cost [10]. We refer to this restriction of only communicating values of variables as *value-only communication*. However, as large-scale systems based on such value-only communication get developed, it is critical to re-examine this commitment to value-only communication that has now become the foundation of DCSP. Indeed, it is feasible that, by unnecessarily subscribing to such value-only communication, researchers may be forced to compromise on correctness or quality of solutions; and/or forced to develop unnecessarily complex algorithms. Could eliminating or diluting this restriction of value-only communication lead to significant speedups, or would that lead to additional overheads? Such a re-examination of the communication commitment in DCSP may imply potentially significant enhancements to the current DCSP algorithms.

We examine the impact of value-only communication in collaborative agent applications, where agents are homogeneous or at least do not face significant difficulties

in communicating their potential choices of values to each other. In such collaborative agent applications, some of the key reasons for restricting to value-only communication do not hold. In particular, there are three key reasons provided in the literature [10] for value-only communication: (i)Maintaining privacy, (ii) Difficulty of knowledge transformation in heterogeneous agent settings; (iii) Overheads of extra communication.

However, collaborative agents have no reason to maintain privacy from other agents, and many domains with homogeneous agents do not have a problem in knowledge transformation. The central remaining question is thus of communication overheads, and loosening the restriction of value-only communication can indeed add to the communication cost in DCSP. This tradeoff in the potential speedup due to extra communication vs the cost of communication is the central tradeoff that is at the heart of this paper. Various aspects of different types of domains need to be considered in the analysis (e.g., communication or local computation cost).

In earlier work, we introduced DCSP techniques with additional information exchange [5]. However, since the investigation was limited to limited settings, the performance of such DCSP techniques was not fully evaluated in a large set of realistic domains. Furthermore, the overhead from extra communication was never analyzed. In this paper, we present a comprehensive, detailed analysis over a large range of realistic domain settings. For the analysis, we develop a new run-time model that takes into account extra communication overhead in various computing and networking environments since the performance metric widely used in the DCSP literature, *cycles* explained in Section 3, does not take into account the overhead of additional information exchange (i.e., increased message size and number).

To evaluate the performance of DCSP techniques based on extra communication in different domains, we systematically investigate more than 300 problem settings in a large problem space with more than 200,000 experimental runs, using the new run-time model. Our investigation (i) shows that DCSP strategies with additional information-exchange can lead to big speedups in a significant range of settings; and (ii) provides categorization of problem settings where big speedups are achieved by the DCSP strategies to guide which DCSP strategy to apply given a domain.

2 Background

DCSP provides an abstract formal framework to model coordination and conflict resolution in many multiagent applications such as distributed sensor networks [6]. DCSP is a distributed version of CSP (Constraint Satisfaction Problems) [10]. CSP is commonly defined by a set of n variables, $X = \{x_1, ..., x_n\}$, each element associated with value domains $D_1, ..., D_n$ respectively, and a set of k constraints, $\Gamma = \{C_1, ..., C_k\}$. A solution in CSP is the value assignment for the variables which satisfies all the constraints in Γ. In DCSP, variables and constraints are distributed among multiple agents. A constraint defined only on variables belonging to a single agent is called a *local constraint*. In contrast, an *external constraint* involves variables of different agents. Solving a DCSP requires that agents not only solve their local constraints, but also communicate with other agents to satisfy external constraints.

A major characteristic of most DCSP algorithms is that they have focused on *value-only communication*: agents communicate only their intended values for the objects on which they need to agree [3, 7, 10]. That is, while the value selection is based on

each agent's local knowledge and local situation, agents do not communicate such information. However, a few different approaches (based on the communication of local information between agents) were recently presented [5, 8, 9].

In this section, we describe two algorithms as representative examples. One is Asynchronous Weak Commitment search algorithm [10], one of the most advanced DCSP algorithms, in which agents communicate only selected values, and the other is Locally Cooperative DCSP algorithm [5] in which agents communicate selected values plus local information and the communicated information is used for value ordering.

2.1 Asynchronous Weak Commitment (AWC) Search Algorithm

In the AWC algorithm, agents asynchronously assign values to their variables and communicate the values to neighboring agents with shared binary constraints. Each variable has a priority that changes dynamically during search. A variable is consistent if its value does not violate any constraints with higher priority variables. A solution is a value assignment in which every variable is consistent.

To simplify the description of the algorithm, suppose that each agent has exactly one variable. When the value of an agent's variable is not consistent with the values of its neighboring agents' variables with higher priorities, there can be two cases: (i) a *good* case where there exists a consistent value in the variable's domain; (ii) a *nogood* case that lacks a consistent value. In the good case with one or more value choices available, an agent selects a value that minimizes the number of conflicts with lower priority agents. On the other hand, in the nogood case, an agent selects a new value that minimizes the number of conflicts with all of its neighboring agents, and increases its priority to *max+1*, where *max* is the highest priority of its neighboring agents.

2.2 Locally Cooperative DCSP (LCDCSP) Algorithm

In the LCDCSP algorithm [5], agents take into account the flexibility (choice of values) given to other agents by their value choices in selecting new values. The LCDCSP algorithm is based on the AWC but has a different mechanism in value ordering (which is enabled by extra communication of local constraints). To elaborate this notion of cooperative value selection, the followings was defined in [5]:

- **Definition 1**: For a value $v \in D_i$ and a set of agents $N_i^{sub} \subseteq N_i$, *flexibility function* is defined as $f^{\oplus}(v, N_i^{sub}) = \oplus(c(v, A_j))$ where (i) $A_j \in N_i^{sub}$; (ii) $c(v, A_j)$ is the number of values of A_j that are consistent with v; and (iii) $\oplus$, referred to as a *flexibility base*, can be sum, min, max, $product$, etc.

Based on the flexibility, four different techniques are defined for value selection:

- $S_{min-conflict}$: Each agent A_i selects a value based on min-conflict heuristics (the original value ordering method in the AWC algorithm);
- S_{high} (S_{low}): Each agent A_i attempts to give maximum flexibility towards its higher (lower) neighboring agents by selecting a value v that maximizes $f^{\oplus}(v, N_i^{high})$ ($f^{\oplus}(v, N_i^{low})$);
- S_{all}: Each agent A_i selects a value v that maximizes $f^{\oplus}(v, N_i)$, i.e. max flexibility to all neighbors.

These four different techniques can be applied to both the *good* and the *nogood* case described in Section 2.1. (Refer to [5] for detailed information.) Therefore, there are sixteen combinations for each flexibility base. While the LCDCSP apporach has relation to

a popular centralized CSP technique, the *least constraining value* heuristic [4], it is not a simple mapping of the *least constraining value* heuristic onto the DCSP framework. Agents can explicitly reason about which agents to consider most with respect to the constrainedness given towards neighboring agents.

3 Performance Measurement

To evaluate approaches with different types of information exchange (as shown above), we need a new run-time model (Section 3.2) that takes into account the overhead of extra communication (required for the LCDCSP algorithm) since existing performance metrics (described in Section 3.1) do not properly assess such communication overhead.

3.1 Existing Method

Since it has been practically difficult to access a real large-scale distributed system (with hundreds of nodes), the standard methodology in the field [3, 9, 10] is to implement a synchronized distributed system which is a model of distributed system where every agent synchronously performs the following three steps (called a *cycle*): (i) Agents receives all the messages sent to them in the previous cycle; (ii) Agents resolve conflicts, if any, and determine which message to send; (iii) Agents send messages to neighboring agents. Given such a synchronized distributed system, it is difficult to directly measure the run-time for real distributed conflict resolution. However, in the literature, as a compromise, researchers have used hardware independent metrics such as *cycles* and *constraint checks* defined below.

- *Cycles*: The number of cycles until a solution is found. Total time for conflict resolution is expected to be proportional to *cycles* [10].
- *Constraint checks*: The total number of the maximum number of constraint checks at each cycle until a solution is found. More specifically, at each cycle, a bottleneck agent (which performs the most constraint checks) is identified, and the numbers of constraint checks from bottleneck agents (which may vary at each cycle) are summed up over all cycles. This is a main indicator for local computation time.

In the DCSP research community, *cycles* is used as a major metric for performance evaluation since the amount of local computation and communication that each agent solves mostly remains same in most of previous DCSP approaches [3, 9, 10]: the difference is in the protocol for passing values and controlling backtracking. However, cycles has the following shortcomings:

- Local computation overhead: For the hardware with limited computing power, the time for local computation may not be ignored, and there can be a variation in local computation depending on constraint checks.
- Message communication overhead: While *cycles* assumes that uniform time is taken at each communication phase, the time for message communication often depends on the size/number of messages.

3.2 Analytical Model for Run-time

While the *cycles* (a major DCSP metric) described above can be used as approximate measurements, it does not properly assess the performance of algorithms like LCDCSP which do not properly assess the additional computation and communication overhead

from the local information exchange. Therefore, we need a new model to take into account such overheads as part of the run-time. In this section, we present an analytical model for run-time measurements which takes into account various message processing/communication overhead in different computing/networking environments.

The local computation processed by an agent at each cycle consists of processing received messages, performing constraint checks, and determining which message to send for its neighbors. The run-time taken by an agent for a cycle is the sum of the local computation time and the communication time for the agent's outgoing messages. Our new run-time model is based on the data collected from the experimentation on a synchronized distributed system. The following terms are defined for the model:

- n_i^k: incoming message number for agent i at cycle k
- $s_i^k(j)$: size of j^{th} incoming message for agent i at cycle k
- $\mathcal{I}$(l): computation time to process one incoming message (whose size is l)
- c_i^k: number of constraint checks by agent i at cycle k
- t: computation time to perform one constraint check
- o_i^k: number of outgoing message for agent i at cycle k
- u_i^k: size of an outgoing message for agent i at cycle k
- $\mathcal{O}(m)$: computation time to process an outgoing message (whose size is m)
- $\mathcal{T}(d)$: communication time to transmit an outgoing message (whose size is d)

In a synchronous distributed system, at each cycle, agents synchronously start their local computation and communication. Thus, the run-time for a cycle is dominated by an agent which requires maximum time for its local computation and communication.

- *Run-time for a cycle k* ($\mathcal{R}(k)$) = $max_{i \in Ag}(\mathcal{L}_i^k + \mathcal{C}_i^k)$ where Ag is a set of agents in a given system, $\mathcal{L}_i^k$ is the *local computation time of agent i at cycle k*, and $\mathcal{C}_i^k$ is the *communication time of agent i at cycle k*.

Here, $\mathcal{L}_i^k$ and $\mathcal{C}_i^k$ are computed by the following equations:

- *Local computation time of agent i at cycle k* ($\mathcal{L}_i^k$) = $\sum_{j=1}^{n_i^k}(\mathcal{I}(s_i^k(j)))$ (time to process received messages) + $c_i^k \times t$ (time to perform constraint checks) + $o_i^k \times \mathcal{O}(u_i^k)$ (time to process outgoing messages)
- *Communication time of agent i at cycle k* ($\mathcal{C}_i^k$) = $o_i^k \times \mathcal{T}(u_i^k)$ (time for transmitting a message whose size is u_i^k for o_i^k times)

Finally, the total run-time is the sum of run-time ($\mathcal{R}(k)$) for each cycle:

- *Total run-time* = $\sum_{k=1}^{K}(\mathcal{R}(k))$ where K is the number of total cycles.

While the above model aims to provide a metric which takes into account message processing/communication overhead (based on message size/number), it is flexible enough to subsume the existing method of performance measurement (Section 3.1):

- *Constraint checks* corresponds to the total run-time (defined above) where $t = 1$, $\mathcal{I}(\cdot) = \mathcal{O}(\cdot) = 0$ and $\mathcal{C}_i^k = 0$ (i.e., no communication/message-processing cost).
- *Cycles* corresponds to the total run-time under the assumption that $t = 0$ (the cost for constraint checks is zero), $\mathcal{I}(\cdot) = \mathcal{O}(\cdot) = 0$ (message processing cost is zero), and $\mathcal{C}_i^k = 1$ (a constant communication time independent of message size/number).

As the first analytical model for the performance measurement in DCSP which takes into account the overhead based on message size and number, the above model could provide a useful method for performance measurement to the DCSP community. Furthermore, as shown below, the model shows interesting results as the parameters for message processing/communication overhead vary.

4 Performance Analysis

While we focus on the domain where agents' interaction topology is regular [3], there can be variations (e.g., problem hardness) in different problem settings that arise within the domain. In this section, we provide various problem settings controlled by several parameters. Systematic changes in the parameters generate a wide variety of problem settings, and enable us to evaluate the performance of the strategies and find their communication vs. computation trade-offs in different situations. Here, parameter selection is motivated by the experimental investigation in the CSP/DCSP literature [10].

First, we vary the density of regular graphs by changing the number of neighboring agents: (i) *Hexagonal topology*: Each agent is surrounded by three agents (separated by 120 degrees); (ii) *Grid topology*: Each agent is surrounded by four neighboring agents (separated by 90 degrees); (iii) *Triangular topology*: Each agent is surrounded by eight neighboring agents (separated by 45 degrees). The purpose of trying three different regular graphs is to investigate the impact on performance by the degree of connectivity (number of interactions for each agent).

Second, given a topology (among the three topologies above), we make variations in constraint compatibility which has shown a great impact on the hardness of problems [10]. We distinguish external constraints from local constraints in defining the constraint tightness to analyze the effect from each constraint:

1. External constraint compatibility: Given an external constraint, for a value in an agent's domain, the percentage of compatibility with neighboring agents' values is defined. The percentage varies from 30% to 90% with intervals of 30%. Note that 0% case and 100% case are not tried since there is no solution for 9% case and every value assignment is a solution for 100% case.
2. Local constraint compatibility: Given a local constraint, a portion of agents' original domains is not allowed. We make the following two variations in local constraint: (i) The percentage of locally constrained agents changes from 0% to 100% (0%, 30%, 60%, 90%, 100%); and (ii) Given a local constraint, the portion of allowed values varies from 25% to 75% (25%, 50%, 75%). Here, 0% and 100% are not tried since 0% case gives agents empty domain and 100% case has no effect of having a local constraint.

Third, we vary the number of domain values from 10 to 80 (10, 40, 80) to check how different domain sizes have an impact on the performance and trade-offs of the strategies. Given the above variations, the total number of settings is 351, and we evaluate the performance of the two DCSP strategies (presented in Section 2) on each setting. Note that the LCDCSP strategy can have different value selection techniques introduced in Section 2.2. For a given problem setting, the performance of strategies is measured on 35 problem instances which are randomly generated by the problem setting (defined with the above parameters).

For each setting, seventeen strategies (sixteen strategies defined in this paper plus the original AWC strategy) are tried for each problem instances. Thus, the total number of experimental runs is 208,845 ($= 351 \times 35 \times 17$). [4] Note that, for the sixteen

[3] In real applications such as sensor networks [6], agents are often arranged in regular networks.

[4] To conduct the experiments within a reasonable amount of time, the number of cycles was limited to 1000 for each run (a run was terminated if this limit exceeded).

LCDCSP strategies, sum is used as a flexibility base (the original AWC strategy is *min-conflict* strategy without extra communication of local information). We set the number of agents as 512 since, in real applications such as sensor networks [6], the number of agents in hundreds is considered to be large-scale.

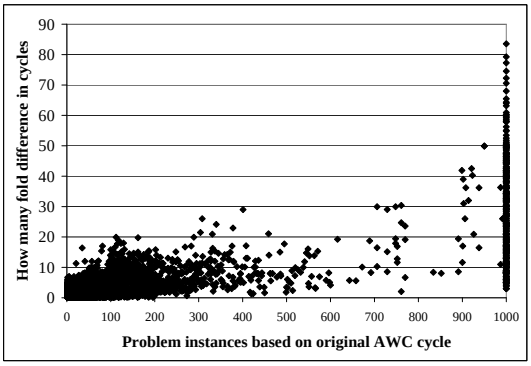

Fig. 1. Speedup by LCDCSP strategies for individual problem instances

4.1 Categorization of Problem Settings with Big Speedups by LCDCSP

In Figure 1, the horizontal axis plots problem hardness for each individual problem instance (based on the *cycles* by the AWC strategy), and the vertical axis plots how many speedup (i.e., how many fold reduction in cycles) is achieved by the best LCDCSP strategies for each problem instance. [5] The results in Figure 1 indicates that LCDCSP strategies show performance improvement for majority of problem instances across different problem hardness: while there is a variation in performance improvement, the speedups do not come from only a few exceptional cases.

Local constraint compatibility	External constraint compatibility	Domain size	Ratio of locally constrained agents	Speedup at Each Topology		
				Hexagonal	Grid	Triangular
25%	30%	10	30%	High		
			60%	Moderate		
			others	Low		
		40 & 80	60%	Low	Moderate	Low
			others	Low		
	60%	40	90%	Low	High	Low
		10 & 80	others	Low		
	90%	*	*	Low		
50% & 75%	*	*	*	Low		

Table 1. Speedups based on problem class

Local constraint compatibility	Domain size	Ratio of locally constrained agents	Speedup
25%	10	0 ∼ 100%	Low
25%	40	90%	High
		others	Moderate
25%	80	0 ∼ 100%	Moderate
50%	10, 40, 80	0 ∼ 100%	Moderate
75%	10, 40, 80	0 ∼ 100%	Moderate

Table 2. Maximum speedup in the problem settings where topology is grid, and external constraint compatibility is 60%

Table 1 and 2 show how much speedu can be achieved by the best LCDCSP strategies for a group of problem settings classified by the parameters introduced above. Note that this categorization is not exhaustive, and focuses on problem settings (not on individual problem instances). In Table 1 and 2, high/moderate/low speedup respectively indicates “more than five”/”between three and four”/”less than two”-fold speedup by LCDCSP strategies over the AWC strategy. The following is the summarized result shown in Table 1 and 2:

[5] Selecting the best LCDCSP strategies were based on empirical results.

– When external constraint compatibility is low (30%),
  - For each topology, high performance improvement is achieved when local constraint compatibility is low (25%) and domain size is small (10).
    * A big speedup by LCDCSP strategies is shown unless agents are either totally unconstrained in local constraints(0%) or totally constrained (100%).
    * For grid topology, a big speedup is also shown when domain size is large (80), and the ratio of locally constrained agents is moderate (60%) or high (90%). However, when all agents are locally constrained (100%), no speedup is shown.
  - When local constraint compatibility increases or domain size gets larger, LCDCSP shows low speedup.
– When external constraint compatibility is moderate (60%) in grid topology,
  - High performance improvement is achieved when local constraint compatibility is low (25%) and domain size is moderate (40).
    * A big speedup by the best LCDCSP strategy is shown when the ratio of locally constrained agents is high (90%). However, note that, when the ratio is 100%, there is no big speedup since all the problems in the setting are easy regardless of strategies to be applied.
– When external constraint compatibility is 90%, the speedup is relatively small since the problem settings with 90% external constraint compatibility is easier than other settings (taking less than 30 *cycles* in general) so that there is no big difference in *cycles* between the AWC strategy and LCDCSP strategies.

4.2 Performance in Run-time Analytical Model

In this section, we present how the performance results (e.g., speedup) changes with the analytical run-time model in Section 3.2 compared with the results based on *cycles*. The parameters specified in this section assume a realistic domain where message communication overhead dominates local computation cost and message processing overhead is relatively smaller than communication overhead (but cannot be ignored). In defining the parameters for such a domain, two different properties for message processing and communication overhead are considered:

– Property 1: Message processing/communication overhead mainly depends on the size of messages to process/communicate.
– Property 2: Message processing/communication overhead mainly depends on the number of messages to process/communicate: Message is processed as a bundle or message communication delay is dominated by message contention.

Message Size as a Main Factor for Message Processing & Communication Overhead For a domain where message size is a main factor for message processing and communication overhead, parameters for the run-time model are set as follows:

– $\mathcal{I}(l) = l \times t \times \alpha$ and $\mathcal{O}(m) = m \times t \times \alpha$: Message processing is assumed to be slower than a constraint check by two order of magnitude. To simulate such a difference, α is set as 100 or 1000.
– $\mathcal{T}(d) = d \times t \times \beta$: To simulate the situation where communication overhead dominates local computation cost, β is set as 1000 or 10000.

Case	Speedup by LCDCSP strategies: Based on *cycles*	Based on run-time model: $\alpha = 100\ \beta = 1000$	$\alpha = 100\ \beta = 10000$	$\alpha = 1000\ \beta = 1000$	$\alpha = 1000\ \beta = 10000$
1	11	7	7	7	7
2	10	9	9	8	9
3	37	21	21	20	21
4	14	4	7	5	7
5	11	7	8	7	8
6	44	33	33	31	33

Table 3. Speedup change in run-time model

Table 3 shows the speedup by the best LCDCSP strategy for prototypical settings given different α and β. In Table 3, the speedup based on the run-time model for different α and β is less than the speedup based on *cycles*: i.e., the performance of LCDCSP strategies with the run-time model appear to be worse than the *cycle*-based performance.

The decrease in speedup with the *run-time* model is due to the fact that LCDCSP strategies have larger message size to process/communicate and more constraint checks (to compute flexibility towards neighbors) than the AWC strategy. The analysis with other α and β values show similar results.

While we present limited data because of space limit, the analysis shows that, as domain size or graph density (i.e., the number of neighbors) increases, the difference in message size and constraint checks between the AWC strategy and LCDCSP strategies also increases, leading to significant decrease in speedup for LCDCSP strategies.

Message Number as a Main Factor for Message Processing & Communication Overhead For a domain where message number is a main factor for message processing and communication overhead (message processing & communication time is independent of message size), parameters for the run-time model are set as follows:

– $\mathcal{I}(l) = t \times \alpha$ and $\mathcal{O}(m) = t \times \alpha$; $\mathcal{T}(d) = t \times \beta$

Case	Speedup by LCDCSP strategies: Based on *cycles*	Based on run-time model: $\alpha = 100\ \beta = 1000$	$\alpha = 100\ \beta = 10000$	$\alpha = 1000\ \beta = 1000$	$\alpha = 1000\ \beta = 10000$
1	11	9	10	9	10
2	10	10	10	9	10
3	37	37	37	38	37
4	14	6	12	9	13
5	11	10	10	9	10
6	44	46	44	54	47

Table 4. Speedup change in run-time model

Here, the values of α and β are same as above. Table 4 shows the speedup by the best LCDCSP strategy for the same prototypical settings (presented in Table 3). In Table 4, the speedup based on the run-time model for different α and β is very similar with the speedup based on *cycles* in general. The main reason is that the number of messages to communicate is decided by the number of neighbors (i.e., graph density) which is static. While there can be a large difference in constraint checks depending on the graph density and the domain size, when the message processing or communication overhead dominates (the difference in *constraint checks* becomes insignificant), the performance of the AWC strategy and LCDCSP strategies depends on *cycles* because of little difference in message size.

This analysis shows that, when the overhead of message processing and communication is mainly decided by message number (not size) and dominates local computation overhead (the difference in *constraint checks* is not significant), *cycles* can be a reasonable measurement to compare strategy performance. Note that, using this analytical model, we can simulate various computing and networking environments by changing (i) the values of α and β (different weights to message processing/communication overheads) or (ii) the cost functions.

5 Related Work and Conclusion

While significant works have focused on variable or agent ordering in DCSP [1, 3, 10], value ordering techniques which exploit additional information-exchange have not received enough attention, and little investigation has been done for performance measurement which takes into account extra communication overhead. While communicating local information has been investigated in DCSP [8, 9], the communication overhead in different computing/networking environments was not properly evaluated. Fernandez et al. investigated the effect of communication delays on the performance of DCSP algorithms [2]. However, their investigation was limited to random effects and did not take into account the impact from extra value communication.

In this paper, we investigate the impact of inter-agent exchange of additional information which has not been exploited in conventional DCSP algorithms. We provide a new run-time model for DCSP performance measurement that takes into account the overhead of extra communication. Experimental results from extensive systematic investigation show that DCSP strategies which exploit additional information-exchange indeed improve performance in a significant range of problem settings, in particular when message processing/communication overhead dominates local computation overhead. We also provide categorization of problem settings with big speedups by the DCSP strategies to guide strategy selection. This paper not only provides a useful method for performance measurement to the DCSP community, but also shows the utility of additional information exchange in DCSP.

References

1. A. Armstrong and E.H. Durfee. Dynamic prioritization of complex agents in distributed constraint satisfaction problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1997.
2. C. Fernandez, R. Bejar, B. Krishnamachari, C. Gomes, and B. Selman. Communication and computation in distributed csp algorithms. In V. Lesser, C. Ortiz, and M. Tambe, editors, *Distributed Sensor Networks*. Kluwer Academic Publishers, 2003.
3. Youssef Hamadi, Christian Bessière, and Joël Quinqueton. Backtracking in distributed constraint networks. In *Proceedings of the European Conference on Artificial Intelligence*, 1998.
4. R. M. Haralick and G. L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.
5. H. Jung and M. Tambe. Performance models for large scale multiagent systems: Using pomdp building blocks. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2003.
6. V. Lesser, C. Ortiz, and M. Tambe, editors. *Distributed Sensor Networks: a Multiagent Perspective*. Kluwer Academic Publishers, 2003.
7. P. Modi, H. Jung, M. Tambe, W. Shen, and S. Kulkarni. A dynamic distributed constraint satisfaction approach to resource allocation. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 2001.
8. E. Monfroy and J. H. Rety. Chaotic iteration for distributed constraint propagation. In *ACM Symposium on Applied Computing*, 1999.
9. M. Silaghi, D. Sam-Haroud, and B. Faltings. Consistency maintenance for abt. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 2001.
10. M. Yokoo. *Distributed Constraint Satisfaction: Foundations of Cooperation in Multi-Agent Systems*. Springer, 2000.