Performance Models for Large Scale Multiagent Systems: Using Distributed POMDP Building Blocks

Hyuckchul Jung, Milind Tambe
Computer Science Department, University of Southern California
941 W. 37th Place, Los Angeles, CA 90089-0781, USA
{jungh,tambe}@usc.edu

ABSTRACT

Given a large group of cooperative agents, selecting the right coordination or conflict resolution strategy can have a significant impact on their performance (e.g., speed of convergence). While performance models of such coordination or conflict resolution strategies could aid in selecting the right strategy for a given domain, such models remain largely uninvestigated in the multiagent literature. This paper takes a step towards applying the recently emerging distributed POMDP (partially observable Markov decision process) frameworks, such as MTDP (Markov team decision process), in service of creating such performance models. To address issues of scale-up, we use small-scale models, called *building blocks* that represent the local interaction among a small group of agents. We discuss several ways to combine building blocks for performance prediction of a larger-scale multiagent system.

We present our approach in the context of DCSPs (distributed constraint satisfaction problems), where we first show that there is a large bank of conflict resolution strategies and no strategy dominates all others across different domains. By modeling and combining *building blocks*, we are able to predict the performance of five different DCSP strategies for four different domain settings, for a large-scale multiagent system. Our approach thus points the way to new tools for strategy analysis and performance modeling in multiagent systems in general.

1. INTRODUCTION

In many large-scale applications such as distributed sensor networks, distributed spacecraft, and disaster response simulations, collaborative agents must coordinate their plans or actions [5, 8, 12]. While such applications require agents to be collaborative, an agent's choice of actions or plans may conflict with its neighboring agents' action or plan choices due to limited (shared) resources. Selecting the right action, plan or resource to resolve conflicts, i.e., selecting the right conflict-resolution strategy, can have a significant impact on the performance of conflict resolution particularly in a large-scale multiagent system. For instance, in distributed sensor networks, tracking targets quickly requires that agents controlling different sensors adopt the right strategy to resolve conflicts involving shared sensors.

Unfortunately, selecting the right conflict resolution strategy is difficult. First, there are often a wide diversity of strategies available, and they can lead to significant variations in the rate of conflict resolution convergence in multiagent systems. For instance, when distributed agents must resolve conflicts over shared resources such as shared sensors, they could select a strategy that offers maximum possible resources to most constrained agents, or that distributes resources equally among all agents requiring such resources, and so on. Each strategy may create a significant variation in the conflict resolution convergence rate [5, 12]. Furthermore, faced with certain types of problem domains, a single agent cannot immediately determine the appropriate coordination or conflict resolution strategy, because it is typically not just this agent's actions, but rather the actions of the entire community that determine the outcome.

Performance modeling of multiagent coordination and conflict resolution could help predict the right strategy to adopt in a given domain. Unfortunately, performance modeling has not received significant attention in mainstream multiagent research community; although within subcommunities such as mobile agents, performance modeling has been considerably investigated [14]. Fortunately, recent research in distributed POMDPs (partially observable Markov decision processes) and MDPs (Markov decision processes) has begun to provide key tools to aid multiagent researchers in modeling the performance of multiagent systems [1, 13, 16]. In the context of this paper, we will use the MTDP model [13] for performance modeling, although other models could be used.

There are at least two major problems in applying such distributed POMDP models. First, while previous work has focused on modeling communication strategies within small numbers of agents [13], we are interested in strategy analysis for large-scale multiagent systems. Second, techniques to apply such models to performance analysis of conflict resolution strategies have not been investigated.

We address these limitations in the context of DCSPs (distributed constraint satisfaction problems), which is a major paradigm of research on conflict resolution [15, 17, 18]. Before addressing the limitations, we introduce DCSPs and our previous work in which we have illustrated the presence of multiple conflict resolution strategies and showed that cooperative strategies can improve performance in conflict resolution convergence. Our first contribution in this paper is to illustrate that more strategies exist and that indeed no single strategy dominates all others. Since the best strategy varies over different domains, given a specific domain, selecting the best strategy is essential to gain maximum efficiency.

Our second key contribution is to illustrate the use of MTDP to model the performance of different strategies to select the right strategy. To address the limitations in the MTDP modeling introduced above, we first illustrate how DCSP strategies can be mod-

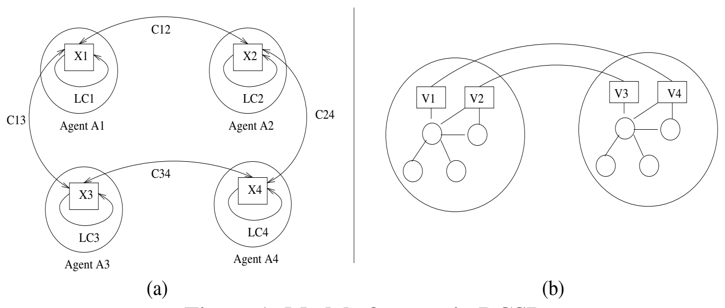


Figure 1: Model of agents in DCSP

eled in MTDP. Next, to address scale-up issues, we introduce small-scale models called "building blocks" that represent the local interaction among a small group of agents. We discuss several ways to combine building blocks for performance prediction of a larger-scale multiagent system.

2. BACKGROUND

DCSP techniques have been used for coordination and conflict resolution in many multiagent applications such as distributed sensor network [8]. In this section, we introduce the DCSP framework and efficient DCSP strategies.

2.1 Distributed Constraint Satisfaction Problems (DCSPs)

A Constraint Satisfaction Problem (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. A DCSP is a CSP in which variables and constraints are distributed among multiple agents [17]. Formally, there is a set of m agents, $Ag = \{A_1, ..., A_m\}$. Each variable (x_i) belongs to an agent A_j. There are two types of constraints based on whether variables in a constraint belong to a single agent or not:

- For a constraint $C_r \in \Gamma$, if all the variables in C_r belong to a single agent $A_j \in Ag$, it is called a *local constraint*.
- For a constraint $C_r \in \Gamma$, if variables in C_r belong to different agents in Ag, it is called an *external constraint*.

Figure 1-a illustrates an example of a DCSP: each agent A_i (denoted by a big circle) has a local constraint LC_i and there is an external constraint C_{ij} between A_i and A_j. As illustrated in Figure 1-b, each agent can have multiple variables. [1] There is no limitation on the number of local/external constraints for each agent. Solving a DCSP requires that agents not only satisfy their local constraints, but also communicate with other agents to satisfy external constraints. Note that DCSPs are not concerned with speeding up centralized CSPs via parallelization; rather, it assumes that the problem is originally distributed among agents.

2.2 Asynchronous Weak Commitment (AWC) Search Algorithm

AWC search algorithm is known to be the best published DCSP algorithm [17]. In the AWC approach, agents asynchronously assign values to their variables from domains of possible values, and communicating the values to neighboring agents with shared constraints.Each variable has a non-negative integer 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 and the constraints between variables are binary. When the value of an agent's variable is not consistent with the values of its neighboring agents' variables, 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 increases its priority to *max+1*, where *max* is the highest priority of its neighboring agents, and selects a new value that minimizes the number of conflicts with all of its neighboring agents. This priority increase makes previously higher agents select new values. Agents avoid the infinite cycle of selecting non-solution values by saving the *nogood* situations.

2.3 Cooperativeness-based Strategies

While AWC is one of the most efficient DCSP algorithms, real-time and dynamism in multi-agent domains demands very fast conflict resolution. Thus, novel strategies were introduced for fast conflict resolution convergence to a complete solution [5]. While AWC relies on the *min-conflict* heuristic [7] that minimizes conflicts with other agents, the new strategies enhanced by local constraint communication consider how much flexibility (choice of values) is given towards other agents by a selected value. By considering neighboring agents' local constraints, an agent can generate a more locally cooperative response, potentially leading to faster conflict resolution convergence.

The concept of local cooperativeness goes beyond merely satisfying constraints of neighboring agents to accelerate convergence. That is, an agent A_i cooperates with a neighbor agent A_j by selecting a value for its variable that not only satisfies the constraint with A_j, but also maximizes A_j's flexibility (choice of values). Then A_j has more choices for a value that satisfies A_j's local constraints and other external constraints with its neighboring agents, which can lead to faster convergence. To elaborate this notion of local cooperativeness, the followings were defined in [5].

- **Definition 1**: For a value $v \in D_i$ and a set of agents $N_i^{sub} \subseteq N_i$ (a set of neighboring agents), a *flexibility function* is defined as $f(v, N_i^{sub}) = \Sigma_j c(v, A_j)$ such that $A_j \in N_i^{sub}$ and $c(v, A_j)$ is the number of values of A_j that are consistent with v.
- **Definition 2**: For a value v of A_i, *local cooperativeness* of v is defined as $f(v, N_i)$. That is, the *local cooperativeness* of v measures how much flexibility (choice of values) is given to all of A_i's neighbors by v.

As an example of the flexibility function $f(v, N_i^{sub})$, suppose agent A_1 has two neighboring agents A_2 and A_3, where a value v leaves 70 consistent values to A_2 and 40 to A_3 while another value v' leaves 50 consistent values for A_2 and 49 to A_3. Now, assuming that values are ranked based on flexibility, an agent will prefer v to v': $f(v, \{A_2, A_3\}) = 110$ and $f(v', \{A_2, A_3\}) = 99$. These definitions of flexibility function and local cooperativeness are applied for the cooperative strategies defined as follows:

- S_{basic}: Each agent A_i selects a value based on min-conflict heuristics (the original strategy in the AWC algorithm);

[1]For simplification, we assume each agent has only one variable.

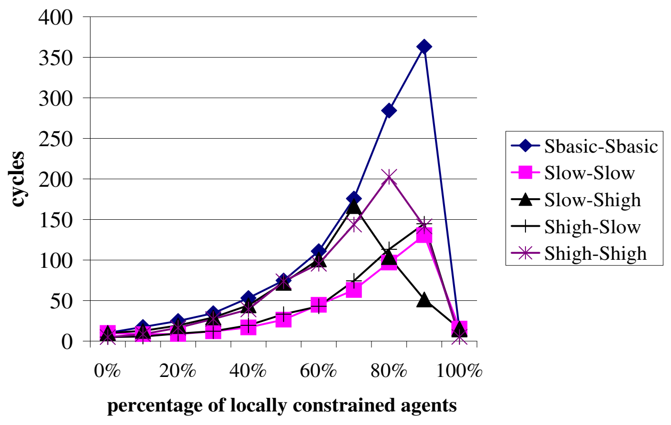


Figure 2: Strategy comparison with cycles

- S_{high}: Each agent A_i attempts to give maximum flexibility towards its higher priority neighbors by selecting a value v that maximizes $f(v, N_i^{high})$;
- S_{low}: Each agent A_i attempts to give maximum flexibility towards its lower priority neighbors by selecting a value v that maximizes $f(v, N_i^{low})$;
- S_{all}: Each agent A_i selects a value v that maximizes $f(v, N_i)$, i.e. max flexibility to all neighbors.

These four strategies can be applied to both the *good* and *nogood* cases. In the *nogood* case, neighboring agents are grouped into higher and lower agents based on the priorities before the priority increase described in Section 2.2. (Refer to [5] for detailed information.) Therefore, there are sixteen strategy combinations for each flexibility base. Since, we will only consider strategy combinations, henceforth, we will refer to them as strategies for short. Note that all the strategies are enhanced with constraint communication and propagation. Here, two exemplar strategies are listed:

- $S_{basic} - S_{basic}$: This is the original AWC strategy. Min-conflict heuristic is used for the good and nogood cases.
- $S_{low} - S_{high}$: For the good case, an agent is most locally cooperative towards its lower priority neighbor agents by using S_{low}. (Note that the selected value doesn't violate the constraints with higher neighbors). On the contrary, for the nogood situations, an agent attempts to be most locally cooperative towards its higher priority neighbors by using S_{high}.

2.4 Experimental Evaluation

An initial principled investigation of these strategies can improve our understanding not only of DCSP strategies, but potentially shed some light on how cooperative an agent ought to be towards its neighbors, and with which neighbors. To that end, a number of DCSP experiments were done with an abstract problem setting. Here, agents (variables) are in a 2D grid configuration (each agent is externally constrained with four neighbors except for the ones on the grid boundary). All agents share an identical binary constraint by which a value in an agent is not compatible with a set of values in its neighboring agent. This grid configuration is motivated by the research on distributed sensor network [8] where multiple agents must collaborate to track targets. For additional justification and domains, refer to [5].

In the experiments, the total number of agents was 512 and each agent has 36 (=6 $\times$ 6) values in its domain. In addition to the external binary constraint, agents can have a unary local constraint that restricts legal values into a set of randomly selected values among its original 36 values. The evaluation followed the method used in [17]. In particular, performance evaluation is measured in terms of cycles consumed until a solution is found, and the experiments ranged over all possible strategies. In Figure 2, for expository purpose, only five strategies are presented, which does not change the conclusions in [5]. The vertical axis plots the number of cycles and the horizontal axis plots the percentage of locally constrained agents. The performance difference between different strategies are proved to be statistically significant by performing t-test.

A key point to note is that choosing the right strategy has significant impact on convergence. Certainly, choosing $S_{basic} - S_{basic}$ (the top line in Figure 2) may lead to significantly slower convergence rate while appropriately choosing $S_{low} - S_{low}$ or $S_{low} - S_{high}$ can lead to significant improvements in convergence. However, we may not need to consider all the strategies for selecting the best since significant performance improvement can be achieved by $S_{low} - S_{low}$ alone.

3. MORE COOPERATIVE STRATEGIES

While novel cooperative strategies in Section 2 improved conflict resolution convergence, two questions remain unanswered: (i) can we apply $S_{low} - S_{low}$ for conflict resolution across different domains?; (ii) in addition to the strategies defined in Section 2, are there more local cooperativeness based strategies?

In this section, to address these two questions, first, we provide more possible DCSP strategies based on the local cooperativeness. Second, experimental results in different types of domains are presented. The results show that there is no single dominant strategy (like $S_{low} - S_{low}$) across all domains.

3.1 New Basis for Cooperativeness

While the novel value ordering strategies (proposed in [5]) improved the performance in conflict resolution, the definition of the flexibility function using summation (Σ) may not be the most appropriate way to compute the local cooperativeness.

- **Example**: suppose we are given two neighboring agents A_1 and A_2, where a value v leaves 99 consistent values to A_1 and 1 to A_2 while another value v' leaves 50 consistent values for A_1 and 49 to A_2. Now, according to the cooperativeness definition in Section 2.3, an agent will prefer v to v': $f(v, \{A_1, A_2\}) = 100$ and $f(v', \{A_1, A_2\}) = 99$. However, leaving one value choice to A_2 by the value v can have high chances of conflicts for A_2 in the future.

Here, we extend the flexibility function to accommodate different types of possible local cooperativeness:

- **Definition 3**: 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.

With this new definition, for the above example, if $\oplus$ is set to min instead of sum, an agent will rank v' higher than v: $f^{min}(v, \{A_1, A_2\}) = 1$ and $f^{min}(v', \{A_1, A_2\}) = 49$. We can apply these extended definitions of the flexibility function to the cooperative strategies defined in [5]. That is, the cooperative strategies definitions hold with the only change in the flexibility function.

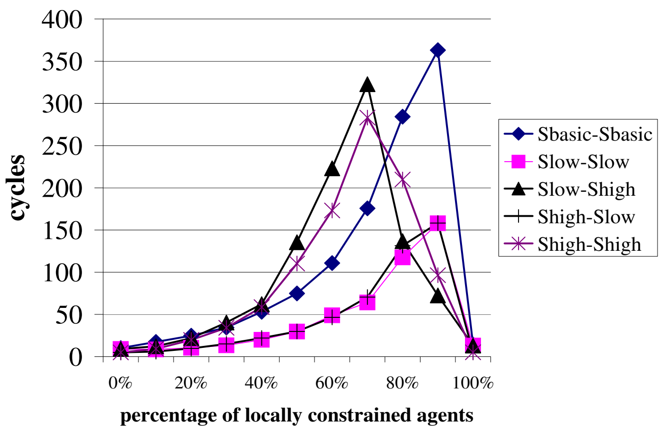


Figure 3: Strategy comparison in high flexibility setting with min as a flexibility base

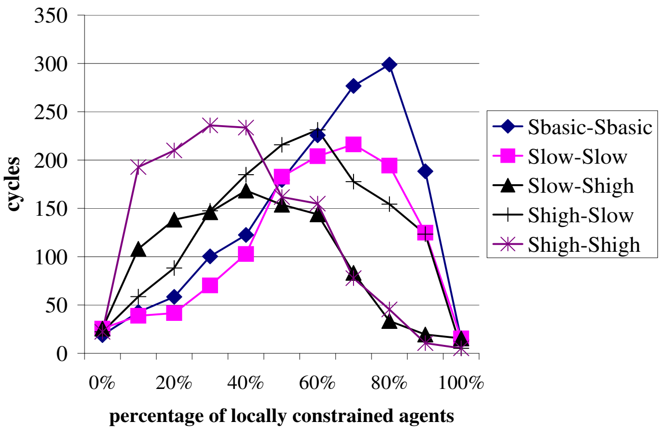


Figure 4: Strategy comparison in low flexibility setting with sum as a flexibility base

3.2 New Experimental Evaluation

To provide the evaluation of these extended strategies, a number of DCSP experiments were done with the problem setting used in [5]. Here, to test the performance of strategies in various problem settings, we make a variation in the problem setting by modifying the external constraint: in a new setting, each agent gives less flexibility (choice of values) towards its neighboring agents given a value in its domain. Henceforth, the previous setting is referred as a *high flexibility setting*, and the new setting with less choice of values to neighbors is referred as a *low flexibility setting*.

Experiments were performed in the two problem settings (the *high flexibility setting* and the *low flexibility setting*), and both sum and min were used as a flexibility base ($\oplus$). Other than this external constraint variation, the parameters for the experiments remain same. Each data point in the figures was averaged over 500 test runs. While all the possible strategies for each flexibility base were tried, for expository purpose, only five strategies are presented in Figure 3 and 4, which does not change our conclusion.

Figure 3 shows the case where min is used as a flexibility base in the *high flexibility setting*. Here, note that $S_{low} - S_{high}$ shows much worse performance than $S_{basic} - S_{basic}$ in some cases. Overall, in this domain, strategies with min as a flexibility base do not improve performance over the strategies with sum as a flexibility base. Figure 4 shows the case where sum is used as a flexibility base, but the problem instances are generated in the *low flexibility setting*. Note that the experimental results shown in Figure 2 were also based on sum as a flexibility base but they were from the *high flexibility setting*. The significant difference between the *high flexibility setting* (Figure 2) and the *low flexibility setting* (Figure 4) is that $S_{low} - S_{low}$ is not the overall dominant strategy in the *low flexibility setting*. Furthermore, $S_{low} - S_{high}$ (that performed well in the *high flexibility setting*) showed worse performance than the original AWC strategy $S_{basic} - S_{basic}$, in particular when the percentage of locally constrained agents is low.

These experimental results show that different flexibility bases also have an impact on performance. For instance, choosing min as a flexibility base ($\oplus$) instead of sum may degrade performance for some cooperative strategies. Furthermore, these results clearly show that choosing the right strategy given a domain has a significant impact on convergence. Certainly, when 80% of agents are locally constrained, $S_{low} - S_{low}$ and $S_{low} - S_{high}$ showed the same performance in the *high flexibility setting* (Figure 2). However, in the *low flexibility setting* (Figure 4), $S_{low} - S_{high}$ showed 10 fold speedup over $S_{low} - S_{low}$.

Here, to check the statistical significance of the performance difference, two-tailed t-test was done for each pair of strategies at each percentage of locally constrained agents. The null hypothesis for each pair of strategies that there was no difference between the two strategies in average cycles was rejected, i.e., the difference is significant, with p-value < 0.01 for each case.

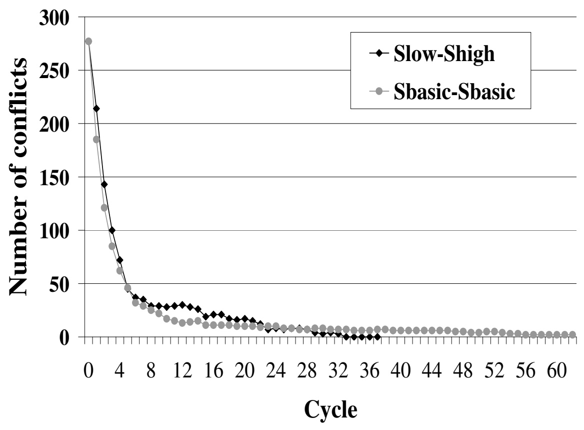


Figure 5: Long-tail convergence example with 90% locally constrained agents in the problem setting for Figure 4

To conclude, no single strategy dominates in different domains. As shown above, the best strategy in a domain could produce 10 times worse performance than others in another domain. Thus, to gain maximum efficiency in conflict resolution, it is essential to predict the right strategy in a given domain.

3.3 Long tail distribution in Convergence

The key factor in determining the performance of strategies is in the long tail where only a small number of agents are in conflicts. Figure 5 shows the number of conflicts at each cycle for two different strategies, $S_{low} - S_{high}$ and $S_{basic} - S_{basic}$. In the beginning of conflict resolution, both strategies show similar performance in resolving conflicts. However, the performance difference appears in the long tail part. While $S_{low} - S_{high}$ quickly solves a given problem, $S_{basic} - S_{basic}$ has a long tail with a small number of conflicts remaining unresolved. This type of long tail distribution has been also reported in many constraint satisfaction problems [4].

4. PERFORMANCE ANALYSIS

Section 3 shows that, given the dynamic nature of multiagent systems, predicting the right strategy to use in a given domain is essential to maximize the speedup of conflict resolution convergence, and the critical factor for strategy performance is in the long tail part where a small number of conflicts exist. Here, we provide formal models for performance analysis and the mapping of DCSP onto the models, and present the results of performance prediction.

4.1 Distributed POMDP-based Model

As a formal framework for strategy performance analysis, we use a distributed POMDP model called MTDP (Multiagent Team Decision Process) model [13]. The MTDP model has been proposed as a framework for teamwork analysis. Distributed POMDP-based model is an appropriate formal framework to model strategy performance in DCSPs since it has distributed agents and the agentView in DCSPs (other agents' values, priorities, etc.) can be modeled as observations. In DCSPs, the exact state of the system is only partially observable to an agent since the received information for the agent is limited to its neighboring agents. Therefore, there is strong correspondence between DCSPs and distributed POMDPs. While we focus on the MTDP model in this paper, other distributed POMDP models such as DEC-POMDP [1] could be used.

Here, we illustrate the actual use of the MTDP model in analyzing DCSP strategy performance. The MTDP model provides a tool for varying key domain parameters to compare the performance of different DCSP strategies, and thus select the most appropriate strategy in a given situation. We first briefly introduce the MTDP model. Refer to [13] for more details.

4.1.1 MTDP model

The MTDP model involves a team of agents operating over a set of world states during a sequence of discrete instances. At each instant, each agent chooses an action to perform and the actions are combined to affect a transition to the next instance's world state. Borrowing from distributed POMDPs, the current state is not fully observed/known and transitions to new world states are probabilistic. Each agent makes its own observations to compute its own beliefs, and the performance of the team is evaluated based on a joint reward function over world states and combined actions.

More formally, an MTDP for a team of agents, α, is a tuple, $< S, A_\alpha, P, \Omega_\alpha, O_\alpha, B_\alpha, R >$. S is a set of world states. $A_\alpha = \prod_{i\in\alpha} A_i$ is a set of combined actions where A_i is the set of agent i's actions. P controls the effect of agents' actions in a dynamic environment: $P(s, a, s') = Pr(S^{t+1} = s'|S^t = s, A^t_\alpha = a)$. Ω_α is a set of combined observations. Observation function, O_α specifies a probability distribution over the observations of an agents team α. Belief state B_α is derived from the observations. $R : S \times A_\alpha \rightarrow \Re$ is a reward function over states and joint actions. A policy in the MTDP model maps individual agents' belief states to actions. A DCSP strategy is mapped onto a policy in the model. Thus, we compare strategies by evaluating policies in this model. Our initial results from policy evaluation in this model match the actual experimental strategy performance results shown before. Thus, the model could potentially form a basis for predicting strategy performance in a given domain.

4.1.2 Mapping from DCSP to MTDP

In a general mapping, the first question is selecting the right state representation for the MTDP. One typical state representation could be a vector of the values for all the variables in a DCSP. However, this representation leads to a huge state space. For instance, if there are 10 variables (agents) and 10 possible values per variable, the number of states is 10^{10}. To avoid this combinatorial explosion in state space, we use an abstract state representation in the MTDP model. In particular, as described in the previous section, each agent can be abstractly characterized as being in a *good* or *nogood* state in the AWC algorithm. We use this abstract characterization in our MTDP model. Henceforth, the *good* state and the *nogood* state are denoted by G and N respectively. Here, an initial state of the MTDP is a state where all the agents are in G state since, in the AWC, an agent finds no inconsistency for its initial values until it receives the values of its neighboring agents. Note that, for simplicity, the case where agents have no violation is not considered.

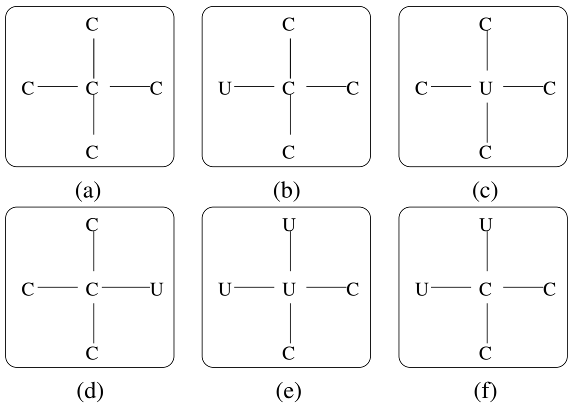


Figure 6: Building block

In this mapping, the reward function R is considered as a cost function. The joint reward (cost) is proportional to the number of agents in the N state. This reward is used for strategy evaluation based on the fact that the better performing strategy has less chance of forcing neighboring agents into the N state than other strategies: as a DCSP strategy performs worse in a given problem setting, more agents will be in the N states.

In the AWC algorithm (a base DCSP algorithm in this paper), each agent receives observations only about the states of its neighboring agents and its current state as well. Thus, the world is not individually observable, but rather it is collectively observable (in the terminology of Pynadath and Tambe [13]), and hence the mapping does not directly reduce to an MDP. Initially, we assume that these observations are perfect (without message loss in communication). This assumption can be relaxed in our future work with unreliable communication.

Here, S_{low}, S_{high}, S_{all}, and S_{basic} in DCSPs are mapped onto the actions for agents in the MTDP model. A DCSP strategy provides a local policy for each agent in the MTDP model, e.g., the $S_{low} - S_{high}$ strategy implies that each agent selects action S_{low} when its local state is *good*, and action S_{high} when its local state is *nogood*. The state transition in the MTDP model is controlled by an agent's own action as well as its neighboring agents' actions. The transition probabilities can be derived from the DCSP simulation.

4.1.3 Building Block

While the abstract representation in the mapping above can reduce the problem space, for a large-scale multiagent system, if we were to model belief states of each agent regarding the state of the entire system, the problem space would be enormous even with the abstract representation. For instance, the number of states in the MTDP model for the system with 512 agents would be 2^{512}: each agent can be either G or N. To further reduce the combinatorial explosion, we use small-scale models, called *building blocks*. Each building block represents the local situation among five agents in the 2D grid configuration.

In the problem domains for the experiments shown in Section 2.4 and 3.2, each agent's local situation depends on whether it has a

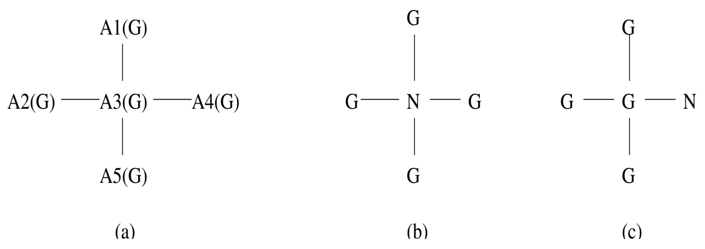


Figure 7: States in a building block

unary local constraint or not: each agents can be either constrained (a portion of its original domain values are not allowed) under a local constraint (C) or unconstrained (U). Figure 6 illustrates some exemplar building blocks for the domain used in the experiments. For instance, Figure 6-a represents a local situation where all the five agents are constrained (C) while Figure 6-b represents a local situation where an agent in the left side is unconstrained (U) but the other four agents are locally constrained (C).

Note that, when the percentage of locally constrained agents is high, most of building blocks would be the one shown in Figure 6-a and a small portion of building blocks would be like the ones shown in Figure 6-b, 6-c, and 6-d. As the percentage of locally constrained agents decreases, more building blocks include unconstrained agents (U) as shown in Figure 6-e and 6-f.

In each building block, as shown in Figure 7, a middle agent (A_3) is surrounded by the other four neighboring agents (A_1, A_2, A_4, A_5). Thus, the state of a building block can be represented as a tuple of local states $<s_1, s_2, s_3, s_4, s_5>$ (e.g., $<G, G, G, G, G>$ if all the five agents are in the *good* (G) state). There are totally 32 states in a building block of the MTDP model (e.g., $<G, G, G, G, G>$, $<G, G, G, G, N>$, $<G, G, G, N, G>$, etc), and the initial state of a building block is $<G, G, G, G, G>$. Here, agents' actions will cause a transition from one state to another. For instance, if agents are in a state $<G, G, G, G, G>$ (Figure 7-a) and all the agents choose the action S_{high}, there is a certain transition probability that the next state will be $<G, G, N, G, G>$ (Figure 7-b) when only the third agent is forced into a *nogood* (N) state. However, the agents may also transition to $<G, G, G, N, G>$ (Figure 7-c) if only the fourth agent enters into the N state.

One may argue that these small-scale models are not sufficient for the performance analysis of the whole system since they represent only local situations. However, as seen in Figure 5, the key factor in determining the performance of strategies is in the long tail where only a small number of agents are in conflicts. Therefore, the performance of strategies are strongly related to the local situation where a conflict may or may not be resolved depending on the local agents' actions. That is, without using a model for the whole system, small scale models for local interaction can be sufficient for performance analysis. Furthermore, while this simple model may appear limiting at first glance, it has already shown promising results presented below.

4.1.4 Building Block Composition

While the building blocks are the basis for performance analysis, we need to deal with multiple building blocks that can exist in a given domain. Each building block has a different impact on conflict resolution convergence. It is expected that the local interaction in a single building block does not totally determine the performance of strategies, but the interactions between building blocks have a great impact on the strategy performance. Here, we propose four methods of building block composition to evaluate MTDP policies (mapping of DCSP strategies) as follows:

- **Single block**: for a given domain, a single building block is selected. The selected block has the highest probability of producing the *nogood* (N) cases within the building block. The performance of a strategy is evaluated based on the value of an initial state of the building block ($<G, G, G, G, G>$) given its mapped policy: As the initial state value gets lower, the policy (strategy) is better.
- **Simple sum**: for a given domain which has multiple building blocks, we compute the value of each building block's initial state. Performance evaluation of a policy is based on the summation of the initial states' values of the multiple building blocks in the domain.
- **Weighted sum**: given multiple building blocks, for each building block, we compute the ratio of the building block in the domain and the value of its initial state. Performance evaluation of a policy is based on the weighted sum of initial states' values where the weight is the ratio of each building block.
- **Interaction**: a sequence of building blocks is considered since the performance difference comes from the long tail part (shown in Figure 5), and their interaction is taken into account: an agent may force its neighboring agents to enter into the *nogood* (N) state given an action under a policy to evaluate. For instance, two blocks, Figure 6-(b) and Figure 6-(c), may interact side by side, so that the rightmost C agent of Figure 6-(b) interacts with the leftmost C agent of Figure 6-(c). Here, with the *interaction* between these two building blocks, the state of the rightmost C agent of Figure 6-(b) and its policy influence the probability that the initial local state of the leftmost C agent of Figure 6-(c) starts in the N state. Without such *interaction*, it would always start in the G state.

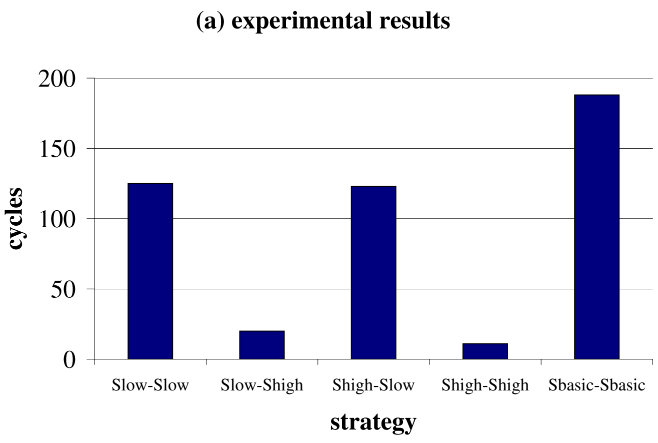
For the *interaction* method, we don't have arbitrary degrees of freedom: for different domains, there can be commonalities in building blocks. That is, for common building blocks, the same transition probabilities within a building block are applied. As we change from the first method (*single block*) to the fourth method (*interaction*), we gradually increase the complexity of composition. The accuracy of the performance prediction with those methods is presented in the next section.

Here, note that the focus of our building block composition is not on computing the optimal policy, but it remains on matching the long-tailed phenomenon shown in Figure 5. Thus, our interactions essentially imply that neighboring blocks affect each other in terms of the values of the policies being evaluated, but we are not computing optimal policies that cross building block boundaries.

4.2 Performance Prediction

To check whether the MTDP based model can effectively predict the performance of strategies, performance analysis is done (based on the four methods of building block composition in Section 4.1.4), and the performance evaluation results from the MTDP model are compared with the real experimental results presented in the previous sections. While there can be various problem domains, in this initial investigation, we focus on two special cases where the percentages of locally constrained agents are 90% and 50% respectively (either most of agents are locally constrained or half of agents are locally constrained). [2] These two cases are considered for both the *high flexibility setting* and the *low flexibility setting*. Among the possible strategies defined in Section 3.1, we focus on the strategies with sum as a flexibility base that were selected in Section 2 for

[2]There is no significant performance difference in the 0% case.



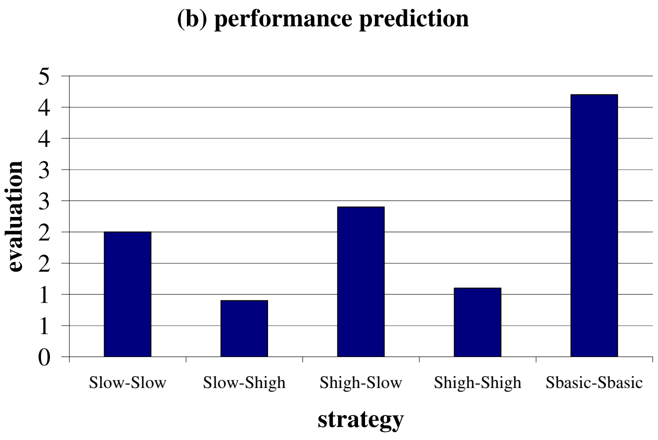


Figure 8: Performance prediction in low flexibility setting with 90% locally constrained agents

expository purpose: $S_{low} - S_{low}$, $S_{low} - S_{high}$, $S_{high} - S_{low}$, $S_{high} - S_{high}$, and $S_{basic} - S_{basic}$.

Figure 8 shows the results when 90% agents are locally constrained in the *low flexibility setting*. Figure 8-(a) (top) shows the experimental results in the case and Figure 8-(b) (bottom) shows the performance evaluation from the MTDP model using the *interaction* composition method. In the performance evaluation, the lower value means the better performance (indicating that less number of cycles will be taken until a solution is found). Figure 8-(a) and (b) show that the performance prediction results match to the real experimental results in the problem setting considered here: the patterns of column graphs in Figure 8-(a) and (b) correspond to each other. That is, the strategies that showed better performance in the experiments ($S_{low} - S_{high}$ and $S_{high} - S_{high}$) are expected to perform better than the others according to the performance prediction. Here, the current goal of performance prediction is mainly to predict the best performing strategy not to predict the magnitude of speedup difference among the given strategies.

Here, a question remains to be answered for whether the other composition method of building blocks can provide the same power of performance prediction as the *interaction* method does. Figure 9 shows the performance prediction results with different composition methods for the same problem domain used in Figure 8. The rightmost part is for the *interaction* method shown in Figure 8-(b). It is shown that the result from the *single block* method does not match at all. While the *simple sum* or *weighted sum* method provides a little improvement over the *single block* method, they are far from matching to the real experimental results (Figure 8-(a)).

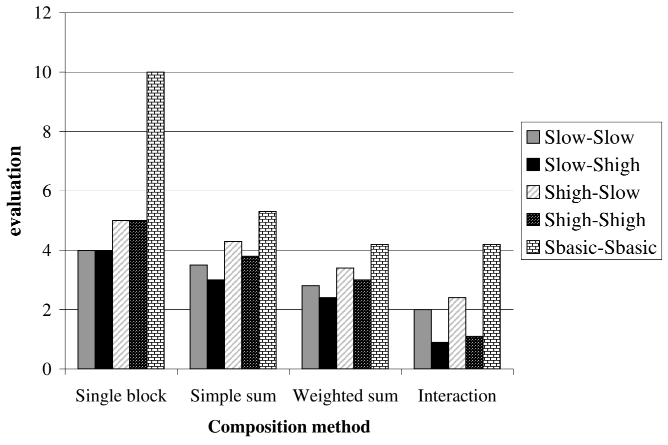


Figure 9: Comparison of composition methods

That is, the composition method that considers the interaction between building blocks shows the best prediction results. Note that 5 building blocks were combined in each prediction (except for the *single block* method where only one selected block was used), while ensuring that the ratio of U (unconstrained) and C (locally constrained) agents were what was required for each domain.

Figure 10 also shows the performance prediction results in different domains using the *interaction* method of building block composition. Here, the performance analysis distinguishes better performing strategies from worse performing strategies, and this difference is statistically significant. The correlation coefficient between the speedup in cycles and the difference in performance evaluations was 0.83. Furthermore, the hypothesis that there is no such correlation was rejected with p-value of 0.0001.

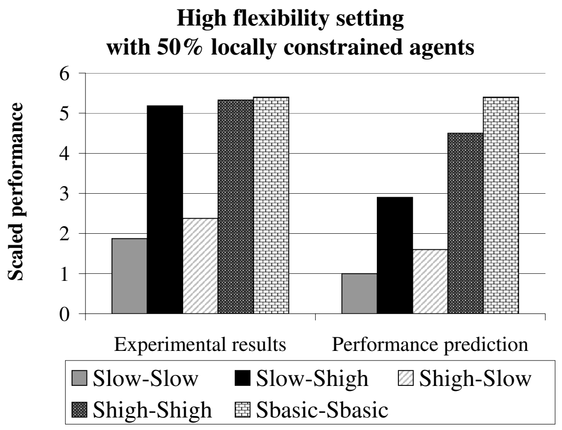
This result illustrates that the MTDP model can be used to predict the right strategy to apply in a given situation (possibly with less computation overhead). That is, given a new domain, agents can analyze different strategies with the simple MTDP model and select the right strategy for the new domain without running a significant number of problem instances for each strategy. Furthermore, this approach will enable agents to flexibly adapt their strategies to changing circumstances. More generally, this result indicates a promising direction for performance analysis in DCSPs, and potentially other multiagent systems.

5. RELATED WORK AND CONCLUSION

Significant works in multiagent learning are focused on learning to select the right coordination strategy [12, 3]. While this goal is related to our goal of choosing the right strategy, one key difference is that the learning work focuses on enabling each agent to select a strategy. Instead, our focus is on a complementary goal of trying to predict the overall performance of the entire multiagent system assuming homogeneous conflict resolution strategies.

In centralized CSPs, the performance prediction for different heuristics has been investigated. However, their methods are based on the estimation of nodes to expand for search[6]. However, this approach is not applicable to DCSPs since multiple agents simultaneously investigate their search space. Theoretic investigations of heuristic performance were also done in centralized CSPs [7, 9]. However, no theoretical investigation has been done for the performance prediction in DCSPs.

While there are related works in composition methods of subproblem solutions in MDPs[2, 10] and in POMDPs[11], we are really interested in applying these composition techniques for performance modeling, not computing an optimal policy. For instance,



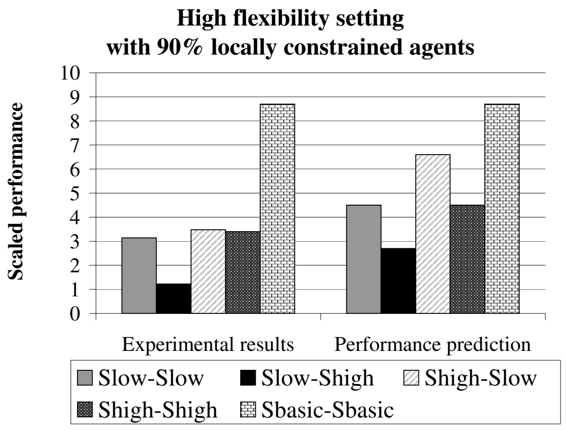


Figure 10: Performance prediction in additional settings

our techniques are heavily influenced by the need to capture the long-tailed phenomena in conflict resolution.

To conclude, in this paper, the recently emerging distributed POMDP frameworks such as MTDP were used to create performance models for conflict resolution strategies in multiagent systems. To address issues of scale-up, we used small-scale models, called *building blocks* that represent the local interaction among a small group of agents. We discussed several ways to combine building blocks for the performance prediction of larger-scale multiagent systems.

These approaches were presented in the context of DCSPs (distributed constraint satisfaction problems), where we first showed that there is a large bank of conflict resolution strategies and no strategy dominates all others across different domains. By modeling and combining *building blocks*, we were able to predict the performance of five different DCSP strategies for four different domain settings, for a large-scale multiagent system. Thus, our approach points the way to new tools for the strategy analysis and performance modeling in multiagent systems in general.

6. ACKNOWLEDGEMENT

The research in this paper was funded by NASA JPL subcontract "Continual Coherent Team Planning". We thank Anthony Barrett and Brad Clement for valuable discussions and inputs.

7. REFERENCES

[1] D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of mdps. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence*, 2000.

[2] T. Dean and S. Lin. Decomposition techniques for planning in stochastic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.

[3] C. Excelente-Toledo and N. Jennings. Learning to select a coordination mechanism. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2002.

[4] C. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomenon in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24, 2000.

[5] H. Jung, M. Tambe, and S. Kulkarni. Argumentation as distributed constraint satisfaction: Applications and results. In *Proceedings of the International Conference on Autonomous Agents*, 2001.

[6] L. Lobjois and M. Lemaitre. Branch and bound algorithm selection by performance prediction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, 1998.

[7] S. Minton, M. D. Johnston, A. Philips, and P. Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the National Conference on Artificial Intelligence*, 1990.

[8] 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.

[9] R. Musick and S. Russell. How long it will take? In *Proceedings of the National Conference on Artificial Intelligence*, 1992.

[10] R. Parr. Flexibile decomposition algorithms for weakly coupled Markov decision problems. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence*, 1998.

[11] J. Pineau, N. Roy, and S. Thrun. A hierarchical approach to POMDP planning and execution. In *Proceedings of the ICML Workshop on Hierarchy and Memory in Reinforcement Learning*, 2001.

[12] N. Prasad and V. Lesser. The use of meta-level information in learning situation-specific coordination. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1997.

[13] D. Pynadath and M. Tambe. The communicative multiagent team decision problem: analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 2002.

[14] O. Rana. Performance management of mobile agent systems. In *Proceedings of the International Conference on Autonomous Agents*, 2000.

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

[16] P. Xuan and V. Lesser. Multi-agent policies: from centralized ones to decentralized ones. In *Proceedings of the International Conference on Autonomous Agents*, 2002.

[17] M. Yokoo and K. Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *Proceedings of the International Conference on Multi-Agent Systems*, 1998.

[18] W. Zhang and L. Wittenburg. Distributed breakout revisited. In *Proceedings of the National Conference on Artificial Intelligence*, 2002.