The Phase Transition in Distributed Constraint Satisfaction Problems: First Results

Katsutoshi Hirayama
Kobe University of Mercantile Marine
5-1-1 Fukae-minami-machi, Higashinada-ku, Kobe 658-0022, JAPAN
hirayama@ti.kshosen.ac.jp

Makoto Yokoo
NTT Communication Science Laboratories
2-4 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0237, JAPAN
yokoo@cslab.kecl.ntt.co.jp

Katia Sycara
The Robotics Institute
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213, U.S.A.
katia@cs.cmu.edu

ABSTRACT

We examine the behavior of a distributed constraint satisfaction algorithm. More specifically, we measured the communication and computation costs of a distributed constraint satisfaction algorithm when varying the numbers of *intra-agent constraints* (constraints which are defined over variables of one agent) and *inter-agent constraints* (constraints which are defined over variables of multiple agents) for distributed graph 3-coloring problems. What we found from our experiments are: 1) the *phase transition* occurs in the narrow region on the x-y plane (where the x-axis is the number of intra-agent constraints and the y-axis is the number of inter-agent constraints), where the sum of intra- and inter-agent constraints is constant, and problems in the phase transition region are likely to be expensive in terms of the median communication and computation costs; 2) among problems in the phase transition region, problems with a smaller number of intra-agent constraints and a larger number of inter-agent constraints can be more expensive; 3) for a fixed total number of variables, problems in the phase transition region can be more expensive as the number of agents increases or the number of variables per agent decreases.

1. INTRODUCTION

Many problems in artificial intelligence can be formalized as constraint satisfaction problems (CSPs). A CSP consists of *variables* with finite and discrete domains and *constraints* over some subsets of variables. A solution to a CSP is a set of all variable values which satisfies all constraints.

A CSP is a typical NP-complete problem and thus it has been said that no domain-independent algorithm has a worst-case search cost that is not exponential in the size of the problem. However, recent studies have observed that for certain CSPs like propositional satisfiability, graph coloring, and binary CSPs, one order parameter exists on which the median computational cost of an algorithm shows an *easy-hard-easy* pattern[3, 12, 13]. Search cost grows very rapidly when a problem size increases in the hard region on the parameter, but relatively slowly in the other easy regions. In other words, really computationally expensive problems are concentrated on some specific region on the parameter.

For example, in a graph coloring problem, which consists of a graph, a specified number of available colors, and the requirement to find a color for each node of the graph such that no adjacent nodes have the same color, such an order parameter is the ratio of the number of links to the number of nodes. For randomly generated graph 3-coloring problems, it has been shown that the peak of the median computational cost is located at 2.3, i.e., at which the number of links is 2.3 times as many as the number of nodes[9].

In this paper, we examine the behavior of a distributed constraint satisfaction algorithm when solving distributed CSPs[16]. A distributed CSP is a CSP where variables and constraints are distributed among multiple agents, each of which tries to solve its own CSP. Various application problems in multiagent systems that are concerned with finding a consistent combination of agent actions (e.g., distributed resource allocation problems[4], distributed scheduling problems[15], distributed interpretation tasks[11], and multi-agent truth maintenance tasks[10]) can be formalized as distributed CSPs.

Distributed algorithms for solving distributed CSPs (distributed constraint satisfaction algorithms) have recently been developed by several authors [1, 7, 8, 16, 17, 18]. For solving a distributed CSP by a distributed constraint satisfaction algorithm, since agents usually have *intra-agent con-*

straints (constraints which are defined over variables of one agent) and *inter-agent constraints* (constraints which are defined over variables of multiple agents), they have not only to perform local computation to satisfy their intra- and inter-agent constraints, but also to communicate with other agents to satisfy their inter-agent constraints. The efficiency of a distributed constraint satisfaction algorithm depends on its communication cost and computation cost, and both can vary with the numbers of intra- and inter-agents constraints. Therefore, it is important to know how the numbers of intra- and inter-agent constraints affect the communication and computation costs of a distributed constraint satisfaction algorithm because such an information may give us a hint to develop a more efficient distributed constraint satisfaction algorithm.

This paper presents first experimental results on the communication and computation costs of a distributed constraint satisfaction algorithm when we vary the numbers of intra- and inter-agent constraints for distributed graph 3-coloring problems[1]. As we will show, both costs show an interesting pattern on the x-y plane (where the x-axis is the number of intra-agent constraints and the y-axis is the number of inter-agent constraints), and its pattern is closely related to the phase transition in the ratio of soluble problems. Since the pattern seems to be independent of algorithms and problem generation methods, we can expect that our first results can be the basis of theoretical analyses.

This paper is organized as follows. In section 2, we provide the definition of distributed CSP and illustrate a distributed graph coloring problem as an example. In section 3, we describe a distributed constraint satisfaction algorithm and a method to generate distributed graph 3-coloring problems used in our experiments. Experimental results and discussion are also included in this section. Finally, in section 4, we summarize our findings and show our future work.

2. DISTRIBUTED CONSTRAINT SATISFACTION PROBLEM

A CSP consists of *variables* with finite and discrete domains and *constraints* over some subsets of variables. A constraint can be described as a *nogood*, i.e., a set of variable values that are prohibited for the variables. A solution to a CSP is an assignment of a value to every variable that violates no nogood.

A distributed CSP is a CSP where variables and constraints are distributed among multiple agents. It consists of:

- a set of agents, $1, 2, \ldots, k$
- a set of CSPs, $P_1, P_2, \ldots, P_k$, such that P_i belongs to agent i ($i = 1, \ldots, k$)

A solution to a distributed CSP is a set of solutions to CSPs of all agents.

[1]Hamadi has reported experimental results on the cost of distributed consistency algorithm[6]. On the other hand, our focus is on the cost of distributed search algorithm

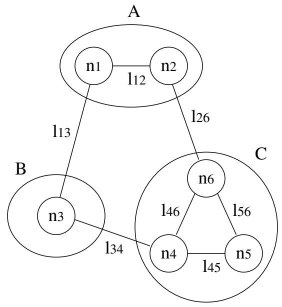

Figure 1: Distributed graph coloring problem

A simplistic distributed CSP is a problem where CSPs of all agents are completely separated. Such a problem is not interesting in this context because it can be solved by each agent that performs a constraint satisfaction algorithm independently. However, we usually assume that each agent has a CSP which includes inter-agent constraints. An *inter-agent constraint* is a constraint which is defined over variables of multiple agents. On the other hand, an *intra-agent constraint* is a constraint which is defined over variables of one agent. By considering inter-agent constraints, the search process over a distributed CSP is made more complicated because communication is required among agents to satisfy inter-agent constraints.

Figure 1 illustrates an example of distributed CSP, a distributed graph coloring problem. A graph coloring problem is a problem where it is required to find a color (among available colors) for each node of a given graph such that no adjacent nodes have the same color. By considering a node as a variable and a link as a constraint, a graph coloring problem can be mapped into a CSP. A distributed graph coloring problem is a problem where nodes and links are distributed among agents such that each agent has some nodes and all links that are connected to the nodes. In figure 1, there are three agents, A, B and C, and each of which has nodes in the corresponding ellipse and links that are connected to the nodes. For example, agent A has the nodes, n_1 and n_2, and the links, l_{12}, l_{13} and l_{26}. In other words, agent A has a CSP with the variables, n_1 and n_2, and the constraints, such as not_equal(n_1, n_2), not_equal(n_1, n_3) and not_equal(n_2, n_6). Note that there are 7 links (constraints) in this figure and among these constraints three are inter-agent constraints, l_{13}, l_{34} and l_{26}, and the other four are intra-agent constraints, l_{12}, l_{45}, l_{46} and l_{56}.

One possible algorithm for solving distributed CSPs is a *centralized method*. In this method, all agents send their CSPs to one specific site, and at this site the entire CSP is solved by a (centralized) constraint satisfaction algorithm. This method has the advantage that an algorithm is able to use global information about a problem. On the other hand, it misses the benefit from performing parallel computation by multiple agents and suffers from the disadvantage caused by gathering all CSPs[2].

[2]For example, in order to gather all CSPs, an agent may have to convert its own problem description into a common one, or an agent may make its

Recently, some researchers have developed distributed algorithms for solving distributed CSPs (distributed constraint satisfaction algorithms) that employ a distributed architecture[1, 7, 8, 16, 17, 18]. When solving a distributed CSP using a distributed constraint satisfaction algorithm, agents have not only to do their local computation to satisfy intra- and inter-agent constraints, but also to do communication among agents to satisfy inter-agent constraints. Thus, we should consider both communication and computation costs when we develop an efficient distributed constraint satisfaction algorithm.

3. EXPERIMENTS

3.1. ALGORITHM

We used a simple backtracking type of algorithm in our experiments. It is basically the same one as the *asynchronous backtracking algorithm*[16] except that it is extended so that it can handle a situation where an agent has multiple local variables. Figure 2 shows the main part of this algorithm (the procedure when an agent receives an *ok?* message) in detail. The outline of this algorithm is as follows. Note that we are using the graph coloring terminology.

- A priority order among agents is introduced such that the agent with more links has higher priority (ties are broken deterministically by comparing agent identifiers). Also, for each agent, a priority order among (internal) nodes is introduced such that the node with more connected links has higher priority (ties are broken deterministically by comparing node identifiers).
- Each agent starts the procedure by randomly assigning a color for each of its nodes and sending these colors to *neighbors* (a set of agents who share inter-agent constraints) via *ok?* messages.
- When received an *ok?* message, an agent updates an *agent_view*, which records the current colors of neighbors' nodes, and then does the following for each of its nodes, say n, in order of node priority.
  - Select a color for n which is consistent with the colors of higher priority agents' nodes and the colors of this agent's higher priority nodes, and then move to the next node. Note that if the current color of n is already consistent, do not change that color.
  - If there is no such consistent color for n, generate a *nogood* by using the method described in [8][3]. A nogood is a set of colors of nodes under which there is no consistent color for n and can be considered as a new constraint discovered during algorithm execution. Since any superset of nogood cannot be a solution, an empty nogood eliminates all possible colors of nodes. Therefore, an agent can detect the fact that there is no solution in a problem if it generates an empty nogood.
  - If there are any of this agent's nodes in the generated nogood, record the nogood as a new constraint, backjump to the lowest priority node among those nodes, and reselect colors of the node and its subsequent nodes. On the other hand, if there isn't any of this agent's nodes in the generated nogood, keep the current color of n, push the nogood into a stack, and move to the next node.

  When finished selecting colors of all its nodes, an agent sends each nogood in the stack via a *nogood* message to the lowest priority agent among those which have nodes in the nogood. Then, it sends changed colors to neighbors via *ok?* messages.
- When received a *nogood* message, an agent records its content as a new constraint and examines all its nodes in the same way as described above. Note that when a received nogood includes an unknown node which belongs to a non-neighboring agent, an agent requests that agent to send a color of the node and adds that agent to its neighbors.

Since agents record all nogoods discovered in their concurrent search processes, this algorithm is complete, i.e., reaches a solution if one exists or finds insolubility if none exists[4].

In our experiments, we implemented this algorithm on a simulator of *synchronous distributed system*, which was implemented in C and ran on an Intel Celeron 333MHz processor, and measured its communication and computation costs using this simulator. A synchronous distributed system is a typical model of distributed system, on which every agent synchronously performs the following cycle.

1. Receive all the messages which were sent toward the agent at the previous cycle.
2. Perform local computation to change its internal state and determine the contents of messages, and send those messages to other agents.

Note that at first cycle an agent does not receive any messages, but it does perform local computation and send messages to other agents[5]. We measured the following using this simulator.

Cycle : the number of cycles consumed until agents find one solution to a distributed CSP or the fact that no solution exists. Since agents communicate with others

confidential information open to others.

[3]In short, for each possible color of n, select an existing nogood that prohibits it; and make a new nogood out of an aggregation of those selected nogoods.

[4]However, this algorithm itself cannot detect the fact that a solution is found. For such a detection, we have to rely on the snapshot algorithm[2].

[5]This algorithm is designed for a *fully asynchronous distributed system*, which is the most general model of distributed system, and thereby working on any type of distributed system. In our experiments, among possible distributed systems, we use a synchronous distributed system for simplicity.

when received (**ok?**, (*sender*, *node*, *color*)) **do**
  add (*sender*, *node*, *color*) to *agent_view*;
  L :=an ordered list of all nodes within this agent (descending order of priority);
  $N := null$;
  if $check\_nodes(L, N)$ **then**
    wait for a next message;
  else
    broadcast "no solution exists";
end do;

procedure $check\_nodes(L, N)$
  if L is empty **then**
    send each nogood in N to the agent whose priority is the lowest in the nogood;
    send all changed colors to *neighbors*;
    return *true*;
  else
    n :=the first node in L;
    if n's color is consistent (with the colors of higher priority agents' nodes and the colors of this agent's higher priority nodes) **then**
      $L := L$ with n deleted; $check\_nodes(L, N)$;
    else
      if there is another consistent color, c, for n **then**
        change n's color to c;
        $L := L$ with n deleted; $check\_nodes(L, N)$;
      else
        nogood :=a set of nodes' colors originating this conflict;
        if *nogood* is empty **then**
          return *false*;
        else
          if *nogood* includes nodes of this agent **then**
            m :=the lowest priority node among these nodes;
            L :=a list of m and all lower priority nodes than m;
            record *nogood* as a new constraint;
            $check\_nodes(L, N)$;
          else
            add *nogood* to N;
            $L := L$ with n deleted; $check\_nodes(L, N)$;

Figure 2: Algorithm

at every cycle, this measure increases with the number of communication among agents. Thus, we use this measure as a communication cost.

Maxcck : the total sum of the maximum number of consistency checks at each cycle until agents find one solution to a distributed CSP or the fact that no solution exists. More specifically, at each cycle we identify the bottleneck agent which performed the most consistency checks for its local computation, and sum up all those maximum numbers of consistency checks over all cycles. In a synchronous distributed system, although the amounts of local computation at each cycle vary among agents, the whole amount of computation is dominated by those bottleneck agents. Thus, we use this measure as a computation cost.

3.2. PROBLEM GENERATION

We used distributed graph 3-coloring problems in our experiments. There are several parameters to specify one distributed graph 3-coloring problem, for example, the number of agents, the number of nodes, the average number of nodes per agent, and so on. Since our goal is to see the relation between the cost of algorithm and the numbers of intra- and inter-agent constraints, we varied the numbers of intra- and inter-agent constraints while fixing the other parameters at certain values.

In our experiments, we generated a distributed graph 3-coloring problem using the following method (the inputs are the number of agents, k, the number of intra-agent constraints, x, and the number of inter-agent constraints, y):

1. Distribute 100 nodes among k agents ($k \in \{2, 5\}$ in our experiments) such that these agents have nodes as equally as possible.
2. Define each of x intra-agent constraints as follows.
   (a) Randomly select one agent from k agents.
   (b) Randomly select two nodes from the nodes which belong to the above agent and define a link between them. If there already exists a link between them, repeat until two nodes with no link between them are found.
3. Define each of y inter-agent constraints as follows.
   (a) Randomly select two agents from k agents.
   (b) Randomly select one node each from the nodes which belong to the above two agents and define a link between these two nodes. If there already exists a link between them, repeat until two nodes with no link between them are found.
4. Accept the graph if it is connected. If it is not connected, delete all the links and go to 2.

Note that we used only a connected graph in our experiments. For the algorithm in section 3.1, a distributed graph 3-coloring problem with a disconnected graph would be easy because it is able to solve such a problem by solving separated components independently.

3.3. RESULTS

Figure 3 shows the ratio of soluble problems for 2 agents with 50 nodes each on the x-y plane (the x-axis is the number of intra-agent constraints and the y-axis is the number of inter-agent constraints). Figure 4 also shows that for 5 agents with 20 nodes each. For each case, we generated 500 problems at every combination of the number of intra-agent constraints, $intra \in \{80, 90, \ldots, 180\}$, and the number of inter-agent constraints, $inter \in \{100, 110, \ldots, 160\}$, and plotted the ratio of soluble problems among 500 problems at each data point. These figures clearly show that when increasing the numbers of intra- and inter-agent constraints, we can observe a rapid drop from near one to zero in the ratio of soluble problems. For each case, the *phase transition region* where such a rapid drop is located lies around $210 \leq intra + inter \leq 240$ on the x-y plane.

Figure 5 and 6 show the median cycle (communication cost) and maxcck (computation cost) for 2 agents with 50 nodes each, respectively, and figure 7 and 8 show those for 5 agents with 20 nodes each. For each case, we measured cycle and maxcck for each of 500 problems at every combination of $intra$ and $inter$, and plotted the median cycle and maxcck of 500 problems at each data point. Note that these are shown using a logarithmic scale. We can observe the following from these results.

- Both the median cycle and maxcck in the phase transition region are higher than those in the other regions. For each of these figures, we can see that there is a "ridge" along the phase transition region and the top of the ridge lies around $230 \leq intra + inter \leq 240$ on the x-y plane, which corresponds the region where the ratios of soluble problems are about between 0.3 and 0.
- When the number of intra-agent constraints decreases and the number of inter-agent constraints increases, the top of the ridge becomes higher.
- The top of the ridge has a steeper slope for 5 agents with 20 nodes each than for 2 agents with 50 nodes each. As a result, the case of 5 agents with 20 nodes each has a higher peak.

It is very interesting that there is the top of the ridge around $230 \leq intra + inter \leq 240$. According to [9], for randomly generated (centralized) graph 3-coloring problems, the median computation cost of depth-first backtracking search with the Brelaz heuristic rises to the peak when the ratio of the number of links to the number of nodes is 2.3, and the location of the peak coincides with the point at which the ratio of soluble problems is 0.5. In our experiments, although we used distributed graph 3-coloring problems and a distributed constraint satisfaction algorithm, we can observe a basically similar result, where the top of the

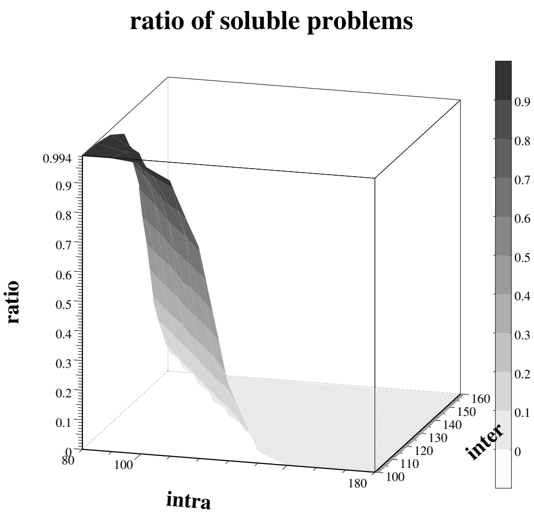

Figure 3: Ratio of soluble problems (2 agents, 50 nodes each)

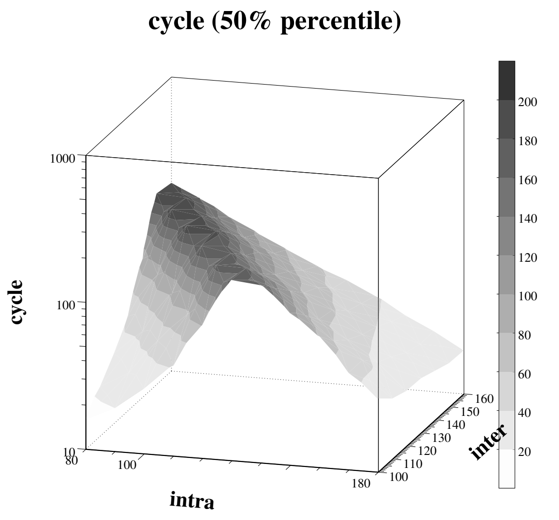

Figure 5: Median cycle (2 agents, 50 nodes each)

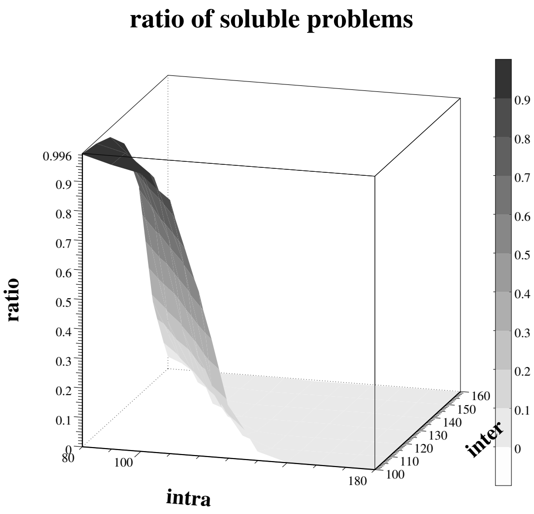

Figure 4: Ratio of soluble problems (5 agents, 20 nodes each)

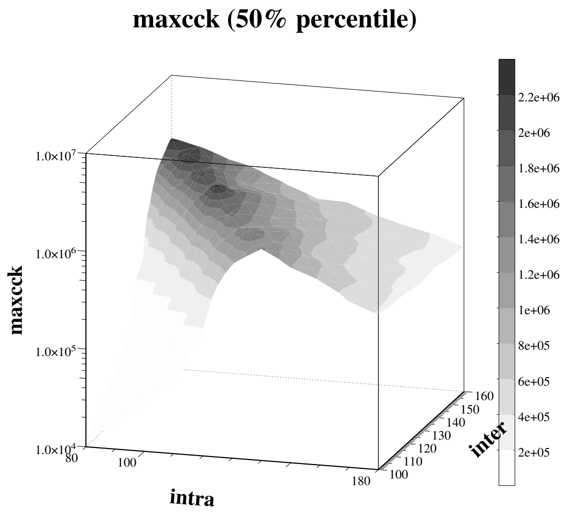

Figure 6: Median maxcck (2 agents, 50 nodes each)

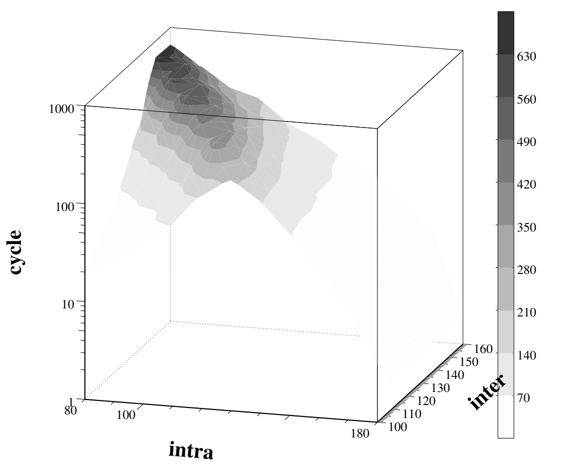

Figure 7: Median cycle (5 agents, 20 nodes each)

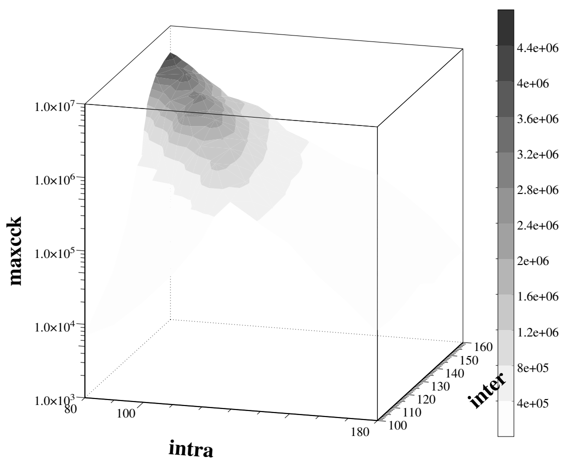

Figure 8: Median maxcck (5 agents, 20 nodes each)

ridge of the median cost appears when the ratio of the number of links ($intra + inter$) to the number of nodes (100) is about 2.3 or 2.4. On the other hand, we should point out that its location does not coincide with the region where the ratio of soluble problems is 0.5.

We conjecture the reason why the top of the ridge becomes higher when the number of intra-agent constraints decreases and the number of inter-agent constraints increases as follows. When solving a distributed graph 3-coloring problem with a small number of intra-agent constraints and a large number of inter-agent constraints, an agent is likely to select a misleading color for its node, that is, the color which appears to be right from its local point of view but actually is wrong from the global point of view. Once an agent (especially a higher priority agent) selects such a misleading color, other agents have to perform a lot of search to make the agent change that misleading color, and thus the communication and computation costs can be high for that kind of problems. On the other hand, for a distributed graph 3-coloring problem with a large number of intra-agent constraints and a small number of inter-agent constraints, an agent is less likely to select such a misleading color, and thus the communication and computation costs can be relatively low.

The top of the ridge has a steeper slope for 5 agents with 20 nodes each (figure 7 and 8) than for 2 agents with 50 nodes each (figure 5 and 6). For example, for 5 agents with 20 nodes each, the maximum and minimum cycles on the top of the ridge are 680 (at $intra = 80$ and $inter = 160$) and 232 (at $intra = 130$ and $inter = 100$), respectively, and for 2 agents with 50 nodes each, those are 200 (at $intra = 80$ and $inter = 150$) and 170 (at $intra = 130$ and $inter = 100$), respectively. We conjecture the reason is that a misleading color mentioned above can have a great impact on the entire performance if the number of agents increases or the number of nodes per agent decreases.

To confirm the generality of our findings, we should make more experiments using other problems and other distributed constraint satisfaction algorithms. For a preliminary report, we show the results of the following combination of problems and an algorithm in figure 9 and 10 (only the median cycle of 500 problems for 5 agents with 20 nodes each):

- soluble distributed graph 3-coloring problems and the algorithm in section 3.1
- soluble distributed graph 3-coloring problems and the *multi-AWC*[18].

A soluble problem is generated using the method in section 3.2 which is modified so that we can specify a solution in advance and avoid any link that would eliminate the solution. Surprisingly, for each case, although there is no phase transition region because all problems are soluble, we can find a ridge almost at the same region as in figure 5–8, and its qualitative characteristics are consistent with our findings.

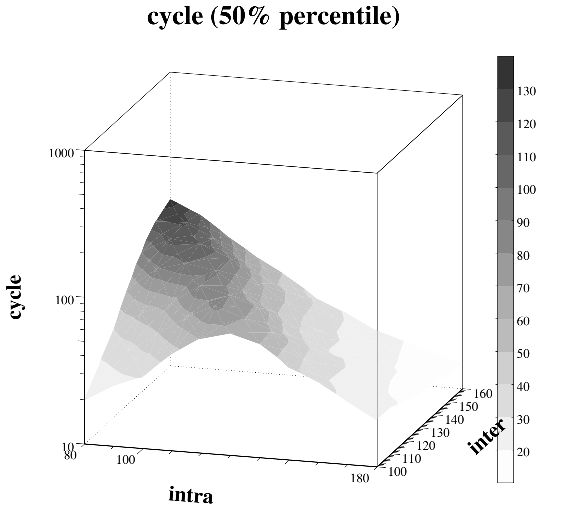

Figure 9: Median cycle (5 agents, 20 nodes each, all soluble)

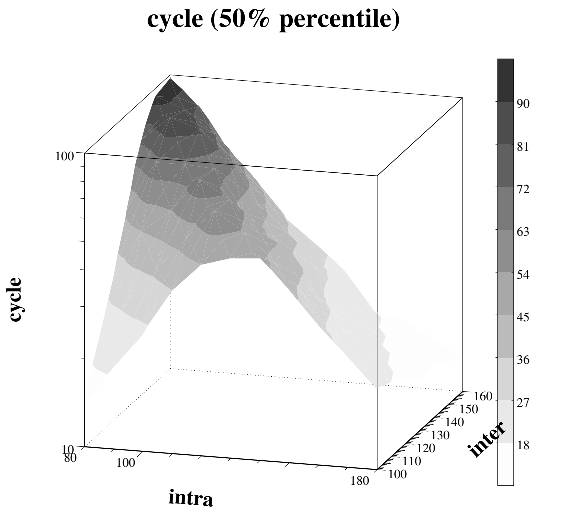

Figure 10: Median cycle (5 agents, 20 nodes each, all soluble, by multi-AWC)

4. CONCLUSIONS AND FUTURE WORK

In this paper, we examine the communication and computation costs of a distributed constraint satisfaction algorithm when varying the numbers of intra- and inter-agent constraints for distributed graph 3-coloring problems. Our findings are as follows.

- The phase transition occurs in the narrow region on the x-y plane where the sum of intra- and inter-agent constraints is constant. Problems in the phase transition region are likely to be expensive in terms of the median communication and computation costs.
- Among problems in the phase transition region, problems with a smaller number of intra-agent constraints and a larger number of inter-agent constraints can be more expensive.
- for a fixed total number of variables, problems in the phase transition region can be more expensive as the number of agents increases or the number of variables per agent decreases.

From these results, we may say that a really hard distributed CSP is a problem in the phase transition region in which each agent has limited amount of knowledge about the problem, i.e., a small number of intra-agent constraints and a small number of variables per agent.

Finally, we will point out our future work. In our experiments, we gathered data for 500 problems at each data point while varying the numbers of intra- and inter-agent constraints in steps of 10. For estimating a full detail of the behavior of a distributed constraint satisfaction algorithm, we may need to make experiments with more data points.

It is well known that there are *exceptionally hard problems* (ehp) in the under-constrained region for (centralized) CSPs[5, 9, 14]. We cannot discuss this issue for lack of space, but in our experiments we observed a sort of ehp in the under-constrained region ($intra + inter < 210$). To confirm this behavior, we may need to make experiments with more data at each data point.

While this paper gives first experimental results on the behavior of a distributed constraint satisfaction algorithm and our interpretation of these results, it gives no theory that would explain the behavior. We are now studying the possibility of theoretical analyses of our experimental results.

ACKNOWLEDGMENTS

Authors wish to thank Takuo Tsuruta, Youssef Hamadi and anonymous reviewers for their helpful comments on this work.

REFERENCES

[1] A. Armstrong and E. Durfee. Dynamic prioritization of complex agents in distributed constraint satisfaction problems. *Proceedings of IJCAI-97*, pp. 620–625, 1997.

[2] K. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transaction on Computer Systems*, Vol. 3, No. 1, pp. 63–75, 1985.

[3] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. *Proceedings of IJCAI-91*, pp. 331–337, 1991.

[4] S. E. Conry, K. Kuwabara, V. R. Lesser, and R. A. Meyer. Multistage negotiation for distributed constraint satisfaction. *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 21, No. 6, pp. 1462–1477, 1991.

[5] I. P. Gent and T. Walsh. Easy problems are sometimes hard. *Artificial Intelligence*, Vol. 70, pp. 335–345, 1994.

[6] Y. Hamadi. Optimal distributed arc-consistency. *Proceedings of CP-99*, pp. 219–233, 1999.

[7] Y. Hamadi, C. Bessière, and J. Quinqueton. Backtracking in distributed constraint networks. *Proceedings of ECAI-98*, pp. 219–223, 1998.

[8] K. Hirayama and M. Yokoo. The effect of nogood learning in distributed constraint satisfaction. *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems*, pp. 169–177, 2000.

[9] T. Hogg and C. P. Williams. The hardest constraint problems: a double phase transition. *Artificial Intelligence*, Vol. 69, pp. 359–377, 1994.

[10] M. N. Huhns and D. M. Bridgeland. Multiagent truth maintenance. *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 21, No. 6, pp. 1437–1445, 1991.

[11] C. Mason and R. Johnson. DATMS: A framework for distributed assumption based reasoning. *Distributed Artificial Intelligence*, Vol. 2, pp. 293–318. Morgan Kaufmann, 1989.

[12] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of sat problems. *Proceedings of AAAI-92*, pp. 459–465, 1992.

[13] B. M. Smith. The phase transition in constraint satisfaction problems: A closer look at the mushy region. Research report 93.41, School of Computer Studies, University of Leeds, 1993.

[14] B. M. Smith and S. A. Grant. Sparse constraint graphs and exceptionally hard problems. *Proceedings of IJCAI-95*, pp. 646–651, 1995.

[15] K. P. Sycara, S. Roth, N. Sadeh, and M. Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 21, No. 6, pp. 1446–1461, 1991.

[16] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 10, No. 5, pp. 673–685, 1998.

[17] M. Yokoo and K. Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction problems. *Proceedings of the Second International Conference on Multi-Agent Systems*, pp. 401–408, 1996.

[18] M. Yokoo and K. Hirayama. Distributed constraint satisfaction algorithm for complex local problems. *Proceedings of the Third International Conference on Multi-Agent Systems*, pp. 372–379, 1998.