Context-Specific Multiagent Coordination and Planning with Factored MDPs

Carlos Guestrin
Computer Science Dept.
Stanford University
guestrin@cs.stanford.edu

Shobha Venkataraman
Computer Science Dept.
Stanford University
shobha@cs.stanford.edu

Daphne Koller
Computer Science Dept.
Stanford University
koller@cs.stanford.edu

Abstract

We present an algorithm for coordinated decision making in cooperative multiagent settings, where the agents' value function can be represented as a sum of context-specific *value rules*. The task of finding an optimal joint action in this setting leads to an algorithm where the coordination structure between agents depends on the current state of the system and even on the actual numerical values assigned to the value rules. We apply this framework to the task of multiagent planning in dynamic systems, showing how a joint value function of the associated Markov Decision Process can be approximated as a set of value rules using an efficient linear programming algorithm. The agents then apply the coordination graph algorithm at each iteration of the process to decide on the highest-value joint action, potentially leading to a different coordination pattern at each step of the plan.

1 Introduction

Consider a system where multiple agents must coordinate in order to achieve a common goal, maximizing their joint utility. Naively, we can consider all possible joint actions, and choose the one that gives the highest value. Unfortunately, this approach is infeasible in all but the simplest settings, as the number of joint actions grows exponentially with the number of agents. Furthermore, we want to avoid a centralized decision making process, letting the agents communicate with each other so as to reach a jointly optimal decision.

This problem was recently addressed by Guestrin, Koller, and Parr (2001a) (GKP hereafter). They propose an approach based on an approximation of the joint value function as a linear combination of local value functions, each of which relates only to the parts of the system controlled by a small number of agents. They show how factored value functions allow the agents to find a globally optimal joint action using a message passing scheme. However, their approach suffers from a significant limitation: They assume that each agent only needs to interact with a small number of other agents. In many situations, an agent can *potentially* interact with many other agents, but not at the *same* time. For example, two agents that are both part of a construction crew might need to coordinate at times when they could both be working on the same task, but not at other times. If we use the approach of GKP, we are forced to represent value functions over large numbers of agents, rendering the approach intractable.

Our approach is based on the use of *context specificity* — a common property of real-world decision making tasks (Boutilier, Dean, & Hanks 1999). Specifically, we assume that the agents' value function can be decomposed into a set of *value rules*, each describing a context — an assignment to state variables and actions — and a value increment which gets added to the agents' total value in situations when that context applies. For example, a value rule might assert that in states where two agents are at the same house and both try to install the plumbing, they get in each other's way and the total value is decremented by 100. This representation is reminiscent of the tree-structured value functions of Boutilier and Dearden (Boutilier & Dearden 1996), but is substantially more general, as the rules are not necessarily mutually exclusive, but can be added together to form more complex functions.

Based on this representation, we provide a significant extension to the GKP notion of a *coordination graph*. We describe a distributed decision-making algorithm that uses message passing over this graph to reach a jointly optimal action. The coordination used in the algorithm can vary significantly from one situation to another. For example, if two agents are not in the same house, they will not need to coordinate. The coordination structure can also vary based on the utilities in the model; e.g., if it is dominant for one agent to work on the plumbing (e.g., because he is an expert), the other agents will not need to coordinate with him.

We then extend this framework to the problem of sequential decision making. We view the problem as a *Markov decision process (MDP)*, where the actions are the joint actions for all of the agents, and the reward is the total reward. Once again, we use context specificity, assuming that the rewards and the transition dynamics are rule-structured. We extend the linear programming approach of GKP to construct an approximate rule-based value function for this MDP. The agents can then use the coordination graph to decide on a joint action at each time step. Interestingly, although the value function is computed once in an offline setting, the online choice of action using the coordination graph gives rise to a highly variable coordination structure.

2 Context-specific coordination

We begin by considering the simpler problem of having a group of agents select a globally optimal joint action in order to maximize their joint value. Suppose we have a collection of agents $\mathbf{A} = \{A_1, \ldots, A_g\}$, where each agent A_j must choose an action a_j from a finite set of possible actions $\mathrm{Dom}(A_j)$. The agents are acting in a space described by a set of discrete state variables, $\mathbf{X} = \{X_1 \ldots X_n\}$, where each X_j takes on values in some finite domain $\mathrm{Dom}(X_j)$. The agents must choose the joint action $\mathbf{a} \in \mathrm{Dom}(\mathbf{A})$ that maximizes the total utility.

As discussed in GKP, the overall utility, or value function is often decomposed as a sum of "local" value functions, associated with the "jurisdiction" of the different agents. For example, if multiple agents are constructing a house, we can decompose the value function as a sum of the values of the tasks accomplished by each agent.

Definition 2.1 *We say that a function f is* restricted *to a* scope $\mathrm{Scope}[f] = \mathbf{C} \subseteq \mathbf{X} \cup \mathbf{A}$ *if $f : \mathbf{C} \mapsto I\!R$.* ∎

Thus, we can specify the value function as a sum of agent-specific value functions Q_j, each with a restricted scope. Each Q_j is typically represented as a table, listing agent j's local values for different combinations of variables in the scope. However, this representation is often highly redundant, forcing us to represent many irrelevant interactions. For example, an agent A_1's value function might depend on the action of agent A_2 if both are trying to install the plumbing in the same house. However, there is no interaction if A_2 is currently working in another house, and there is no point in making A_1's entire value function depend on A_2's action. We represent such context specific value dependencies using *value rules*:

Definition 2.2 *Let $\mathbf{C} \subseteq \mathbf{X} \cup \mathbf{A}$ and $\mathbf{c} \in \mathrm{Dom}(\mathbf{C})$. We say that $\mathbf{c}$ is* consistent *with $\mathbf{b} \in \mathrm{Dom}(\mathbf{B})$ if $\mathbf{c}$ and $\mathbf{b}$ assign the same value to $\mathbf{C} \cap \mathbf{B}$. A* value rule *$\langle \rho; \mathbf{c} : v \rangle$ is a function $\rho : \mathrm{Dom}(\mathbf{X}, \mathbf{A}) \mapsto I\!R$ such that $\rho(\mathbf{x}, \mathbf{a}) = v$ when $(\mathbf{x}, \mathbf{a})$ is consistent with $\mathbf{c}$ and 0 otherwise.* ∎

In our construction example, we might have a rule:

$$\langle \rho; A_1, A_2 \text{ in-same-house} = \textit{true} \; \wedge \\ A_1 = \textit{plumbing} \wedge A_2 = \textit{plumbing} : -100 \rangle .$$

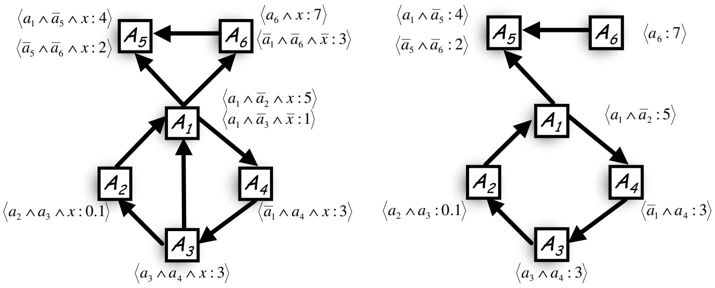
This definition of rules adapts the definition of rules for exploiting context specific independence in inference for Bayesian networks by Zhang and Poole (1999). Note that a value rule $\langle \rho; \mathbf{c} : v \rangle$ has a scope $\mathbf{C}$.

Definition 2.3 *A* rule-based function *$f : \{\mathbf{X}, \mathbf{A}\} \mapsto I\!R$ is composed of a set of rules $\{\rho_1, \ldots, \rho_n\}$ such that $f(\mathbf{x}, \mathbf{a}) = \sum_{i=1}^{n} \rho_i(\mathbf{x}, \mathbf{a})$.* ∎

This notion of a rule-based function is related to the tree-structure functions used by Boutilier and Dearden (1996) and by Boutilier et al. (1999), but is substantially more general. In the tree-structure value functions, the rules corresponding to the different leaves are mutually exclusive and exhaustive. Thus, the total number of different values represented in the tree is equal to the number of leaves (or rules). In the rule-based function representation, the rules are not mutually exclusive, and their values are added to form the overall function value for different settings of the variables. Different rules are added in different settings, and, in fact, with k rules, one can easily generate 2^k different possible values. Thus, the rule-based functions can provide a compact representation to a much richer class of value functions.

We represent the local value function Q_j associated with agent j as a rule-based function:

$$Q_j = \sum_i \rho_i^j \; .$$



(a) (b)

Figure 1: (a) Coordination graph for a 6-agent problem, the rules in Q_j are indicated in the figure by the rules next to A_j. (b) Graph becomes simpler after conditioning on the state $X = x$.

Note that if each rule ρ_i^j has scope $\mathbf{C}_i^j$, then Q_j will be a restricted scope function of $\cup_i \mathbf{C}_i^j$. The scope of Q_j can be further divided into two parts: The state variables

$$Obs[Q_j] = \{X_i \in \mathbf{X} \mid X_i \in \mathrm{Scope}[Q_j]\}$$

are the observations agent j needs to make. The agent decision variables

$$Agents[Q_j] = \{A_i \in \mathbf{A} \mid A_i \in \mathrm{Scope}[Q_j]\}.$$

are the agents with whom j interacts directly.

3 Cooperative action selection

Recall that the agents' task is to select a joint action $\mathbf{a}$ that maximizes $Q = \sum_j Q_j(\mathbf{x}, \mathbf{a})$. The fact that the Q_j's depend on the actions of multiple agents forces the agents to coordinate their action choices. As we now show, this process can be performed using a very natural data structure called a *coordination graph*.

Intuitively, a coordination graph connects agents whose local value functions interact with each other. This definition is the directed extension of the definition proposed in GKP, and is the collaborative counterpart of the relevance graph proposed for competitive settings by Koller and Milch (2001).

Definition 3.1 *A* coordination graph *for a set of agents with local utilities $Q = \{Q_1, \ldots, Q_g\}$ is a directed graph whose nodes are $\{A_1, \ldots, A_g\}$, and which contains an edge $A_i \rightarrow A_j$ if and only if $A_i \in Agents[Q_j]$.* ∎

An example of a coordination graph with 6 agents and one state variable is shown in Fig. 1(a). See, for example, that agent A_3 has the parent A_4, because A_4's action affects Q_3.

Recall that our task is to find a coordination strategy for the agents to maximize $\sum_j Q_j$ at each state $\mathbf{x}$. First, note that the scope of the Q_j functions that comprise the value can include both action choices and state variables. We assume that each agent j has full observability of the relevant state variables $Obs[Q_j]$. Given a particular state $\mathbf{x} = \{x_1, \ldots, x_n\}$, agent j *conditions* on the current state by discarding all rules in Q_j not consistent with the current state $\mathbf{x}$. Note that agent j only needs to observe $Obs[Q_j]$, and not the entire state of the system, substantially reducing the sensing requirements.

Interestingly, after the agents observe the current state the coordination graph may become simpler, in our example the edges $A_3 \rightarrow A_1$ and $A_1 \rightarrow A_6$ disappear after agents observe that $X = x$, as shown in Fig. 1(b). Thus, agents A_1 and A_6 will only need to coordinate directly in the context of $X = \bar{x}$.

After conditioning on the current state, each Q_j will only depend on the agents' action choices $\mathbf{A}$. Now, our task is to select a joint action $\mathbf{a}$ that maximizes $\sum_j Q_j(\mathbf{a})$. Maximization in a graph structure suggests the use of *non-serial dynamic programming* (Bertele & Brioschi 1972), or variable elimination. To exploit structure in rules, we use a variable elimination algorithm similar to variable elimination in a Bayesian network with context specific independence (Zhang & Poole 1999).

Intuitively, the algorithm operates by having an individual agent "collect" value rules relevant to them from their children. The agent can then decide on its own strategy, taking all of the implications into consideration. The choice of optimal action and the ensuing payoff will, of course, depend on the actions of agents whose strategies have not yet been decided. The agent therefore simply communicates the value ramifications of its strategy to other agents, so that they can make informed decisions on their own strategies.

Thus, our algorithm "eliminates" agents one by one, where the elimination process performs a maximization step over the agent's action choice. We use $MaxOut\,(f, A_i)$ to denote a procedure that takes a rule function f and a variable A_i and returns a rule function g such that: $g(A) = \max_{a_i} f(A)$. Such a procedure is a fairly straightforward extension of the variable elimination algorithm of (Zhang & Poole 1999). We omit details for lack of space. The algorithm proceeds by repeatedly selecting some undecided agent, until all agents have decided on a strategy. For a selected agent A_l:

1. A_l receives messages from its children, with all the rules $\langle \rho; \mathbf{c} : v \rangle$ such that $A_l \in \mathbf{C}$. These rules are added to Q_l. After this step, A_l has no children in the coordination graph and can be optimized independently.
2. A_l performs the local maximization step $g_l = MaxOut\,(Q_l, A_l)$; This local maximization corresponds to a conditional strategy decision.
3. A_l distributes the rules in g_l to its parents. At this point, A_l's strategy is fixed, and it has been "eliminated".

Once this procedure is completed, a second pass in the reverse order can be performed to compute the optimal action choice for all of the agents. Note that the initial distribution of rules among agents and the procedure for distributing messages among the parent agents in step 3 do not alter the final action choice and have a limited significance in the final communication required for solving the coordination problem.

The cost of this algorithm is polynomial in the number of new rules generated in the maximization operation $MaxOut\,(Q_l, A_l)$. The number of rules is never larger and in many cases exponentially smaller than the complexity bounds on the table-based coordination graph in GKP, which, in turn, was exponential only in the *induced width* of the graph (Dechter 1999). However, the computational costs involved in managing sets of rules usually imply that the computational advantage of the rule-based approach will only manifest in problems that possess a fair amount of context specific structure.

More importantly, the rule based coordination structure exhibits several important properties. First, as we discussed, the structure often changes when conditioning on the current state, as in Fig. 1. Thus, in different states of the world, the agents may have to coordinate their actions differently. In our example, if the situation is such that the plumbing is ready to be installed, two qualified agents that are at the same house will need to coordinate. However, they may not need to coordinate in other situations.

More surprisingly, interactions that seem to hold between agents even after the state-based simplification can disappear as agents make strategy decisions. For example, if $Q_1 = \{\langle a_1 \wedge a_2 : 5\rangle, \langle \overline{a_1} \wedge a_2 \wedge \overline{a_3} : 1\rangle\}$, then A_1's optimal strategy is to do a_1 regardless, at which point the added value is 5 regardless of A_3's decision. In other words, $MaxOut\,(Q_1, A_1) = \{\langle a_2 : 5\rangle\}$. In this example, there is an *a priori* dependence between A_2 and A_3. However, after maximizing A_1, the dependence disappears and agents A_2 and A_3 may not need to communicate. In the construction crew example, suppose electrical wiring and plumbing can be performed simultaneously. If there is an agent that can do both tasks and another that is only a plumber, then a priori agents need to coordinate so that they are not both working on plumbing. However, when the first agent is optimizing his strategy, he decides that electrical wiring is a dominant strategy, because either the other agent will do the plumbing and both tasks are done or the other agent will perform a different task, in which case the first agent can get to plumbing in the next time step, achieving the same total value.

The context-sensitivity of the rules also reduces communication between agents. In particular, agents only need to communicate relevant rules to each other, reducing unnecessary interaction. For example, in Fig. 1(b), when agent A_1 decides on its strategy, agent A_5 only needs to pass the rules that involve A_1, i.e., only $\langle a_1 \wedge \overline{a_5} : 4\rangle$. The rule involving A_6 is not transmitted, avoiding the need for agent A_1 to consider agent A_6's decision in its strategy.

Finally, we note that the rule structure provides substantial flexibility in constructing the system. In particular, the structure of the coordination graph can easily be adapted incrementally as new value rules are added or eliminated. For example, if it turns out that two agents intensely dislike each other, we can easily introduce an additional value rule that associates a negative value with pairs of action choices that puts them in the same house at the same time.

4 One-step lookahead

Now assume that the agents are trying to maximize the sum of an immediate reward and a value that they expect to receive one step in the future. We describe the dynamics of such system τ using a *dynamic decision network (DDN)* (Dean & Kanazawa 1989). Let X_i denote the ith variable at the current time and X'_i the variable at the next step. The *transition graph* of a DDN is a two-layer directed acyclic graph G whose nodes are $\{A_1, \ldots, A_g, X_1, \ldots, X_n, X'_1, \ldots, X'_n\}$, and where only nodes in $\mathbf{X}'$ have parents. We denote the parents of X'_i in the graph by *Parents*(X'_i). For simplicity of exposition,

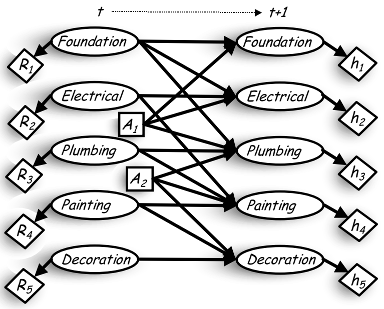


Figure 2: A DDN for a 2-agent crew and 1 house setting.

we assume that *Parents*$(X'_i) \subseteq \mathbf{X} \cup \mathbf{A}$, i.e., all of the parents of a node are in the previous time step. Each node X'_i is associated with a *conditional probability distribution (CPD)* $P(X'_i \mid$ *Parents*$(X'_i))$. The transition probability $P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a})$ is then defined to be $\prod_i P(x'_i \mid \mathbf{u}_i)$, where $\mathbf{u}_i$ is the value in $\mathbf{x}, \mathbf{a}$ of the variables in *Parents*(X'_i). The immediate rewards are a set of functions $r_1, \ldots, r_g$, and the next-step values are a set of functions $h_1, \ldots, h_g$.

Fig. 2 shows a DDN for a simple two-agent problem, where ovals represent the variables X_i (features of a house) and rectangles the agent actions (tasks). The arrows to the next time step variables represent dependencies, e.g., painting can only be done if both electrical wiring and plumbing are done and agent A_2 decides to paint. The diamond nodes in the first time step represent the immediate reward, while the h nodes in the second time step represent the future value associated with a subset of the state variables.

In most representations of Bayesian networks and DDNs, tables are used to represent the utility nodes r_i and h_i and the transition probabilities $P(X'_i \mid$ *Parents*$(X'_i))$. However, as discussed by Boutilier *et al.* (1999), decision problems often exhibit a substantial amount of context specificity, both in the value functions and in the transition dynamics. We have already described a rule-based representation of the value function components. We now describe a rule representation (as in (Zhang & Poole 1999)) for the transition model.

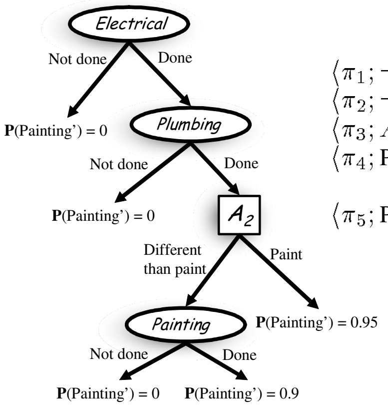
Definition 4.1 *A* probability rule $\langle \pi; \mathbf{c} : p \rangle$ *is a function* $\pi : \{\mathbf{X}, \mathbf{X}', \mathbf{A}\} \mapsto [0,1]$*, where the* context $\mathbf{c} \in \mathrm{Dom}(\mathbf{C})$ *for* $\mathbf{C} \subseteq \{\mathbf{X}, \mathbf{X}', \mathbf{A}\}$ *and* $p \in [0,1]$*, such that* $\pi(\mathbf{x}, \mathbf{x}', \mathbf{a}) = p$ *if* $\{\mathbf{x}, \mathbf{x}', \mathbf{a}\}$ *is consistent with* $\mathbf{c}$ *and is* 1 *otherwise. A* rule-based conditional probability distribution *(rule CPD)* P *is a function* $P : \{\mathbf{X}'_i, \mathbf{X}, \mathbf{A}\} \mapsto [0,1]$*, composed of a set of probability rules* $\{\pi_1, \pi_2, \ldots\}$*, such that:*

$$P(\mathbf{x}'_i \mid \mathbf{x}, \mathbf{a}) = \prod_{i=1}^{n} \pi_i(\mathbf{x}'_i, \mathbf{x}, \mathbf{a});$$

and where every assignment $(\mathbf{x}'_i, \mathbf{x}, \mathbf{a})$ *is consistent with the context of only one rule.*

We can now define the conditional probabilities $P(X'_i \mid$ *Parents*$(X'_i))$ as a rule CPD, where the context variables $\mathbf{C}$ of the rules depend on variables in $\{X'_i \cup$ *Parents*$(X'_i)\}$. An example of a CPD represented by a set of probability rules is shown in Fig. 3.

In the one-step lookahead case, for any setting $\mathbf{x}$ of the



$\langle \pi_1; \neg\text{Electrical} : 0 \rangle$
$\langle \pi_2; \neg\ \text{Plumbing} : 0 \rangle$
$\langle \pi_3; A_2 = \neg\ \text{paint} \wedge \neg\ \text{Painting} : 0 \rangle$
$\langle \pi_4; \text{Plumbing} \wedge \text{Electrical}$
$\wedge\ A_2 = \text{paint} : 0.95 \rangle$
$\langle \pi_5; \text{Plumbing} \wedge \text{Electrical} \wedge \text{Painting}$
$\wedge\ A_2 = \neg\ \text{paint} : 0.9 \rangle$

(a) **(b)**

Figure 3: (a) Example CPD, represented as a decision tree, for the probability that the painting will be done at time $t+1$. (b) Equivalent set of probability rules.

state variables, the agents aim to maximize:

$$\begin{aligned} Q(\mathbf{x}, \mathbf{a}) &= \sum_{j=1}^{g} Q_j(\mathbf{x}, \mathbf{a}) \\ Q_j(\mathbf{x}, \mathbf{a}) &= r_j(\mathbf{x}, \mathbf{a}) + \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) h_j(\mathbf{x}'). \end{aligned}$$

In the previous section, we showed that if each Q_j is a rule-based function, it can be optimized effectively using the coordination graph. We now show that, when system dynamics, rewards and values are rule-based, the Q_j's are also rule based, and can be computed effectively. Our approach extends the factored backprojection of Koller and Parr (1999).

Each h_j is a rule function, which can be written as $h_j(\mathbf{x}') = \sum_i \rho_i^{(h_j)}(\mathbf{x}')$, where $\rho_i^{(h_j)}$ has the form $\left\langle \rho_i^{(h_j)}; \mathbf{c}_i^{(h_j)} : v_i^{(h_j)} \right\rangle$. Each rule is a restricted scope function; thus, we can simplify:

$$\begin{aligned} g_j(\mathbf{x}, \mathbf{a}) &= \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) h_j(\mathbf{x}') \\ &= \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) \sum_i \rho_i^{(h_j)}(\mathbf{x}'); \\ &= \sum_i \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) \rho_i^{(h_j)}(\mathbf{x}'); \\ &= \sum_i v_i^{(h_j)} P(\mathbf{c}_i^{(h_j)} \mid \mathbf{x}, \mathbf{a}); \end{aligned}$$

where the term $v_i^{(h_j)} P(\mathbf{c}_i^{(h_j)} \mid \mathbf{x}, \mathbf{a})$ can be written as a rule function. We denote this backprojection operation by $RuleBackproj(\rho_i^{(h_j)})$; its implementation is straightforward, and we omit details for lack of space.

As an example, consider the backprojection of a simple rule, $\langle \rho;$ Painting done at $t+1 : 10 \rangle$, through the CPD in Fig. 3:

$$\begin{aligned} RuleBackproj(\rho) &= \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a})\rho(\mathbf{x}'); \\ &= \sum_{\text{Painting}'} P(\text{Painting}' \mid \mathbf{x}, \mathbf{a})\rho(\text{Painting}'); \\ &= 10\ \pi_1\pi_2\pi_3\pi_4\pi_5\ . \end{aligned}$$

Note that the product $\pi_1\pi_2\pi_3\pi_4\pi_5$ will be equal to 0 in most contexts, e.g., when electricity is not done at time t. This product will only be non-zero in two contexts, in the context associated with rule π_4 and in the one for π_5. Thus, we can express the backprojection operation as:

$$\begin{aligned} &RuleBackproj(\rho) = \\ &\quad \langle \text{Plumbing} \wedge \text{Electrical} \wedge\ A_2 = \text{ paint} : 9.5\rangle + \\ &\quad \langle \text{Plumbing} \wedge \text{Electrical} \wedge \text{Painting} \wedge\ A_2 = \neg\,\text{paint} : 9\rangle; \end{aligned}$$

which is a rule-based function composed of two rules.

Thus, we can now write the *backprojection* of the next step utility h_j as:

$$g_j(\mathbf{x}, \mathbf{a}) = \sum_i RuleBackproj(\rho_i^{(h_j)}); \tag{1}$$

where g_j is a sum of rule-based functions, and therefore also a rule-based function. Using this notation, we can write $Q_j(\mathbf{x}, \mathbf{a}) = r_j(\mathbf{x}, \mathbf{a}) + g_j(\mathbf{x}, \mathbf{a})$, which is again a rule-based function. This function is exactly the case we addressed in Section 3. Therefore, we can perform efficient one-step lookahead planning using the same coordination graph.

5 Multiagent sequential decision making

We now turn our attention to the substantially more complex case where the agents are acting in a dynamic environment and are trying to jointly maximize their expected long-term return. The *Markov Decision Process (MDP)* framework formalizes this problem.

An MDP is defined as a 4-tuple $(\mathbf{X}, \mathcal{A}, \mathcal{R}, P)$ where: $\mathbf{X}$ is a finite set of $N = |\mathbf{X}|$ states; $\mathcal{A}$ is a set of actions; $\mathcal{R}$ is a *reward function* $\mathcal{R} : \mathbf{X} \times \mathcal{A} \mapsto \mathrm{I\!R}$, such that $\mathcal{R}(\mathbf{x}, a)$ represents the reward obtained in state $\mathbf{x}$ after taking action a; and P is a *Markovian transition model* where $P(\mathbf{x}' \mid \mathbf{x}, a)$ represents the probability of going from state $\mathbf{x}$ to state $\mathbf{x}'$ with action a. We assume that the MDP has an infinite horizon and that future rewards are discounted exponentially with a discount factor $\gamma \in [0, 1)$. Given a value function $\mathcal{V}$, we define $Q_{\mathcal{V}}(\mathbf{x}, a) = R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a)\mathcal{V}(\mathbf{x}')$, and the *Bellman operator* $\mathcal{T}^*$ to be $\mathcal{T}^*\mathcal{V}(\mathbf{x}) = \max_a Q_{\mathcal{V}}(\mathbf{x}, a)$. The optimal value function $\mathcal{V}^*$ is the fixed point of $\mathcal{T}^*$: $\mathcal{V}^* = \mathcal{T}^*\mathcal{V}^*$. For any value function $\mathcal{V}$, we can define the policy obtained by acting greedily relative to $\mathcal{V}$: *Greedy*$(\mathcal{V})(\mathbf{x}) = \arg\max_a Q_{\mathcal{V}}(\mathbf{x}, a)$. The greedy policy relative to the optimal value function $\mathcal{V}^*$ is the optimal policy $\pi^* =$ *Greedy*$(\mathcal{V}^*)$.

There are several algorithms for computing the optimal policy. One is via linear programming. Our variables are $V_1, \ldots, V_N$, where V_i represents $\mathcal{V}(\mathbf{x}^{(i)})$ with $\mathbf{x}^{(i)}$ referring to the ith state. One simple variant of the LP is:

$$\begin{aligned} &\text{Minimize:} && 1/N \textstyle\sum_i V_i\ ; \\ &\text{Subject to:} && V_i \geq R(\mathbf{x}^{(i)}, a) + \gamma \textstyle\sum_j P(\mathbf{x}^{(j)} \mid \mathbf{x}^{(i)}, a) V_j \\ &&& \forall i \in \{1, \ldots, N\}, a \in \mathcal{A}. \end{aligned}$$

In our setting, the state space is exponentially large, with one state for each assignment $\mathbf{x}$ to $\mathbf{X}$. We use the common approach of restricting attention to value functions that are compactly represented as a linear combination of *basis functions* $H = \{h_1, \ldots, h_k\}$. A *linear value function* over H is a function $\mathcal{V}$ that can be written as $\mathcal{V}(\mathbf{x}) = \sum_{j=1}^k w_j h_j(\mathbf{x})$ for some coefficients $\mathbf{w} = (w_1, \ldots, w_k)'$. The linear programming approach can be adapted to use this value function representation (Schweitzer & Seidmann 1985) by changing the objective function to $\sum_i w_i h_i$, and modifying the constraints accordingly. In this approximate formulation, the variables are $w_1, \ldots, w_k$, i.e., the weights for our basis functions. The LP is given by:

$$\begin{aligned} &\text{Variables:} && w_1, \ldots, w_k\ ; \\ &\text{Minimize:} && \textstyle\sum_{\mathbf{x}} 1/N \sum_i w_i\ h_i(\mathbf{x})\ ; \\ &\text{Subject to:} && \textstyle\sum_i w_i\ h_i(\mathbf{x}) \geq \\ &&& \quad R(\mathbf{x}, a) + \gamma \textstyle\sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) \sum_i w_i\ h_i(\mathbf{x}') \\ &&& \quad \forall \mathbf{x} \in \mathbf{X}, \forall a \in A. \end{aligned}$$

This transformation has the effect of reducing the number of free variables in the LP to k (one for each basis function coefficient), but the number of constraints remains $|\mathbf{X}| \times |\mathcal{A}|$. We address this issue by combining assumptions about the structure of the system dynamics with a particular form of approximation for the value function. First, we assume that the system dynamics of the MDP are represented using a DDN with probability rule CPDs, as described in Section 4. Second, we propose the use of value rules as basis functions, resulting in a rule-based value function. If we had a value function $\mathcal{V}$ represented in this way, then we could implement *Greedy*$(\mathcal{V})$ by having the agents use our message passing coordination algorithm of Section 4 at each step.

Our formulation is based on the approach of GKP, who show how to exploit the factorization of the basis functions and system dynamics in order to replace the constraints in the approximate LP by an equivalent but exponentially smaller set of constraints. First, note that the constraints can be replaced by a single, nonlinear constraint:

$$0 \geq \max_{\mathbf{x}, \mathbf{a}} \left[R(\mathbf{x}, \mathbf{a}) + \sum_i (\gamma g_i(\mathbf{x}) - h_i(\mathbf{x})) w_i \right];$$

where $g_i = RuleBackproj(h_i) = \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) h_i(\mathbf{x}')$, which can be computed as described in Section 4. Although a naive approach to maximizing over the state space would require the enumeration of every state, as we have shown in Section 3, the structure in rule functions allow us to perform such maximization very efficiently. The same intuition allows us to decompose this nonlinear constraint into a set of linear constraints, whose structure is based on the intermediate results of the variable elimination process. The algorithm is directly analogous to that of GKP, except that it is based on the use of rule-based variable elimination rather than standard variable elimination. We refer the reader to (Guestrin, Koller, & Parr 2001a) for the details.

The approximate LP computes a rule-based value function, which approximates the long-term optimal value function for the MDP. These value functions can be used as the one-step lookahead value in Section 4. In our rule-based

Prob.	♯houses	Agent skills	♯states	♯actions	Time (m)
1	1	$A_1 \in \{\text{Found, Elec, Plumb}\}$; $A_2 \in \{\text{Plumb, Paint, Decor}\}$	2048	36	1.6
2	2	$A_1 \in \{\text{Paint, Decor}\}$, moves $A_2 \in \{\text{Found, Elec, Plumb, Paint}\}$, at House 1 $A_3 \in \{\text{Found, Elec}\}$ and $A_4 \in \{\text{Plumb, Decor}\}$, at House 2	33,554,432	1024	33.7
3	3	$A_1 \in \{\text{Paint, Decor}\}$, moves $A_2 \in \{\text{Found, Elec, Plumb}\}$, at House 1 $A_3 \in \{\text{Found, Elec, Plumb, Paint}\}$, at House 2 $A_4 \in \{\text{Found, Elec, Plumb, Decor}\}$, at House 3	34,359,738,368	6144	63.9
4	2	$A_1 \in \{\text{Found}\}$, moves; $A_2 \in \{\text{Decor}\}$, moves $A_3 \in \{\text{Found, Elec, Plumb, Paint}\}$, at House 1 $A_4 \in \{\text{Elec, Plumb, Paint}\}$, at House 2	8,388,608	768	5.7

Figure 4: Summary of results on the building crew problem.

models, the overall one-step value function is also rule-based, allowing the agents to use the coordination graph in order to select an optimal joint action (optimal relative to the approximation for the long-term value function). It is important to note that, although the same value function is used at all steps in the MDP, the actual coordination structure varies substantially between steps.

Finally, we observe that the structure of the computed value rules determines the nature of the coordination. In some cases, we may be willing to introduce another approximation into our value function, in order to reduce the complexity of the coordination process. In particular, if we have a value rule $\langle \rho; \mathbf{c} : v \rangle$ where v is relatively small, then we might be willing to simply drop it from the rule set. If $\mathbf{c}$ involves the actions of several agents, dropping ρ from our rule-based function might substantially reduce the amount of coordination required.

6 Experimental results

We implemented our rule-based factored approximate linear programming and the message passing coordination algorithms in C++, using CPLEX as the LP solver. We experimented with a construction crew problem, where each house has 5 features $\{\text{Foundation, Electric, Plumbing, Painting, Decoration}\}$. Each agent has a set of skills and some agents may move between houses. Each feature in the house requires two time steps to complete. Thus, in addition to the variables in Fig. 2, the DDN for this problem contains "action-in-progress" variables for each house feature, for each agent, e.g., "A_1-Plumbing-in-progress-House 1". Once an agent takes an action, the respective "action-in-progress" variable becomes true with high probability. If one of the "action-in-progress" variables for some house feature is true, that feature becomes true with high probability at the next time step. At every time step, with a small probability, a feature of the house may break, in which case there is a chain reaction and features that depend on the broken feature will break with probability 1. This effect makes the problem dynamic, incorporating both house construction and house maintenance in the same model. Agents receive 100 reward for each completed feature and -10 for each "action-in-progress". The discount factor is 0.95. The basis functions used were rules over the settings of the parents of the CPDs for the house feature variables in the DDN.

Fig. 4 summarizes the results for various settings. Note that, although the number of states may grow exponentially from one setting to the other, the running times grow polynomially. Furthermore, in Problem 2, the backprojections of the basis functions had scopes with up to 11 variables, too large for the table-based representation to be tractable.

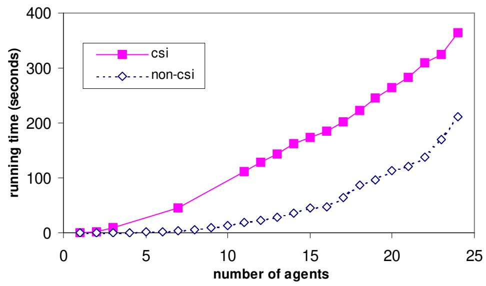
The policies generated in these problems seemed very intuitive. For example, in Problem 2, if we start with no features built, A_1 will go to House 2 and wait as its painting skills are going to be needed there before the decoration skill are needed in House 1. In Problem 1, we get very interesting coordination strategies: If the foundation is completed, A_1 will do the electrical fitting and A_2 will do the plumbing. Furthermore, A_1 makes its decision not by coordinating with A_2, but by noting that electrical fitting is a dominant strategy. On the other hand, if the system is at the state where both foundation and electrical fitting is done, then agents coordinate to avoid doing plumbing simultaneously. Another interesting feature of the policies occurs when agents are idle, e.g., in Problem 1, if foundation, electric and plumbing are done, then agent A_1 repeatedly performs the foundation task. This avoids a chain reaction starting from the foundation of the house. Checking the rewards, there is actually a higher expected loss from the chain reaction than the cost of repeatedly checking the foundation of the house.

For small problems with one house, we can compute the optimal policy exactly. In the table in Fig. 5, we present the optimal values for two such problems. Additionally, we can compute the actual value of acting according to the policy generated by our method. As the table shows, these values are very close, indicating that the policies generated by our method are very close to optimal in these problems.

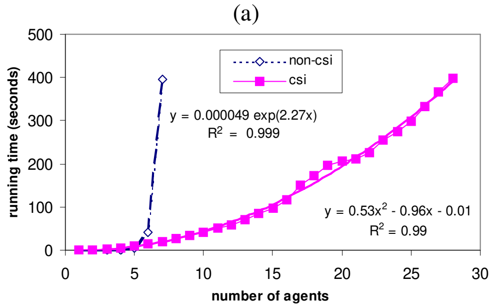
We also tested our rule-based algorithm on a variation of the multiagent SysAdmin problem of GKP. In this problem, there is a network of computers, each is associated with an administrator agent. Each machine runs processes and receives a reward if a process terminates. Processes take longer to terminate in faulty machines and dead machines can send bad packets to neighbors, causing them to become faultye. The rule-based aspect in this problem comes from a selector variable which chooses which neighboring machine to receive packets from. We tested our algorithm on a variety of network topologies and compared it to the table-based approach in GKP. For a bidirectional ring, for example, the total number of constraints generated grows *linearly* with the number of agents. Furthermore, the rule-based (CSI) approach generates considerably fewer constraints than the table-based approach (non-CSI). However, the constant overhead of managing rules causes the

Agent skills	*Actual* value of rule-based policy	Optimal value
$A_1 \in \{\text{Found, Elec}\}$; $A_2 \in \{\text{Plumb, Paint, Decor}\}$	6650	6653
$A_1 \in \{\text{Found, Elec, Plumb}\}$; $A_2 \in \{\text{Plumb, Paint, Decor}\}$	6653	6654

Figure 5: Comparing the actual expected value of acting according to the rule-based policy obtained by our algorithm starting from the state with no features built in the house to the value of the true optimal policy for problems with one house.



(a)



(b)

Figure 6: Running times: (a) Bidirectional ring; (b) Inverted star.

rule-based approach to be about two times slower than the table-based approach, as shown in Fig. 6(a).

However, note that in ring topologies the the induced width of the coordination graph is constant as the number of agents increases. For comparison, we tested on a reverse star topology, where every machine can affect the status of a central server machine, so that the number of parents of the server increases with the number of computers in the network. Here, we observe a very different behavior, as seen in Fig. 6(b). In the table-based approach, the tables grow exponentially with the number of agents, yielding an exponential running time. On the other hand, the size of the rule set only grows linearly, yielding a quadratic total running time.

Notice that in all topologies, the sizes of the state and action spaces are growing exponentially with the number of machines. Nonetheless, the total running time is only growing quadratically. This exponential gain has allowed us to run very large problems, with over 10^{124} states.

7 Conclusion

We have provided a principled and efficient approach to planning in multiagent domains where the required interactions vary from one situation to another. We have shown that our results scale to very complex problems, including problems where traditional table-based representations of the value function blow up exponentially. In problems where the optimal value could be computed analytically for comparison purposes, the value of the policies generated by our approach were within 0.05% of the optimal value. From a representation perspective, our approach combines the advantages of the factored linear value function representation of (Koller & Parr 1999; Guestrin, Koller, & Parr 2001b; 2001a) with those of the tree-based value functions of (Boutilier & Dearden 1996).

We showed that the task of finding an optimal joint action in our approach leads to a very natural communication pattern, where agents send messages along a *coordination* graph determined by the structure of the value rules. The coordination structure dynamically changes according to the state of the system, and even on the actual numerical values assigned to the value rules. Furthermore, the coordination graph can be adapted incrementally as the agents learn new rules or discard unimportant ones. We believe that this graph-based coordination mechanism will provide a well-founded schema for other multiagent collaboration and communication approaches.

Acknowledgments We are very grateful to Ronald Parr for many useful discussions. This work was supported by the ONR under the MURI program "Decision Making Under Uncertainty" and by the Sloan Foundation. C. Guestrin was also supported by a Siebel Scholarship.

References

Bertele, U., and Brioschi, F. 1972. *Nonserial Dynamic Programming*. New York: Academic Press.

Boutilier, C., and Dearden, R. 1996. Approximating value trees in structured dynamic programming. In *Proc. ICML*, 54–62.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11:1 – 94.

Dean, T., and Kanazawa, K. 1989. A model for reasoning about persistence and causation. *Computational Intelligence* 5(3).

Dechter, R. 1999. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence* 113(1–2):41–85.

Guestrin, C.; Koller, D.; and Parr, R. 2001a. Multiagent planning with factored MDPs. In *Proc. NIPS-14*.

Guestrin, C.; Koller, D.; and Parr, R. 2001b. Max-norm projections for factored MDPs. In *Proc. IJCAI*.

Koller, D., and Milch, B. 2001. Multi-agent influence diagrams for representing and solving games. In *Proc. IJCAI*.

Koller, D., and Parr, R. 1999. Computing factored value functions for policies in structured MDPs. In *Proc. IJCAI*.

Schweitzer, P., and Seidmann, A. 1985. Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications* 110:568 – 582.

Zhang, N., and Poole, D. 1999. On the role of context-specific independence in probabilistic reasoning. In *Proc. IJCAI*.