AAAI Spring Symposium, Stanford, California, March 2002.

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 a new, principled and efficient algorithm for decision making and planning cooperative multi-agent dynamic systems. We consider systems where the agents' value function is a sum of local *value rules*, that specify an increment to the value in certain *contexts*, which can depend both on the current state and on the actions of some subset of the agents. We show that the task of finding an optimal joint action relative to this type of value function leads to a very natural communication pattern, where agents send messages along a *coordination* graph determined by the structure of the value rules. We show that the coordination structure depends on the state of the system, and even on the actual numerical values assigned to the value rules. We then show how to apply this framework to the task of multi-agent planning in dynamic systems. We view the entire multi-agent system as a single, large Markov decision process (MDP). We assume that the agents' reward functions and the system dynamics are described in terms of factored rules. We show how to use an efficient linear programming algorithm to derive a rule-based value function which is an approximation to the optimal joint value function. Given this value function, the agents then apply the coordination graph algorithm at each iteration of the process to decide on a joint action, potentially leading to a different coordination pattern at each step of the plan.

1 Introduction

Consider a system where multiple agents, each with its own set of possible actions and its own observations, must coordinate in order to achieve a common goal. We want to find a mechanism for coordinating the agents' actions so as to maximize their joint utility. A naive approach is to simply 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. In this paper, we provide a framework for context-specific coordination in multi-agent systems. We provide a simple communication algorithm between the agents based on an elegant and intuitive data structure which we call the *coordination graph*. We also show how this framework can be applied to sequential decision making using the framework of factored MDPs.

Our approach is based on *context specificity* — a common property of real-world decision making tasks [2]. 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 agent 1 and agent 2 are facing each other in a narrow hallway, and if they both take the action of moving forward, then the total value is decremented by 1000 points. This representation is reminiscent of the tree-structured value functions of Boutilier and Dearden [3], 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 define a notion of a *coordination graph*, which describes the dependencies implied by the value rules between the agents' actions. We provide a distributed decision-making algorithm that uses message passing over the communication graph to reach a jointly optimal action. This algorithm allows the coordination structure between the agents to vary from one situation to another. For example, it can change in different states; e.g., if the two agents are not near each other, they do not have to coordinate their motion. The coordination structure can also vary based on the exact choice of the values in the value rules; e.g., if the rewards are such that it is simply not optimal for one agent to enter the hallway, then no coordination will ever be required regardless of agent 2's action.

We then extend this framework to the problem of sequential decision making under uncertainty. We use the framework of *Markov decision processes (MDPs)*, viewing the "action" as a joint action for all of the agents and the reward is the total reward for all of the agents. Once again, we use context specificity, assuming that the rewards and the probabilities defining the transition dynamics are all rule-structured. We show how to use an efficient linear programming algorithm to construct an approximate *value function* for this MDP, one that exhibits a similar rule structure. 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 task

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, where each agent j must choose an action a_j from a finite set of possible actions $\mathrm{Dom}(A_j)$. We use $\mathbf{A}$ to denote

$\{A_1, \ldots, A_g\}$. 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 $\mathrm{Dom}(X_j)$. A state $\mathbf{x}$ defines a setting $x_j \in \mathrm{Dom}(X_j)$ for each variable X_j and an action $\mathbf{a}$ defines an action $a_j \in \mathrm{Dom}(A_j)$ for each agent. The agents must choose the joint action $\mathbf{a}$ that maximizes the total utility.

In a multi-agent coordination problem, the overall value function is often decomposed as a sum of "local" value function, associated with the "jurisdiction" of the different agents. For example, if multiple agents are collaborating in a task to clear the debris from some large area, our overall value is the total amount of debris cleared and the total resources spent. But, we can often decompose this into a set of local value functions, where the value function for each agent represents the amount of debris cleared from its area of responsibility, and the amount of resources it spent.

A standard solution is to simply specify a table for each agent, listing its local values for different combinations of variables on which the value depends. However, this representation is often highly redundant, forcing us to represent many irrelevant interactions. In our example, an agent A_1's value function might depend on the actions of a neighboring agent A_2, e.g., if A_2's action is moving debris into A_1's territory. However, this is only one situation, and there is no point in making A_1's entire value function depend on A_2's action in all other situations. As another example, the agent's value function might depend on whether it is currently raining, but only if there is a hole in the roof in his area; there is no point depending on this attribute in all other situations.

To exploit such context specific independencies, we define *value rules*:

Definition 2.1 *A* value rule $\langle \rho, \mathbf{c} : v \rangle$ *is a function* $\rho : \{\mathbf{X}, \mathbf{A}\} \mapsto I\!R$*, where the* context $\mathbf{c}$ *is an assignment to a subset of the variables* $\mathbf{C} \subseteq \{\mathbf{X}, \mathbf{A}\}$ *and* $v \in I\!R$*, such that:*

$$\rho(\mathbf{x}, \mathbf{a}) = \begin{cases} v, & \textit{if } \{\mathbf{x}, \mathbf{a}\} \textit{ is consistent with } \mathbf{c}; \\ 0, & \textit{otherwise;} \end{cases}$$

where an assignment $\mathbf{c}$ *to some set of variables* $\mathbf{C}$ *is* consistent *with some other assignment* $\mathbf{b}$ *to a subset of variables* $\mathbf{B}$ *if* $\mathbf{c}$ *and* $\mathbf{b}$ *assign the same value to every variable in* $\mathbf{C} \cap \mathbf{B}$. ∎

In our hallway example from the introduction, we might have a rule

$$\langle \rho_h, A_1, A_2 \text{ in-hallway} = \textit{true} \quad \wedge \quad A_1 = \textit{straight} \quad \wedge \quad A_2 = \textit{straight} : -1000 \rangle.$$

This definition of rules adapts the definition of rules for exploiting context specific independence in inference for Bayesian networks by Zhang and Poole [13].

Note that the value of the rule $\rho(\mathbf{x}, \mathbf{a})$ can be defined by observing only the context variables in the assignment $\{\mathbf{x}, \mathbf{a}\}$. We call such function a *restricted domain function*:

Definition 2.2 *We say that a function* f *is* restricted *to a domain* $\mathrm{Dom}[f] = \mathbf{C} \subseteq \mathbf{X}$ *if* $f : \mathbf{C} \mapsto I\!R$*. If* f *is restricted to* $\mathbf{Y}$ *and* $\mathbf{Y} \subset \mathbf{Z}$*, we will use* $f(\mathbf{z})$ *as shorthand for* $f(\mathbf{y})$ *where* $\mathbf{y}$ *is the part of the instantiation* $\mathbf{z}$ *that corresponds to variables in* $\mathbf{Y}$. ∎

Under this definition, a value rule $\langle \rho, \mathbf{c} : v \rangle$ has domain restricted to $\mathbf{C}$, e.g., $\mathrm{Dom}[\rho_h] = \{\text{agent 1 in-hallway}, \text{agent 2 in-hallway}, A_1, A_2\}$.

The immediate local utility Q_j of agent j can be represented by a set of value rules, we call such function a *rule-based function* or rule function for short:

Definition 2.3 *A* rule-based function f *is a function* $f : \{\mathbf{X}, \mathbf{A}\} \mapsto I\!R$*, composed of a set of rules* $\{\rho_1, \ldots, \rho_n\}$ *such that:*

$$f(\mathbf{x}, \mathbf{a}) = \sum_{i=1}^{n} \rho_i(\mathbf{x}, \mathbf{a}). \quad ∎$$

Thus, each Q_j is defined by a set of rules $\{\rho_1^j, \ldots, \rho_{n_j}^j\}$.

The notion of a rule-based function is related to the tree-structure functions used by Boutilier and Dearden [3] and by Boutilier et al. [2], 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.

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

$$\textit{Observable}[Q_j] = \{X_i \in \mathbf{X} \mid X_i \in \mathrm{Dom}[Q_j]\};$$

and the relevant agent decision variables:

$$\textit{Relevant}[Q_j] = \{A_i \in \mathbf{A} \mid A_i \in \mathrm{Dom}[Q_j]\}.$$

This distinction will allow us to characterize the observations each agent needs to make and the type of communication needed to obtain the joint optimal action, i.e., the joint action choice that maximizes the total utility $Q = \sum_j Q_j$.

3 Cooperative action selection

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

3.1 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 we proposed in previous work [8] and is the collaborative counterpart of the relevance graph proposed for competitive settings by Koller and Milch [11].

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 corresponds to agent decisions* $\{A_1, \ldots, A_g\}$*, and which contains an edge* $A_j \to A_i$ *if and only if* $A_i \in \textit{Relevant}[Q_j]$. ∎

An example of a coordination graph with 6 agents and one state variable is shown in Fig. 1(a). We see, for example, that agent A_3 has the parents A_1 and A_2, because both their value functions depend on A_3's action; conversely, A_3 has the child A_4, because A_4's action affects Q_3.

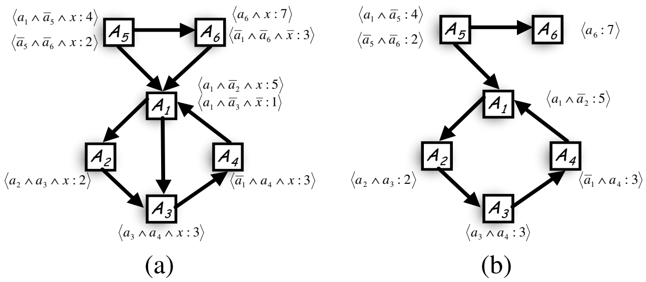


Figure 1: (a) Coordination graph for a 6-agent problem, the rules in Q_j are indicated in the figure by the rules next to A_j. (b) Graph after conditioning on the state $X = x$.

Recall that our task is to task to find a coordination strategy for the agents to maximize $\sum_j Q_j$ at state $\mathbf{x}$. First, note that the domain of the Q_j functions that comprise the value can include both action choices and state variables. We assume that the agents have full observability of the relevant state variables, i.e., agent j can observe $Observable[Q_j]$. Given a particular state $\mathbf{x} = \{x_1, \ldots, x_n\}$, agent j discards all rules in Q_j not consistent with the current state $\mathbf{x}$. We call this process conditioning on the current state.

Note that the agents do not need to have access to all of the state variables: An agent j only needs to observe the variables that appear in its Q_j function, i.e., $Observable[Q_j]$, thereby decreasing considerably the amount of information each agent needs to observe. Interestingly, after the agents observe the current state the coordination graph may become simpler, in our example the edges $A_1 \rightarrow A_3$ and $A_6 \rightarrow A_1$ 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 $\mathbf{A}$. Our task now is to select a joint action $\mathbf{a}$ that maximizes $\sum_j Q_j(\mathbf{a})$. The structure of the coordination graph will allow us to design an efficient coordination strategy for the agents. Maximization in a graph structure suggests the use of *non-serial dynamic programming* [1], 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 [13].

Intuitively, the algorithm operates by having an individual agent "collect" value rules relevant to them from their parents. 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. For example, A_1 optimal "strategy" might be to go left if A_2 goes left, with an incremental value of 3, and to go right if A_2 goes right, with an incremental value of -1. However, A_1 cannot make a final decision on what to do without knowing A_2's final decision, which might depend on various other factors as well. 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.

We now describe the algorithm more formally. In the description of our algorithm, we will need to perform a maximization step, i.e., "maximizing out" some variable. We will 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: $g(A) = \max_{a_i} f(A)$. We will describe the implementation of such procedure in Section 3.2.

Our rule-base coordination algorithm computes the maximum utility by repeating the following loop, until all agents have decided on a strategy.

1. Select some undecided agent A_l;
2. Agent A_l receives a message from its parents in the graph with all the rules that depend on A_l, i.e., all rules $\langle \rho, \mathbf{c} : v\rangle$ such that $A_l \in \mathbf{C}$. These rules are added to Q_l. At this point the graph structure will change, in particular A_l has no parents in the coordination graph and can be optimized independently.
3. Agent A_l computes its local maximization, i.e., computes $g_l = MaxOut\,(Q_l, A_l)$. This local maximization corresponds to a strategy decision.
4. Agent A_l distributes the rules in g_l to its children. At this point the graph structure changes: A_l's strategy is now fixed, and therefore 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. We note that this algorithm is essentially a context specific extension of the algorithm used to solve influence diagrams with multiple parallel decisions [9] (as is the one in the next section). However, to our knowledge, these ideas have not been applied to the problem of coordinating the decision making process of multiple collaborating agents.

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 our previous work [8], which, in turn, was exponential only in the *induced width* of the graph [6]. However, the computational costs involved in managing sets of rules usually imply that the computational advantage of the rule-based approach will only be prominent 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 a certain passageway is blocked, agent A_1 might have to transport his debris through the territory of agent A_2, requiring coordination between them.

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.

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 the floor in some room is too weak to bear the weight of two agents, it is easy to introduce an additional value rule that associates a negative value with pairs of action choices that lead to two agents going into the room at the same time.

3.2 Rule-based maximization

In our coordination algorithm, we need a maximization operator: the $MaxOut\,(f, A_i)$ procedure which takes a rule function f and a variable A_i and returns a rule function g such that: $g(\mathbf{a}) = \max_{a_i} f(\mathbf{a})$. We will assume that every rule in f contains variable A_i. Our procedure is based on the rule-based variable elimination developed by Zhang and Poole [13] for Bayesian network inference. We briefly describe our construction here, as it is slightly different from theirs.

First, we are going to need to define two basic operations: rule splitting and residual. A rule $\langle \rho, \mathbf{c} : v\rangle$ can be split on an uneliminated variable B with domain $\{b_1, \ldots, b_k\}$ and replaced by an equivalent set of rules $\{\langle \rho_1, \mathbf{c} \wedge b_1 : v\rangle, \ldots, \langle \rho_k, \mathbf{c} \wedge b_k : v\rangle\}$. We have just split a rule on a variable, but we can also split a rule on a context $\mathbf{b}$: Given a rule $\langle \rho, \mathbf{c} : v\rangle$ and a context $\mathbf{b}$ compatible with $\mathbf{c}$, denote *Split* $(\rho, \mathbf{b})$ by the successive splitting of ρ on each variable assigned in $\mathbf{b}$ which is not assigned in $\mathbf{c}$.

When we split ρ on $\mathbf{b}$ we are left with a single rule compatible with $\mathbf{b}$; all other rules have contexts incompatible with $\mathbf{b}$. This set of incompatible rules is called the residual, denoted by $Residual\,(\rho, \mathbf{b})$. More formally, we have that $Residual\,(\rho, \mathbf{b}) = \{\}$, if $\mathbf{b} \subseteq \mathbf{c}$; otherwise, select a variable B assigned as b_j in $\mathbf{b}$, but not assigned in $\mathbf{c}$ and set $Residual\,(\rho, \mathbf{b}) = \{\langle \mathbf{c} \wedge B = b_i : v\rangle \mid i \neq j\} \cup Residual\,(\langle \mathbf{c} \wedge B = b_j : v\rangle, \mathbf{b})$.

Note that we can now separate the rule that is compatible with the splitting context $\mathbf{b}$, that is, *Split* $(\rho, \mathbf{b}) = Residual\,(\rho, \mathbf{b}) \cup \langle \mathbf{c} \wedge \mathbf{b} : v\rangle$. None of the other rules will be compatible with $\mathbf{b}$. Thus, if we want to add two compatible rules $\langle \rho_1, \mathbf{c}_1 : v_1\rangle$ and $\langle \rho_2, \mathbf{c}_2 : v_2\rangle$, then all we need to do is replace these rules by the set: $Residual\,(\rho_1, \mathbf{c}_2) \cup Residual\,(\rho_2, \mathbf{c}_1) \cup \langle \mathbf{c}_1 \wedge \mathbf{c}_2 : v_1 + v_2\rangle$.

We are now ready to describe the procedure. Note that we need to add a set of value rules with zero value to guarantee that our rule function f is complete, i.e., assigns a value to every context. The procedure in Fig. 2 maximizes out variable B from rule function f.

$MaxOut\,(f, B)$
$g = \{\}$.
Add completing rules to f:
$\langle \rho_i, B = b_i : 0\rangle, i = 1, \ldots, k$.
// Summing compatible rules:
While there are two compatible rules $\langle \rho_1, \mathbf{c}_1 : v_1\rangle$ and $\langle \rho_2, \mathbf{c}_2 : v_2\rangle$:
replace these two rules by the set:
$Residual\,(\rho_1, \mathbf{c}_2) \cup Residual\,(\rho_2, \mathbf{c}_1) \cup \langle \mathbf{c}_1 \wedge \mathbf{c}_2 : v_1 + v_2\rangle$.
// Maximizing out variable B:
Repeat until f is empty:
If there are rules $\langle B = b_i \wedge \mathbf{c} : v_i\rangle, \forall b_i \in \mathrm{Dom}(B)$, then remove these rules from f and add rule $\langle \mathbf{c} : \max_i v_i\rangle$ to g;
Else select two rules: $\langle \rho_i, B = b_i \wedge \mathbf{c}_i : v_i\rangle$ and $\langle \rho_j, B = b_j \wedge \mathbf{c}_j : v_j\rangle$ such that $\mathbf{c}_i$ is compatible with $\mathbf{c}_j$, but not identical, and replace them with
Split $(\rho_i, \mathbf{c}_j) \cup$ *Split* $(\rho_j, \mathbf{c}_i)$

Figure 2: Maximizing out variable B from rule function f.

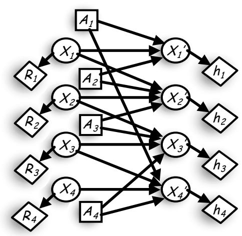


Figure 3: A DDN for a 4-agent MDP.

4 One-Step Lookahead

We now consider an extension to the action selection problem: we 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)* [5].

4.1 Dynamic Decision Network

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, 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. (This assumption can be relaxed, but our algorithm becomes somewhat more complex.) Each node X_i' is associated with a *conditional probability distribution (CPD)* $P(X_i' \mid \textit{Parents}(X_i'))$. 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. 3 shows a DDN for a simple four-agent problem, where the ovals represent the variables X_i and the rectangles the agent actions. 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. The actions of agents $A_1, \ldots, A_4$ are represented as squares.

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 $P(X_i' \mid \textit{Parents}(X_i'))$. However, as discussed by Boutilier *et al.* [2], 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, which exploits context specificity. We now describe how the rule representation can also be used for the probabilities specifying the transition dynamics; our formulation follows the lines of [13].

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 $\mathbf{c}$ *is an assignment to a subset of the variables* $\mathbf{C} \subseteq \{\mathbf{X}, \mathbf{X}', \mathbf{A}\}$ *and* $p \in [0,1]$*, such that:*

$$\pi(\mathbf{x}, \mathbf{x}', \mathbf{a}) = \begin{cases} p, & \textit{if } \{\mathbf{x}, \mathbf{x}', \mathbf{a}\} \textit{ is consistent with } \mathbf{c}; \\ 1, & \textit{otherwise;} \end{cases}$$

if $\{\mathbf{x}, \mathbf{x}', \mathbf{a}\}$ *is consistent with* $\mathbf{c}$ *we say that the rule* π *is* applicable *to* $\{\mathbf{x}, \mathbf{x}', \mathbf{a}\}$*. 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:*

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

where for every assignment of $(\mathbf{x}'_i, \mathbf{x}, \mathbf{a})$ *one and only one rule should be applicable.*

We can now define the conditional probabilities $P(X'_i \mid \mathit{Parents}(X'_i))$ as a rule CPD, where the context variables $\mathbf{C}$ of the rules depend on variables in $\{X'_i \cup \mathit{Parents}(X'_i)\}$. Using this compact DDN representation, we will be able to compute a one-step lookahead plan very efficiently.

4.2 One-step lookahead action selection

In the one-step lookahead case, for any setting $\mathbf{x}$ of the state variables, the agents aim to maximize:

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

We can decompose this global utility, or Q function, into local Q_j functions:

$$Q_j(\mathbf{x}, \mathbf{a}) = r_j(\mathbf{x}, \mathbf{a}) + \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) h_j(\mathbf{x}').$$

The objective of the agents is to coordinate and find a joint action that maximizes $Q(\mathbf{x}, \mathbf{a}) = \sum_{j=1}^{g} Q_j(\mathbf{x}, \mathbf{a})$. However, computing Q_j's naively would require an enumeration of every one of the exponentially many states. This computation can be simplified dramatically by using a factored backprojection [10] to replace the above expectation. Here, we extend this construction to exploit rule functions.

First, note that 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 domain function; thus, we can write the expected value g_j of receiving h_j at the next time step, a procedure called *backprojection*, as:

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

$\mathit{RuleBackproj}(\rho)$, where ρ is given by $\langle \rho, \mathbf{c} : v\rangle$.
  $g = \{\}$.
  Generate a set $\mathcal{P}$ of relevant probability rules: $\mathcal{P} = \{\pi_j \in P(X'_i \mid \mathit{Parents}(X'_i)) \mid X_i$ assigned in $\mathbf{c}$ and $\mathbf{c}$ is compatible with $\mathbf{c}_j\}$.
  Remove $\mathbf{X}'$ assignments from the context of all rules in $\mathcal{P}$.
  // Multiply compatible rules:
  While there are two compatible rules $\langle \rho_1, \mathbf{c}_1 : p_1\rangle$ and $\langle \rho_2, \mathbf{c}_2 : p_2\rangle$:
    replace these two rules by the set:
    $\mathit{Residual}(\rho_1, \mathbf{c}_2) \cup \mathit{Residual}(\rho_2, \mathbf{c}_1) \cup \langle \mathbf{c}_1 \wedge \mathbf{c} : p_1 p_2\rangle$.
  // Generate value rules:
  For each rule π_i in $\mathcal{P}$:
    $g = g \cup \{\langle \mathbf{c}_i : p_i v\rangle\}$.

Figure 4: Rule-based backprojection.

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 operation by $\mathit{RuleBackproj}(\rho_i^{(h_j)})$ and describe the procedure in Fig. 4. Thus, we can write the backprojection of h_j as:

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

where g_j is a sum of rule-based functions, and therefore also a rule-based function. For notation purposes, we will use $g_j = \mathit{RuleBackproj}(h_j)$ to refer to this definition of backprojection.

Using this notation, we can write the local utilities Q_j as:

$$Q_j(\mathbf{x}, \mathbf{a}) = r_j(\mathbf{x}, \mathbf{a}) + g_j(\mathbf{x}, \mathbf{a});$$

which is again a rule-based function. Thus, we can decompose Q as a set of rule-based functions Q_j, one for each agent, which depend only on the current state $\mathbf{x}$ and action choice $\mathbf{a}$. This is exactly the case we addressed in Section 3.1. Therefore, we can perform efficient one-step lookahead planning using the same coordination graph.

5 Multi-Agent 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. We begin by describing the MDP framework in general, and then discussing how the techniques described in the previous section can be used to allow agents to make optimal collaborative decisions in context-sensitive dynamic settings.

5.1 Markov Decision Processes

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* $\mathcal{R} : \mathbf{X} \times \mathcal{A} \mapsto \mathbb{R}$, 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 $\gamma \in [0, 1)$. A stationary policy π for an MDP is a mapping $\pi : \mathbf{X} \mapsto A$, where $\pi(\mathbf{x})$ is the action the agent takes at state $\mathbf{x}$. The optimal value function $\mathcal{V}^*$ is defined so that the value of a state must be the maximal value achievable by any action at that state. More precisely, 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. Our LP is:

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

Where the state relevance weights α can be any positive weight vector such that $\alpha(\mathbf{x}) > 0, \forall \mathbf{x}$ and $\sum_{\mathbf{x}} \alpha(\mathbf{x}) = 1$, i.e., we can think of α as a probability distribution over the states.

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 $\mathbf{w} = (w_1, \ldots, w_k)'$. It is useful to define an $|\mathbf{X}| \times k$ matrix A whose columns are the k basis functions, viewed as vectors. Our approximate value function is then represented by $A\mathbf{w}$.

The linear programming approach can be adapted to use this value function representation [12] by changing the objective function to $\alpha \cdot A\mathbf{w}$, and changing the constraints to have the form, $A\mathbf{w} \geq \mathcal{T}^* A\mathbf{w}$. 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:

$$\begin{array}{ll} \text{Variables:} & w_1, \ldots, w_k \; ; \\ \text{Minimize:} & \sum_{\mathbf{x}} \alpha(\mathbf{x}) \sum_i w_i \; h_i(\mathbf{x}) \; ; \\ \text{Subject to:} & \sum_i w_i \; h_i(\mathbf{x}) \geq \\ & \quad R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) \sum_i w_i \; h_i(\mathbf{x}') \\ & \qquad \forall \mathbf{x} \in \mathbf{X}, \forall a \in A. \end{array} \tag{2}$$

In other words, this formulation takes the exact LP and substitutes the explicit state value function by a linear value function representation $\sum_i w_i \; h_i(\mathbf{x})$, or, in our more compact notation, $\mathcal{V}$ is replaced by $A\mathbf{w}$. There is, in general, no guarantee as to the quality of the approximation $A\mathbf{w}$, but the recent work of de Farias and Van Roy [4] provides some analysis of the error relative to that of the best possible approximation in the subspace, and some guidance as to selecting α so as to improve the quality of the approximation. 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 $|\mathbf{X}| \times |\mathcal{A}|$. In the next section, we discuss how we can use the structure of a *factored MDP* to provide a compact representation and efficient solution of this LP.

5.2 Factored MDPs

Factored MDPs [2] allow the representation of large structured MDPs by using a dynamic Bayesian network to represent the transition model. Our representation of the one-step transition dynamics in Section 4 is precisely a factored MDP, where we factor not only the states but also the actions.

In [10], we proposed the use of *factored linear value functions* to approximate the value function in a factored MDP. These value functions are a weighted linear combination of basis functions, as above, but where each basis function is restricted to depend only on a small subset of state variables. The h functions in Fig. 3 are an example. In this paper, we extend the factored value function concept to rules. Thus, each basis function h_j is a rule function. If we had a value function $\mathcal{V}$ represented in this way, then we could use our algorithm of Section 4 to implement *Greedy*$(\mathcal{V})$ by having the agents use our message passing coordination algorithm at each step.

First, consider the objective function of the LP in (2):

$$\mathit{Objective} = \sum_{\mathbf{x}} \alpha(\mathbf{x}) \sum_j w_j \; h_j(\mathbf{x}).$$

Representing such an objective function explicitly would require a summation over all the exponentially large state space. However, we can use structure in the rule functions to represent this objective function compactly. Note that:

$$\begin{aligned} \mathit{Objective} &= \sum_j w_j \sum_{\mathbf{x}} \alpha(\mathbf{x}) \; h_j(\mathbf{x}); \\ &= \sum_j w_j \sum_i \sum_{\mathbf{x}} \alpha(\mathbf{x}) \rho_i^{(h_j)}(\mathbf{x}); \\ &= \sum_j w_j \sum_i v_i^{(h_j)} \alpha(\mathbf{c}_i^{(h_j)}) = \sum_j w_j \alpha_j; \end{aligned}$$

where $\alpha(\mathbf{c}_i^{(h_j)}) = \sum_{[\mathbf{x} \text{ consitent with } \mathbf{c}_i^{(h_j)}]} \alpha(\mathbf{x})$, i.e., we are marginalizing out from α all variables that do not appear in the context of the rule. Such computation can be very efficient if we use a factored representation for the state relevance weights α, e.g., a Bayesian network. In our experiments, we used an uniform distribution. Here, $\alpha(\mathbf{c}_i^{(h_j)})$ is simply $1/\left|\mathbf{C}_i^{(h_j)}\right|$. Using this method, we can precompute one coefficient α_j for each basis function and our objective function simply becomes $\sum_j w_j \alpha_j$.

The second step is the representation of the constraints:

$$\sum_i w_i \; h_i(\mathbf{x}) \geq R(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) \sum_i w_i \; h_i(\mathbf{x}'); \quad \forall \mathbf{x} \in \mathbf{X}, \forall \mathbf{a} \in \mathbf{A}.$$

Although there are exponentially many constraints, we can replace these constraints by an equivalent set which is exponentially smaller. In previous work, we have applied such transformation in the context of single agent problems [7] and table-based multiagent factored MDPs [8]. We will now extend these ideas to exploit the rule-based representation of our reward and basis functions.

First, note that the constraints above 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 $g_i = \mathit{RuleBackproj}(h_i) = \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) h_i(\mathbf{x}')$, which can be computed as described in Section 4.2. Although, a naive approach to maximizing over the state space would require the enumeration of every state, as we have shown in Section 3.1, the structure in rule functions allow us to perform such maximization very efficiently. We are going to apply the same intuition to decompose this nonlinear constraint into a set of linear constraints following a very similar procedure to variable elimination.

More generally, suppose we wish to enforce the constraint $0 \geq max_{\mathbf{y}} F^{\mathbf{w}}(\mathbf{y})$, where $F^{\mathbf{w}}(\mathbf{y}) = \sum_j f_j^{\mathbf{w}}(\mathbf{y})$ such that each f_j is a rule; in the multiagent case we are considering $\mathbf{Y} = \mathbf{X} \cup \mathbf{A}$. The superscript $\mathbf{w}$ means that f_j might depend on $\mathbf{w}$. Specifically, if f_j comes from a basis function, it would be multiplied by some weight w_i; if f_j is a rule from the reward function, it will not.

In our factored linear program, we generate LP variables, which we are going to associate with contexts; we call these *LP rules*. An LP rule has the form $\langle e, \mathbf{c} : u \rangle$; it is associated with a context $\mathbf{c}$ and a variable u in the linear program. We begin by transforming all our original rules $f_j^{\mathbf{w}}$ into LP rules as follows: If rule f_j has the form $\langle \rho_j, \mathbf{c}_j : v_j \rangle$; and comes from basis function i, we introduce an LP rule $\langle e_j, \mathbf{c}_j : u_j \rangle$ and the equality constraint $u_j = w_i v_j$; if f_j has the same form but comes from a reward function, we introduce an LP rule of the same form, but the equality constraint $u_j = v_j$.

Now, we have only LP rules and need to represent the constraint: $0 \geq max_{\mathbf{y}} \sum_j e_j(\mathbf{y})$. To represent such constraint, we follow an algorithm very similar to the variable elimination procedure in Section 3.1. The main difference occurs in the $\mathit{MaxOut}(f, B)$ operation in Fig. 2. Instead of generating new value rules, we generate new LP rules, with associated new variables and new constraints. The simplest case occurs when computing a residual or adding two LP rules. For example, when we would add two value rules in the original algorithm, we instead perform the following operation on their associated LP rules: If the LP rules are $\langle e_i, \mathbf{c}_i : u_i \rangle$ and $\langle e_j, \mathbf{c}_j : u_j \rangle$, we create a new rule with context $\mathbf{c}_i \wedge \mathbf{c}_j$ whose value should be $u_i + u_j$. To enforce this last constraint, we simply create a new LP variable u_k associated with the rule $\langle e_k, \mathbf{c}_i \wedge \mathbf{c}_j : u_k \rangle$ and add an additional constraint $u_k = u_i + u_j$. A similar procedure can be followed when computing the residuals.

More interesting constraints are generated when we perform a maximization. In the original algorithm in Fig. 2 this occurs when we create a new rule $\langle \mathbf{c} : \max_i v_i \rangle$. Following the same process as in the LP rule summation above, if we were maximizing $\langle e_i, B = b_i \wedge \mathbf{c}_i : u_i \rangle, \forall b_i \in \mathrm{Dom}(B)$, we can generate a new LP variable u_k associated with the rule $\langle e_k, \mathbf{c} : u_k \rangle$. However, we cannot add the nonlinear constraint $u_k = \max_i u_i$, but we can add a set of equivalent linear constraints $u_k \geq u_i$, for each i.

Therefore, using these simple operations, we can exploit structure in the rule functions to represent the nonlinear constraint $e_n \geq max_{\mathbf{y}} \sum_j e_j(\mathbf{y})$, where e_n is the very last LP rule we generate. A final constraint $u_n = 0$ implies that we are representing exactly the constraints in the LP of (2), without having to enumerate every state and action. This exponential saving will allow us to solve very large rule-based factored MDPs very efficiently.

More interestingly, this structure can give rise to very interesting coordination behavior. The value functions that we construct using this LP algorithm are rule-based functions. These are the value functions used as the one-step lookahead in Section 4. Assuming that the reward function and transition model have a similar structure, we will get an overall value function that is also a rule-based function. This value function is the one that is used at each stage of the MDP for the agents to select an optimal joint action.[1] As discussed in Section 3.1, the use of rule-based functions allows the agents to use the coordination graph as a data structure which the agents can use to jointly select optimal actions in a distributed way.

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. As we discussed, the coordination graph depends on the state variable assignment $\mathbf{x}$, so that in different states the coordination pattern changes.

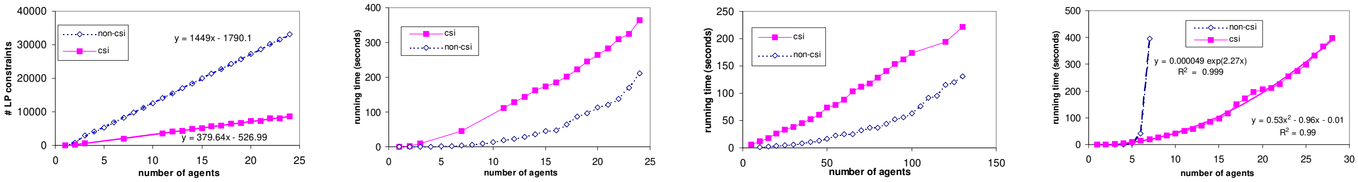
Furthermore, as we discussed, the coordination also depends on the numerical values of the value rules; i.e., sometimes dominance of one action over another will allow the elimination of a coordination step without losing optimality. Thus, even an MDP with the same structure and the same set of value rules, a different set of reward values might lead to a different coordination strategy.

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 tested our new rule-based algorithm on a variation of the multiagent SysAdmin problem presented in [8]. In this problem, there is a network of computers and each computer is associated with an administrator agent. Here, the dynamic system associated with each machine is represented with an agent A_i and three variables: Status $S_i \in$ {good, faulty, dead}, Load $L_i \in$ {idle, loaded, process successful} and NeighborMessage $M_i \in \{1, \ldots, \sharp$ neighbors$\}$. The system receives a reward of 1 if a process terminates successfully (in the ring topologies, one machine receives reward of 2 to break symmetry). If the Status is faulty, processes take longer to terminate. If the machine dies, the process is lost. Each agent A_i must decide whether machine i should be rebooted, in which case the Status becomes good and any running process is lost. Machines become faulty and die with small probabilities. However, a dead neighbor on the network may transmit bad packets, considerably increasing the probability that a machine will become faulty and die. This neighbor interaction is governed by the NeighborMessage variable, which, at every time step, uniformly selects one neighboring machine

[1] It is important to note that, by optimal, we mean optimal relative to the computed value function. The value function itself is an approximation of the long-term optimal value function, and therefore does not guarantee truly optimal behavior.



(a) (b) (c) (d)

Figure 5: Bidirectional ring: (a) Number of LP constraints; (b) Running time. Other running times: (c) Unidirectional ring; (d) Inverted star.

to receive packets from. For a network of n machines, the number of states in the MDP is 9^n and the joint action space contains 2^n possible actions, e.g., a problem with 30 agents has over 10^{28} states and a billion possible actions.

We implemented the rule-based factored approximate linear programming and the message passing coordination algorithms in C++, using CPLEX as the LP solver. We experimented with "single" basis functions, which contains a rule basis function for each joint value of each S_i and L_i. We use $\gamma = 0.95$. Three network topologies were considered: bidirectional ring, where machine i can communicate with machine $i+1$ and $i-1$ (mod n); unidirectional ring, where machine i can communicate only with machine $i+1$ (mod n); and reverse star, where every machine can send packets only to machine 1.

For bidirectional ring, for example, as shown in Fig. 5(a), 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 rule-based approach to be about two times slower than the table-based approach, as shown in Fig. 5(b). Similar observations can be made in the unidirectional ring case in Fig. 5(c).

However, note that in ring topologies the number of parents for each variable in the DDN and the induced width of the coordination graph are constant as the number of agents increases. On the other hand, in the reverse star topology, every machine in the network can affect the status of machine 1, as all machines can send it (potentially bad) packets. Thus, the number of parents of S_1 increases with the number of computers in the network. In this case, we observe quite a different behavior, as seen in Fig. 5(d). 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 principled and efficient approach to planning in multiagent domains. We show that the task of finding an optimal joint action relative to a rule-based value function leads to a very natural communication pattern, where agents send messages along a *coordination* graph determined by the structure of the value rules. Rather than placing *a priori* restrictions on the communication structure between agents, 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.

Our initial experimental results are very promising. By exploiting rule structure in both the state and action spaces, we can deal with considerably larger MDPs than those described in previous work. In a family of multiagent network administration problems, the algorithm tackles MDPs with over 10^{124} states and over 10^{39} actions. Furthermore, we demonstrated that the rule-based method can scale polynomially as the size of parent sets increases, as opposed to the exponential growth of the table-based method. We believe that the adaptive coordination graph presented here 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. The first author was also supported by a Siebel Scholarship.

References

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

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

[3] C. Boutilier and R. Dearden. Approximating value trees in structured dynamic programming. In *Proc. ICML*, 1996.

[4] D.P. de Farias and B. Van Roy. The linear programming approach to approximate dynamic programming. *Submitted to the IEEE Transactions on Automatic Control*, January 2001.

[5] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Comp. Intelligence*, 5(3):142–150, 1989.

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

[7] Carlos Guestrin, Daphne Koller, and Ronald Parr. Max-norm projections for factored MDPs. In *Proc. of IJCAI-01*, 2001.

[8] Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored MDPs. In *NIPS-14*, 2001.

[9] F. Jensen, F. Jensen, and S. Dittmer. From influence diagrams to junction trees. In *UAI-94*, 1994.

[10] D. Koller and R. Parr. Computing factored value functions for policies in structured MDPs. In *IJCAI-99*, 1999.

[11] D. Koller and B. Milch. Multi-agent influence diagrams for representing and solving games. In *IJCAI-01*, 2001.

[12] P. Schweitzer and A. Seidmann. Generalized polynomial approximations in Markovian decision processes. *Journ. of Math. Analysis and Applications*, 110:568 – 582, 1985.

[13] N.L. Zhang and D. Poole. On the role of context-specific independence in probabilistic reasoning. In *IJCAI-99*, 1999.