Problem Structure and Subproblem Sharing in Multi-Agent Systems

Dorothy L. Mammen and Victor R. Lesser[*]
Department of Computer Science
University of Massachusetts
Amherst, MA USA 01003
{mammen, lesser}@cs.umass.edu

Abstract

In multi-agent problem-solving systems in which agents work asynchronously and in parallel on parts of a problem distributed among them, subproblem communication and integration protocols can have a major impact on problem solving efficiency. Transmitting too soon can necessitate having to "take back" transmitted partial solutions, wasting communication and processing time. Transmitting too late can complicate integration of partial solutions derived independently. The trade-off between these depends on the nature of the subproblem interdependency among the agents. We present a parameterized distributed constraint satisfaction problem (CSP) generator and a parameterized multi-agent problem-solving simulator. Using these tools, we offer some empirical observations of the trade-off between strategies for subproblem communication timing. Our goal is to derive domain-independent answers to questions of subproblem sharing in multi-agent systems for problems that can be represented within the CSP paradigm.

1. Introduction

In multi-agent problem-solving systems in which agents work asynchronously and in parallel on parts of a problem distributed among them, then combine their partial solutions into an overall solution, subproblem communication and integration protocols can have an enormous impact on problem-solving efficiency. While the differential efficacy of various protocols in different situations has been documented (for example, [4, 3]), there is no general theory of how communication and subproblem integration should be organized as a function of the structural characteristics of the problem, including its decomposition among the agents.

Domain information as a basis for coordinating agent problem solving is not always available. For example, whereas in distributed vehicle monitoring it is clear that sharing processing results for a track at the edge of an agent's sensor region can facilitate its neighbor's computations [2], in a furniture placement problem there may be no *a priori* knowledge to suggest that one agent's placement of some pieces might facilitate another agent's tasks. Depending on the floor plan, furniture to be placed, and distribution of responsibility among the agents with respect to the furniture, the nature of advantageous coordination among the agents can change from one problem to another, or dynamically as pieces are placed. Identification of underlying relationships between problem structure, problem-solving strategy, and performance could be useful for both online agent interaction during problem solving, and for off-line design of multi-agent problem-solving systems.

One aspect of communication is the question of when an agent should share a partial subproblem solution with other agents, who will generally incorporate it as a constraint into their own problem-solving efforts. In particular, in this paper we consider what size subproblem, in terms of number of variables, an agent should share, based on information about the structure of its subproblem and its relationship to the subproblems of other agents.

To share partial subproblem solutions most advantageously, an agent must balance two competing pressures:

1. An agent does not want to communicate a partial subproblem solution *too soon*, because it wants to maximize the probability that the shared partial result is locally extendible. That is, the agent does not want to continue its local problem solving only to discover that it must "take back" a partial result that it has already communicated to other agents, as this will result in wasted communication and processing time, both for the sharing agent and for receiving agents as well. In general, the more complete an agent's partial solu-

[*]This material is based upon work supported by the National Science Foundation under Grants No. IRI-9321324 and IRI-9523419. Any opinions, finding, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

tion, the more certain the agent can be that the partial solution will at least be locally extendible. This results in pressure on agents to share partial subproblem solutions later rather than sooner.

2. An agent does not want to communicate a partial subproblem solution *too late*, because other agents will simultaneously, independently, have extended their own subproblem solutions further as well. The longer problem solving progresses without the sharing of partial solutions, the more likely it is that work will have to be undone to reconcile independently-derived subproblem solutions. Earlier sharing of partial solutions allows receiving agents to incorporate received partial solutions as constraints, and ensure that their own partial solutions are globally consistent as they are developed. In addition, receiving agents may discover that communicated subproblem solutions are not consistent with their own subproblem solution in progress, and generate new constraints or initiate a resolution process sooner. These factors result in pressure on agents to share partial subproblem solutions sooner rather than later.

In this paper we look at the trade-off among these two pressures, and begin to show how problem structure, including distribution among agents, can have implications for coordinating the sharing of subproblem solutions. Specifically, we consider what size subproblem agents should share, based on the degree to which problem constraints are between agents versus within agents.

We view distributed problem solving as distributed search, and use a distributed constraint satisfaction problem (DCSP) formalism, as do others, such as [9, 13]. The CSP formalism can be used to describe many types of problems, both theoretical and in the real world, including design, sensor interpretation, circuit analysis, image understanding, and scheduling. It can be used to describe problems ranging in difficulty from trivially easy to NP-hard, such as graph coloring. As a problem representation, one of its greatest strengths is the explicit representation of subproblem interaction.

Our work is a mix of empirical and analytical approaches, but we present just empirical results in this paper. We generate DCSP problems using a parameterized problem generator to tailor problem structure along particular dimensions. We solve the problems with a multi-agent simulator parameterized with regard to particular aspects of problem-solving strategy. The problem generator is described in Section 2, the multi-agent simulator in Section 3. Our experiments and results are presented in Section 4, and we conclude and summarize in Section 5.

2. Parameterized problem generator

Our goal in problem generation is to be able to generate problems with particular problem structure characteristics, for use in evaluating distributed problem-solving strategies as a function of aspects of problem structure. Our problem generator generates distributed problems based on domain-independent problem characteristics, specifying problem size and various aspects of the density and distribution of constraints. Use of a problem characterization independent of any particular problem domain means that the results are applicable to any problem domain, based on the problem structure characteristics of the domain.

In earlier work on the relationship between problem structure, problem-solving algorithms, and solution cost for centralized CSPs [6], we specified problems using just three parameters: number of problem variables, n, domain size of variables, b, and number of binary nogoods, m. Each nogood defines a constraint in the form of one inconsistent variable-value pair. In this work, we augment the centralized problem description to include number of agents as well as how variables and nogoods are distributed among agents, for the distributed context.

The problem generator creates random constraint problems parameterized along nine dimensions of problem structure, enumerated below. Problems generated may either be solvable or unsolvable. For each problem parameter, we specify a frequency distribution of possible values from which the generator will choose one or more values, depending on the parameter. Thus a given parameterization describes a class of problems, of which each generated problem is one instance. The nine problem parameters and their relevance with respect to characterizing problem structure are described below.

2.1. Problem class parameters

- Number of problem variables, n; set of problem variables denoted $V = \{v_1 \ldots v_n\}$
  Specify μ_n, σ_n; normal distribution.
  Number of variables is one measure of problem size.

- Problem variable domain sizes, $b_{v_1} \ldots b_{v_n}$
  Specify μ_{b_v}, σ_{b_v}; normal distribution.
  Problem variable domain sizes are another measure of problem size.

- Number of binary nogoods (constraints), m
  Specify m_{min}, m_{max}; uniform distribution.
  Number of binary nogoods[1] is a measure of problem

[1]Generator could be extended to include constraints of arity greater than two

constrainedness. Problem constrainedness has a well-documented relationship to problem difficulty (for example, see [1, 7, 10, 5]).

- Number of agents, g; set of agents denoted $A = \{a_1 \dots a_g\}$
  Specify μ_g, σ_g; normal distribution.
  Number of agents is a measure of problem distribution.

- Number of problem variables per agent, $n_{a_1} \dots n_{a_g}$; set of problem variables belonging to agent a denoted V_a.
  Mean number of variables per agent, $\mu_{n_a} = n/g$.
  Specify σ_{n_a} as coefficient of variation; normal distribution. Generator chooses number of problem variables n_a for each agent a from the specified distribution, with the additional constraints (1) $\mu_{n_a} \pm 2\sigma_{n_a}$ contains n_a, and (2) $\sum_{a\in A} n_a = n$.
  Variance in number of variables per agent is a measure of problem distribution.

- Number of nogood vertices per agent, $m_{a_1} \dots m_{a_g}$
  Mean number of nogood vertices per agent,

$$\mu_{m_a} = 2m \frac{\sum\limits_{v\in V_a} b_v \sum\limits_{v\in V} b_v - \sum\limits_{v\in V_a} b_v^2}{\left(\sum\limits_{v\in V} b_v\right)^2 - \sum\limits_{v\in V} b_v^2}. \qquad (1)$$

  Specify σ_{m_a} as coefficient of variation; normal distribution; constraints (1) $\mu_{m_a} \pm 2\sigma_{m_a}$ contains m_a, and (2) $\sum_{a\in A} m_a = 2m$. Note: when $\sigma_{b_v} = \sigma_{m_a} = 0.0$, $\mu_{m_a} = 2m\,(n_a/n)$.
  Variance in number of nogood vertices among agents is a measure of variation in inter-agent constrainedness.

- Number of nogood vertices per variable, $m_{v_1} \dots m_{v_n}$, within an agent
  Mean number of nogood vertices for variable v_i,

$$\mu_{m_{v_i\in V_a}} = m_a \frac{b_{v_i} \sum\limits_{v\in V} b_v - b_{v_i}^2}{\sum\limits_{v\in V_a} b_v \sum\limits_{v\in V} b_v - \sum\limits_{v\in V_a} b_v^2}. \qquad (2)$$

  Specify $\sigma_{m_{v_i}\in V_a}$ as coefficient of variation; normal distribution; constraints (1) $\mu_{m_{v_i}} \pm 2\sigma_{m_{v_i}}$ contains m_{v_i}, and (2) $\sum\limits_{v\in V_a} m_v = m_a$. Note: when $\sigma_{b_v} = \sigma_{m_v} = 0.0$, $\mu_{m_{v\in V_a}} = m_a/n_a$.
  It is possible to specify different values in this parameter for different agents.
  Variance in number of nogoods among variables within one agent is a measure of intra-agent variation in constrainedness, or variable tightness.

- Relative proportion of local nogood vertices per agent, $f_{a_1} \dots f_{a_g}$
  Specify $0.0 \le f_{a_{min}}, f_{a_{max}} \le 2.0$; uniform distribution, indicating proportion of nogoods to be local (intra-agent), relative to frequency of internal versus total possible constraints, which we call par, where

$$par = \frac{\left(\sum\limits_{v\in V_a} b_v\right)^2 - \sum\limits_{v\in V_a} b_v^2}{\sum\limits_{v\in V_a} b_v \sum\limits_{v\in V} b_v - \sum\limits_{v\in V_a} b_v^2}. \qquad (3)$$

  Parameter f_a is used to compute the actual proportion of local nogoods, p_a, as follows: for $0.0 \le f_a \le 1.0$, $p_a = f_a \times par$; for $1.0 < f_a \le 2.0$, $p_a = par + (f_a - 1.0)(1.0 - par)$. Given p_a, actual number of local nogood vertices, l_a, can be calculated as $l_a = m_a \times 2p_a/(1 + p_a)$ vertices; $m_a(1 - p_a)/(1 + p_a)$ vertices will be involved in nogoods involving another agent. Example: if $p_a = 0.5$, 0.67 of agent a's m_a nogood vertices will be involved in local nogoods, while 0.33 of its nogood vertices will be involved in nogoods with other agents, so that 0.5 of the nogoods the agent is involved are local.
  Relative proportion of local nogoods is a measure of agent independence versus interdependence.

- Number of local nogoods per agent variable, $l_{v_1} \dots l_{v_n}$, within an agent
  Mean number of local nogoods for variable v_i,

$$\mu_{l_{v_i\in V_a}} = m_{v_i} \frac{b_{v_i} \sum_{v\in V_a} b_v - b_{v_i}^2}{b_{v_i} \sum_{v\in V} b_v - b_{v_i}^2}. \qquad (4)$$

  Specify $\sigma_{l_{v_i}}$ as a coefficient of variation; normal distribution; constraints (1) $\mu_{l_{v_i}} \pm 2\sigma_{l_{v_i}}$ contains l_{v_i}, and (2) $\sum\limits_{v\in V_a} l_v = l_a$.
  Variance in number of local nogoods per agent variable is a measure of the distribution of an agent's external constraints among its variables.

2.2. Example

Figure 1 shows an example of a set of problem class parameters and one problem instance generated within the specified class. We believe these problem class parameters characterize problem structure and decomposition fairly completely. They allow for control of overall problem size using total number of variables and their domain sizes; for overall problem difficulty using number of nogoods; for distribution of problem using number of agents and variance in number of variables per agent; for distribution of variable

Problem variables ($\mu_n = 10$, $\sigma_n = 2.0$): 10
Vbl dom sizes ($\mu_{b_v} = 3$, $\sigma_{b_v} = 1.0$): 3,2,2,3,1,4,3,4,4,1
Nogoods ($m_{min} = 50$, $m_{max} = 60$): 54
Agents ($\mu_g = 4$, $\sigma_g = 1.0$): 4
Variables/Agent ($\sigma_{n_a} = 0.25$): 2, 2, 2, 4
Nogood vertices/agent ($\sigma_{m_a} = 0.25$): 19, 24, 18, 47
Ng verts/vbl ($\sigma_{m_{v_i}} = 0.25$): 10,9,9,15,4,14,10,13,12,12
Loc ng verts/agt ($f_{a_{min}} = 1.0$, $f_{a_{max}} = 1.0$): 4, 4, 2, 24
Loc ng/vbl ($\sigma_{l_{v_i}} = 0.0$): 2, 2, 2, 2, 1, 1, 6, 8, 8, 2

Figure 1. Example of a set of problem class parameters and a problem instance drawn from that class.

tightness among and within agents using variance in number of nogood vertices per agent and variance in number of nogood vertices per variable within an agent; for relative independence versus interdependence of agents using relative proportion of local nogoods per agent; and for distribution of local versus external constraints within an agent using variance in local nogoods per variable within an agent. Parameters varied for the experiments reported herein are number of nogoods and relative proportion of local nogood vertices per agent.

3. Parameterized multi-agent simulator

We use a simple, complete, reasonably efficient, multi-agent problem solving protocol as a basis for studying the question of when to transmit partial solutions. Based on Yokoo's asynchronous weak-commitment search algorithm [12], our algorithm allows for parallel processing among agents and is guaranteed to either find a solution if one exists, or report that a solution does not exist if none does.

Our version extends the asynchronous weak-commitment search algorithm in various ways. We allow each agent to be responsible for multiple variables, using the same priority-raising scheme within agents as among them. We include a termination detection mechanism, based on each agent sending a **consistent** message to every other agent when its variables are consistent with respect to its view of the problem. The simulation is finished when all agent message queues are empty and each agent has received consistent messages from each other agent since the last time at which any local variable was assigned or nogood generated. Our agents send **ok?** messages only to those agents for whom the information is relevant, namely those agents with whom a constraint is shared, rather than to all agents. As problem solving progresses and implicit constraints are discovered, which agents are relevant changes dynamically. We also include the flexibility for agents to send more information than assignment value and priority along with a variable assignment, for other agents to use in decision making during problem solving. This allows us to explore more sophisticated problem-solving strategies.

Finally, we derive new nogoods by computing the minimal subsets of assignments that are responsible for a discovered inconsistency, rather than using the entire *agent-view* (agent's record of variable values). During problem solving, each time a local assignment conflict is discovered, we record exactly which assignments (or set of assignments) in the *agent-view* are responsible. When an inconsistency is discovered, we use that information to derive the set of minimal nogoods. There is a trade-off between computational time and utility of nogoods, smaller nogoods requiring more computational effort to derive, but being more useful in problem solving.

We have also parameterized the algorithm along several dimensions of problem-solving strategy, for studying the efficiency of different strategies. In addition, we have parameterized the timing of various types of problem-solving events. The parameterization of the problem solver with respect to algorithm and problem-solving events is described below.

3.1. Problem-solving algorithm parameters

- Communication strategy
  Specifies at what point during problem solving a subproblem solution will be communicated.
  Options: after assignment of every x variables, where $1 \leq x \leq n_a$ (default, with $x = 1$); only in the case of backtracking; only in the case of any constraint violations with one or more other agent(s)'s variables; after local variable assignments have satisfied a certain number or percentage of an agent's local nogoods

- Initial variable ranking strategy
  Specifies method for determination of initial variable priorities.
  Options: random equivalent (default); random unique; in order of number of nogoods.

- Variable ordering
  Specifies method for selection of assignment variable.
  Options: random, priority order (default), most remaining nogoods, most relevant agents.

- Value ordering
  Specifies which value an agent selects for a variable assignment.
  Options: random, min-conflict (default).

3.2. Problem-solving system parameters

- Communication delay
  Specifies any delay between when an agent sends a

message and when it is received.
Specify mean and variance; normal distribution.

- Communication packaging overhead
  Specifies time to package one or more pieces of information to communicate
  Specify mean and variance; normal distribution.

- Communication processing overhead
  Specifies overhead of interrupting local problem-solving to process received information
  Specify mean and variance; normal distribution.

- Problem-solving action timing
  This is actually multiple parameters, specifying the time required for all problem-solving actions, such as assigning a variable with or without backtracking, consistency checking, generating new nogoods, etc.
  Specify mean and variance; normal distribution.

4. Experiments and results

For the experiments reported in this paper, we generated distributed constraint satisfaction problems consisting of ten variables with three possible values for each one, distributed evenly (3 - 4 - 3) among three agents, with no variance in these values. We work with these small problems to allow for analysis of problem structure and problem solving at a level of detail that is simply not possible with larger problems. Again, our goal is understanding relationships between problem structure and problem-solving strategies.

For all problems generated, nogood vertices were distributed among agents in proportion to the number of variables belonging to each agent, and evenly among variables within each agent. We generated 1000 problems at each multiple of ten nogoods from 30 to 130, for three points along a continuum of interdependence among agents (specified as relative proportion of local nogood vertices per agent, f_{a_i}): completely *independent* ($f_{a_i} = 2.0$: all intra-agent nogoods, no inter-agent nogoods), *par* ($f_{a_i} = 1.0$, intra- and inter-agent nogoods in proportion to the number of intra- and inter-agent nogoods possible), and completely *interdependent* ($f_{a_i} = 0.0$: no intra-agent nogoods, all inter-agent nogoods). The *independent* and *interdependent* classes mark extremes of the continuum. We use them for comparing simulation results against intuition, acknowledging that real-world problems for multi-agent systems would rarely fall at either extreme. In all three classes, the intra-agent and inter-agent nogoods belonging to a given agent were distributed evenly among its variables.

We would like to be able to compare the three problem classes as a function of number of nogoods. However, we must take into account the fact that the way in which the nogoods are distributed in the three classes affects solvability.

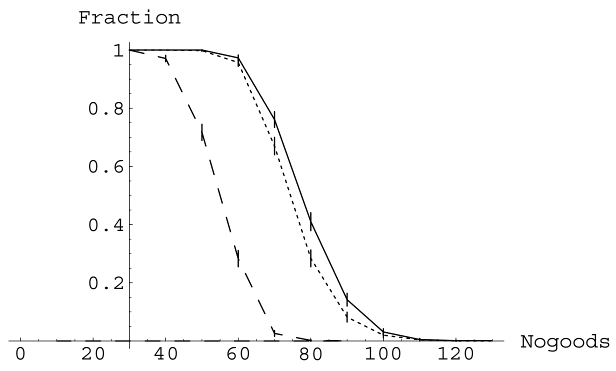

Figure 2. Solvability as a function of number of nogoods for the *independent* (dashed line), *interdependent* (dotted line), and *par* (solid line) problem classes. Each point is based on 1000 problems of 10 variables and domain size 3. Error bars showing 95% confidence intervals (given approximately by $f \pm 2\sqrt{f(1-f)/N}$, where f is the estimated value of the fraction) are included.

Figure 2 shows the proportion of solvable problems for the *independent*, *interdependent*, and *par* problem classes as a function of number of nogoods. The *independent* class of problems, with intra-agent nogoods only, becomes unsolvable much faster than the *par* and *interdependent* classes, which are quite comparable. This is due to the distribution of problem nogoods among a smaller number of variable-value pairs in the *independent* class. Assuming variances of 0.0, if there are g agents, n_g variables per agent, and variable domain size is b, then in the *independent* class of problems, the nogoods are distributed among

$$g \times \binom{n_g}{2} \times b^2$$

possible variable-value pairs, whereas in the *interdependent* class, they are distributed among

$$\binom{g}{2} \times {n_g}^2 \times b^2$$

possible pairs, the former always being smaller than the latter. Because the peak of problem difficulty usually occurs near the point at which 50% of the problems are solvable (see, for example, [8, 11]), the peak in problem difficulty will be shifted left for the *independent* class. Thus we will want to compare the patterns in problem solving behavior rather than comparing for specific numbers of nogoods.

For the simulations reported in this paper, we used two different subproblem communication strategies. In the *every* strategy, each agent sends every assignment it makes to

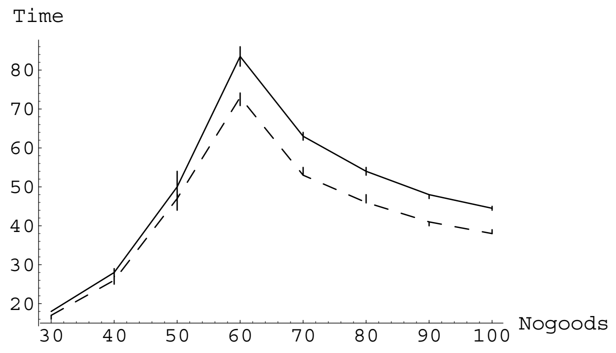

Figure 3. Solution time as a function of number of nogoods for the *independent* class of problems, for the *every* strategy (solid line) and the *locally-consistent* strategy (dashed line). Each point is the median for 100 problems of 10 variables and domain size 3, solved 10 times each. Error bars showing 95% confidence intervals are included.

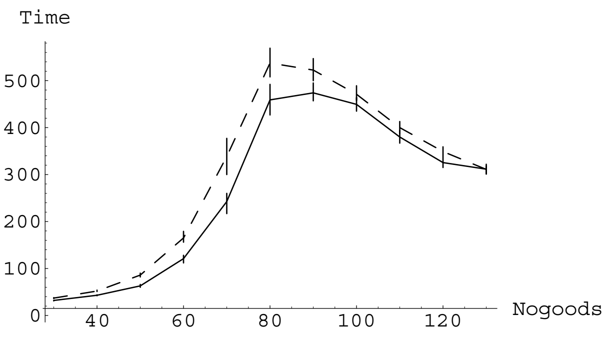

Figure 4. Solution time as a function of number of nogoods for the *interdependent* class of problems, for the *every* strategy (solid line) and the *locally-consistent* strategy (dashed line). Each point is the median for 100 problems of 10 variables and domain size 3, solved 10 times each. Error bars showing 95% confidence intervals are included.

all relevant agents (those agents with whom it shares constraints for the assigned variable). In the *locally-consistent* strategy, each agent waits until it has reached local consistency for its own subproblem (or discovered that its own subproblem is unsolvable) before sending any partial results. For all simulations, we used equivalent initial variable priorities, priority-order variable ordering during problem solving, and min-conflict value ordering. Communication delay was zero, and communication packaging and processing costs were negligible: only the actual computational time of simulating them. For the simulations reported here, solution time is reported in terms of internal run-time. On later versions of the simulator we use parameterized times.

Figures 3, 4,and 5 show how solution time changes as a function of number of nogoods for each of the three problem classes, for the two communication strategies.

Figure 3 shows that for the *independent* class of problems, it is always better for agents to reach local consistency first, or discover that their own subproblem is unsolvable, and then communicate their results. This makes intuitive sense, as the agents' subproblems are completely independent of each other.

At the other extreme, Figure 4 shows that for the *interdependent* class of problems, it is always better for agents to send every assignment than to reach local consistency first. This makes sense as well, as there are no intra-agent constraints to be satisfied, and checking just wastes time.

Figure 5 shows that for the *par* class of problems, with nogoods distributed in proportion to the frequency at which they are possible, neither communication strategy is best all

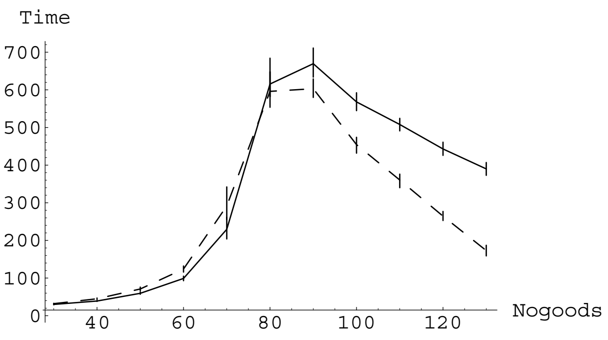

Figure 5. Solution time as a function of number of nogoods for the *par* class of problems, for the *every* strategy (solid line) and the *locally-consistent* strategy (dashed line). Each point is the median for 100 problems of 10 variables and domain size 3, solved 10 times each. Error bars showing 95% confidence intervals are included.

the time. For problems with fewer than about 78 nogoods, it is better to send every assignment as it is made, whereas for more than 78 nogoods, it is better to reach local consistency before sending. Interestingly, 78 nogoods is approximately the transition point for this class of problems: the point at which half the problems are unsolvable (Figure 2). The fact that the two strategies differ in efficiency on either side of the transition point suggests that they may perform differ-

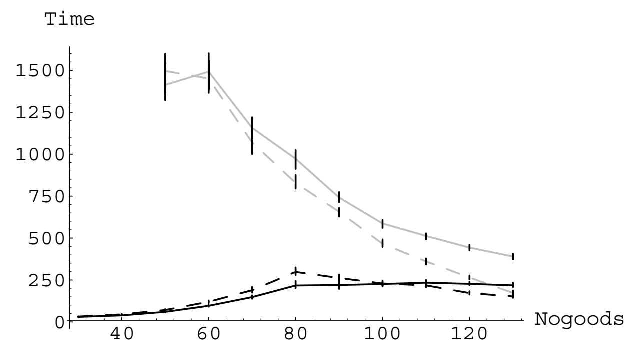

Figure 6. Solution time as a function of number of nogoods for the *par* class of problems, for solvable (black lines) and unsolvable (grey lines) problems considered separately, using the *every* (solid line) and *locally-consistent* (dashed line) strategies. Each point is the median for 100 problems of 10 variables and domain size 3, solved 10 times each. Error bars showing 95% confidence intervals are included.

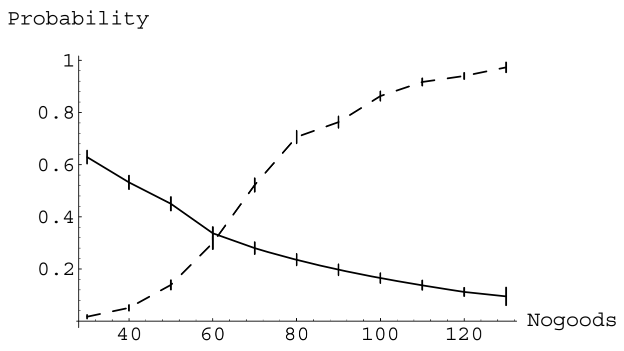

Figure 7. Probability that a complete local solution is locally consistent (solid line), and that a locally-complete, locally-consistent solution is globally consistent (dashed line), for solvable problems, as a function of number of nogoods. Each point is the mean for 500 problems of 10 variables and domain size 3, solved 10 times each, except for 130 nogoods, which is based on 100 problems. Error bars showing 95% confidence intervals, calculated as in Figure 2, are included.

ently for solvable versus unsolvable problems.

Figure 6 shows how the strategies compare for solvable and unsolvable problems considered separately. For unsolvable problems, it is almost always better to use the *locally-consistent* strategy, whereas for solvable problems, it is better to use the *every* strategy up to about 100 nogoods, and then the *locally-consistent* strategy. Thus the relative performance of the two strategies for solvable and unsolvable problems, combined with the crossover in solvability at about 78 nogoods, explains the crossover in advantageous strategy at the same point.

The relative advantage of the two strategies depends in part on the degree to which local work contributes to the global solution. Roughly speaking, if a subproblem solution derived in the absence of external information is very likely to advance problem solving, then agents should employ the *locally-consistent* strategy, which postpones communication until a complete locally-consistent partial solution is reached. If not, then agents should communicate sooner. Clearly, this trade-off is affected by communication costs, including packaging and integration costs. Those costs were minimal in the results reported here.

Figure 7 shows, for solvable problems, (1) the probability that a complete set of local assignments is locally consistent, and (2) the probability that a complete, locally-consistent solution is globally consistent. For solvable problems with more than 100 nogoods, over 90% of complete locally-consistent solutions are globally consistent. Global consistency is so likely that inter-agent exchange of partial solutions, even if relatively inexpensive, is not worthwhile. At lower numbers of nogoods, two factors come into play: (1) a given set of local assignments is less likely to be locally consistent, so more effort is required to reach a locally-complete, locally-consistent solution; and (2) once a complete, locally-consistent solution has been derived, it is less likely to be globally consistent than at greater numbers of nogoods. Both of these factors favor inter-agent sharing of partial subproblem solutions. In this paper, communication overhead was minimal relative to computation time. Higher communication costs would lower the number of nogoods at which the *locally-consistent* strategy becomes advantageous over the *every* strategy.

For unsolvable problems, of course, locally-consistent solutions cannot be extended to global solutions. The fact that the *locally-consistent* strategy is always advantageous except at 30 nogoods suggests that as agents work locally, they derive new nogoods more quickly, and thus reach unsolvability sooner, than they would if they were sharing every assignment made.

5. Conclusions

For problems of the *par* class used in this study, with nogoods distributed in proportion to the actual ratio of intra- versus inter-agent interactions, we can reach some tentative conclusions with respect to the two subproblem sharing strategies examined.

If solvability is known in advance, for example it is known that the class of problems to be solved is solvable, it is better to use the *every* strategy up to the point at which solvability becomes negligible in randomly-generated problems. At this point, solvable problems generally have only one solution, and locally-consistent solutions have a very high probability of global consistency. This point can be approximated analytically [5]. If solvability is not known in advance, one should use the *every* strategy up to the solvability transition point, which can also be approximated analytically, then switch to the *locally-consistent* strategy. The exact point at which the *locally-consistent* strategy will become advantageous over the *every* strategy depends on communication overhead. In our simulations, communication overhead was very low.

We are currently evaluating these strategies for other distributions of nogoods, for problems of different sizes and numbers of agents, and for different amounts of communication overhead, to evaluate the generality of these results and to extend them. We are also studying more complex subproblem-sharing strategies, and examining in detail exactly what agents are doing during problem solving under different scenarios. Finally, we are comparing our empirical results with theoretical predictions on probability of global consistency given locally consistent results of a certain size.

Our goal in this research is to discover and explain domain-independent relationships between problem structure, problem-solving strategies, and problem-solving efficiency. We believe that understanding these relationships and deriving generalizations that can be applied across problem-solving domains is very important. At the same time, it is important to be clear about the assumptions upon which any model is based, or generalizations derived. Our results are derived within the paradigm of constraint satisfaction, and furthermore, are based on a model in which problem variables are assigned to agents (though they could be shared by more than one agent) and constraints are within or between agents. In addition, we make certain assumptions about problem and constraint representation, type of information communicated, and so on. Our results are applicable for problems that can be represented within the CSP paradigm, with variables owned by agents, using the basic problem-solving model that we assumed.

Although the CSP model as we have instantiated it here has wide applicability, there are also many places where it does not apply. Our results will not apply to problems that cannot be represented as CSPs, or to problem-solving systems in which constraints are instantiated as agents, or to systems that deviate in other significant ways from the problem-solving model we have presented here, such as by communication of ranges of values rather than particular assignments or constraint equations rather than nogoods. Some of these represent ways that this model could be extended; others call for alternative models altogether. Our results are domain-independent, but they are not independent of the basic problem-solving model we used; the problem-solving architecture is one piece of the puzzle. There is certainly room for much work in this area.

References

[1] P. Cheeseman, B. Kanefsky, and W. Taylor. Where the *really* hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 331–337, Sydney, Australia, 1991.

[2] K. S. Decker. Distributed artificial intelligence testbeds. In G. O'Hare and N. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, chapter 3. Wiley Inter-Science, 1995. Forthcoming.

[3] K. S. Decker and V. R. Lesser. Quantitative modeling of complex computational task environments. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 217–224, Washington, DC, USA, 1993.

[4] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers*, 36(11):1275–1291, 1987.

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

[6] D. L. Mammen and T. Hogg. A new look at the easy-hard-easy pattern of combinatorial search difficulty. *Journal of Artificial Intelligence Research*, 6:47–66, 1997.

[7] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 459–465, San Jose, CA, USA, 1992.

[8] B. M. Smith. Phase transition and the mushy region in constraint satisfaction problems. In A. Cohn, editor, *Proceedings of the ECAI-94*, pages 100–104. John Wiley and Sons, 1994.

[9] K. Sycara, S. Roth, N. Sadeh, and M. Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1446–1461, 1991.

[10] C. P. Williams and T. Hogg. Using deep structure to locate hard problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 472–477, San Jose, CA, USA, 1992.

[11] C. P. Williams and T. Hogg. Exploiting the deep structure of constraint problems. *Artificial Intelligence*, 70:73–117, 1994.

[12] M. Yokoo. Asynchronous weak-commitment search for solving distributed constraint satisfaction. In *Proceedings of the First International Conference on Principles and Practices of Constraint Programming (CP-95)*, pages 407–422, Cassis, France, 1995.

[13] M. Yokoo and E. Durfee. Distributed search formalisms for distributed problem solving. In *Proceedings of the Eleventh International Workshop on Distributed Artificial Intelligence*, pages 371–390, Glen Arbor, MI, USA, 1992.