Exploiting Factored Representations for Decentralized Execution in Multi-agent Teams

Maayan Roth*
Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA, USA
mroth@andrew.cmu.edu

Reid Simmons
Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA, USA
reids@cs.cmu.edu

Manuela Veloso
Computer Science
Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA, USA
veloso@cs.cmu.edu

ABSTRACT

In many cooperative multi-agent domains, there exist some states in which the agents can act independently and others in which they need to coordinate with their teammates. In this paper, we explore how factored representations of state can be used to generate factored policies that can, with minimal communication, be executed distributedly by a multi-agent team. The factored policies indicate those portions of the state where no coordination is necessary, automatically alert the agents when they reach a state in which they do need to coordinate, and determine what the agents should communicate in order to achieve this coordination. We evaluate the success of our approach experimentally by comparing the amount of communication needed by a team executing a factored policy to a team that needs to communicate in every timestep.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms, Design, Performance

Keywords

decentralized execution, multi-agent MDP, communication

1. INTRODUCTION

The problem of coordinating teams of cooperative agents operating under uncertainty is a difficult one that has received a great deal of recent attention. This problem is a challenging one because, while each agent observes only its own local point of view, the agents must reason about both the states and actions of their teammates in order to act in coordination. Several extensions of the Markov decision process have been proposed to enable decision-theoretic modeling of multi-agent teams. Decentralized MDPs (Dec-MDP) can be used to model teams that, like the ones addressed in this paper, operate under *collective observability* [1]. In a collectively observable domain, while each agent is unable to independently identify the world state, the union of all teammate observations at a given timestep renders the state fully observable.

Although Dec-MDPs provide a comprehensive semantics for modeling multi-agent teams in collectively observable domains, their use in planning for such teams is limited by the intractability of generating their policies. Even for teams of as few as two agents, the problem of generating an optimal policy for the Dec-MDP representing that team is NEXP-complete [1]. One approach to overcoming the high complexity of planning for Dec-MDPs has been the identification of classes of domains that are easier to solve. One such class, *transition-independent* Dec-MDPs are domains in which the global state is partitionable into local states for each agent. This local state depends only on the individual actions of a single agent, rather than on the joint actions of the team. However, because the team shares a joint reward that depends on the joint state and action, a transition-independent Dec-MDP cannot be solved as a set of independent MDPs. Policy-generation for a transition-independent Dec-MDP is NP-complete [6], a significant reduction in complexity from the problem of solving a general Dec-MDP. Unfortunately, not all multi-agent domains are transition-independent.

Factoring is another approach that has been applied to the problem of planning for multi-agent teams. Factoring has been applied successfully to the problem of speeding up policy computation for large MDPs (e.g. [2, 7]). Unlike standard MDP representations which enumerate all possible states, factored MDPs take advantage of conditional independences among state variables to allow more compact problem representations, and in some cases, more compact policies. The presence of an additional type of independence, called *context-specific independence*, can lead to even greater savings in the size of problem representation and the efficiency of planning [3]. Work has been done in applying factoring to decentralized planning for multi-agent MDPs [8]. In that approach, it is assumed that agents have *full observability* of all relevant state variables and the chal-

*The first author is a student.

AAMAS'07 May 14–18 2007, Honolulu, Hawai'i, USA.

lenge is to enable those agents to plan independently without reasoning over the full set of joint actions.

When agents must coordinate without communication, generating a policy for the team is NEXP-complete. However, if it is assumed that agents will be allowed to communicate with their teammates at every timestep during execution, then a centralized policy, which we refer to as the *free-communication policy*, can be generated in polynomial time, as if the entire team were being modeled by a single-agent MDP [12]. In general, though, communication is not free, and it is necessary to minimize the use of communication resources during execution. Like Roth *et al.* [13], we assume that each agent in our decentralized team can independently compute the free-communication policy, and does so at plan-time. In this paper, we explore the challenge of decentralized execution of this centralized policy while minimizing the use of communication resources.

In our work, we employ techniques for solving factored MDPs to generate the centralized, free-communication plan for our team. We believe that there is a natural synergy in applying factored representations to cooperative multi-agent teams. When represented for centralized planning, even assuming the presence of free communication, multi-agent teams can quickly form large domains with many relevant state variables and joint actions. Factored representations and solution techniques can speed up policy-generation in large domains, but only if those domains contain significant amounts of conditional and context-specific independence. It is our claim that many multi-agent domains exhibit a large amount of context-specific independence, allowing them to be represented compactly and solved efficiently using factored representations. Although there will be times when agents need to coordinate their actions or gain knowledge of their teammates' local state variables, multi-agent domains also contain situations in which agents can act independently for long periods of time. The challenge is to identify the independences present in any given domain, and thus enable agents to coordinate only when necessary. Through addressing this challenge, factoring can enable the efficient solution of and execution in some Dec-MDP domains.

In the remainder of this paper, we show how factored policies can enable a team of cooperative agents to execute independently, with each agent choosing *when* and *what* to communicate if it is necessary for team coordination. We introduce an algorithm for transforming a centralized factored policy into a distributed policy, and evaluate the success of our approach experimentally.

2. FACTORED DEC-MDP MODEL

The Dec-MDP model is a Markov decision process extended to handle joint actions and observations by a cooperative multi-agent team [1]. A Dec-MDP is composed of the tuple $\langle \alpha, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ where: α is the number of agents on the team, $\mathcal{S}$ is a finite set of world states, $\mathcal{A}$ is the set of possible joint actions, with $\mathcal{P}$ defining a transition function that specifies the likelihood of the team transitioning from one state to the next given a particular joint action, and $\mathcal{R}$ specifies the joint reward function of the team, given the state and a joint action.

The state, $\mathcal{S}$, is comprised of n state features or variables $\mathcal{X} = \langle \mathcal{X}_1 \ldots \mathcal{X}_n \rangle$, where each state s_i is an assignment of values to each feature in $\mathcal{X}$. The features may be binary or multi-valued, or both. While in the most general case, a Dec-MDP model also contains a set of joint observations, Ω, we limit ourselves to domains in which the state features are can be partitioned into α sets of local state features, one for each agent, with each agent observing its local features perfectly, and a set of joint features that are fully observable by all of the agents.

Each agent i has a set $\mathcal{A}^i$ of possible individual actions. Every joint action in $\mathcal{A}$ is comprised of one individual action for each agent on the team. Because our model is factorable, the relationship between state features and joint actions can be represented as a Dynamic Decision Network [4]. For each joint action, there is a Bayes network (such as the example network shown in Figure 1) that specifies the relationship between state variables at time t and time $t+1$. State features at time $t+1$ are dependent on the values of their parent features in time t and are conditionally independent of all other features. Associated with each variable is a conditional probability table describing the probability of that variable given the different possible values of its parents. In many cases, a variable will depend on certain parents in some contexts but not others. When this is the case, the conditional probability table can be represented more compactly as a tree [3]. For example, in the conditional probability tree for variable $\mathcal{X}_2$ at time $t+1$, shown in Figure 1, $\mathcal{X}_2$ depends on the value of $\mathcal{X}_3$ at time t when $\mathcal{X}_1$ has the value 1, but is independent of $\mathcal{X}_3$ otherwise. This type of independence is referred to as *context-specific independence* [3].

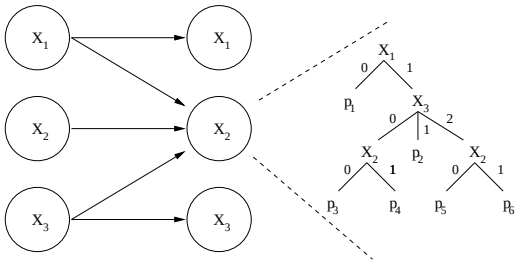

Figure 1: Example action network for a joint action $a_k \in \mathcal{A}$, with tree-structured CPT for $\mathcal{X}_2$.

Our use of a factored state representation in modeling our multi-agent teams enables us to generate tree-structured policies like the ones described in Boutilier *et al.* [2]. These factored policies take the form of a decision tree, where the internal nodes are state variables and the leaves are joint actions. Structured Policy Iteration (SPI) is an algorithm that can be used to compute decision tree-structured policies for factored MDPs. We refer you to [2] for the details of the algorithm, but point out that it requires the MDP to be represented as a Dynamic Decision Network, with conditional probability tables for the variables represented as trees, and that the reward function, also represented as a tree, depend only on state features and not on the selected action. This constraint is easily met by adding additional state variables to the domain representation. We use SPI to generate a factored free-communication policy for our team. Figure 2 shows an example of one such factored policy. The actions at the leaves are joint actions, and $\mathcal{V}1$ and $\mathcal{V}2$ are state variables.

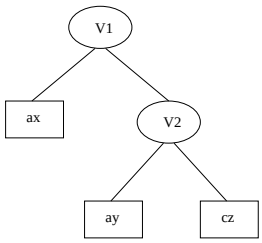

Figure 2: A factored policy over joint actions. For example, the joint action ax indicates that one agent executes action a while the other agent executes action x.

3. DECENTRALIZED EXECUTION OF AN INDIVIDUAL FACTORED POLICY

In order to execute distributedly, each agent on our team must have a policy that maps assignments of state features to individual actions. In Section 4 we detail an algorithm for converting a factored joint policy generated by SPI into individual factored policies. For now, we note that an agent's individual policy may depend on state features that it cannot observe directly. For example, in the policy for agent i shown in Figure 3, the feature labeled $\mathcal{X}^j$ a local feature of agent j. An agent executes a factored policy by traversing the policy tree, choosing branches according to the value assignments of the state variables that it encounters, until it reaches an action at a leaf. Paths through the tree that depend only on agent i's features indicate portions of the state space where that agent has context-specific independence from its teammates. Communication is needed to facilitate the execution of those portions of the policy without this context-specific independence.

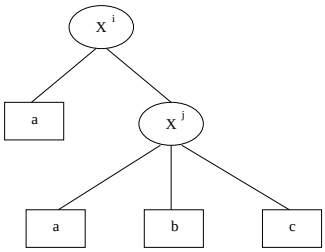

Figure 3: A factored policy over individual actions. $\mathcal{X}^i$ is a state variable observed by agent i, and $\mathcal{X}^j$ is a variable observed by agent j.

Previous approaches to communication for Dec-MDPs and Dec-POMDPs has utilized either the *tell* paradigm of communication, in which each agent reasons about its local knowledge and decides whether to send information to its teammates (e.g. [13]), or the *synchronize* paradigm, in which if at least one agent initiates communication, all the teammates broadcast their full observation histories (e.g. [11, 5]). In this paper, we consider a different option, *query* communication. Instead of trying to predict if its information will be useful to its teammates, each agent asks its teammates for information when needed. In order to *query*, agents must reason both about *when* communication is necessary, as well as *what* information is needed. Unlike other approaches which use broadcast communication, *query* communication is peer-to-peer, with each agent asking for information only from the teammate who possess it.

Table 1 shows how communication enables the distributed execution of a factored policy. Each agent observes its local state features, as well as any global state variables that are observable to the team, and traverses the policy tree according to the values of those features until it either reaches a leaf, whereupon it executes the individual action at that leaf, or until it reaches a decision point in the policy that depends on a feature to which it does not have local access. When this happens, the agent knows it must ask its teammate to communicate the feature value. The amount of communication needed during execution measures of the degree of context-specific independence among agents in a domain. Agents executing in domains where they are often independent of their teammates will need to communicate less than agents in domains where they must coordinate frequently.

Table 1: Recursively execute a factored policy, communicating when necessary.

function EXECUTE(*policy*, *features*, *teammate_features*)
returns an *action*
input: *policy*, a tree-structured policy for agent i
features, the current values of i's state features
teammate_features, a set of teammate features for which values are known (from previous communication)

1. **if** *policy* is a leaf **return** *policy.action*
2. **if** *policy.variable* is in *features*
   a. $x \leftarrow$ *features*[*policy.variable*]
   b. **return** EXECUTE(*policy.child*$_x$, *features*, *teammate_features*)
3. **if** *teammate_features* is empty
   a. Find all the variables in *policy.*
   b. Build *teammate_variables* by asking teammates for the values of those variables.
4. $x \leftarrow$ *teammate_features*[*policy.variable*]
5. **return** EXECUTE(*policy.child*$_x$, *features*, *teammate_features*)

To avoid the possibility that an agent will have to communicate more than once per timestep, when an agent reaches a decision point where it discovers that it needs a piece of information local to one of its teammates, the agent attempts to predict what other information it may need to know. The agent searches for all of the variables contained in the subtree of its policy rooted at this decision point and asks its teammates for all of those feature values. This means that, in some cases, the agent will be requesting information that it later discovers is irrelevant. However, we believe that in most domains where it is necessary to conserve communication resources, it is desirable to minimize the total number of messages sent, not the size of those messages. Only a minor change is required in the execution algorithm to accommodate domains in which it is desirable to minimize the total number of features communicated.

4. GENERATING INDIVIDUAL FACTORED POLICIES FOR MULTI-AGENT TEAMS

Structured Policy Iteration relies on several tree operations defined by Boutilier *et al.*, namely the ability to MERGE multiple policy trees such that the resulting tree contains the maximum policy attainable among the trees, and to SIMPLIFY trees to reduce redundant branches and subtrees. In order to convert a factored policy for a centralized team into factored policies for individual agents, we extend the policy tree representation to allow for multiple actions at any given leaf, indicating the presence of a tie, and introduce two additional tree operations, INTERSECT, which simplifies a tree by combining branches whose leaves have non-empty action-set intersections, and INDEPENDENT, which computes, for a single leaf, the set of individual actions that can be performed without considering the action choices of a teammate.

The ability to detect and propagate ties between joint actions in a policy improves the discovery of context-specific independences among agents. If there is a portion of the state where an agent i can act independently, without needing to coordinate with its teammates' actions, the values of the joint actions composed of i's optimal action and all possible actions that i's teammates could choose will be the same. To this end, we modify MERGE, which in its original form picks one joint action at random when it encounters a tie, to instead build trees where leaves may contain a set of optimal joint actions. There is now an additional simplification that can be performed on a policy tree, INTERSECT, detailed in Table 2 and in the example in Figure 4. At each non-leaf node in the policy tree, INTERSECT is recursively called on the node's children. After this, if every child of the node is a leaf, and the action sets of the node's children have a non-empty intersection, the node is redundant and can be replaced with a new leaf containing the intersection. If all children but one are leaves, a node may still be redundant. This redundancy is detected by examining the subtree that is not a leaf. If all of the actions at the leaves of that subtree are present in the intersection of the node's leaf-children's action sets, the node may be replaced by its non-leaf child.

Table 2: Simplify a policy tree through intersection of subtrees that contain the same actions.

function INTERSECT(*policy*) **returns** a *policy*
input: *policy*, a tree-structured policy

1. **if** *policy* is a leaf **return** *policy*
2. Recursively call INTERSECT on each *child* in *policy*.
3. **if** every *child* of *policy* is a leaf
   return $\bigcap_i$ $child_i.actions$ **for each** $child_i$ of *policy*
4. **if** exactly one *child*, $child_{NL}$, of *policy* is a non-leaf
   a. get $\mathcal{A}_L$, the intersection of the leaf-children's action sets
   b. get $\mathcal{A}_{NL}$, the set of all actions at the leaves of $child_{NL}$'s subtree
   c. **if** $\mathcal{A}_{NL}$ is a subset of $\mathcal{A}_L$ **return** $child_{NL}$
5. **return** *policy*

An individual action is considered INDEPENDENT for an agent i in a particular leaf of a policy tree if the action is optimal when paired with any of the other individual actions that its teammates may choose at that leaf. Figure 5 shows an example of a leaf in which an INDEPENDENT action can be discovered for agent 1, and one in which no INDEPENDENT action exists.

(a) (b)

Figure 5: Independent action examples. (a) In this leaf, action a is can be performed independently by agent 1, since it forms an optimal joint action with x and y, all the possible actions that agent 2 might perform at this leaf. (b) There is no Independent action for agent 1 in this leaf.

The full process of transforming a factored joint policy into factored individual policies for each agent can be found in Table 3. First, the joint policy is re-written for each agent so as to move the state features visible to that agent to the upper-most nodes of the tree. This is done to increase the likelihood that, during execution, an agent will be able to traverse portions of its policy without requiring communication. This rewritten tree is processed to discover the INDEPENDENT actions for the agent at each leaf. Leaves containing independent actions are labeled with those actions, while those where the set of independent actions is empty retain their joint action sets. If any ties remain among the joint actions, they are now broken according to a predetermined canonical action ordering. This is necessary to avoid equilibrium-selection mis-coordinations like the one shown in Figure 6. At this point, the joint actions in the policy tree are converted to individual actions. Finally, INTERSECT is performed and the tree is simplified one final time.

Table 3: Transform a factored joint policy into factored individual policies for each agent..

function GENERATEINDIVIDUALPOLICIES
for each agent:

1. Make a joint policy for that agent, with its state variables at the root.
2. **For each** leaf in the policy, find INDEPENDENT actions
3. Break ties among remaining joint actions using canonical action ordering.
4. Convert joint actions to individual actions.
5. INTERSECT and SIMPLIFY.

5. EXPERIMENTAL RESULTS

We demonstrate the effectiveness of our approach on an example problem, a Meeting-Under-Uncertainty domain. In this problem, two agents must meet at a predetermined location in a grid world, and when both are at the goal location, simultaneously send up a SIGNAL. The other individual actions available to the agents are NORTH, SOUTH,

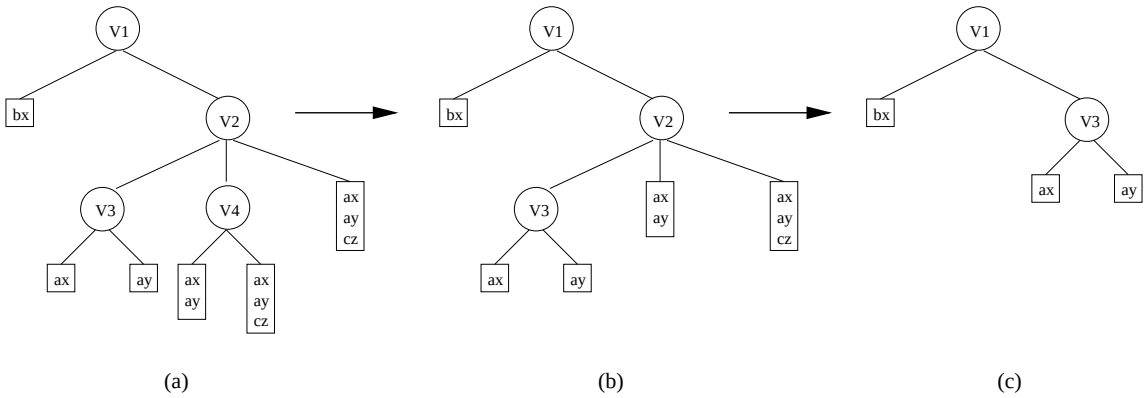

Figure 4: Intersect example. Agent 1 can take individual actions $\{a, b, c\}$ and agent 2 can take actions $\{x, y, z\}$. (a) The node labeled V4 is simplified by Intersect because all of its children are leaves. The intersection of the leaves' action sets is $\{ax, ay\}$. (b) V2 is simplified because it has only one child that is not a leaf (the subtree rooted at V3). Its non-leaf children's action sets have the intersection $\{ax, ay\}$, which includes all of the actions at the leaves of the non-leaf subtree. (c) No further intersection is possible.

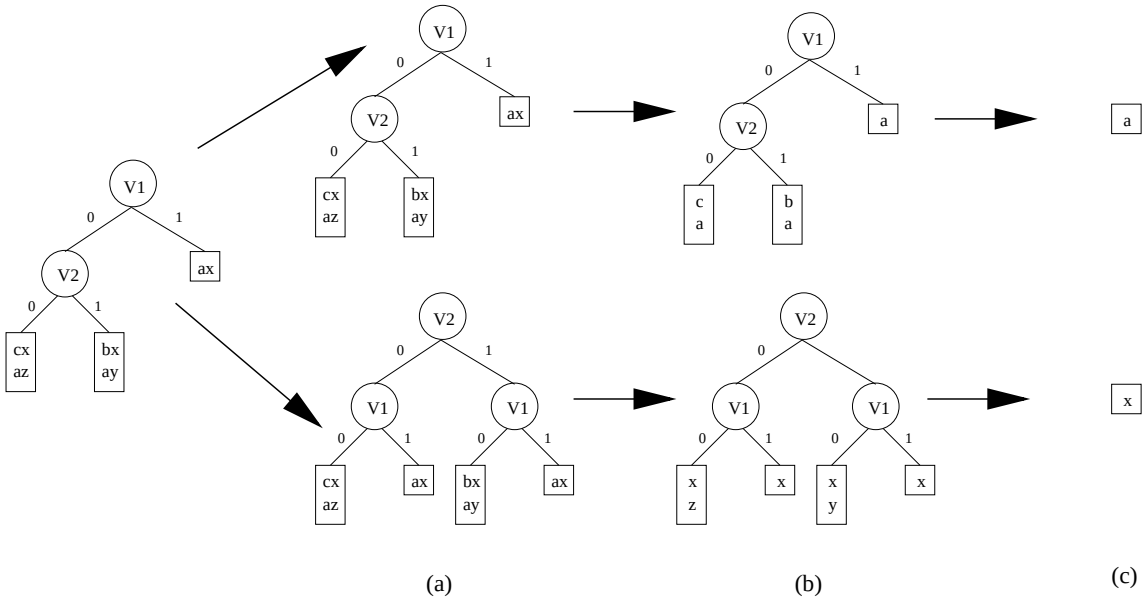

Figure 6: An example of mis-coordination that can occur if ties are not broken using a canonical action ordering. V1 is a state feature local to agent 1, and V2 is local to agent 2. (a) An individual policy is generated for each agent, with its local variable at the root. (b) The policies are converted into policies over individual actions, without an intermediate tie-breaking step. (c) The resulting simplified policies indicate that agent 1 should always perform action a and agent 2 should always perform x, making the joint action ax. This is an uncoordinated action whenever V1 = 0.

EAST, WEST, and STOP, with movement actions succeeding with 0.9 probability. The agents receive a reward of +20 for signaling together, and receive penalties for either mis-coordinating their SIGNAL actions or signaling from the wrong location. The team incurs a cost of -1 for each timestep that it takes them to reach the goal and SIGNAL. Each agent observes its own location ($\mathcal{X}^1$ and $\mathcal{Y}^1$ for agent 1, and $\mathcal{X}^2$ and $\mathcal{Y}^2$ for agent 2) but does not know the position of its teammate. This is a domain that admits a high degree of factorability and which allows for considerable context-specific independence among the two agents. While the agents are moving toward the goal location, they can act independently. They only need to coordinate once they have reached the goal.

We applied our algorithm to grid worlds of various sizes. Figure 8 shows a factored individual policy for agent 1 operating in the 3-by-3 grid world shown in Figure 7, with the goal at location (2,3). A visual inspection of the policy shows that, as expected, for most of the state space, agents can act independently. We verified this by measuring the amount of communication needed in an average run of the problem.

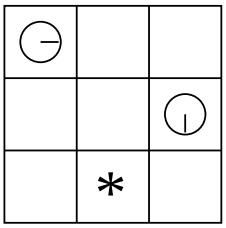

Figure 7: Agents in a 3-by-3 grid world. The goal is at (2,3).

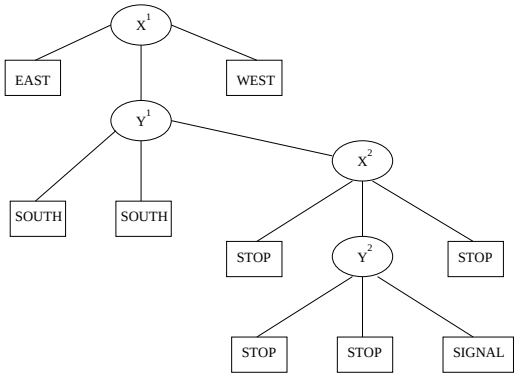

Figure 8: A factored policy for Agent 1 in a 3-by-3 Meeting-Under-Uncertainty problem.

We compared our execution method to a team operating with free communication and to DEC-COMM, an execution-time communication algorithm that chooses joint actions by requiring agents to maintain estimates of the possible joint beliefs of the team [13]. We ran 1000 trials of each method, starting the agents in random locations. The complete results for a 3-by-3 grid world are shown in Table 4.

Our factored algorithm achieves the same average reward as a team communicating at every timestep, but requires significantly fewer instances of communication. This confirms our intuition that the Meeting-Under-Uncertainty domain admits a great deal of context-specific independence between agents. The DEC-COMM algorithm, an approach specifically designed to minimize communication usage, is able to communicate much less frequently than a team executing a factored policy, but does so by sacrificing some amount of expected reward.

Table 4: Complete results for the 3-by-3 Meeting-Under-Uncertainty domain.

	Mean Reward (σ^2)	Mean Messages Sent (σ^2)	Mean Variables Sent (σ^2)
FREE COMMUNICATION	17.484 (1.069)	7.032 (1.069)	14.064 (1.069)
FACTORED EXECUTION	17.484 (1.069)	3.323 (1.050)	6.646 (4.199)
DEC-COMM	16.378 (1.234)	1.153 (1.153)	2.306 (0.498)

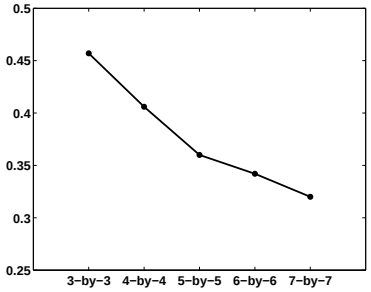

Figure 9: Communication usage as a function of problem size.

Figure 9 shows the amount of communication used by a team executing a factored policy, as a percentage of the amount of communication used by a team with full communication for grid worlds of increasing size, from 3-by-3 (82 states) up to 7-by-7 (2402 states). As the problem size increases, in this domain, the number of states in which agents can act without coordinating increases, leading to even greater communication savings over a team that must communicate at every timestep.

One major challenge posed by multi-agent problems is the exponential growth in the number of states and joint actions as the number of agents increases. However, as the agents may be independent of their teammates for large portions of the state in some multi-agent domains, the single-agent factored policies needed to control the agents may be very compact. Table 5 shows that the number of leaves in each agent's factored individual policy grows linearly as the number of agents increases from two to five in a 3-by-3 Meeting-Under-Uncertainty domain.

Table 5: Policy-tree size grows linearly with the number of agents in a 3-by-3 Meeting-Under-Uncertainty domain, even as the number of states and joint actions grow exponentially.

	States	Joint Actions	Policy Tree Leaves
2 AGENTS	82	36	9
3 AGENTS	730	216	13
4 AGENTS	6562	1296	17
5 AGENTS	59050	7776	21

6. CONCLUSION

In this work, we show how factored policy representations can be used for coordinated decentralized execution of a multi-agent team, providing an efficient answer to the questions of when and what a team should communicate and allowing agents to act independently of their teammates until they reach a state in which potential mis-coordination could take place. We provide a post-processing technique for transforming a centralized factored policy into individual factored policies for each member of a team, and show how this technique discovers context-specific independences between agents. Our preliminary experimental results are promising, showing that our work enables agents to act independently and without communication under favorable state conditions, and discovers automatically those conditions in which agents must coordinate with their teammates.

Although the Structured Policy Iteration algorithm that we discuss in this paper generates optimal policies for MDP domains, it is important to note that our approach does not require optimal joint policies as input, nor does it depend on the use of one particular MDP solution technique. The algorithms that we present in this paper can be applied to transform a joint policy of any quality into single-agent policies with equal expected reward. All that is required is that the joint policies be transformable into decision trees over state variables. For example, the SPUDD algorithm efficiently solves Markov decision processes in which relationships among state variables are represented as algebraic decision diagrams (ADDs) [10], which can easily be transformed into trees. Thus, our work can build on any advances that arise in the area of generating tree-structured policies for MDPs to solve increasingly large problems.

There are several directions in which our work can be extended. Currently, our execution algorithm does not take into account the cost of communication. An analysis of the value of information could be integrated into the reasoning to allow agents to decide if the cost of a potential mis-coordination is high enough to justify communication. Additionally, our work focuses on *collectively observable* multi-agent domains modeled by Dec-MDPs. It is possible that work on factored POMDPs (e.g. [9] could be applied in a similar fashion to the decentralized execution of Dec-POMDPs, modeling teams with *collective partial observability*. There is also significant work to be done in characterizing multi-agent domains that exhibit a high degree of context-specific independence between teammates, as those are the domains that would benefit most from the application of our techniques.

7. ACKNOWLEDGMENTS

This research was sponsored in part by the Boeing Corporation, NASA grant NNA04CK90A, and by BBNT Solutions, LLC under contract no. FA8760-04-C-0002. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution or any other entity.

8. REFERENCES

[1] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of centralized control of Markov decision processes. *Mathematics of Operations Research*, 2002.

[2] C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 2000.

[3] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *Uncertainty in Artificial Intelligence*, 1996.

[4] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence Journal*, 1989.

[5] C. V. Goldman and S. Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *International Joint Conferences on Autonomous Agents and Multi-agent Systems*, 2003.

[6] C. V. Goldman and S. Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of AI Research*, 2004.

[7] C. Guestrin and G. Gordon. Distributed planning in hierarchical factored MDPs. In *Uncertainty in Artificial Intelligence*, 2002.

[8] C. Guestrin, S. Venkataraman, and D. Koller. Context specific multiagent coordination and planning with factored MDPs. In *AAAI Spring Symposium*, 2002.

[9] E. Hansen and Z. Feng. Dynamic programming for POMDPs using a factored state representation. In *International Conference on AI Planning Systems*, 2000.

[10] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Uncertainty in Artificial Intelligence*, 1999.

[11] R. Nair, M. Roth, M. Yokoo, and M. Tambe. Communication for improving policy computation in distributed POMDPs. In *International Joint Conferences on Autonomous Agents and Multi-agent Systems*, 2004.

[12] D. V. Pynadath and M. Tambe. The communicative Multiagent Team Decision Problem: Analyzing teamwork theories and models. *Journal of AI Research*, 2002.

[13] M. Roth, R. Simmons, and M. Veloso. Reasoning about joint beliefs for execution-time communication decisions. In *International Joint Conferences on Autonomous Agents and Multi-agent Systems*, 2005.