Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2006-022 March 20, 2006

Computing action equivalences for planning under time-constraints

Natalia H. Gardiol and Leslie Pack Kaelbling

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Computing action equivalences for planning under time-constraints

Natalia H. Gardiol
MIT Computer Science and Artificial Intelligence Lab
Cambridge, MA 02139
nhg@mit.edu

Leslie Pack Kaelbling
MIT Computer Science and Artificial Intelligence Lab
Cambridge, MA 02139
lpk@mit.edu

December 31, 2005

Abstract

In order for autonomous artificial decision-makers to solve realistic tasks, they need to deal with the dual problems of searching through large state and action spaces under time pressure. We study the problem of planning in domains with lots of objects. Structured representations of action can help provide guidance when the number of action choices and size of the state space is large. We show how structured representations of action effects can help us partition the action space in to a smaller set of approximate equivalence classes. Then, the pared-down action space can be used to identify a useful

subset of the state space in which to search for a solution. As computational resources permit, we then allow ourselves to elaborate the original solution. This kind of analysis allows us to collapse the action space and permits faster planning in much larger domains than before.

1 Introduction

In many logical planning domains, the crux of finding a solution often lies in overcoming an overwhelmingly large action space. Consider, as an illustration, the classic blocks world domain: the number of ways to make a stack of a certain height grows exponentially with the number of blocks on the table; and if the outcomes of actions are uncertain, this apparently simple task becomes daunting. We want planning techniques that can deal with large state spaces and large, stochastic action sets, since most compelling, realistic domains have these characteristics.

One way to describe large stochastic domains compactly is to use relational representations. Such a representation allows dynamics of the domain to be expressed in terms of object *properties* rather than object identities, and, thus, yields a much more compact representation of a domain than the equivalent propositional version can.

Even planning techniques that use relational representations, however often end up operating in a fully-ground state and action space when it comes time to find a solution, since such spaces are conceptually much simpler to handle. In this case, a key insight gives us leverage: often, several action instances produce similar effects. For example, in a blocks world it often does not matter which block is picked up first as long as a stack of blocks is produced in the end. If it were possible to identify under what conditions actions produce equivalent kinds of effects, the planning problem could be simplified by considering a representative action (from each equivalence class) rather

than the whole action space.

This work is about taking advantage of structured, relational action representations. First, we want to identify logically similar effects in order to reduce the effective size of the action space; second, we want to limit the state space in which we search for policies to an informative, reachable subset.

2 Relational Envelope-based Planning

Decision-making agents are often faced with complicated problems and not much time in which to find a solution. In such situations, the agent is better off acting quickly – finding *some* reasonable solution fast – than acting perfectly. Relational Envelope-based Planning (REBP) [11] is a planning approach designed for time-pressured decision-making problems.

REBP proceeds in two phases. First, given a planning problem, an initial plan of action is found quickly. Knowledge about the structure of action effects is used to eliminate potentially redundant information and focus the search onto high-probability sequences of actions, known as an *envelope* of states [7]. Second, if the agent is given additional time, it can elaborate the original plan by considering lower-probability consequences of its action choices. Figure 1 shows a very high-level system diagram of the main parts of the REBP system.

REBP lets us reason with ground states and actions, which have the advantage of conceptual simplicity, but it is designed to limit how much of the ground state and action space is considered at a time. REBP explicitly inhabits the space between a plan (a sequence of actions computed for a given state/goal pair) and a policy (a mapping from all states in a space to the appropriate action). This allows an agent to make a plan that hedges against the most likely deviations from the expected course of action, without requiring

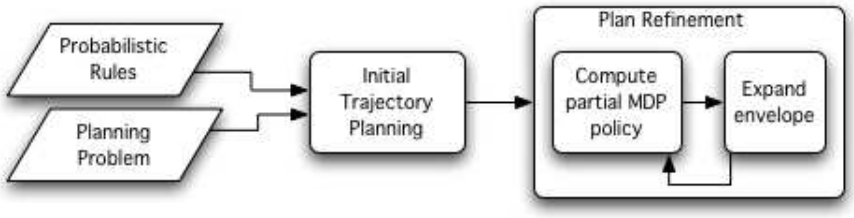

Figure 1: A high-level schematic of the REBP system. The input to the system is twofold: a set of probabilistic transition schemas. and a description of the planning problem at hand. The next process is to find an initial plan quickly. The final process is to refine the initial plan as resources permit.

construction of a complete policy. Then, as time permits, envelope-growing techniques [7, 11] are used to improve the partial policies incrementally.

A key step, however, is to produce the initial envelope efficiently. In our previous work [11], a useful insight was to partition the actions into equivalence classes; then, we search only over representative actions for each equivalence class instead of over the set of ground actions. If done properly, the resulting reduction in branching factor results in huge planning efficiency gains.

In the previous work, we used a heuristic method to partition the actions into classes. In this paper, we provide a formal basis for computing action equivalence classes.

2.1 Relational representation of actions and states

We cast our planning problem in the framework of a Markov decision process (MDP) [17]. An MDP is a tuple, $M = \langle \mathcal{S}, \mathcal{A}, T, R \rangle$ where: $\mathcal{S}$ is a set of states; $\mathcal{A}$ is a set of actions, where each action a maps a state to a distribution over next states; R is a reward function mapping each state to a real number; and $\mathcal{T}$ is a transition function mapping each state-action-state triple to a

probability. A *solution* for an MDP is a mapping from states to actions that maximizes long-term reward. This function, π, is called a *policy*.

In the past, much work on finding policies for MDPs considered a state to be an atomic entity; this approach is known not to scale to large state spaces, and much recent literature has been devoted to smarter ways of representing problems.

In this work, we take advantage of a more compact way of representing state transitions. Rather than viewing a state as a set of propositional features, we think of it as a set of logical relationships between domain objects. Since these relations can make assertions about logical *variables*, a single relation may in fact represent a large number of ground propositions. This lets us use a single logical schema to represent many ground state transitions.

We define a *relational* MDP *(*RMDP*)* as a tuple $M' = \langle \mathcal{P}, \mathcal{O}, \mathcal{Z}, \mathcal{T}, R \rangle$. $\mathcal{P}$ is a set of logical predicates, denoting the properties and relations that can hold among the finite set of domain objects, $\mathcal{O}$. $\mathcal{Z}$ is the set of transition schemas in a subset of the PPDDL language[1] [20]. $\mathcal{T}$ is a set of object *types*. R is a scalar reward function.

An RMDP M' induces a fully-ground MDP M as follows:

States: The set of ground states $\mathcal{S}$ is determined by $\mathcal{P}$ and $\mathcal{O}$. In other words, each ground state s is an application of the domain predicates over the domain objects; any relation that is not known to be true is assumed to be false. We treat a ground state as a conjunction of positive and/or negative ground terms.

Actions: The set of ground actions is determined by $\mathcal{Z}$ and $\mathcal{O}$. A rule z *applies* in a state s if its precondition can be logically unified against some subset of the state's ground relations. This unification yields a set of ground

[1]We do not consider conditional outcomes.

```
(:action pick-up-block-from
 :parameters (?top - block ?bot - object)
 :precondition
   (and (on-top-of ?top ?bot) (not (= ?top ?bot)) (on-top-of ?top ?bot)
        (forall (?b - block) (not (holding ?b)))
        (forall (?b - block) (not (on-top-of ?b ?top))))
 :effect (probabilistic
   0.9 (and (holding ?top) (not (on-top-of ?top ?bot)))
   0.1 (and (forall (?b - block)) (not (on-top-of ?b ?top))
                                  (on-top-of ?top table))))

(:action put-down-block-on
 :parameters (?top - block ?bottom - object)
 :precondition
   (and (not (= ?top ?bottom))
        (holding ?top)
        (not (holding ?bottom))
        (or (= ?bottom table)
            (and (on-top-of ?bottom ?any)
                 (forall (?b - block) (not (on-top-of ?b ?bottom))))))
 :effect (and
             (not (holding ?top))
             (probabilistic 0.75 (on-top-of ?top ?bottom)
                            0.25 (on-top-of ?top table))))
```

Figure 2: Probabilistic relational schema for blocks-world dynamics in the PPDDL formalism. Each rule schema contains the action name, arguments, precondition, and a set of outcomes.

actions $z|_s$, each element of which an instantiation of the rule schema over the objects in state s.

Transition Dynamics: We assume only one rule can apply per state transition, so the distribution over next states is given compactly by the distribution over outcomes encoded in the rule schema. When a rule is applied to a state s, the ground action $a \in z|_s$ maps s to an outcome which specifies the set of changes: the set of facts that are now true because of a, $add(a)$, and the set of things that are no longer true because of a, $del(a)$. However, an outcome only describes the changes to a subset of the domain relations; Thus, to compute the complete successor state of the i^{th} outcome, $\gamma_i(a, s)$, we assume a static frame. The state relations not directly changed by a are assumed to remain the same as in s.

Rewards: A ground state is mapped to a scalar reward according to $R(s)$.

The original version of envelope-based planning [7] used an atomic MDP representation. To extend envelope-based planning to relational domains, then, we need two things. First, we need a set of probabilistic rule schemas, which tell us the transition dynamics for a domain; and second, we need a problem description, which tells us the states and reward. This is the structure shown in Figure 1.

Rules may be designed by hand or obtained via learning [21]. Figure 2 shows an example rule.

A to define a planning problem we have to specify the following elements. The *initial world state* s_0 is a set of ground relations that are true in the starting state; relations that aren't stated are assumed to be false. The *goal condition* g is a logical sentence; planning terminates upon successful achievement of the goal. The *reward* is specified by list of logical conditions mapping states to a scalar reward value. If a state in the current MDP does not match a reward condition, the default value is 0. Additionally, we must specify how

to consider states that are out of the envelope. In our case, we assign them a penalty that is an estimate of the cost of having to recover from falling out (such as having to re-plan back to the envelope, for example).

We adopt the following notational conventions. We identify each relation p by the predicate's name (designated $name(p)$) and the number and type of its arguments. For example, the relation $\mathsf{on}/2$ is fully specified as $\mathsf{on}(Block, Surface)$. A domain element, or object, $o \in \mathcal{O}$ is uniquely identified by its name ($name(o)$). The function $type(o)$ returns the type of the object o. A type $t \in \mathcal{T}$ has a set of more specific types, $subtypes(t)$ (which may be empty), and a set of more general types, $supertypes(t)$ (also possibly empty). Each domain element and each variable in the argument list of a relation or rule schema must be assigned a type. A domain element o can only be substituted for a variable v if $type(o) \in subtypes(v)$. We also adopt the following shorthand: if a ground relation $p(o_1, ..., o_n)$ appears in s, we say $p(o_1, ..., o_n) \in \mathcal{P}_s$; if a domain object o_1 appears in a relation in s, we say $o \in \mathcal{O}_s$;

2.2 Initial trajectory planning

Given a set of schemas and the problem description, the first step in envelope-based planning is finding the initial envelope. In a relational setting, when the underlying MDP space implied by the full instantiation of the representation is potentially huge, a good initial envelope is crucial. It determines the quality of the early envelope policies and sets the stage for more elaborate policies later on.

Blum and Langford [3] describe a probabilistic extension to the Graphplan algorithm [2], called TGraphplan (TGP), that can quickly find the shortest straight-line plan from start to goal that satisfies a minimum probability. We use the trajectory found by TGP to populate our initial envelope.

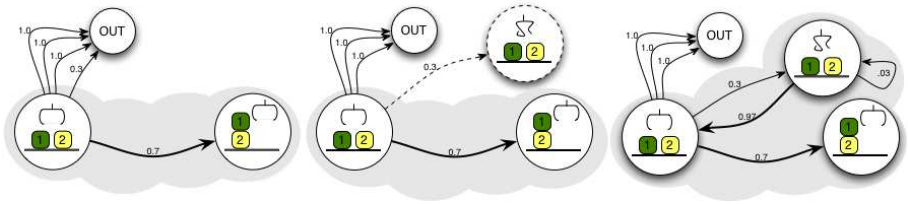

Figure 3: An illustration of the envelope-based approach to planning. The task is to making a two-block stack in a domain with two blocks. The initial envelope is shown (far left), followed by a step of deliberation (i.e., consideration of deviations from the initial plan) (middle), and finally after envelope expansion and computation of a new MDP policy (far right).

Initial plan construction essentially follows the TGP algorithm described by Blum and Langford [3]. The TGP algorithm starts with the initial world state as the first layer in the graph, a probability threshold for the plan, and a maximum plan depth. The output of the TGP algorithm is a sequence of actions that achieves the goal with probability higher than the threshold.

We run into difficulties, however, when the number of domain elements is large. The relational MDP describes a large underlying MDP and when a schema is grounded, it generates a number of actions exponential in the number of domain objects. Large numbers of actions mean a huge branching factor in the plan graph, which grinds TGP to a near halt. Thus, we want to avoid considering *all* the actions during our plan search.

To cope with this problem, we have identified a technique called *equivalence-class sampling*. We partition into equivalence classes the actions that produce the same effects on the properties of the variables in their scope. Then, the plan graph can be constructed by chaining forward only a sampled action from each class. The sampled action is *representative* of the effects of any action from that class. Sampling reduces the branching factor at each step in the plan graph, so significantly larger domains can be handled.

2.3 Equivalence in relational domains

Before we proceed to define equivalence between objects, however, we make the following crucial assumption:

Assumption 2.1 (Sufficiency of Object Properties)**.** *We assume a domain object's function is determined solely by its properties and relations to other objects, and not by its name.*[2]

So, for example, consider a blocks world in which the only two properties are the relation on() and the attribute color(). Then if two blocks block14 and block37 are both red, are both on the table, and have nothing on them, they would be considered functionally equivalent. If block37 had another block on top of it, however, it would not be equivalent to block14.

Intuitively, we mean to say that two objects are equivalent to each other if they are related in the same way to other objects that are, in turn, equivalent.

Evaluating equivalence is tricky, of course, because computing whether two objects are equivalent requires looking at the objects that they are related to; we have to "push" through each relation an object participates in. Previous work on object equivalence, or symmetry, has used single, unary relations as a basis for computing similarity [8, 9, 10].

We want to study object equivalence when more complex relationships are present. To generalize our concept of equivalence, we view a relational state description as a graph. The nodes in the graph correspond to objects in the domain, and the binary relations between the objects correspond to the edges. For each pair of related nodes, we construct an edge representing

[2]What if we are in a setting in which a few objects' identities are in fact necessary? One could encode this information via supplementary properties, by adding a relation such as block14(X) that would only be true for block14. Obviously, if identity matters for a large number of objects, the approach described here would not be suitable.

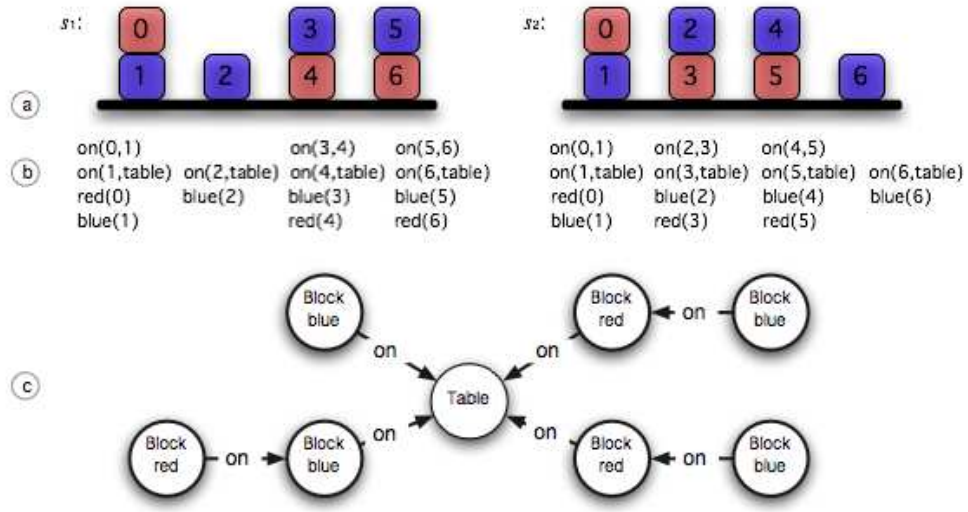

Figure 4: Two blocks-world states depicted pictorially (a), and as a set of ground relations (b). The corresponding state relation graph is constructed by labeling each edge with the predicate's name and each node with the object's type and unary properties. In (c), we see the state relation graph. Since it is the same for both blocks-world states, they are *equivalent*.

the relation. In addition, nodes and edges are *labeled* with a string (or set of strings). Each node is labeled with the object's type, and each edge is labeled with the predicate name. If an object also participates in a unary relation, we augment its label set with that predicate's name. [3]

Definition 2.2 (State Relation Graph)**.** A *state relation graph* $\mathcal{G}$ for a state s is a labeled, directed graph. The set $V(\mathcal{G})$ of vertices is:

$$V(\mathcal{G}) = \{v_o | o \in \mathcal{O}_s\}, \text{ where}$$

$$\forall v_o.\ label(v_o) = \{type(o)\} \cup \{name(p) | \exists p(o) \in \mathcal{P}_s\}.$$

[3] At present, we consider up to binary relations. In the case of relations with more than two arguments, we would have to consider a hypergraph representation to allow for edges of more than two nodes — a non-trivial extension.

The set $E(\mathcal{G})$ of edges is:

$$E = \{(v_{o_1}, v_{o_2}, p) | o_1, o_2 \in \mathcal{O}_s, p(o_1, o_2) \in \mathcal{P}_s\}.$$

□

An isomorphism between two labeled graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ is a bijective function, $\Phi : V(\mathcal{G}_1) \rightarrow V(\mathcal{G}_2)$, such that $\forall v \in V(G), label(v) = label(\Phi(v))$, and $(v_{o_1}, v_{o_2}, p) \in E(\mathcal{G}_1)$ if and only if $(\Phi(v_{o_1})\Phi(v_{o_2}), p) \in E(\mathcal{G}_2)$.

Now we extend the meaning of Φ, which is in principle a mapping between nodes, to allow us to map graphs, sentences, and actions. First, let us define $\Phi(\mathcal{G}_1) = \mathcal{G}_2$, the graph that results from applying Φ to all the vertices of $\mathcal{G}_1$:

$$V(\mathcal{G}_2) = \{\Phi(v_o) | v \in \mathcal{G}_1\}, \text{and}$$

$$E(\mathcal{G}_2) = \{(\Phi((v_{o_1}), \Phi(v_{o_2}), p)) | (v_{o_1}, v_{o_2}, p) \in \mathcal{G}_1\}.$$

Second, we define $\Phi(\sigma_1) = \sigma_2$, the sentence 0that results from applying Φ to all the objects of ground sentence σ_1:

σ_2 =

$p(\Phi(o_1), \ldots, \Phi(o_n))$, if σ_1 is an n-ary relation $p(o_1, \ldots, o_n)$, or

$\neg\Phi(\rho)$, if σ_1 is a negation $\neg\rho$, or

$\vee_i \Phi(\rho_i)$, if $\Phi(\sigma_1)$ is a conjunction $\vee_i \rho_i$, or

$\wedge_i \Phi(\rho_i)$, if $\Phi(\sigma_1)$ is a disjunction $\wedge_i \rho_i$, or

And finally, we extend Φ to actions. This follows easily from the above case, since the precondition and all outcomes of an action are ground sentences. We define $\Phi(a_1) = a_2$ to refer to the re-written action that results from applying Φ to the action a_1:

$$precondition(a_2) = \Phi(precondition(a_1)), \text{ and}$$

$$\forall i \ outcome_i(a_2) = \Phi(outcome_i(a_1))$$

Definition 2.3 (State equivalence)**.** Two *states* are *equivalent*, written $s_1 \sim s_2$, if there exists an isomorphism, Φ, between the respective state relation graphs such that $\Phi(\mathcal{G}_{s_1}) = \mathcal{G}_{s_2}$.

Definition 2.4 (Action Equivalence)**.** The applications of action schema z in states s_1 and s_2 yield the sets of ground actions $z|_{s_1}$ and $z|_{s_2}$. Two ground actions $a_1 \in z|_{s_1}$ and $a_2 \in z|_{s_2}$ are equivalent if and only if there exists a Φ such that $\Phi(\mathcal{G}_{s_1}) = \mathcal{G}_{s_2}$ and $\Phi(a_1) = a_2$.

Figure 5 contains an example of computing the equivalence between pairs of ground actions that result from applying a rule schema z in a state s.[4]

To show that the relations defined in Definitions 2.3 and 2.4 are equivalence relations, we have to show that they are reflexive, symmetric, and transitive.

Lemma 2.5. *Relation-graph isomorphism defines an equivalence relation on states and actions.*

Proof. First, a state s produces a unique relation graph $\mathcal{G}_s$, and there always exists the identity mapping from $\mathcal{G}_s$ to $\mathcal{G}_s$, so we conclude $s \sim s$. Next, if $s_1 \sim s_2$, then there exists Φ such that $\Phi(\mathcal{G}_{s_1}) = \mathcal{G}_{s_2}$. Since Φ is bijective, it has an inverse, $\Phi^{-1}(\mathcal{G}_{s_2}) = \mathcal{G}_{s_1}$, and so we conclude $s_2 \sim s_1$. Finally, if $s_1 \sim s_2$ and $s_2 \sim s_3$, then there exist Φ_1 such that $\Phi_1(\mathcal{G}_{s_1}) = \mathcal{G}_{s_2}$ and Φ_2 such that $\Phi_2(\mathcal{G}_{s_2}) = \mathcal{G}_{s_3}$. Thus $\Phi_2(\Phi_1(\mathcal{G}_{s_1})) = \mathcal{G}_{s_3}$, which implies $s_1 \sim s_3$. The argument for actions is analogous. □

Since the relation $\sim$ is an equivalence relation, we denote the equivalence class of item x as $[x]$.

Next, we see that if a logical sentence is satisfied in a state s, then it can be satisfied in any state $\tilde{s} \in [s]$. We must be clear about the setting: we assume that a non-ground sentence contains *no* constants, and that a ground state is a fully ground list of facts (which we can treat as a conjunction or set

[4]Not that in this case, the Identity mapping always exists. Since it provides no useful information, however, it is omitted for simplicity.

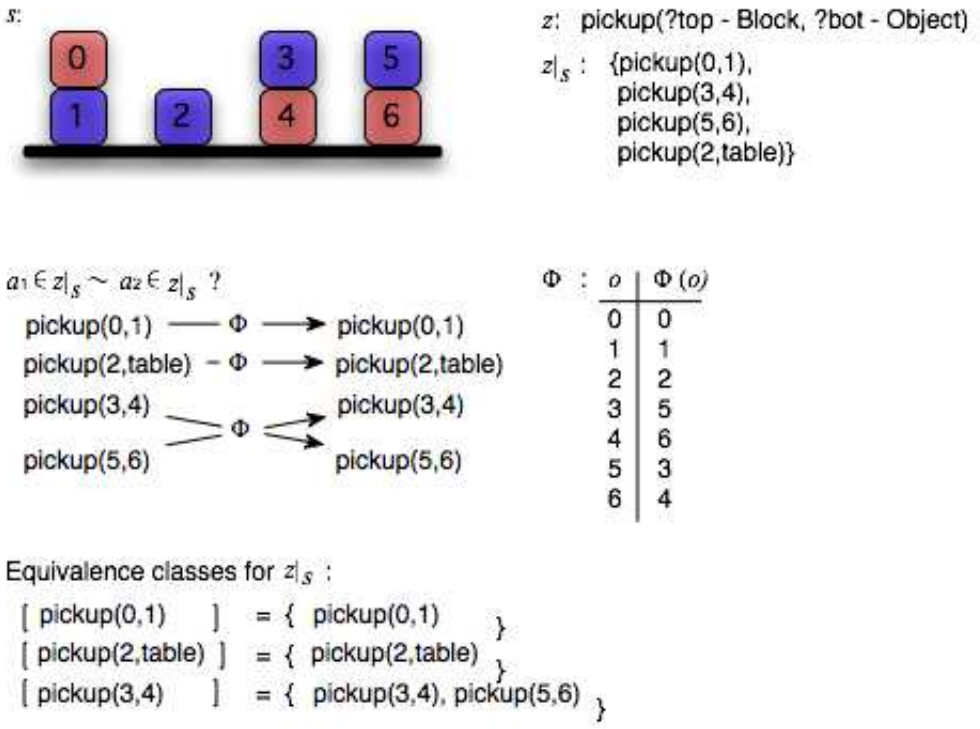

Figure 5: In the top row of the figure, we see a state s, rule schema z, and the set of ground actions that results from applying z in s. The middle row shows the computation of equivalence classes: for each pair of ground actions a_1 and a_2, we compute whether there exists an isomorphism such that $\Phi(s) = s$ and $\Phi(a_1) = a_2$. The isomorphism Φ, at right, exists. In the bottom row of the figure, we see the three equivalence classes that were computed.

of ground relations). When we say that a state entails a sentence, we are speaking purely of syntactic entailment.

Lemma 2.6. *If a sentence σ is entailed by a state s, then it is entailed by any $\tilde{s} \in [s]$*

Proof. Let $\Phi(s) = \tilde{s}$. If sentence σ is entailed by s, then there is some substitution ψ for the variables in σ such that $\psi(\sigma)$ is a subset of s. In other words, $s \models \sigma$ if and only if $\exists\psi.\psi(\sigma) \subseteq s$. Assume σ is entailed by s, and let $\psi(\sigma) \subseteq s$. We know by equivalence of s and $\tilde{s}$ that:

$$\begin{aligned}
\Phi'(\tilde{s}) &= s. \\
\text{So, } \psi(\sigma) &\subseteq \Phi'(\tilde{s}), \\
\text{and, } (\Phi')^{-1}(\psi(\sigma)) &\subseteq \tilde{s}. \\
\text{Let } \Phi'' &= (\Phi')^{-1} \circ \psi. \\
\text{Then, } \Phi''(\sigma) &\subseteq \tilde{s}, \text{ and, thus} \\
\tilde{s} &\models \sigma.
\end{aligned}$$

□

The next establishes the equivalence of the states produced by taking equivalent ground actions in equivalent states.

Lemma 2.7. *If two actions $a_1 \in z|_{s_1}$ and $a_2 \in z|_{s_2}$ are equivalent, then the successor states determined by their respective outcomes are equivalent.*

Proof. By definition, for a given outcome i of z, $\gamma_i(a_1, s_1) = s_1 \cup add_i(a_1) \setminus del_i(a_1)$, so:

$$\begin{aligned}
\Phi(\gamma_i(a_1, s_1)) &= \Phi(s_1 \cup add_i(a_1) \setminus del_i(a_1)) \\
&= \Phi(s_1) \cup \Phi(add_i(a_1)) \setminus \Phi(del_i(a_1)) \\
&= s_2 \cup add_i(a_2) \setminus del_i(a_2) \\
&= \gamma_i(a_2, s_2) \\
\text{thus, } \gamma_i(a_1, s_1) &\sim \gamma_i(a_2, s_2)
\end{aligned}$$

□

In the case of deterministic actions, a solution plan is said to exist if there is a sequence of actions that leads from the starting state to the goal. In the case of stochastic actions, however, we have no control over the actual outcome; we only know the distributions over outcomes. What does it mean for a solution straight-line plan to exist in this case? Since a straight-line plan considers only a single outcome at each step, we designate the *anticipated* outcome to be the one expected by the planning procedure; usually, this is just the most likely outcome. Thus, in this case, a plan exists if there is a sequence of actions whose expected outcomes yield a state sequence that leads to the goal. We accept as a goal state any state that entails the goal conditions g.

Now we almost have all the pieces to state the main theorem. We know that equivalent schema applications produce equivalent successor states. Now, we must show that a sequence of schema applications can be replaced by an equivalent sequence to produce equivalent ending states.

Definition 2.8 (Equivalent Planning Procedures)**.** Let P be a planning procedure such at at each state s, P selects an action a. Consider a planning procedure P' such that at each state $\tilde{s} \sim s$, P' chooses an action $\tilde{a} \sim a$. Then P and P' are defined to be *equivalent planning procedures.*

Theorem 2.9. *Let P be a complete planning procedure.*[5]*. Any planning procedure P'* equivalent *to P is also a complete planning procedure. That is,*

$$\gamma(a_1, \ldots, a_n, s_0) \rightarrow g \Rightarrow \gamma(\tilde{a_1}, \ldots, \tilde{a_n}, s_0) \rightarrow g$$

Proof. We prove the theorem by induction. First, consider the initial step. If a_1 in s_0 is equivalent to $\tilde{a_1}$ in s_0, then $\gamma(a_1, s_0) \sim \gamma(\tilde{a_1}, s_0)$ (by Lemma 2.7). Next, we need to show that if a_{i+1} in $\gamma(a_1, \ldots, a_i, s_0)$ is equivalent to $\tilde{a}_{i+1}$ in $\gamma(\tilde{a_1}, \ldots, \tilde{a_i}, s_0)$, then $\gamma(a_1, \ldots, a_{i+1}, s_0) \sim \gamma(\tilde{a_1}, \ldots, \tilde{a}_{i+1}, s_0)$.

[5]A *complete* planning procedure is one which is guaranteed to find a path to the goal if one exists.

Again, Lemma 2.7 guarantees that

$$\begin{aligned} \gamma(a_{i+1}, \gamma(a_1, \ldots, a_i, s_0)) &\sim \gamma(\tilde{a}_{i+1}, \gamma(\tilde{a_1}, \ldots, \tilde{a_i}, s_0)), \text{thus,} \\ \gamma(a_1, \ldots, a_{i+1}, s_0)) &\sim \gamma(\tilde{a_1}, \ldots, \tilde{a}_{i+1}, s_0)). \\ \text{Hence, } \gamma(a_1, \ldots, a_n, s_0) &\sim \gamma(\tilde{a_1}, \ldots, \tilde{a_n}, s_0), \end{aligned}$$

and by Lemma 2.6, $\gamma(\tilde{a_1}, \ldots, \tilde{a_n}, s_0) \rightarrow g$. □

Thus, any serial plan that existed before in the full action space will have an equivalent version in the new, collapsed action space.

Planning in the reduced action space consisting of representatives from each equivalence class preserves completeness. It does, however, have an effect on plan parallelism. Since we are limited to only one action of each class on each step, a planning procedure that might have used two instances of the same class in parallel would have to serialize them.

2.4 Planning with equivalent actions in TGraphplan

Now we want to use this notion of action equivalence in the REBP planning framework.

When we apply this definition into the TGraphplan setting, however, the following issue arises: in each layer of the plan graph, there is no notion of "current state." In the Graphplan algorithm, the first level in the graph contains the propositions corresponding to the facts in the initial state. Each level beyond the first contains two layers: one layer for all the actions that could possibly be enabled based on the propositions on the previous level, and a layer for all of the possible effects of those actions. Thus, each level of the plan graph simply contains a list of all propositions that could possibly be true. The only information at our disposal is that of which propositions are mutually exclusive from one another.

In order to partition actions into equivalence classes, we adopt the following criterion. We define the *extended state* of an action to be all those propositions in the current layer that are not mutually exclusive with any of the action's preconditions. Thus, we group two actions together if the ground objects in each argument list are isomorphic to each other with respect to each action's *extended state*.

Computing equivalence based on extended states will create a finer set of equivalence classes as compared to ground states: the set of propositions that could be possibly true is greater than or equal to the set that will actually become true. Thus, the equivalence classes produced by this criterion will be at least as fine as those produced by Theorem 2.9.

3 Experimental Validation

As a check, we did a small study to illustrate the computational savings of planning with equivalence class sampling. Figures 6, 7, and 8 show these results. The experiments were done in the ICAPS 2004 blocks-world domain, varying the number of blocks and the number of colors. In each case, the goal was to stack all of the blocks, and the starting state was with all blocks on the table. The x-axis of the graphs is the time (in milliseconds) spent planning. This was measured using a monitoring package, which measures only the time spent in computing the plan, not just elapsed CPU time. Each data point shows the time at which a plan graph layer was added, and the number of actions added to that layer.

Figure 6 shows the results for the two-block domain. The correct plan in this case is two actions: first is a pick-up action, and second is a put-down action to create a stack with the first block on top of the second. In the case where there is only one color, REBP is able to identify the equivalence of

both pick-up actions in the first step, and so it only adds one planning action to the plan graph in that step. In the case where there are two colors, both actions belong to different classes, so there is no difference between sampling from the classes or using all of the actions. In each case, however, the cost of computing the classes is higher.

In Figures 7 and 8, the number of blocks increases. With just four blocks in the domain, the branching factor when using all actions is already such that the computational savings of computing the action classes is significant.

Some notes: In Figure 8 it is not clear why the domain with a single color should take the all-actions search so much longer than in the case with two colors. This may be an artifact of the search ordering in TGraphplan. Also, in Figures 7 and 8, the dip in the number of actions added after steps 3 and 4 (respectively) of the sampling search is due to the fact that, as the plan graph is extended, the mutex relationships between the propositions tend to disappear. This reduces the number of distinctions between the actions and enables more of them to be in the same equivalence class.

4 Conclusion

Our objective, ultimately, is to plan in large domains in which there is time-pressure to begin acting appropriately. To this end, we seek to take advantage of an envelope-based planning framework, which explicitly considers low-complexity plans first, and higher-complexity plans as time permits. The difficulty of envelope-based approaches, however, is always this: what is the best way to populate the initial envelope?

If our domain is represented relationally, then it makes sense to leverage the fast classical planning techniques for finding straight-line plans, such as Graphplan. However, when our domains have many objects in them (which

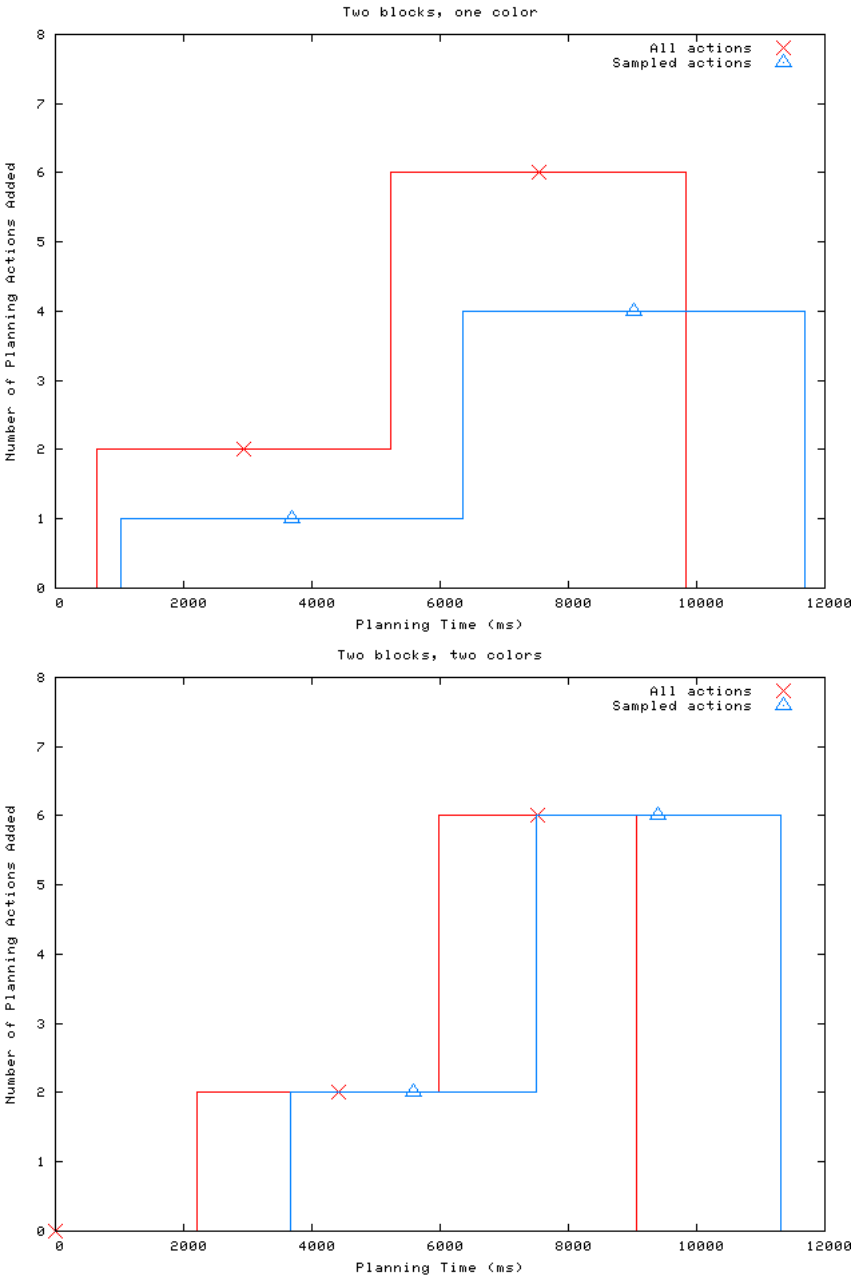

Figure 6: Planning with equivalence-class sampling vs. with all actions in domains with two blocks and one color (top), and two colors (bottom).

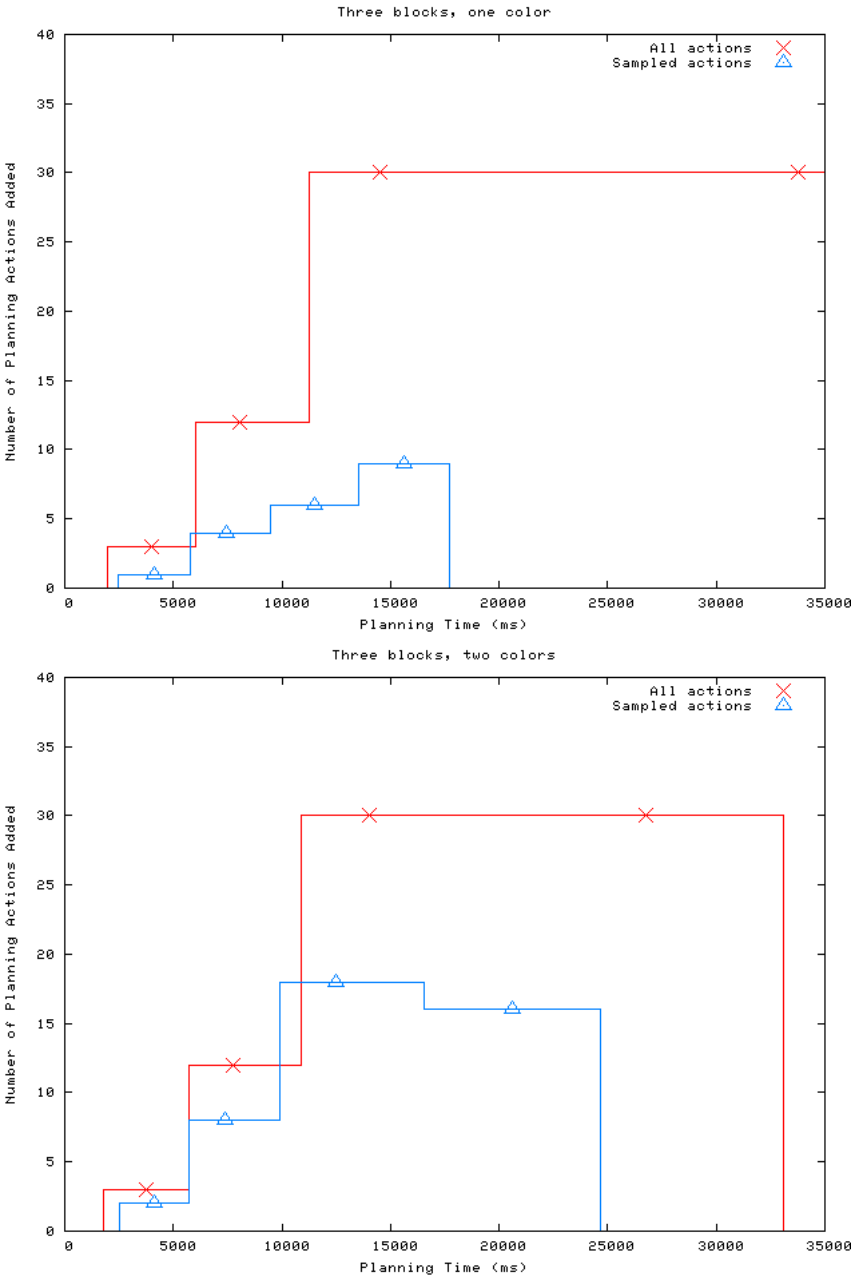

Figure 7: Planning with equivalence-class sampling vs. with all actions in domains with three blocks and one color (top), and two colors (bottom).

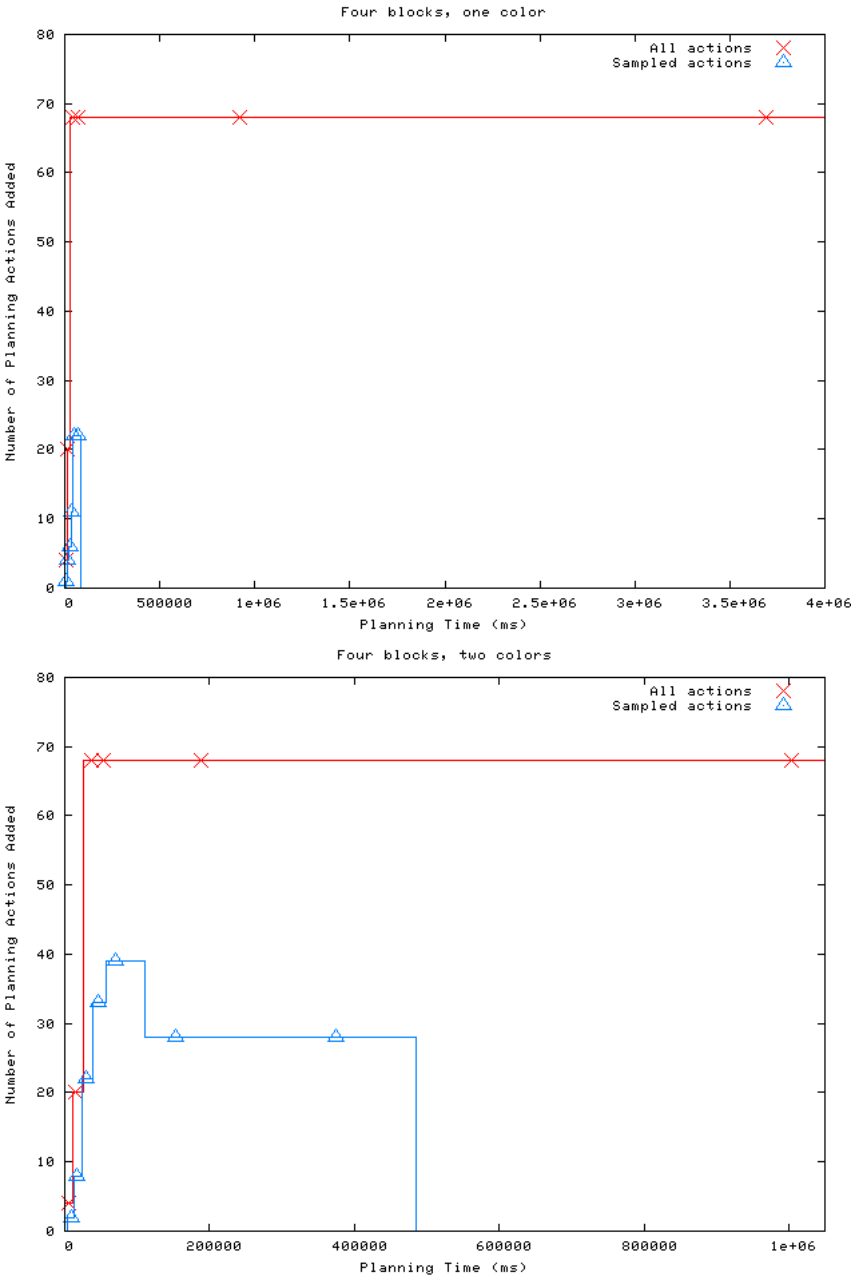

Figure 8: Planning with equivalence-class sampling vs. with all actions in domains with four blocks and one color (top), and two colors (bottom).

is precisely the setting we want to consider), the number of ground action instantiations creates an crippling branching factor for Graphplan and descendent algorithms. They are unable to find the straight-line plan we need in order to create an initial envelope.

Thus, for envelope-based approaches to scale, it is crucial to find a way to prune out action instantiations that do not achieve qualitatively different outcomes.

There is a host of preceding approaches that attempt to identify symmetries in the problem to collapse actions together [8, 9, 10]. Unfortunately, in the existing work in this area there is a rather weak notion of what makes objects equivalent. These notions rely primarily on unary relations, and they are hard to reconcile with a relational domain in which relations have more than one argument.

This is the first work that we know of that explicitly attempts to define what it means for planning operators to be equivalent in the presence of complex relational structure. This is work is an initial attempt at formalizing such a definition and to apply it within the context of envelope-based planning.

4.1 Related Work

Techniques for planning in large state spaces need to employ some way of restricting the search space. There are a few main ways that this is done in the literature.

First, one can use a heuristic to guide search. This group includes heuristic search methods [1, 16, 4] in the context of MDPs. Heuristic planning approaches seek to avoid the evaluation of a whole, potentially large, state space by using a heuristic estimate. However, computing a good heuristic can often be rather expensive, and it may not always be obvious how to

choose a good heuristic. Furthermore, these approaches are agnostic on the problem of potentially large action spaces, which is impossible to ignore for relational domains.

Next, one can seek to identify symmetries in the problem structure to collapse actions together. This group includes work in the context of constraint satisfaction and probabilistic STRIPS planning [8, 9, 10]. Unfortunately, in the existing work in this area the notion of what makes objects equivalent relies primarily on unary properties; thus, the contributions are difficult to extend to relational domain in which relations have more than one argument.

More recently, there have been a number of approaches that learn from small, concrete, examples and ramp up. In the setting of relational MDPs, the first set of approaches seeks to learn a generalizable value function [15, 14] The general strategy is to solve one problem instance, compute value function that will generalize to larger instances, and directly tackle the next instance. The complementary approach [18] employs a similar strategy, but in the space of policies rather than value functions. The difference with our work is that these compute solutions over the entire RMDP space, whereas our approach explicitly attempts to solve only a subset of it.

Finally, one can analyze the transition dynamics collapse together states that behave the same. This set includes techniques that model MDPs in which certain groups of states are grouped together and treated as an atomic, abstract state [5, 12, 6, 13]. This work is clearly related to ours, but important here is the notion of actions being equivalent in *all* states to achieve a reduced version of the whole domain. By contrast, our work seeks to dynamically compute action equivalence given the current state and task, i.e., to locally collapse states/actions for the purposes of forward planning.

References

[1] A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1), 1995.

[2] Avrim L. Blum and Merrick L. Furst. Fast plannning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[3] Avrim L. Blum and John C. Langford. Probabilistic planning in the graphplan framework. In *5th European Conference on Planning*, 1999.

[4] Blai Bonet and Hector Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *ICAPS-03*, 2003.

[5] Thomas Dean and Robert Givan. Model minimization in markov decision processes. In *AAAI*, 1997.

[6] Thomas Dean, Robert Givan, and Sonia Leach. Model reduction techniques for computing approximately optimal solutions for markov decision processes. In *UAI*, 1997.

[7] Thomas Dean, Leslie Pack Kaelbling, Jak Kirman, and Ann Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76, 1995.

[8] T. Ellman. Abstraction via approximate symmetry. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 1993.

[9] Maria Fox and Derek Long. The detection and exploitation of symmetry in planning problems. In *16th International Joint Conference on Artificial Intelligence*, 1999.

[10] Maria Fox and Derek Long. Extending the exploitation of symmetries in planning. In *AIPS*, 2002.

[11] Natalia H. Gardiol and Leslie P. Kaelbling. Envelope-based planning in relational MDPs. In *Advances in Neural Information Processing 16 (NIPS-2003)*, Vancouver, 2004.

[12] Robert Givan and Thomas Dean. Model minimization, regression, and propositional strips planning. In *15th IJCAI*, 1997.

[13] Robert Givan, Tom Dean, and Matthew Greig. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, 147:163–223, 2003.

[14] Carlos Guestrin. *Planning Under Uncertainty in Complex Structured Environments*. PhD thesis, Stanford University, 2003.

[15] Carlos Guestrin, Daphne Koller, Chris Gearhart, and Neal Kanodia. Generalizing plans to new environments in relational MDPs. In *International Joint Conference on Artificial Intelligence (IJCAI 2003)*, 2003.

[16] Eric Hansen and Shlomo Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 2001.

[17] M. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 1994.

[18] SungWook Yoon, Alan Fern, and Robert Givan. Inductive policy selection for first-order MDPs. In *18th International Conference on Uncertainty in Artificial Intelligence*, 2002.

[19] H. Younes and M. Littman. PPDDL1.0: An extension to pddl for expressing planning domains with probabilistic effects. In *In Proceedings of the 14th International Conference on Automated Planning and Scheduling*, 2003.

[20] H. Younes and M. Littman. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, Carnegie Mellon University, 2004.

[21] Luke Zettlemoyer, Hanna Pasula, and Leslie Pack Kaelbling. Learning planning rules in noisy stochastic worlds. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, 2005.