Generalizing Plans to New Environments in Relational MDPs

Carlos Guestrin Daphne Koller Chris Gearhart Neal Kanodia

Computer Science Department, Stanford University

{*guestrin, koller, cmg33, nkanodia*}*@cs.stanford.edu*

Abstract

A longstanding goal in planning research is the ability to generalize plans developed for some set of environments to a new but similar environment, with minimal or no replanning. Such generalization can both reduce planning time and allow us to tackle larger domains than the ones tractable for direct planning. In this paper, we present an approach to the generalization problem based on a new framework of *relational Markov Decision Processes (RMDPs)*. An RMDP can model a set of similar environments by representing objects as instances of different classes. In order to generalize plans to multiple environments, we define an approximate value function specified in terms of classes of objects and, in a multiagent setting, by classes of agents. This class-based approximate value function is optimized relative to a sampled subset of environments, and computed using an efficient linear programming method. We prove that a polynomial number of sampled environments suffices to achieve performance close to the performance achievable when optimizing over the entire space. Our experimental results show that our method generalizes plans successfully to new, significantly larger, environments, with minimal loss of performance relative to environment-specific planning. We demonstrate our approach on a real strategic computer war game.

1 Introduction

Most planning methods optimize the plan of an agent in a fixed environment. However, in many real-world settings, an agent will face multiple environments over its lifetime, and its experience with one environment should help it to perform well in another, even with minimal or no replanning.

Consider, for example, an agent designed to play a strategic computer war game, such as the *Freecraft* game shown in Fig. 1 (an open source version of the popular *Warcraft* game). In this game, the agent is faced with many scenarios. In each scenario, it must control a set of agents (or units) with different skills in order to defeat an opponent. Most scenarios share the same basic elements: *resources*, such as gold and wood; *units*, such as peasants, who collect resources and build structures, and footmen, who fight with enemy units; and *structures*, such as barracks, which are used to train footmen. Each scenario is composed of these same basic building blocks, but they differ in terms of the map layout, types of units available, amounts of resources, etc. We would like the agent to learn from its experience with playing some scenarios, enabling it to tackle new scenarios without significant amounts of replanning. In particular, we would like the agent to generalize from simple scenarios, allowing it to deal with other scenarios that are too complex for any effective planner.

The idea of generalization has been a longstanding goal in Markov Decision Process (MDP) and reinforcement learning

Figure 1: Freecraft strategic domain with 9 peasants, a barrack, a castle, a forest, a gold mine, 3 footmen, and an enemy, executing the generalized policy computed by our algorithm.

research [15; 16], and even earlier in traditional planning [5]. This problem is a challenging one, because it is often unclear how to translate the solution obtained for one domain to another. MDP solutions assign values and/or actions to states. Two different MDPs (*e.g.*, two Freecraft scenarios), are typically quite different, in that they have a different set (and even number) of states and actions. In cases such as this, the mapping of one solution to another is not well-defined.

Our approach is based on the insight that many domains can be described in terms of objects and the relations between them. A particular domain will involve multiple objects from several classes. Different tasks in the same domain will typically involve different sets of objects, related to each other in different ways. For example, in Freecraft, different tasks might involve different numbers of peasants, footmen, enemies, etc. We therefore define a notion of a *relational MDP (RMDP)*, based on the *probabilistic relational model (PRM)* framework [10]. An RMDP for a particular domain provides a general schema for an entire suite of environments, or worlds, in that domain. It specifies a set of classes, and how the dynamics and rewards of an object in a given class depend on the state of that object and of related objects.

We use the class structure of the RMDP to define a value function that can be generalized from one domain to another. We begin with the assumption that the value function can be well-approximated as a sum of *value subfunctions* for the different objects in the domain. Thus, the value of a global Freecraft state is approximated as a sum of terms corresponding to the state of individual peasants, footmen, gold, etc. We then assume that individual objects in the same class have a very similar value function. Thus, we define the notion of a *class-based value function*, where each class is associated with a *class subfunction*. All objects in the same class have the value subfunction of their class, and the overall value function for a particular environment is the sum of value subfunctions for the individual objects in the domain.

A set of value subfunctions for the different classes immediately determines a value function for any new environment in the domain, and can be used for acting. Thus, we can compute a set of class subfunctions based on a subset of environments, and apply them to another one without replanning.

We provide an optimality criterion for evaluating a class-based value function for a distribution over environments, and show how it can, in principle, be optimized using a linear program. We can also "learn" a value function by optimizing it relative to a sample of environments encountered by the agent. We prove that a polynomial number of sampled environments suffice to construct a class-based value function which is close to the one obtainable for the entire distribution over environments. Finally, we show how we can improve the quality of our approximation by automatically discovering subclasses of objects that have "similar" value functions.

We present experiments for a computer systems administration task and two Freecraft tasks. Our results show that we can successfully generalize class-based value functions. Importantly, our approach also obtains effective policies for problems significantly larger than our planning algorithm could handle otherwise.

2 Relational Markov Decision Processes

A *relational MDP* defines the system dynamics and rewards at the level of a template for a task domain. Given a particular environment within that domain, it defines a specific MDP instantiated for that environment. As in the PRM framework of [10], the domain is defined via a *schema*, which specifies a set of *object classes* $\mathcal{C} = \{C_1, \ldots, C_c\}$. Each class C is also associated with a set of *state variables* $\mathcal{S}[C] = \{C.S_1, \ldots, C.S_k\}$, which describe the state of an object in that class. Each state variable $C.S$ has a *domain* of possible values $\mathrm{Dom}[C.S]$. We define $\boldsymbol{S}_C$ to be the set of possible states for an object in C, *i.e.*, the possible assignments to the state variables of C.

For example, our Freecraft domain might have classes such as Peasant, Footman, Gold; the class Peasant may have a state variable *Task* whose domain is $\mathrm{Dom}[\mathsf{Peasant}.\mathit{Task}] = \{\mathit{Waiting, Mining, Harvesting, Building}\}$, and a state variable *Health* whose domain has three values. In this case, $\boldsymbol{S}_{\mathsf{Peasant}}$ would have $4 \cdot 3 = 12$ values, one for each combination of values for *Task* and *Health*.

The schema also specifies a set of *links* $\mathcal{L}[C] = \{L_1, \ldots, L_l\}$ for each class representing links between objects in the domain. Each link $C.L$ has a *range* $\rho[C.L] = C'$. For example, Peasant objects might be linked to Barrack objects — $\rho[\mathsf{Peasant}.\mathit{BuildTarget}] = \mathsf{Barrack}$, and to the global Gold and Wood resource objects. In a more complex situation, a link may relate C to many instances of a class C', which we denote by $\rho[C.L] = \{C'\}$, for example, $\rho[\mathsf{Enemy}.\mathit{My\_Footmen}] = \{\mathsf{Footman}\}$ indicates that an instance of the enemy class may be related to many footman instances.

A particular instance of the schema is defined via a *world* ω, specifying the set of objects of each class; we use $\mathcal{O}[\omega][C]$ to denote the objects in class C, and $\mathcal{O}[\omega]$ to denote the total set of objects in ω. The world ω also specifies the links between objects, which we take to be fixed throughout time. Thus, for each link $C.L$, and for each $o \in \mathcal{O}[\omega][C]$, ω specifies a set of objects $o' \in \rho[C.L]$, denoted $o.L$. For example, in a world containing 2 peasants, we would have $\mathcal{O}[\omega][\mathsf{Peasant}] = \{\mathrm{Peasant1}, \mathrm{Peasant2}\}$; if Peasant1 is building a barracks, we would have that $\mathrm{Peasant1}.\mathit{BuildTarget} = \mathrm{Barrack1}$.

The dynamics and rewards of an RMDP are also defined at the schema level. For each class, the schema specifies an *action* $C.A$, which can take on one of several values $\mathrm{Dom}[C.A]$. For example, $\mathrm{Dom}[\mathsf{Peasant}.A] = \{\mathit{Wait, Mine, Harvest, Build}\}$. Each class C is also associated with a *transition model* P^C, which specifies the probability distribution over the next state of an object o in class C, given the current state of o, the action taken on o, and the states and actions of all of the objects linked to o:

$$P^C(\boldsymbol{S}'_C \mid \boldsymbol{S}_C, C.A, \boldsymbol{S}_{C.L_1}, C.L_1.A, \ldots, \boldsymbol{S}_{C.L_l}, C.L_l.A). \quad (1)$$

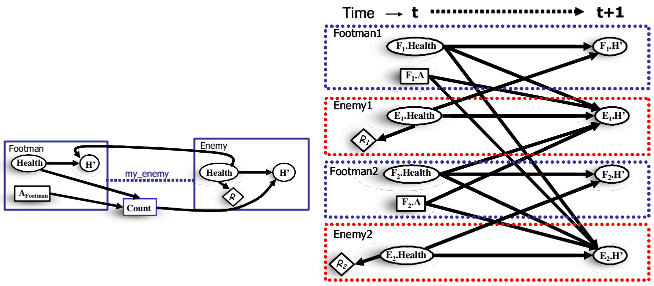
For example, the status of a barrack, $\mathsf{Barrack}.\mathit{Status}'$, depends on its status in the previous time step, on the task performed by any peasant that could build it ($\mathsf{Barrack}.\mathit{BuiltBy}.\mathit{Task}$), on the amount of wood and gold, etc.

The transition model is conditioned on the state of $C.L_i$, which is, in general, an entire set of objects (*e.g.*, the set of peasants linked to a barrack). Thus we must now provide a compact specification of the transition model that can depend on the state of an unbounded number of variables. We can deal with this issue using the idea of *aggregation* [10]. In Freecraft, our model uses the *count* aggregator $\sharp$, where the probability that $\mathsf{Barrack}.\mathit{Status}$ transitions from *Unbuilt* to *Built* depends on $\sharp[\mathsf{Barrack}.\mathit{BuiltBy}.\mathit{Task} = \mathit{Built}]$, the *number* of peasants in $\mathsf{Barrack}.\mathit{BuiltBy}$ whose *Task* is *Build*.

Finally, we also define rewards at the class level. We assume for simplicity that rewards are associated only with the states of individual objects; adding more global dependencies is possible, but complicates planning significantly. We define a reward function $R^C(\boldsymbol{S}_C, C.A)$, which represents the contribution to the reward of any object in C. For example, we may have a reward function associated with the Enemy class, which specifies a reward of 10 if the state of an enemy object is *Dead*: $R^{\mathsf{Enemy}}(\mathsf{Enemy}.\mathit{State} = \mathit{Dead}) = 10$. We assume that the reward for *each object* is bounded by R_{max}.

Given a world, the RMDP uniquely defines a *ground* factored MDP Π_ω, whose transition model is specified (as usual) as a dynamic Bayesian network (DBN) [3]. The random variables in this factored MDP are the state variables of the individual objects $o.S$, for each $o \in \mathcal{O}[\omega][C]$ and for each $S \in \mathcal{S}[C]$. Thus, the state $\boldsymbol{s}$ of the system at a given point in time is a vector defining the states of the individual objects in the world. For any subset of variables $\boldsymbol{X}$ in the model, we define $\boldsymbol{s}[\boldsymbol{X}]$ to be the part of the instantiation $\boldsymbol{s}$ that corresponds to the variables $\boldsymbol{X}$. The ground DBN for the transition dynamics specifies the dependence of the variables at time $t+1$ on the variables at time t. The parents of a variable $o.S$ are the state variables of the objects o' that are linked to o. In our example with the two peasants, we might have the random variables $\mathrm{Peasant1}.\mathit{Task}$, $\mathrm{Peasant2}.\mathit{Task}$, $\mathrm{Barrack1}.\mathit{Status}$, etc. The parents of the time $t+1$ variable $\mathrm{Barrack1}.\mathit{Status}'$ are the time t variables $\mathrm{Barrack1}.\mathit{Status}'$, $\mathrm{Peasant1}.\mathit{Task}$, $\mathrm{Peasant2}.\mathit{Task}$, $\mathrm{Gold1}.\mathit{Amount}$ and $\mathrm{Wood1}.\mathit{Amount}$.

The transition model is the same for all instances in the same class, as in (1). Thus, all of the $o.\mathit{Status}$ variables for



(a) (b)

Figure 2: Freecraft tactical domain: (a) Schema; (b) Resulting factored MDP for a world with 2 footmen and 2 enemies.

barrack objects o share the same conditional probability distribution. Note, however, that each specific barrack depends on the particular peasants linked to it. Thus, the actual parents in the DBN of the status variables for two different barrack objects can be different.

The reward function is simply the sum of the reward functions for the individual objects:

$$R(\boldsymbol{s}, \boldsymbol{a}) = \sum_{C \in \mathcal{C}} \sum_{o \in \mathcal{O}[\omega][C]} R(\boldsymbol{s}[\boldsymbol{S}_o], \boldsymbol{a}[o.A]).$$

Thus, for reward function for the Enemy class described above, our overall reward function in a given state will be 10 times the number of dead enemies in that state.

It remains to specify the actions in the ground MDP. The RMDP specifies a set of possible actions for every object in the world. In a setting where only a single action can be taken at any time step, the agent must choose both an object to act on, and which action to perform on that object. Here, the set of actions in the ground MDP is simply the union $\cup_{o \in \omega} \mathrm{Dom}[o.A]$. In a setting where multiple actions can be performed in parallel (say, in a multiagent setting), it might be possible to perform an action on every object in the domain at every step. Here, the set of actions in the ground MDP is a vector specifying an action for every object: $\times_{o \in \omega} \mathrm{Dom}[o.A]$. Intermediate cases, allowing degrees of parallelism, are also possible. For simplicity of presentation, we focus on the multiagent case, such as Freecraft, where, an action is an assignment to the action of every unit.

Example 2.1 (Freecraft tactical domain) *Consider a simplified version of Freecraft, whose schema is illustrated in Fig. 2(a), where only two classes of units participate in the game:* $\mathcal{C} = \{\mathsf{Footman}, \mathsf{Enemy}\}$. *Both the footman and the enemy classes have only one state variable each, Health, with domain* $\mathrm{Dom}[\mathit{Health}] = \{\mathit{Healthy}, \mathit{Wounded}, \mathit{Dead}\}$. *The footman class contains one single-valued link:* $\rho[\mathsf{Footman}.\mathit{My\_Enemy}] = \mathsf{Enemy}$. *Thus the transition model for a footman's health will depend on the health of its enemy:* $P^{\mathsf{Footman}}(\boldsymbol{S}'_{\mathsf{Footman}} \mid \boldsymbol{S}_{\mathsf{Footman}}, \boldsymbol{S}_{\mathsf{Footman}.\mathit{My\_Enemy}})$, *i.e., if a footman's enemy is not dead, than the probability that a footman will become wounded, or die, is significantly higher. A footman can choose to attack any enemy. Thus each footman is associated with an action* $\mathsf{Footman}.A$ *which selects the enemy it is attacking.*[1] *As a consequence, an enemy could end up being linked to a set of footmen,* $\rho[\mathsf{Enemy}.\mathit{My\_Footmen}] = \{\mathsf{Footman}\}$. *In this case, the transition model of the health of an enemy may depend on the number of footmen who are not dead and whose action choice is to attack this enemy:* $P^{\mathsf{Enemy}}(\boldsymbol{S}'_{\mathsf{Enemy}} \mid \boldsymbol{S}_{\mathsf{Enemy}}, \sharp[\boldsymbol{S}_{\mathsf{Enemy}.\mathit{My\_Footmen}}, \mathsf{Enemy}.\mathit{My\_Footmen}.A])$. *Finally, we must define the template for the reward function. Here there is only a reward when an enemy is dead:* $R^{\mathsf{Enemy}}(\boldsymbol{S}_{\mathsf{Enemy}})$.

We now have a template to describe any instance of the tactical Freecraft domain. In a particular world, we must define the instances of each class and the links between these instances. For example, a world with 2 footmen and 2 enemies will have 4 objects: $\{\mathit{Footman1}, \mathit{Footman2}, \mathit{Enemy1}, \mathit{Enemy2}\}$. *Each footman will be linked to an enemy:* $\mathit{Footman1}.\mathit{My\_Enemy} = \mathit{Enemy1}$ *and* $\mathit{Footman2}.\mathit{My\_Enemy} = \mathit{Enemy2}$. *Each enemy will be linked to both footmen:* $\mathit{Enemy1}.\mathit{My\_Footmen} = \mathit{Enemy2}.\mathit{My\_Footmen} = \{\mathit{Footman1}, \mathit{Footman2}\}$. *The template, along with the number of objects and the links in this specific ("2vs2") world yield a well-defined factored MDP,* Π_{2vs2}, *as shown in Fig. 2(b).* ∎

3 Approximately Solving Relational MDPs

There are many approaches to solving MDPs [15]. An effective one is based on linear programming (LP): Let $\mathcal{S}(\Pi)$ denote the states in an MDP Π and $\mathcal{A}(\Pi)$ the actions. If $\mathcal{S}(\Pi) = \{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N\}$, our LP variables are $V_1, \ldots, V_N$, where V_i represents $\mathcal{V}(\boldsymbol{s}_i)$, the value of state $\boldsymbol{s}_i$. The LP formulation is:

$$\begin{array}{ll} \text{Minimize:} & \sum_i \alpha(\boldsymbol{s}_i) V_i \, ; \\ \text{Subject to:} & V_i \geq R(\boldsymbol{s}_i, \boldsymbol{a}) + \gamma \sum_k P(\boldsymbol{s}'_k \mid \boldsymbol{s}_i, \boldsymbol{a}) V_k \\ & \qquad\qquad \forall \boldsymbol{s}_i \in \mathcal{S}(\Pi), \boldsymbol{a} \in \mathcal{A}(\Pi). \end{array}$$

The *state relevance weights* $\alpha(\boldsymbol{s}_1), \ldots, \alpha(\boldsymbol{s}_N)$ in the objective function are any set of positive weights, $\alpha(\boldsymbol{s}_i) > 0$.

In our setting, the state space is exponentially large, with one state for each joint assignment to the random variables $o.\boldsymbol{S}$ of every object (*e.g.*, exponential in the number of units in the Freecraft scenario). In a multiagent problem, the number of actions is also exponential in the number of agents. Thus this LP has both an exponential number of variables and an exponential number of constraints. Therefore the exact solution to this linear program is infeasible.

We address this issue using the assumption that the value function can be well-approximated as a sum of *local value subfunctions* associated with the individual objects in the model. (This approximation is a special case of the factored linear value function approach used in [6].) Thus we associate a value subfunction $\mathcal{V}_o$ with every object in ω. Most simply, this local value function can depend only on the state of the individual object $\boldsymbol{S}_o$. In our example, the local value subfunction V_{Enemy1} for enemy object *Enemy1* might associate a numeric value for each assignment to the variable *Enemy1.Health*. A richer approximation might associate a value function with pairs, or even small subsets, of closely related objects. Thus, the

[1] A model where an action can change the link structure in the world requires a small extension of our basic representation. We omit details due to lack of space.

function $\mathcal{V}_{\textit{Footman1}}$ for *Footman1* might be defined over the joint assignments of *Footman1.Health* and *Enemy1.Health*, where *Footman1.My_Enemy* = *Enemy1*. We will represent the complete value function for a world as the sum of the local value subfunction for each individual object in this world. In our example world ($\omega = 2vs2$) with 2 footmen and 2 enemies, the global value function will be: $V_{2vs2}(\textit{F1.Health}, \textit{E1.Health}, \textit{F2.Health}, \textit{E2.Health}) = \mathcal{V}_{\textit{Footman1}}(\textit{F1.Health}, \textit{E1.Health}) + \mathcal{V}_{\textit{Enemy1}}(\textit{E1.Health}) + \mathcal{V}_{\textit{Footman2}}(\textit{F2.Health}, \textit{E2.Health}) + \mathcal{V}_{\textit{Enemy2}}(\textit{E2.Health})$.

Let $\boldsymbol{T}_o$ be the *scope* of the value subfunction of object o, *i.e.*, the state variables that $\mathcal{V}_o$ depends on. Given the local subfunctions, we approximate the global value function as:

$$\mathcal{V}_\omega(\boldsymbol{s}) = \sum_{o \in \mathcal{O}[\omega]} \mathcal{V}_o(\boldsymbol{s}[\boldsymbol{T}_o]). \tag{2}$$

As for any linear approximation to the value function, the LP approach can be adapted to use this value function representation [14]. Our LP variables are now the local components of the individual local value functions:

$$\{\mathcal{V}_o(\boldsymbol{t}_o) \,:\, o \in \mathcal{O}[\omega], \boldsymbol{t}_o \in \mathrm{Dom}[\boldsymbol{T}_o]\}. \tag{3}$$

In our example, there will be one LP variable for each joint assignment of *F1.Health* and *E1.Health* to represent the components of $\mathcal{V}_{\textit{Footman1}}$. Similar LP variables will be included for the components of $\mathcal{V}_{\textit{Footman2}}$, $\mathcal{V}_{\textit{Enemy1}}$, and $\mathcal{V}_{\textit{Enemy2}}$.

As before, we have a constraint for each global state $\boldsymbol{s}$ and each global action $\boldsymbol{a}$:

$$\begin{aligned}\textstyle\sum_o \mathcal{V}_o(\boldsymbol{s}[\boldsymbol{T}_o]) \geq \sum_o R^o(\boldsymbol{s}[\boldsymbol{S}_o], \boldsymbol{a}[o.A]) + \\ \gamma \textstyle\sum_{\boldsymbol{s}'} P_\omega(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}) \sum_o \mathcal{V}_o(\boldsymbol{s}'[\boldsymbol{T}_o]);\ \forall \boldsymbol{s}, \boldsymbol{a}.\end{aligned} \tag{4}$$

This transformation has the effect of reducing the number of free variables in the LP to n (the number of objects) times the number of parameters required to describe an object's local value function. However, we still have a constraint for each global state and action, an exponentially large number.

Guestrin, Koller and Parr [6] (GKP hereafter) show that, in certain cases, this exponentially large LP can be solved efficiently and exactly. In particular, this compact solution applies when the MDP is factored (*i.e.*, represented as a DBN), and the approximate value function is decomposed as a weighted linear combination of local basis functions, as above. Under these assumptions, GKP present a decomposition of the LP which grows exponentially only in the *induced tree width* of a graph determined by the complexity of the process dynamics and the locality of the basis function.

This approach applies very easily here. The structure of the DBN representing the process dynamics is highly factored, defined via local interactions between objects. Similarly, the value functions are local, involving only single objects or groups of closely related objects. Often, the induced width of the resulting graph in such problems is quite small, allowing the techniques of GKP to be applied efficiently.

4 Generalizing Value Functions

Although this approach provides us with a principled way of decomposing a high-dimensional value function in certain types of domains, it does not help us address the generalization problem: A local value function for objects in a world ω does not help us provide a value function for objects in other worlds, especially worlds with different sets of objects.

To obtain generalization, we build on the intuition that different objects in the same class behave similarly: they share the transition model and reward function. Although they differ in their interactions with other objects, their local contribution to the value function is often similar. For example, it may be reasonable to assume that different footmen have a similar long-term chance of killing enemies. Thus, we restrict our class of value functions by requiring that all of the objects in a given class share the same local value subfunction.

Formally, we define a *class-based local value subfunction* $\mathcal{V}_C$ for each class. We assume that the parameterization of this value function is well-defined for every object o in C. This assumption holds trivially if the scope of $\mathcal{V}_C$ is simply $\boldsymbol{S}_C$: we simply have a parameter for each assignment to $\mathrm{Dom}[\boldsymbol{S}_C]$. When the local value function can also depend on the states of neighboring objects, we must define the parameterization accordingly; for example, we might have a parameter for each possible joint state of a linked footman-enemy pair. Specifically rather than defining separate subfunctions $\mathcal{V}_{\textit{Footman1}}$ and $\mathcal{V}_{\textit{Footman2}}$, we define a class-based subfunction $\mathcal{V}_{\textit{Footman}}$. Now the contribution of *Footman1* to the global value function will be $\mathcal{V}_{\textit{Footman}}(\textit{F1.Health}, \textit{E1.Health})$. Similarly *Footman2* will contribute $\mathcal{V}_{\textit{Footman}}(\textit{F2.Health}, \textit{E2.Health})$.

A class-based value function defines a specific value function for each world ω, as the sum of the class-based local value functions for the objects in ω:

$$\mathcal{V}_\omega(\boldsymbol{s}) = \sum_{C \in \mathcal{C}} \sum_{o \in \mathcal{O}[\omega][C]} \mathcal{V}_C(\boldsymbol{s}[\boldsymbol{T}_o]). \tag{5}$$

This value function depends both on the set of objects in the world and (when local value functions can involve related objects) on the links between them. Importantly, although objects in the same class contribute the same function into the summation of (5), the argument of the function for an object is the state of that specific object (and perhaps its neighbors). In any given state, the contributions of different objects of the same class can differ. Thus, every footman has the same local value subfunction parameters, but a dead footman will have a lower contribution than one which is alive.

5 Finding Generalized MDP Solutions

With a class-level value function, we can easily generalize from one or more worlds to another one. To do so, we assume that a single set of local class-based value functions $\mathcal{V}_C$ is a good approximation across a wide range of worlds ω. Assuming we have such a set of value functions, we can act in any new world ω without replanning, as described in Step 3 of Fig. 3. We simply define a world-specific value function as in (5), and use it to act.

We must now optimize $\mathcal{V}_C$ in a way that maximizes the value over an entire set of worlds. To formalize this intuition, we assume that there is a probability distribution $P(\omega)$ over the worlds that the agent encounters. We want to find a single set of class-based local value functions $\{\mathcal{V}_C\}_{C \in \mathcal{C}}$ that is a good fit for this distribution over worlds. We view this task as one of optimizing for a single "meta-level" MDP Π^*, where nature first chooses a world ω, and the rest of the dynamics are then determined by the MDP Π_ω. Precisely, the state space of Π^* is $\{s_0\} \cup \bigcup_\omega \{(\omega, s) \;:\; s \in \mathcal{S}(\Pi_\omega)\}$. The transition model is the obvious one: From the initial state s_0, nature chooses a world ω according to $P(\omega)$, and an initial state in ω according to the initial starting distribution $P^0_\omega(s)$ over the states in ω. The remaining evolution is then done according to ω's dynamics. In our example, nature will choose the number of footmen and enemies, and define the links between them, which then yields a well-defined MDP,*e.g.*, Π_{2vs2}.

5.1 LP Formulation

The meta-MDP Π^* allows us to formalize the task of finding a generalized solution to an entire class of MDPs. Specifically, we wish to optimize the class-level parameters for $\mathcal{V}_C$, not for a single ground MDP Π_ω, but for the entire Π^*.

We can address this problem using a similar LP solution to the one we used for a single world in Sec. 3. The variables are simply parameters of the local class-level value subfunctions $\{\mathcal{V}_C(t_C) \;:\; C \in \mathcal{C}, t_C \in \mathrm{Dom}[T_C]\}$. For the constraints, recall that our object-based LP formulation in (4) had a constraint for each state s and each action vector $a = \{a_o\}_{o \in \mathcal{O}[\omega]}$. In the generalized solution, the state space is the union of the state spaces of all possible worlds. Our constraint set for Π^* will, therefore, be a union of constraint sets, one for each world ω, each with its own actions:

$$\begin{aligned}\mathcal{V}_\omega(s) \geq \textstyle\sum_o R^o(s[S_o], a_o) + \gamma \sum_{s'} P_\omega(s' \mid s, a)\mathcal{V}_\omega(s');\\ \forall \omega,\ \forall s \in \mathcal{S}(\Pi_\omega), a \in \mathcal{A}(\Pi_\omega);\end{aligned} \tag{6}$$

where the value function for a world, $\mathcal{V}_\omega(s)$, is defined at the class level as in Eq. (5). In principle, we should have an additional constraint for the state s_0. However, with a natural choice of state relevance weights α, this constraint is eliminated and the objective function becomes:

$$\text{Minimize:} \quad \frac{1+\gamma}{2} \sum_\omega \sum_{s \in S_\omega} P(\omega) P^0_\omega(s) \mathcal{V}_\omega(s); \tag{7}$$

if $P^0_\omega(s) > 0, \forall s$. In some models, the potential number of objects may be infinite, which could make the objective function unbounded. To prevent this problem, we assume that the $P(\omega)$ goes to zero sufficiently fast, as the number of objects tends to infinity. To understand this assumption, consider the following generative process for selecting worlds: first, the number of objects is chosen according to $P(\sharp)$; then, the classes and links of each object are chosen according to $P(\omega_\sharp \mid \sharp)$. Using this decomposition, we have that $P(\omega) = P(\sharp)P(\omega_\sharp \mid \sharp)$. The intuitive assumption described above can be formalized as: $\forall n, P(\sharp = n) \leq \kappa_\sharp e^{-\lambda_\sharp n}$; for some $\kappa_\sharp \geq \lambda_\sharp > 0$. Thus, the distribution $P(\sharp)$ over number of objects can be chosen arbitrarily, as long as it is bounded by some exponentially decaying function.

5.2 Sampling worlds

The main problem with this formulation is that the size of the LP — the size of the objective and the number of constraints — grows with the number of worlds, which, in most situations, grows exponentially with the number of possible objects, or may even be infinite. A practical approach to address this problem is to *sample* some reasonable number of worlds from the distribution $P(\omega)$, and then to solve the LP for these worlds only. The resulting class-based value function can then be used for worlds that were not sampled.

We will start by sampling a set $\mathcal{D}$ of m worlds according to $P(\omega)$. We can now define our LP in terms of the worlds in $\mathcal{D}$, rather than all possible worlds. For each world ω in $\mathcal{D}$, our LP will contain a set of constraints of the form presented in Eq. (4). Note that in all worlds these constraints share the variables $\mathcal{V}_C$, which represent our class-based value function. The complete LP is given by:

$$\begin{aligned}
&\text{Variables:} && \{\mathcal{V}_C(t_C) \;:\; C \in \mathcal{C}, t_C \in \mathrm{Dom}[T_C]\}.\\
&\text{Minimize:} && \tfrac{1+\gamma}{2m} \textstyle\sum_{\omega \in \mathcal{D}} \sum_{C \in \mathcal{C}} \sum_{o \in \mathcal{O}[\omega][C]}\\
&&& \qquad \textstyle\sum_{t_o \in T_o} P^0_\omega(t_o) \mathcal{V}_C(t_o).\\
&\text{Subject to:} && \textstyle\sum_{C \in \mathcal{C}} \sum_{o \in \mathcal{O}[\omega][C]} \mathcal{V}_C(s[T_o]) \geq\\
&&& \textstyle\sum_{o \in \mathcal{O}[\omega][C]} R^C(s[S_o], a[o.A]) +\\
&&& \gamma \textstyle\sum_{s'} P_\omega(s' \mid s, a) \sum_{C \in \mathcal{C}}\\
&&& \qquad \textstyle\sum_{o \in \mathcal{O}[\omega][C]} \mathcal{V}_C(s'[T_o]);\\
&&& \forall \omega \in \mathcal{D},\ \forall s \in \mathcal{S}(\Pi_\omega), a \in \mathcal{A}(\Pi_\omega);
\end{aligned} \tag{8}$$

where $P^0_\omega(T_o)$ is the marginalization of $P^0_\omega(S_o)$ to the variables in T_o. For each world, the constraints have the same form as the ones in Sec. 3. Thus, once we have sampled worlds, we can apply the same LP decomposition techniques of GKP to each world to solve this LP efficiently. Our generalization algorithm is summarized in Step 2 of Fig. 3.

The solution obtained by the LP with sampled worlds will, in general, not be equal to the one obtained if all worlds are considered simultaneously. However, we can show that the quality of the two approximations is close, if a sufficient number of worlds are sampled. Specifically, with a *polynomial* number of sampled worlds, we can guarantee that, with high probability, the quality of the value function approximation obtained when sampling worlds is close to the one obtained when considering all possible worlds.

Theorem 5.1 *Consider the following class-based value functions (each with k parameters): $\widehat{\mathcal{V}}$ obtained from the LP over all possible worlds by minimizing Eq. (7) subject to the constraints in Eq. (6); $\widetilde{\mathcal{V}}$ obtained from the LP with the sampled worlds in (8); and $\mathcal{V}^*$ the optimal value function of the meta-MDP Π^*. For a number of sampled worlds m polynomial in $(1/\varepsilon, \ln 1/\delta, 1/(1-\gamma), k, \lambda_\sharp, 1/\kappa_\sharp)$, the error is bounded by:*

$$\|\mathcal{V}^* - \widetilde{\mathcal{V}}\|_{1,P_\Omega} \leq \|\mathcal{V}^* - \widehat{\mathcal{V}}\|_{1,P_\Omega} + \varepsilon R_{max};$$

with probability at least $1-\delta$, for any $\delta > 0$ and $\varepsilon > 0$; where $\|\mathcal{V}\|_{1,P_\Omega} = \sum_{\omega, s \in S_\omega} P(\omega)P^0_\omega(s)\,|\mathcal{V}_\omega(s)|$, and R_{max} is the maximum per-object reward. ∎

The proof, which is omitted for lack of space (see online version of this paper), uses some of the techniques developed by de Farias and Van Roy [2] for analyzing constraint sampling in general MDPs. However, there are two important differences: First, our analysis includes the error introduced when sampling the objective, which in our case is a sum only over a subset of the worlds rather than over all of them as in the LP for the full meta-MDP. This issue was not previously addressed. Second, the algorithm of de Farias and Van Roy relies on the assumption that constraints are sampled according to some "ideal" distribution (the stationary distribution of the optimal policy). Unfortunately, sampling from this distribution is as difficult as computing a near-optimal policy. In our analysis, after each world is sampled, our algorithm exploits the factored structure in the model to represent the constraints exactly, avoiding the dependency on the "ideal" distribution.

6 Learning Classes of Objects

The definition of a class-based value function assumes that all objects in a class have the same local value function. In many cases, even objects in the same class might play different roles in the model, and therefore have a different impact on the overall value. For example, if only one peasant has the capability to build barracks, his status may have a greater impact. Distinctions of this type are not usually known in advance, but are learned by an agent as it gains experience with a domain and detects regularities.

We propose a procedure that takes exactly this approach: Assume that we have been presented with a set $\mathcal{D}$ of worlds ω. For each world ω, an approximate value function $\mathcal{V}_\omega = \sum_{o \in \mathcal{O}[\omega]} \mathcal{V}_o$ was computed as described in Sec. 3. In addition, each object is associated with a set of features $\mathcal{F}_\omega[o]$. For example, the features may include local information, such as whether the object is a peasant linked to a barrack, or not, as well as global information, such as whether this world contains archers in addition to footmen. We can define our "training data" $\mathcal{D}$ as $\{\langle \mathcal{F}_\omega[o], \mathcal{V}_o \rangle \ : \ o \in \mathcal{O}[\omega], \omega \in \mathcal{D}\}$.

We now have a well-defined learning problem: given this training data, we would like to partition the objects into classes, such that objects of the same class have similar value functions. There are many approaches for tackling such a task. We choose to use decision tree regression, so as to construct a tree that predicts the local value function parameters given the features. Thus, each split in the tree corresponds to a feature in $\mathcal{F}_\omega[o]$; each branch down the tree defines a subset of local value functions in $\mathcal{D}$ whose feature values are as defined by the path; the leaf at the end of the path is the average value function for this set. As the regression tree learning algorithm tries to construct a tree which is predictive about the local value function, it will aim to construct a tree where the mean at each leaf is very close to the training data assigned to that leaf. Thus, the leaves tend to correspond to objects whose local value functions are similar. We can thus take the leaves in the tree to define our subclasses, where each subclass is characterized by the combination of feature values specified by the path to the corresponding leaf. This algorithm is summarized in Step 1 of Fig. 3. Note that the mean subfunction at a leaf is not used as the value subfunction for the corresponding class; rather, the parameters of the value subfunction are optimized using the class-based LP in Step 2 of the algorithm.

1. **Learning Subclasses:**
   - **Input:**
     - A set of training worlds $\mathcal{D}$.
     - A set of features $\mathcal{F}_\omega[o]$.
   - **Algorithm:**
     (a) For each $\omega \in \mathcal{D}$, compute an object-based value function, as described in Sec. 3.
     (b) Apply regression tree learning on $\{\langle \mathcal{F}_\omega[o], \mathcal{V}_o \rangle \ : \ o \in \mathcal{O}[\omega], \omega \in \mathcal{D}\}$.
     (c) Define a subclass for each leaf, characterized by the feature vector associated with its path.
2. **Computing Class-Based Value Function:**
   - **Input:**
     - A set of (sub)class definitions $\mathcal{C}$.
     - A template for $\{\mathcal{V}_C : C \in \mathcal{C}\}$.
     - A set of training worlds $\mathcal{D}$.
   - **Algorithm:**
     (a) Compute the parameters for $\{V_C : C \in \mathcal{C}\}$ that optimize the LP in (8) relative to the worlds in $\mathcal{D}$.
3. **Acting in a New World:**
   - **Input:**
     - A set of local value functions $\{\mathcal{V}_C : C \in \mathcal{C}\}$.
     - A set of (sub)class definitions $\mathcal{C}$.
     - A world ω.
   - **Algorithm:** Repeat
     (a) Obtain the current state $\boldsymbol{s}$.
     (b) Determine the appropriate class C for each $o \in \mathcal{O}[\omega]$ according to its features.
     (c) Define $\mathcal{V}_\omega$ according to (5).
     (d) Use the coordination graph algorithm of GKP to compute an action $\boldsymbol{a}$ that maximizes $R(\boldsymbol{s}, \boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}'} P(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}) \mathcal{V}_\omega(\boldsymbol{s}')$.
     (e) Take action $\boldsymbol{a}$ in the world.

Figure 3: The overall generalization algorithm.

7 Experimental results

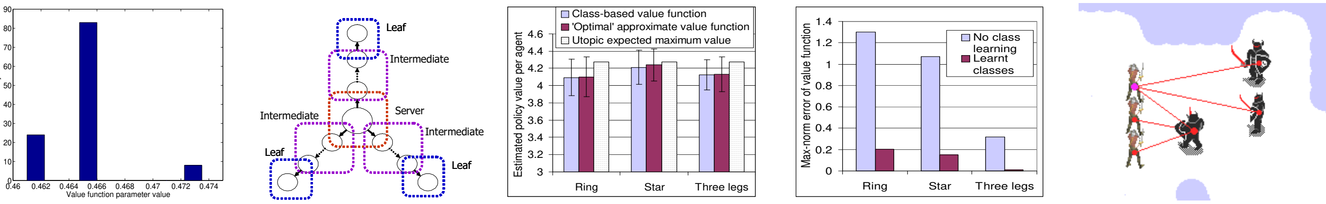
We evaluated our generalization algorithm on two domains: computer network administration and Freecraft.

7.1 Computer network administration

For this problem, we implemented our algorithm in Matlab, using CPLEX as the LP solver. Rather than using the full LP decomposition of GKP [6], we used the constraint generation extension proposed in [13], as the memory requirements were lower for this second approach. We experimented with the multiagent computer network examples in [6], using various network topologies and "pair" basis functions that involve states of neighboring machines (see [6]). In one of these problems, if we have n computers, then the underlying MDP has 9^n states and 2^n actions. However, the LP decomposition algorithm uses structure in the underlying factored model to solve such problems very efficiently [6].

We first tested the extent to which value functions are shared across objects. In Fig. 4(a), we plot the value each object gave to the assignment *Status* $=$ *working*, for instances of the 'three legs' topology. These values cluster into three classes. We used $CART^{®}$ to learn decision trees for our class partition. In this case, the learning algorithm partitioned the computers into three subclasses illustrated in Fig. 4(b): 'server', 'intermediate', and 'leaf'. In Fig. 4(a), we see that 'server' (third column) has the highest value, because a broken server can cause a chain reaction affecting the whole network, while 'leaf' value (first column) is lowest, as it cannot affect any other computer.

We then evaluated the generalization quality of our class-based value function by comparing its performance to that of planning specifically for a new environment. For each topology, we computed the class-based value function with 5 sampled networks of up to 20 computers. We then sampled a



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

Figure 4: Network administrator results: (a) Training data for learning classes; (b) Classes learned for 'three legs'; (c) Generalization quality (evaluated by 20 Monte Carlo runs of 100 steps); (d) Advantage of learning subclasses. Tactical Freecraft: (e) 3 footmen against 3 enemies.

new network and computed for it a value function that used the same factorization, but with no class restrictions. This value function has more parameters — different parameters for each object, rather than for entire classes, which are optimized for this particular network. This process was repeated for 8 sets of networks. The results, shown in Fig. 4(c), indicate that the value of the policy from the class-based value function is very close to the value of replanning, suggesting that we can generalize well to new problems. We also computed a utopic upper bound on the *expected* value of the optimal policy by removing the (negative) effect of the neighbors on the status of the machines. Although this bound is loose, our approximate policies still achieve a value close to it.

Next, we wanted to determine if our procedure for learning classes yields better approximations than the ones obtained from the default classes. Fig. 4(d) compares the max-norm error between our class-based value function and the one obtained by replanning. The graph suggests that, by learning classes using our decision trees regression tree procedure, we obtain a much better approximation of the value function we would have, had we replanned.

7.2 Freecraft

In order to evaluate our algorithm in the Freecraft game, we implemented the methods in C++ and used CPLEX as the LP solver. We created two tasks that evaluate two aspects of the game: long-term strategic decision making and local tactical battle maneuvers. Our Freecraft interface, and scenarios for these and other more complex tasks are publicly available at: `http://dags.stanford.edu/Freecraft/`. For each task we designed an RMDP model to represent the system, by consulting a "domain expert". After planning, our policies were evaluated on the actual game. To better visualize our results, we direct the reader to view videos of our policies at a website: `http://robotics.stanford.edu/~guestrin/Research/Generalization/`. This website also contains details on our RMDP model. It is important to note that, our policies were constructed relative to a very approximate model of the game, but evaluated against the real game.

In the tactical model, the goal is to take out an opposing enemy force with an equivalent number of units. At each time step, each footman decides which enemy to attack. The enemies are controlled using Freecraft's hand-built strategy. We modelled footmen and enemies as each having 5 "health points", which can decrease as units are attacked. We used a simple aggregator to represent the effect of multiple attackers. To encourage coordination, each footman is linked to a "buddy" in a ring structure. The local value functions include terms over triples of linked variables. We solved this model for a world with 3 footmen and 3 enemies, shown in Fig. 4(e). The resulting policy (which is fairly complex) demonstrates successful coordination between our footmen: initially all three footmen focus on one enemy. When the enemy becomes injured, one footman switches its target. Finally, when the enemy is very weak, only one footman continues to attack it, while the others tackle a different enemy. Using this policy, our footmen defeat the enemies in Freecraft.

The factors generated in our planning algorithm grow exponentially in the number of units, so planning in larger models is infeasible. Fortunately, when executing a policy, we instantiate the current state at every time step, and action selection is significantly faster [6]. Thus, even though we cannot execute Step 2 in Fig. 3 of our algorithm for larger scenarios, we can generalize our class-based value function to a world with 4 footmen and enemies, without replanning using only Step 3 of our approach. The policy continues to demonstrate successful coordination between footmen, and we again beat Freecraft's policy. However, as the number of units increases, the position of enemies becomes increasingly important. Currently, our model does not consider this feature, and in a world with 5 footmen and enemies, our policy loses to Freecraft in a close battle.

In the strategic model, the goal is to kill a strong enemy. The player starts with a few peasants, who can collect gold or wood, or attempt to build a barrack, which requires both gold and wood. All resources are consumed after each *Build* action. With a barrack and gold, the player can train a footman. The footmen can choose to attack the enemy. When attacked, the enemy loses "health points", but fights back and may kill the footmen. We solved a model with 2 peasants, 1 barrack, 2 footmen, and an enemy. Every peasant was related to a "central" peasant and every footman had a "buddy". The scope of our local value function included triples between related objects. The resulting policy is quite interesting: the peasants gather gold and wood to build a barrack, then gold to build a footman. Rather than attacking the enemy at once, this footman waits until a second footman is built. Then, they attack the enemy together. The stronger enemy is able to kill both footmen, but it becomes quite weak. When the next footman is trained, rather than waiting for a second one, it attacks the now weak enemy, and is able to kill him. Again, planning in large scenarios is infeasible, but action selection can be performed efficiently. Thus, we can use our generalized value function to tackle a world with 9 peasants and 3 footmen, without replanning. The 9 peasants coordinate to gather resources. Interestingly, rather than attacking with 2 footmen, the policy now waits for 3 to be trained before attacking. The 3 footmen kill the enemy, and only one of them dies. Thus, we have successfully generalized from a problem with about 10^6 joint state-action pairs to one with over 10^{13} pairs.

8 Discussion and Conclusions

In this paper, we have tackled a longstanding goal in planning research, the ability to generalize plans to new environments. Such generalization has two complementary uses: First we can tackle new environments with minimal or no replanning. Second it allows us to generalize plans from smaller tractable environments to significantly larger ones, which could not be solved directly with our planning algorithm. Our experimental results support the fact that our class-based value function generalizes well to new plans and that the class and subclass structure discovered by our learning procedure improves the quality of the approximation. Furthermore, we successfully demonstrated our methods on a real strategic computer game, which contains many characteristics present in real-world dynamic resource allocation problems.

Several other papers consider the generalization problem. Several approaches can represent value functions in general terms, but usually require it to be hand-constructed for the particular task. Others [12; 8; 4] have focused on reusing solutions from isomorphic regions of state space. By comparison, our method exploits similarities between objects evolving in parallel. It would be very interesting to combine these two types of decomposition. The work of Boutilier *et al.* [1] on symbolic value iteration computes first-order value functions, which generalize over objects. However, it focuses on computing exact value functions, which are unlikely to generalize to a different world. Furthermore, it relies on the use of theorem proving tools, which adds to the complexity of the approach. Methods in deterministic planning have focused on generalizing from compactly described policies learned from many domains to incrementally build a first-order policy [9; 11]. Closest in spirit to our approach is the recent work of Yoon *et al.* [17], which extends these approaches to stochastic domains. We perform a similar procedure to discover classes by finding structure in the value function. However, our approach finds regularities in compactly represented value functions rather than policies. Thus, we can tackle tasks such as multiagent planning, where the action space is exponentially large and compact policies often do not exist.

The key assumption in our method is interchangeability between objects of the same class. Our mechanism for learning subclasses allows us to deal with cases where objects in the domain can vary, but our generalizations will not be successful in very heterogeneous environments, where most objects have very different influences on the overall dynamics or rewards. Additionally, the efficiency of our LP solution algorithm depends on the connectivity of the underlying problem. In a domain with strong and constant interactions between many objects (*e.g.*, Robocup), or when the reward function depends arbitrarily on the state of many objects (*e.g.*, Blocksworld), the solution algorithm will probably not be efficient. In some cases, such as the Freecraft tactical domain, we can use generalization to scale up to larger problems. In others, we could combine our LP decomposition technique with constraint sampling [2] to address this high connectivity issue. In general, however, extending these techniques to highly connected problems is still an open problem. Finally, although we have successfully applied our class-value functions to new environments without replanning, there are domains where such direct application would not be sufficient to obtain a good solution. In such domains, our generalized value functions can provide a good initial policy, which could be refined using a variety of local search methods.

We have assumed that relations do not change over time. In many domains (*e.g.*, Blocksworld or Robocup), this assumption is false. In recent work, Guestrin *et al.* [7] show that *context-specific independence* can allow for dynamically changing coordination structures in multiagent environments. Similar ideas may allow us to tackle dynamically changing relational structures.

In summary, we believe that the class-based value functions methods presented here will significantly further the applicability of MDP models to large-scale real-world tasks.

Acknowledgements We are very grateful to Ron Parr for many useful discussions. This work was supported by the DoD MURI program, administered by the Office of Naval Research under Grant N00014-00-1-0637, and by Air Force contract F30602-00-2-0598 under DARPA's TASK program.

References

[1] C. Boutilier, R. Reiter, and B. Price. Symbolic dynamic programming for first-order MDPs. In *IJCAI-01*, 2001.

[2] D.P. de Farias and B. Van Roy. On constraint sampling for the linear programming approach to approximate dynamic programming. *Submitted to Math. of Operations Research*, 2001.

[3] T. Dean and K. Kanazawa. Probabilistic temporal reasoning. In *AAAI-88*, 1988.

[4] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[5] R. E. Fikes, P. E. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artf. Intel.*, 3(4):251–288, 1972.

[6] C. E. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored MDPs. In *NIPS-14*, 2001.

[7] C. E. Guestrin, S. Venkataraman, and D. Koller. Context specific multiagent coordination and planning with factored MDPs. In *AAAI-02* , 2002.

[8] M. Hauskrecht, N. Meuleau, L. Kaelbling, T. Dean, and C. Boutilier. Hierarchical solution of Markov decision processes using macro-actions. In *UAI*, 1998.

[9] R. Khardon. Learning action strategies for planning domains. *Artificial Intelligence*, 113:125–148, 1999.

[10] D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *AAAI*, 1998.

[11] M. Martin and H. Geffner. Learning generalized policies in planning using concept languages. In *KR*, 2000.

[12] R. Parr. Flexible decomposition algorithms for weakly coupled markov decision problems. In *UAI-98*, 1998.

[13] D. Schuurmans and R. Patrascu. Direct value-approximation for factored MDPs. In *NIPS-14*, 2001.

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

[15] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[16] S. Thrun and J. O'Sullivan. Discovering structure in multiple learning tasks: The TC algorithm. In *ICML-96*, 1996.

[17] S. W. Yoon, A. Fern, and B. Givan. Inductive policy selection for first-order MDPs. In *UAI-02*, 2002.