Journal of Artificial Intelligence Research ? (200?) 1-? Submitted 2/09; published ?

Adaptive Relational Envelope MDPs

Natalia H. Gardiol NHG@CSAIL.MIT.EDU
Leslie Pack Kaelbling LPK@CSAIL.MIT.EDU
MIT CSAIL,
Cambridge, MA 02139 USA

Abstract

We describe a method for using structured representations of the environment's dynamics to constrain and speed up the planning process. Given a problem domain described in a probabilistic logical description language, we develop an anytime technique that incrementally improves on an initial, partial policy. This partial solution is found by first reducing the number of predicates needed to represent a relaxed version of the problem to a minimum, and then dynamically partitioning the action space into a set of equivalence classes with respect to this minimal representation. Our approach uses the *envelope* MDP framework, which creates a Markov decision process out of a subset of the full state space as determined by the initial partial solution. This strategy permits an agent to begin acting within a restricted part of the full state space and to expand its envelope judiciously as resources permit.

1. Introduction

For an intelligent agent to operate efficiently in a highly complex domain, it must identify and gain leverage from structure in its domain. Household robots, office assistants, and logistics support systems, for example, cannot count on problems that are carefully formulated by humans to contain only domain aspects actually relevant to achieving the goal. Generally speaking, planning in a formal model of the agents entire raw environment will be intractable; instead, the agent will have to find ways to reformulate a problem into a more tractable version at run time. Not only will such domains require an adaptive representation, but adaptive aspirations as well: if the agent is under time pressure to act, we must be willing to accept some trade-off in the quality of behavior. However, as time goes on, we would expect the agent's behavior to become more robust and to improve in quality. Algorithms with this characteristic are called *anytime* algorithms (Dean & Boddy, 1988). Anytime algorithms can operate either off-line (working until a specified time limit) or by interleaving refinement with execution.

Consider the task of going to the airport. There are many features available your world view (the current time, what kind of shoes you are wearing, etc), but you might start the planning process by considering only road connections. Then, with a basic route in place, you might then make modifications to the plan by considering traffic levels, the amount of gas currently in the tank, how late you are, and so forth. The point is that by starting with a reduced representation to solve a principled approximation of the problem, we can begin to act sooner and expect that our solution will improve upon more reflection.

The basic idea we are interested in, then, is this: first, find a simple plan; then, elaborate the plan. One early technique in this spirit was proposed by Dean *et al.* (1995), who

introduced the idea of *envelope* MDP*s* in the context of an algorithm called Plexus. Given a planning problem represented as an atomic-state MDP (Puterman, 1994), Plexus finds an initial subset of states by executing a depth-first search to the goal, forming a restricted MDP out of this subset. The state space for the restricted MDP is called the *envelope*: it consists of a subset of the whole system state space, and it is augmented by a special state called OUT representing any state outside of the envelope. The algorithm then works by alternating phases of *policy generation*, which computes a policy for the given envelope, and *envelope alteration*, which adds states to or removes states from the envelope. This deliberation can produce increasingly robust and sophisticated plans.

The difficulty of planning effectively in complex problems, however, lies in maintaining an efficient, compact model of the world in spite of potentially large ground state and action spaces. This requires moving beyond atomic-state MDPs. Therefore, we will represent problems in the richer language of *relational* MDPs and present an extension of the envelope MDP idea into this setting. One additional advantage of relational representation is that it exposes the structure of the domain in a way that permits modifying, or adapting, the representation with respect to the goal. For example, if our goal of getting to the airport is not merely to arrive there, but to avoid security delays, we might take our shoes into consideration from the outset.

Our technique takes the basic envelope MDP idea, but extends it to efficiently handle relational domains. These steps are: 1) reformulating the given problem in terms of the most parsimonious representation, β for the given task; 2) finding an initial plan in the space expressed by β; 3) constructing an *abstract* MDP from the initial subset of states; and, 4) expanding the abstract MDP by both adding new states and/or refining the representation, β. Whereas the original Plexus algorithm refined its plan by manipulating the set of states in an envelope, the Relational Envelope-based Planning approach (REBP), provides a framework for also modifying the *dimensions* for representing those states.

2. Compactly modeling large problems

The problem of planning has been an important research area of AI since the inception of the field. However, even its "simplest" setting, that of deterministic STRIPS planning, has been found to be PSPACE-complete (Bylander, 1994). Nonetheless, with the use of powerful logical representations that enable structural features of the state and action spaces to be leveraged for efficiency, traditional AI planning techniques are often able to manage very large state spaces. Simultaneously, work in the operations research community (OR) has developed the framework of MDPs, which specifically addresses uncertainty in dynamical systems. Being able to address uncertainty (not only in acting, but, also in sensing, which we do not address in this work) is a primary requirement for any system to be applicable to a wide range of real-world problems.

Our aim is to bring together some of these complementary features of AI planning and MDP solution techniques to produce a system that can build on the strengths of both. An important result that enables our approach is that problems of goal-achievement, as typically seen in AI planning problems, are equivalent to general reward problems (and vice versa). Thus, it will be possible for us to take a given planning planning problem and convert it to an equivalent MDP (Majercik & Littman, 2003).

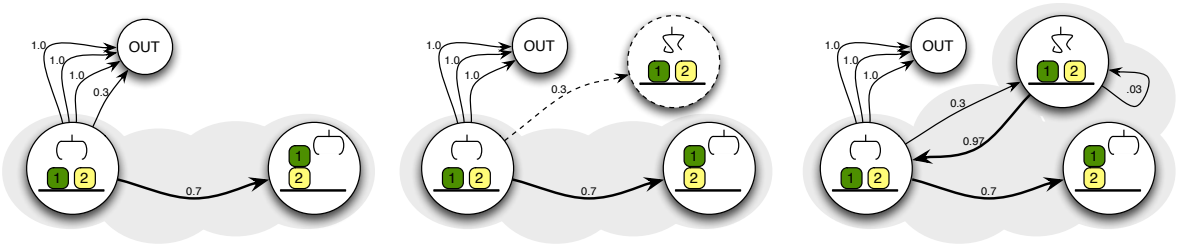

Figure 1: A toy example of envelope-based planning. The task is to make a two-block stack in a domain with two blocks. The initial plan is consists of a single *move* action, and the initial envelope (far left) reflects this action sequence. The next step is to sample from this policy, and the potentially bad outcome of breaking the gripper is noticed (middle). After expanding the envelope to include this outcome, the policy is revised to include executing a "repair" action from the newly incorporated state (far right).

2.1 Envelope MDP background

The original Plexus algorithm works by considering a subset of states with which to form a restricted MDPand then searching for an optimal policy in this restricted MDP. The state space for the restricted MDP is called the *envelope*: it consists of a subset of the whole system state space, and it is augmented by a special state called OUT representing any state outside of the envelope. The algorithm then works by alternating phases of *envelope alteration*, which adds states to or removes states from the envelope, and *policy generation*, which computes a policy for the given envelope. In order to guarantee the anytime behavior of the algorithm, Dean *et al.* extensively study the issue of *deliberation scheduling* to determine how best to devote computational resources between envelope alteration and policy generation.

A small example of refining an initial plan is shown in Figure 1, which consists of a sequence of fringe sampling and envelope expansion. A complete round of deliberation involves sampling from the current policy to estimate which *fringe* states — states one step outside of the envelope — are likely. The figure shows the incorporation of an alternative outcome of the policy action in which the gripper breaks. After the envelope is expanded to include the new state, the policy is re-computed. In the figure, the policy now specifies the *fix* action in case of gripper breakage.

Thus, deliberation can produce increasingly sophisticated plans. The initial planner needs to be quick, and as such may not be able to find conditional plans, though it can develop one through deliberation; conversely, searching for this conditional plan in the space of all MDP policies, without the benefit of the initial envelope, could potentially have taken too long.

The Plexus algorithm was originally developed for atomically represented robot-navigation domains, which generally have the characteristics of high solution density, low dispersion rate (i.e., a small number of outgoing transitions at each state), and continuity (i.e., that the value of a state can be reasonably estimated by considering nearby states). These features made it reasonable to execute a depth-first search in order to find the first set of states

for the envelope. Arbitrary relational planning domains may not necessarily share these characteristics.

2.2 Representation issues in MDPs

Much preceding work on finding policies for MDPs considered a state to be an atomic, indivisible entity. More recently, advances have been made in representing a MDP states in terms of *factored* state spaces: that is, a particular state is seen to be a combination of state features. However, though a factored state representation is an improvement over an atomic state representation, it still has limitations. The state features correspond to *propositions* about a state: e.g., "the x-coordinate has value 3." This means that the policy π, the transition function T, and the reward function R must cover possible combinations of values of all of the state features. Having large number of features, or features that can take on a wide range of values, becomes problematic as the size of the state space grows combinatorially.

In this work, rather than a thinking of a state as being composed of a set of propositional features, we think of it as being composed instead of a set of logical relationships between classes of domain objects. Because these predicates can make assertions about logical *variables*, a single predicate may in fact represent a large group of ground propositions. Thus, a single transition rule in a logical language may represent multiple ground state transitions. If the transition function and the reward function have a compact representation that can be exploited for efficiency during planning, then that makes it possible to be relatively insensitive to the size of the state space.

2.3 Encoding Markovian dynamics with rules

A well-specified AI planning problem contains two basic elements: a domain description and a problem instance. The domain description specifies the *dynamics* of the world, the *types* of objects that can exist, and the set of logical *predicates* which comprise the set of relationships and properties that can hold for the objects in this domain. To specify a given problem instance, we need an *initial world state*, which is the set of ground predicates that are initially true for a given set of objects. We also need a *goal condition,* which is a first-order sentence that defines the task to be achieved. The dynamics of the domain must be expressed in a particular rule language. In our case, the language used is the Probabilistic Planning and Domain Definition Language (PPDDL) of Younes and Littman (2004), which extends the classical STRIPS language (Fikes & Nilsson, 1971; Kushmerick, Hanks, & Weld, 1995) to probabilistic domains. We will use the term "rule" or "operator" when we mean an abstract rule such as it appears in the domain description, and we will use "action" to denote a ground instance of a rule.

A domain description together with a particular problem instance induce a *relational* MDP for the problem instance. An classical MDP is defined as a tuple, $\langle \mathcal{Q}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ where: $\mathcal{Q}$ is a set of states; $\mathcal{A}$ is a set of actions; $\mathcal{T}$ is a transition function; and $\mathcal{R}$ is a reward function.

By extension, we define a relational MDP (RMDP) as a tuple $\langle \mathcal{P}, \mathcal{Z}, \mathcal{O}, \mathcal{T}, R \rangle$, consisting of a set of states, actions, transitions, and rewards:

States: The set of states $\mathcal{Q}$ in an RMDP is defined by a finite set $\mathcal{P}$ of relational predicates, representing the relations that can hold among the finite set of domain objects, $\mathcal{O}$. Each RMDP state is an *interpretation* of the domain predicates over the domain objects.

Actions: Analogously, the set of ground actions, is induced, by the set of rules $\mathcal{Z}$ and the objects in the world.

Transition Dynamics: The transition dynamics are given by a compact set of rules $\mathcal{Z}$ as given in the domain description. A rule is said to apply in a state if its precondition is true in the interpretation associated with the state.

For each action, the distribution over next states is given compactly by the distribution over outcomes encoded in the rule schema. The rule outcomes themselves usually only specify a subset of the domain predicates, effectively describing a set of possible resulting ground states. To fill in the rest, we assume a static frame: state predicates not directly changed by the rule are assumed to remain the same.

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

2.4 Example: describing a planning domain

To ground the discussion, let us consider an example using one of our test domains, the "slippery" blocksworld. This domain is an extension of the standard blocks world in which some blocks (the green ones) are "slipperier" than the other blocks. The pick-up and put-down actions are augmented with a conditional effect that produces a different distribution on successful pick-up and put-down when the block in-hand is green. While color may be ignored for the purposes of sketching out a solution quickly, higher quality policies result from detecting that the color green is informative (Gardiol, 2007).

The domain description contains the *types* of objects available in the world (in this case, blocks and tables), and a list of the relationships that can hold between objects of particular type (e.g., $on(A, B)$, where A is a Block and B is a block or a table). Finally, the rules in the slippery blocks domain consist of two operators, each containing a conditional effect that can produce a different outcome distribution, as in Figure 2.

A particular problem instance consists of a ground initial state, such as, e.g.:

on(block0,block2), on(block2,table), on(block4,table), isblue(block0),
isred(block2), isblue(block4)

and a goal, such as:

$$\forall B.type(B, block) \wedge on(B, table),$$

or,

$$\exists B1.type(B1, block) \wedge \exists B2.type(B2, block) \wedge B1 \neq B2 \wedge on(B1, B2) \wedge on(B2, table).$$[1]

1. Unless the context is ambiguous, we will henceforth leave objects' *type()* specifications implicit.

$$
\begin{array}{l}
pickup(A,B): \\
\quad on(A,B) \wedge \forall C.\neg holding(C) \wedge \forall D.\neg on(D,A)) \\
\qquad\qquad \longrightarrow \left\{ \begin{array}{ll} .9 & holding(A) \wedge \neg on(A,B) \\ .1 & on(A,table) \wedge \forall E.\neg on(A,E) \\ when(isgreen(A)) & \\ \qquad \longrightarrow \left\{ \begin{array}{ll} .6 & holding(A) \wedge \neg on(A,B) \\ .4 & on(A,table) \wedge \forall E.\neg on(A,E) \end{array} \right. & \end{array} \right.
\end{array}
$$

$$
\begin{array}{l}
put(A,B): \\
\quad holding(A) \wedge A \neq B \wedge \forall C.\neg on(C,B)) \\
\qquad\qquad \longrightarrow \left\{ \begin{array}{ll} .9 & \neg holding(A) \wedge on(A,B) \\ .1 & \neg holding(A) \wedge on(A,table) \\ when(isgreen(A)) & \\ \qquad \longrightarrow \left\{ \begin{array}{ll} .6 & \neg holding(A) \wedge on(A,B) \\ .4 & \neg holding(A) \wedge on(A,table) \end{array} \right. & \end{array} \right.
\end{array}
$$

Figure 2: Example operators from the slippery blocksworld domain. Both the *pickup* and the *put* rules contain a conditional effect that produces a different outcome distribution when the block being held is green.

3. Equivalence-based planning

In developing an envelope MDP approach for the relational case, the first task is to find a suitable method for finding the initial envelope of states. The original Plexus algorithm did this by executing a depth-first search to the goal; in an arbitrary planning problem, however, this may be a poor strategy. However, because our planning task is expressed as an RMDP, with the transition dynamics as a set of logical rules, then we can exploit techniques from classical AI planning to find an initial envelope more efficiently.

The complexity of classical planning is driven primarily by the length of the solution and the branching factor of the search. The solution length can sometimes be effectively reduced using hierarchical techniques. The branching factor can often be reduced, in effect, by an efficient heuristic. The equivalence-based planning of Gardiol and Kaelbling (2007) algorithm provides a novel method for reducing the branching factor by dynamically grouping the agent's actions into *state-dependent equivalence classes*, and only considering a single action from each class in the search. This method can dramatically reduce the size of the search space, while preserving correctness and completeness of the planning algorithm. It can be combined with heuristic functions and other methods for improving planning speed.

3.1 Assumptions and definitions

To be precise, we will review some assumptions and definitions introduced by Gardiol and Kaelbling:

Assumption 1 (Sufficiency of Object Properties)**.** *A domain object's function is assumed to be determined solely by its properties and relations to other objects, and not by its name.*

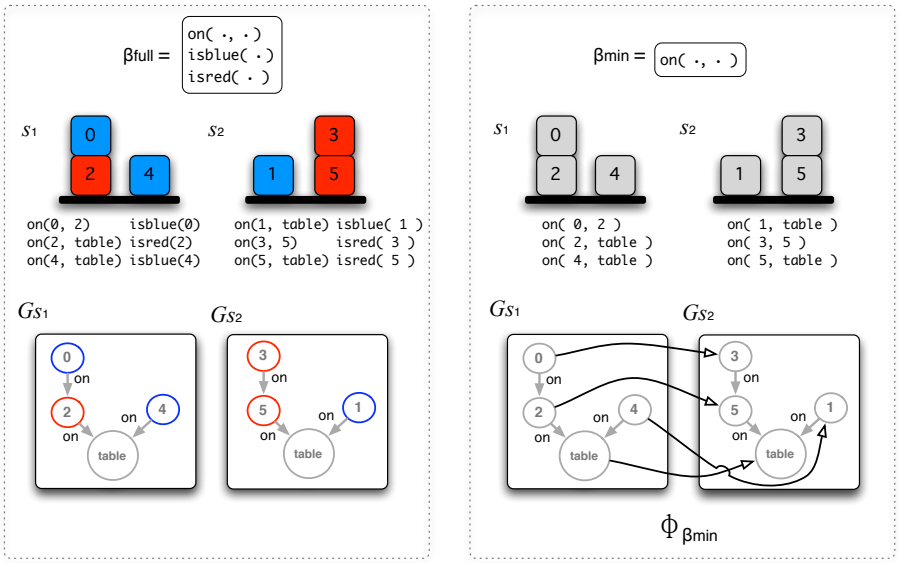

Figure 3: In the original set of predicates representing the problem, β_{full}, states s_1 and s_2 are distinct. But in the reduced basis β_{min}, without colors, an isomorphism Φ can be found between the state relation graphs G_{s1} and G_{s2}.

The algorithm uses a graph isomorphism-based definition of equivalence among states. State equivalence is determined by representing a state s as a *state relation graph*, G_s, where each node in the graph represents an object in the domain and each edge between nodes represents a relation between the corresponding objects. Nodes are labeled by their *type* and with any unary relations (or properties) that apply to them.

More formally, the definitions are as follows:

State equivalence Two *states* s_1 and s_2 are *equivalent*, denoted $s_1 \sim s_2$, if there exists an isomorphism, ϕ, between the respective state relation graphs: $\phi(\mathcal{G}_{f_\infty}) = \mathcal{G}_{f_\in}$.

Intuitively, two objects are equivalent to each other if they are related in the same way to other objects that are, in turn, equivalent. Thus, two *objects* o_1 and o_2, in states s_1 and s_2 respectively, are defined to be equivalent if o_2 is the image of o_1 under an isomorphism that maps s_1 to s_2.

We use the notation $\gamma(a, s)$ to refer to the state that results from taking action a in state s. If a is an action following the PDDL syntax, then we calculate $\gamma(a, s)$ by removing from s all the atoms in a's *delete list* and adding all the atoms in a's *add list*. We will sometimes refer to this calculation of $\gamma(a, s)$ as "propagating" s through the dynamics of a. More precisely, we write:

$$\gamma(a, s)) = \phi(s \cup add(a) \setminus del(a)),$$

Where $del(a)$ refers to set the atoms that are negated in the effect of a, and $add(a)$ refers to the set of atoms that are non-negated in the effect of a.

With respect to a given state s, then, we define two ground actions a_1 and a_2 to be equivalent if they produce equivalent successor states, $\gamma(a_1, s)$ and $\gamma(a_2, s)$:

Action equivalence Two *actions* a_1 and a_2 are *equivalent* in a state s, denoted $a_1 \sim a_2$, iff $\gamma(a_1, s) \sim \gamma(a_2, s)$

However, using this definition to directly calculate a set of of equivalent actions: it requires propagation of the state through each action's dynamics in order to look for pairs of resulting states that are equivalent. This is unwieldy. A better tactic is to *overload* the notion of isomorphism to apply to actions and develop a test on the starting state and actions directly, without propagation. In PPDDL and related formalisms, actions can be thought of as ground applications of predicates. Thus, each argument in a ground action will correspond to an object in the state, and, thus, to a node in the state relation graph. So, two actions applicable in a state s are provably equivalent if: (1) they are each ground instances of the same operator, and (2) there exists a mapping $\phi(s) = s$ that will map the arguments of one action to arguments of the other. In this case, since the isomorphism ϕ that we seek is a mapping between s and itself, it is called an *automorphism*. We compute action equivalence via the notion of action isomorphism, defined formally as follows:

Action isomorphism Two actions a_1 and a_2 are *isomorphic* in a state s, denoted $a_1 \sim_s a_2$, iff there exists an automorphism for s, $\phi(s) = s$, such that $\phi(a_1) = a_2$.

It can be shown that the application of isomorphic actions in equivalent states produces equivalent successor states. Thus, isomorphic actions are equivalent actions. It is straightforward to prove that a sequence of actions can be replaced by a sequence of equivalent actions to produce equivalent ending states. Thus, using a complete planning procedure, such as forward search, with a reduced action space consisting of *canonical representatives* from each equivalence class preserves completeness of the original planning procedure with the full action set.

We will describe a state or action *canonical* as canonical to mean that it is a member, and thus represents, an equivalence class of states or actions. How the canonical state or action is selected is immaterial to the algorithm; any member of the class is sufficient. We will use the bracket notation $[o]$ to denote that the object o represents the class of objects equivalent to o.

The original envelope-based algorithm is based on forward search with the Fast Forward heuristic (Hoffmann & Nebel, 2001), which is a complete search procedure.[2]

3.2 Extending equivalence-based forward search

In this paper, we will extend the power of state equivalence by allowing the representational vocabulary of the state relation graph itself to be adaptive. The motivation is that the fewer predicates we have in our representation, the fewer edges or labels there will be in the state relation graph, resulting in more states being considered equivalent.

2. While not affecting completeness, reducing the action set can have an effect on plan parallelism. Since the algorithm is 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.

Specifically, the planning process begins by making two simplifications. First, as before, is a deterministic approximation of the original planning problem: operators are assumed to deterministically produce their most likely outcome. Second, however, is the identification of the minimal set of predicates, or basis, necessary to solve a relaxed version of the problem. We mean "relaxed" in the sense of the well-known Fast-Forward heuristic (FF)of Hoffman and Nebel (2001), the heuristic used by the forward-search algorithm to guide the search. Computing the heuristic value of the initial state gives a lower bound on the number of steps required to achieve the goal. It also gives an estimate of the smallest set (though not the order) of actions necessary to achieve the goal from the initial state. The minimal basis is found by taking this set of actions and including only those predicates that appear in the goal statement and in the preconditions of those actions. The planning problem is then reformulated with respect to this minimal basis set, β. See Figure 3 for an example of formulating a ground state with different bases.

The extended equivalence-based forward search algorithm is given in Algorithm 1. By extending the machinery of the envelope-deliberation framework, which we will see shortly, β can be subsequently refined and augmented to improve policy robustness.

Input: Initial state s_0, goal condition g, set of rules Z
Output: Sequence of actions from s_0 to a goal state
Find a minimal representational basis, β ;
Find canonical initial state, $\tilde{s_0}$;
Initialize agenda with $\tilde{s_0}$;
while *agenda is not empty* **do**
  Select and remove a state s from the agenda ;
  if *s satisfies goal condition g* **then**
    return *path from root of search tree to s ;*
  else
    Find representative set $\mathcal{A}'$ of actions applicable in s ;
    foreach $a \in \mathcal{A}'$ **do**
      Add the successor of s under a to the agenda ;

Algorithm 1: Equivalence-based forward-search algorithm. $\mathcal{A}'$ denotes the set of *canonical* actions with respect to β; each canonical action stands for an equivalence class of actions.

4. Computing an abstract envelope

Now we will use the output of the forward search, which is a deterministic sequence of actions, to bootstrap an MDP and directly address uncertainty in the domain. The output of the planning phase, above, is a sequence of canonical actions, which induces a sequence of canonical states. The canonical states are represented in a basis set of predicates that may be smaller than or equal to the set of predicates originally given in our domain description.

We will use this abstract state sequence to initialize an abstract envelope MDP, which we will manipulate this envelope MDP in two ways: first, as in the original Plexus algorithm, we will sample from our policy and incorporate states off the initial path; second, new to this work, we will incorporate additional dimensions to the representation to increase the accuracy of the estimated value of the current policy.

Constructing an MDP with abstract states introduces an additional subtlety to the algorithm: it is possible that the dynamics driving the transition between abstract states may differ depending on which underlying ground states are participating in the transition. Thus, each transition probability is properly represented as an interval rather than a scalar probability. Interval MDPs, and the corresponding Interval Value Iteration algorithm, were first shown by Givan *et al.* (1997, 2000).

Let us formally define an abstract-state, interval envelope MDP (called an AMDP for short) as an extension of the basic relational MDP.

An AMDP M is a tuple $\langle \mathcal{Q}, \beta, \mathcal{Z}, \mathcal{O}, \mathcal{T}, R \rangle$, where:

States: The full abstract state space, $\mathcal{Q}^*$, is defined by a basis set β of relational predicates, representing the relations that hold among the equivalence classes of domain objects, $\mathcal{O}$. The set of states $\mathcal{Q}$, is the union of the set $\mathcal{Q}' \subseteq \mathcal{Q}^*$ and a special state q_{out}. That is, $\mathcal{Q} = \mathcal{Q}' \cup \{q_{out}\}$. The set $\mathcal{Q}'$, also called the *envelope*, is a subset of the entire abstract state space, and q_{out} is an additional special state that captures transitions from any $q \in \mathcal{Q}'$ to a state outside the envelope. Through the process of envelope expansion, the set of states $\mathcal{Q}$ will change over time.

Actions: The set of actions, $\mathcal{A}$, is calculated by finding the ground instances of the set of rules $\mathcal{Z}$ applicable in $\mathcal{Q}'$.

Transition Dynamics: In an interval MDP, $\mathcal{T}$ gives the interval of probabilities that a state and action pair will transition to another state: $\mathcal{T} : \mathcal{Q} \times \mathcal{A} \times \mathcal{Q} \rightarrow [\mathbb{R}, \mathbb{R}]$. We will see how to compute this transition function in the sections below.

Rewards: As before, state is mapped to a scalar reward according to function $R(s)$.

4.1 Initializing the abstract-state envelope

In this section, we look at how to compute the initial set of states $\mathcal{Q}$ by bootstrapping from the output of the planning phase.

Each state $q_i \in \mathcal{Q}'$ of our AMDP M is a composite structure consisting of:

1. $\tilde{s}_i$: a canonical state, in which we represent only the canonical members of each object equivalence class and the relations between them.

2. S_i: a set of underlying ground states, s_i, consistent with the above canonical state. As we will see below, these underlying ground states will be collected as we go.

The first state, q_0, is computed from the initial state of the planning problem straightforwardly: the set S_0 is initialized with the ground initial state of the planning problem, and the canonical state $\tilde{s}_0$ is the canonical representative, with respect to basis β.

We compute the second state, q_1, by taking the first action, a_0, in our plan. The next canonical state $\tilde{s}_1$ is computed by propagating $\tilde{s}_0$ through a_0. The ground state of q_0 can be efficiently propagated as well, and, we add the result to S_1. This procedure is repeated until we've processed the last action. More formally, the procedure to compute the envelope, with respect to basis β, from a plan p is in Algorithm 2.

Input: Canonical initial state $\tilde{s_0}$, Plan p, Basis β
Output: Set of envelope MDP states $\mathcal{Q}'$
Initialize q_0 with $\tilde{s}_0$ and with $S_0 = \{s_0\}$;
Initialize $\mathcal{Q}' = \{q_0\}$;
foreach *action* a_i *in* p, $i = 0 \ldots n$ **do**
  Propagate $\tilde{s}_i$ to obtain $\tilde{s}_{i+1}$;
  Propagate each s_i in S_i to obtain s_{i+1} ;
  Initialize q_{i+1} with $\tilde{s}_{i+1}$ and with $S_{i+1} = \{s_{i+1}\}$;
  $\mathcal{Q}' = \mathcal{Q}' \cup \{q_{i+1}\}$;

Algorithm 2: Computation of envelope given a plan.

At this point, we have a set of MDP states $\mathcal{Q}' = \cup_{i=0}^{n+1}\{q_i\}$. To complete the set of states, $\mathcal{Q}$, we add the special state q_{out}.

This procedure lets us keep a record of the true ground state sequence, the s_i's, as we construct our model. Why do this, when we've gone through so much trouble to compute the canonical states? The problem is not that any individual ground state is too large or difficult to represent, but that the search space over all the ground states is prohibitive in combination. Nonetheless, conserving ground information is necessary in order to determine how to modify the basis set down the line.

While each MDP state tracks its underlying ground state, it is only the canonical state that is used for determining behavior. Since a canonical state represents a collection of underlying ground states, the policy we compute using this approach effectively encompasses more of the state space than we physically visit during the planning process.

4.2 Computing transition probabilities

Given a set of states $\mathcal{Q}$, the next step is to compute the transition function $\mathcal{T}$. First, we compute the *nominal* interval probabilities of transitioning between the canonical states. The nominal probabilities represent the best estimate of the intervals given the abstract model and the current ground information stored in the model. Second, we sample from our underlying state space to flesh out the interval of probabilities describing each transition. That is, we start from the ground state, and sample a sequence of ground transitions. If any of these sampled transitions corresponds to an action outcome with a probability outside of the current interval estimate, we add that informative ground state to the collection of ground states associated with the corresponding abstract state, and update the interval accordingly. We will speak of *updating* a probability interval $P = [a, b]$ with the probability p, which means: if $p < a$, then P becomes $[p, b]$; if $p > b$, then P becomes $[a, p]$. We adopt a sampling approach, rather than an analytic approach based on rule syntax, to avoid having to use, e.g, theorem proving, to decide whether two logical outcomes in fact refer

to equivalent successor states. These two steps, the nominal interval computation and the sampling estimation, are executed as part of a loop that includes the envelope expansion phase: each time a new state is added, it will be necessary to re-estimate the transition probabilities.

We compute the nominal interval probabilities by:

1. For each state $q_i \in \mathcal{Q}$, find the set of actions A_i applicable in $\tilde{s}_i$.
2. For each action $a_k \in A_i$ and state $q_j \in \mathcal{Q}'$ compute the transition probability between q_i and q_j:
   (a) Initialize the ground transition probability. That is, take the first ground state in S_i and propagate it through action a_k. If the resulting ground state is equivalent to q_j with respect to the basis β, and p is the probability of the outcome of a_k corresponding to that transition, then set the probability of transitioning from q_i to q_j via action k as the interval $P_{ijk} = [p, p]$.
   (b) For each remaining ground state $s_n^i \in S_i$, compute the probability p' of transitioning to q_j via action a_k.[3] *Update* interval P_{ijk} with p'.
3. Compute the probability of transitioning to q_{out} from q_i and a_k. This involves keeping track, as we execute the above steps, of the overall out-of-envelope probability for each ground application of the action a_k. More precisely: for each $s \in S_i$, when we apply a_k and detect a transition of probability p to a state within the envelope, we update a_k's out-of-envelope probability with $1 - p$. This ensures that the out-of-envelope probabilities are consistent for each representative action, a_k.

Figure 4 shows an example of an abstract MDP M after the initial round of nominal interval computation. In this example, the initial world state is s_0^0. The goal is to put three blocks in a stack, regardless of color, and the forward-search has returned the sequence: $pickup(3, table)$, $put(3, 1)$. The abstract MDP has been computed from this plan to produce the set of states induced by this action sequence, and the collection S_i of each abstract state contains only a single element at this point.

Next, in order to improve the interval estimates, a round of sampling is done from the current model. The idea is to try to uncover, via this sampling, any ground states that yield transition probabilities outside of the current interval estimates. This proceeds as follows:

1. For each state $q_i \in \mathcal{Q}'$, action $a_k \in A_i$, and state $q_{j \neq i} \in \mathcal{Q}'$: let the ground state s' be the result of propagating a state $s_n^i \in S_i$ through a_k.
   (a) If there exists a state q_k such that the probability of transitioning from s' to q_k under an action is outside of the current interval for transition of q_j to q_k for that action, add s' to the set S_j of q_j.

The result of carrying out this procedure is show in Figure 5.

3. Strictly speaking, we will need to use the inverse of the mapping ϕ between s and $\tilde{s}_i$ to translate the action a_k into the analogous action applicable in s. This is because, while s may belong to the equivalence class of states represented by $\tilde{s}_i$, it may have objects of different actual identities belonging to each object equivalence class.

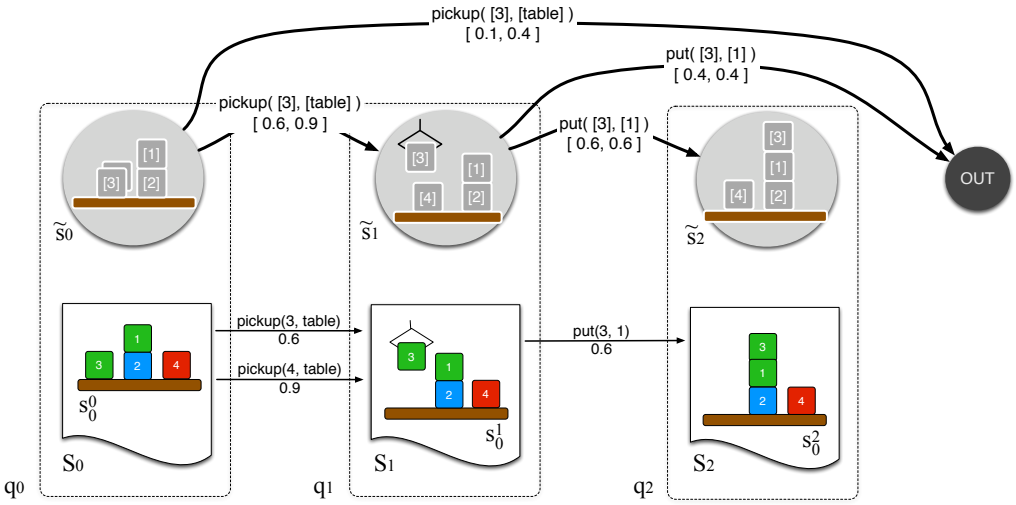

Figure 4: The abstract MDP after computing the nominal transition probabilities. For simplicity, we only show a subset of the possible actions. For example, the action *pickup*([1], [2]), which is applicable in state $\tilde{s}_0$, would have a transition to OUT with probability $[1.0, 1.0]$ since there is no state in the current model that represents the outcome of taking this action.

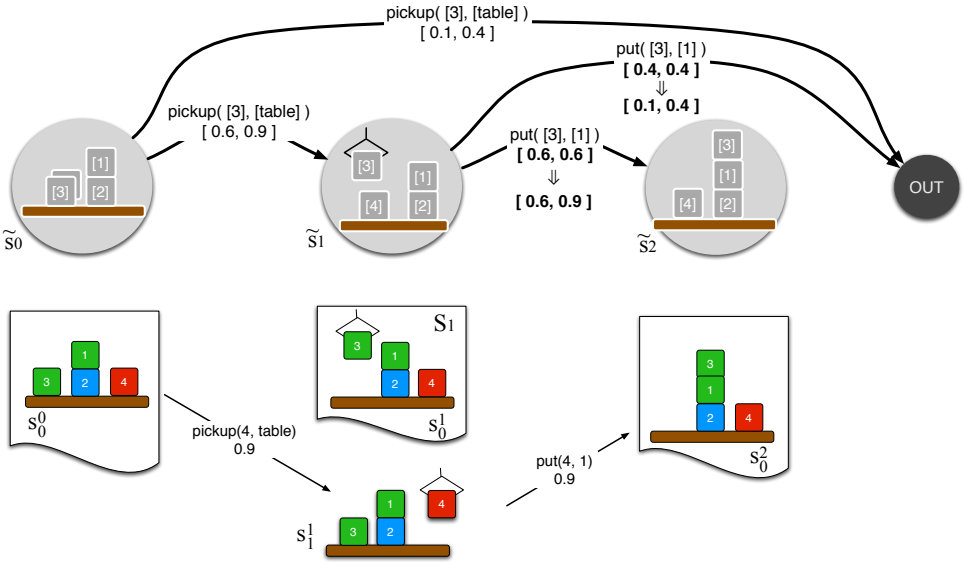

Figure 5: The abstract MDP after a round of sampling to update transition interval estimates. State $s' = s_1^1$ was sampled from s_0^0, and it yielded a transition probability to $\tilde{s}_2$, outside of the current estimate of P_{120} (assuming the action *put*([3], [1]) has index 0 for $\tilde{s}_1$). Thus, we add ground state s_1^1 to the collection S_1 and update our estimate for P_{120}.

5. Changing the representation

At this point, we know how to construct an AMDP with an abstract state space and with probabilities represented as intervals. Because of the anytime nature of the algorithm, it is possible to carry out Interval Value Iteration on this model and begin acting according to it. If, however, there is computation time available, it may be beneficial to add a predicate, or set of predicates, into the basis β in order to tighten these intervals and lessen the uncertainty in the value estimates. In addressing the issue of modifying the basis, we are confronted very naturally with a type of *structure search* problem, which we describe next.

The point of augmenting the basis set is to be able to express transition probabilities, and thus the expected value of a policy, more precisely. To construct a procedure for modifying the basis, we begin by noting that the transition probabilities are encoded in the rule schemas given as part of our domain description. Therefore, in our case, a β that is missing some potentially useful predicates will lack the capacity to determine the applicability of either an action containing such predicates or a conditional outcome depending on such predicates. For example, consider the slippery blocks world: the minimal basis may ignore color completely, depending on the goal. While this minimal representation speeds up the planning by allowing blocks of different colors to be put into the same equivalence class, it does not model the fact that blocks of color green will experience a different transition probability via the conditional outcome of the pick-up and put-down actions.

So, to propose candidate predicates to augment the current β, the operator examines its precondition and/or conditional effects, and it proposes a set of predicates missing from β. For example, consider the *pickup* operator, which has the condition $isgreen(A)$ on an effect. To produce a new representation β' which can determine if the condition is applicable, then the operator must propose the set (which may simply be a singleton) of required predicates missing from the current β. In the case of our example, this is simply the predicate $isgreen()$. If more than one additional predicate is required to express a condition (e.g., $isgreen(A) \wedge isblue(B)$), then a classic structure search problem occurs: no benefit will be observed by augmenting β with $isgreen$ until $isblue$ has also been added. Thus, because we know we are dealing with rule schemas of this sort, we can take the shortcut of proposing complete sets of missing predicates for a given condition.

The natural place in the algorithm in which to augment the representation is as a part of the envelope-refinement loop. In this loop, we keep a sorted list of the transitions in our AMDP with the maximally uncertain transitions at the top.[4] Then, to propose a refinement, we get the top transition and compute the operator's proposal as described above.

Once we have a proposal to try, we initialize a *new* AMDP using the original plan and the *new* basis. Then, the regular phases of policy improvement and envelope expansion happen for both AMDPs in parallel. We can add as many parallel AMDPs as desired. In our current implementation, we set the limit at n=5 interval AMDPs in parallel. The reason for keeping a number of AMDPs in parallel is that the basis-refinement algorithm is greedy. The first time a proposal is requested from the first abstract AMDP the interval with the widest range is chosen. But it might turn out to be that the second and third proposals jointly produce the highest-performing representation. An AMDP tracks its proposals and does not make

4. We could imagine sorting this list by other metrics. For example, we could be risk-averse and sort them by the lower value bound.

```
Input: Init. state s_0, Goal condition g, Set of rules Z
Output: An abstract MDP, M
Compute minimal basis representation, β ;
Let plan P = REBPForwardSearch( s_0, g, Z ) ;
begin
    Initialize envelope MDP M with P and β ;
    Compute transitions and transition probabilities for M ;
    Do interval value iteration in M until convergence ;
end
Initialize a list of MDPs m = {M} ;
while have time to deliberate do
    foreach MDP M_i in m do
        Do a round of envelope expansion in M_i ;
        if failure to find applicable action in a state q' then
            Remove the q' from M_i ;
            Select the first non-empty proposal basis, β', corresponding to the sequence
            of actions between q' and q_0 ;
            if β' not empty then append to the front of the list of proposals, l_i;
        else
            Sort transitions of M_i in descending order ;
            Compute a proposal basis β' from the top transition ;
            if β' not empty then append β to end of list l_i ;

        Do interval value iteration in M_i until convergence ;
        if l_i not empty then
            Select a basis β' from the list ;
            Construct a new MDP M' with plan P and basis β'. ;
            Append M' to list m of MDPs.;

    Sort the list m by decreasing average policy value ;
    Let M be the MDP at the top of the list m ;
```

Algorithm 3: Overall REPB algorithm.

the same one twice. We would never discover this better performing representation if we only kept the first modification. Again, this is a common structure search issue. Keeping a list of the top performing AMDPs is a simple way of avoiding local minima caused by greedy search, but certainly more sophisticated search control should be considered.

At any given time, the policy of the system is that of the AMDP with the highest policy value. This approach, called Relational Envelope-based Planning, REBP, is given in Algorithm 3.

6. Experiments

In this section we examine a set of experiments done in three different domains. The objective in each domain is to compute a high-value policy with as compact a model as possible. We will look at the various ways of combining the techniques described in this work with the aim of identifying the impact of each on the behavior we observe. The different algorithms are:

Complete Basis + Initial Plan (`full-init`): This is the basic relational envelope-based planning algorithm with no basis reduction. A plan is found in the original representation basis, and this plan initializes a scalar-valued envelope MDP.

Minimal Basis + Initial Plan (`min-init`): This is an extension of REBP that first computes a minimal basis set for the representation. No further basis modification is done, so we use a scalar-valued MDP in this approach as well.
item[Adaptive Basis + Initial Plan] (`adap-init`): This is the full technique: a minimal basis plus an interval MDP for basis and envelope expansion.

No initial plan (`full-null`, `min-null`, `adap-null`): To control for the impact of the initial plan by combining each style of equivalence-class representation with a trivial initial envelope consisting of just the initial planning state.

Propositional (`prop-init`, `prop-null`): Finally, to control for the impact of the equivalence classes, we initialize a scalar-valued MDP in the full, propositional (no equivalence classes) with an initial plan, and with the initial state, respectively.

The domains are:

Slippery blocksworld: This is the same domain described in section 2. For reference, ground problem size in this domain ranges from $4,000$ states and 50 actions in the 5-block problem to 3×10^{79} states and $5,000$ actions in the 50-block problem.

Zoom blocksworld: a different extension of blocks world in which the standard action set is augmented by a one-step *move* action. This action achieves the same effect as, but is less reliable than, a sequence of *pick-up* and *put-down*. However, in order to switch to using the *pick-up* action, the "holding" predicate must be in the representation. The state spaces are the same as the above problem, but the ground action space ranges from 175 actions in the 5-block problem to $130,000$ actions in the 50-block problem.

MadRTS world: this domain is an adaptation of a real-time military logistics planning problem. [5] The world consists of a map (of varying size; six territories in the b

5. Our PPDDL version of this problem was adapted from a scenario originally described by the Mad Doc Software company of Andover, MA in a proposal to address the Naval Research Lab's TIELT military

problems and 11 in the c), some soldiers (ranging from two to six in each problem series), some enemies (ranging from one to four), and some food resources (from one to five). The goal is to move the soldiers between territories so as to outnumber the enemies at their location (enemies don't move). However, the success of a *move* action depends on the health of the soldier. A soldier can transfer, collect, and consume food resources in a territory in order to regain good health. Ground problem size ranges from $12,000$ states and 30 actions in the smallest ($b0$) problem to 1×10^{20} states and 606 actions in the largest ($c2$).

Trials were carried out as follows: REBP forward search was executed to find an intial plan, if required, and then an MDP was initialized (with or without an initial partial solution); then, a specified number (about 100) rounds of deliberation were executed. This number was selected somewhat arbitrarily, but it was enough to allow the adaptive-basis and the fixed, minimal-basis algorithms to converge to a locally optimal policy. To compute the accumulated reward during execution, about 900 steps of simulation were run in each problem (corresponding to roughly 8 successful trials in the blocks worlds), selecting actions according to the policy, and selecting an action randomly %15 of the time. This was done to see how the policy behaves over a broader part of the state space. In the interval MDPs, action selection is done by choosing the action with the highest *average* value. A reward of 1.0 was given upon attainment to the goal, and we report the average accumulated reward per step. All results are averaged over 10-12 trials, initialized with different random seeds, for each algorithm. Complete results are available in (Gardiol, 2007).

Figure 6 shows the accumulated reward (averaged per step) obtained during execution in the simulated domains. Due to space limits, we do not show the corresponding results for the null-envelope approaches, since they generally performed poorly compared to the initial-plan-based approaches. In addition, for those algorithms that achieved a non-zero policy, we indicate the approximate average size of the MDP at convergence to this policy and the computation time. If a data point is missing from the graph, the trials ran out of memory before finishing the specified number of deliberation rounds.

In the Slippery blocks-world, the adaptive-basis approach performs slightly better since it can incorporate the `green` predicate in order to distinguish the green blocks and formulate a policy to avoid them. In the Zoom blocks-world, the difference is more marked: the adaptive-basis approach can formulate a completely new policy out of the more reliable *pick-up* and *put-down* actions, avoiding the less reliable, but faster, single-step *zoom* action. The representation in the fixed-minimal-basis approach contains the necessary predicates to enable the *zoom* action but not *pick-up*. In the MadRTS experiments, being able to identify a minimal predicate set proved crucial to gain traction in the domain. We also note that, in general, the adaptive-basis algorithms are able to provide the highest expected value for a given model size.

The essential message from these experiments is:

1. Equivalence classes improve the efficiency of envelope expansion.

challenge problem (Molineaux & Aha, 2005). While no longer taking place in a real-time system, we call this planning domain the MadRTS domain to signal this origin.

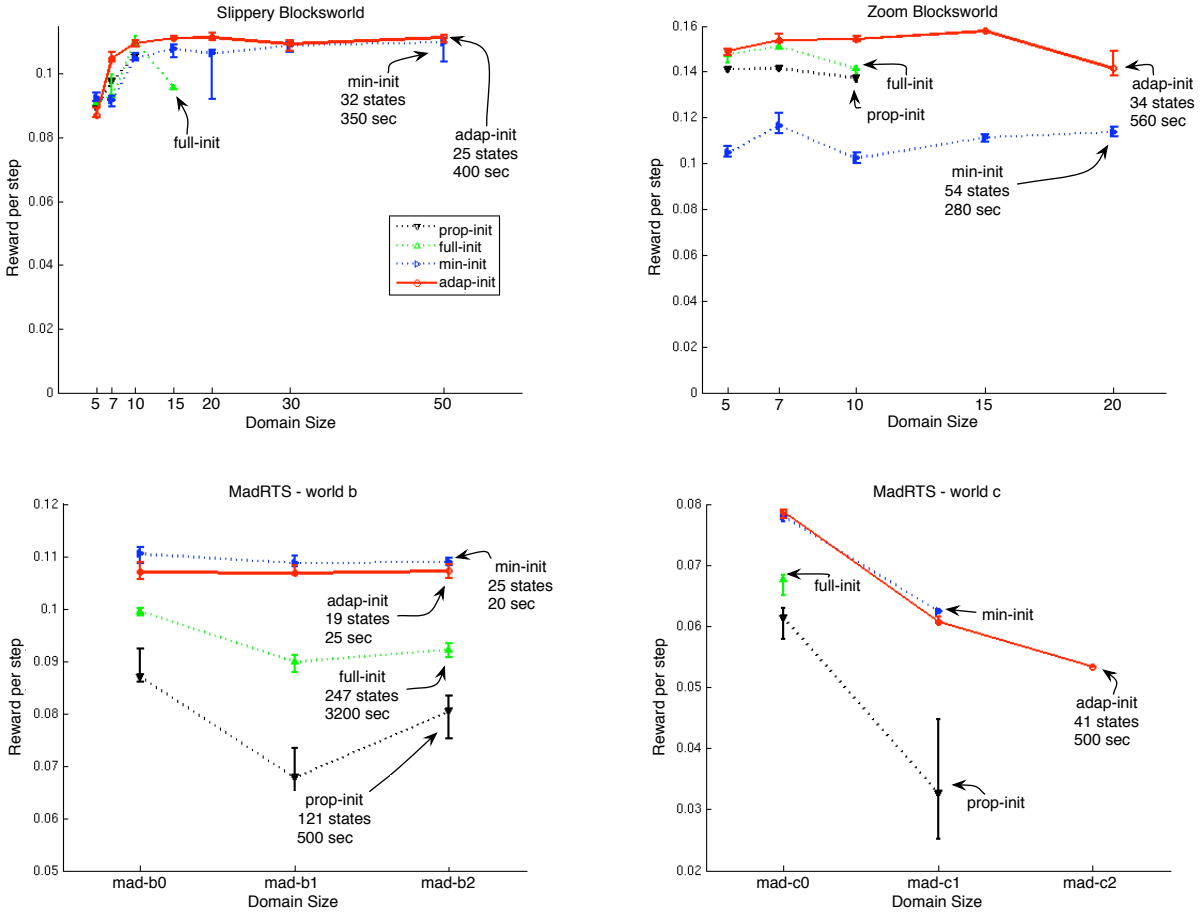

Figure 6: Comparison of accumulated reward for the initial-plan-based algorithms.

Slippery

Domain Size	prop-init	full-init	min-init	adap-init
5	.089	.090	.092	.088
7	.093	.092	.091	**.104**
10	.105	.110	.105	.110
15	-	.092	.106	**.111**
20	-	-	.105	**.111**
30	-	-	.110	.110
50	-	-	.110	.111
			32 states	25 states
			350 sec	400 sec

Zoom

Domain Size	prop-init	full-init	min-init	adap-init
5	.140	.150	.105	.151
7	.140	.151	.115	.152
10	.138	.140	.101	**.153**
15	-	-	.110	**.155**
20	-	-	.114	**.143**
			54 states	34 states
			280 sec	560 sec

MadRTS - World B

Domain Size	prop-init	full-init	min-init	adap-init
b0	.086	.099	.110	.106
b1	.066	.090	.108	.106
b2	.080	.091	.108	.107
			25 states	19 states
			20 sec	25 sec

MadRTS - World C

Domain Size	prop-init	full-init	min-init	adap-init
c0	.061	.067	.077	.078
c1	.031	-	.062	.061
c2	-	-	-	**.054**
				41 states
				500 sec

Figure 7: Full numerical results corresponding to Figure 6.

2. Adapting the basis can yield more accurate and better performing model for a given MDP size.

3. Finding minimal basis representation, in conjunction with an initial plan, produces the highest expected value per number of states in the MDP.

In general, better policies are found when gradually elaborating an initial solution than are found by trying to solve a problem all at once. The equivalence classes further aid this elaboration because they constrain the sampling done in the envelope MDP during envelope expansion.

7. Related Work

The idea of selective abstraction has a rich history. Apart from the original work by Dean *et al.* (1995), our work is perhaps most closely related to that of Baum and Nicholson (1998), who consider approximate solutions to MDP problems by selectively ignoring dimensions of the state space in an atomic-state robot navigation domain. The work of Lane and Kaelbling (2002) also exploits the idea of not exploring all aspects of a problem at once, decoupling local navigation from global routefinding with dedicated, approximate models for each.

Conceptually, the notion of abstraction by selectively removing predicates was explored early on in work by Sacerdoti (1974) and Knoblock (1994). THese approaches produce a hierarchy of "weakenings" from the ground problem up. Following explicitly in this vein is work by Armano *et al.* (2003), who describe an extension of PDDL that describes a hierarchy of problems, as well as a semi-automatic method for producing these descriptions.

8. Conclusions

We have described a technique for bootstrapping the solution of planning problems in uncertain domains by implementing envelope-based planning as an AMDP. The bootstrapping is done by taking advantage of a formalism for planning with equivalence classes of objects which is dynamic, domain-indpendent, and works under arbitrarily complex relational structure. We have also presented some experiments that show the advantage of this anytime approach to refinement of policy. To our knowledge, this is a novel approach to planning in relational domains, and the initial results presented show promise for planners of this kind.

However, since the work described in this paper is an initial step towards planners of this kind, there are many ways in which the approach could be improved.

8.1 Improving the forward search

As our implementation currently stands, the biggest bottleneck in the forward-search algorithm, which is implemented as a best-first search with random tie-breaking, is the heuristic evaluation of states. This is because we have modified the FF heuristic to be an *admissible heuristic*, which ultimately yields solutions of optimal length, but which involves searching the relaxed plan graph for the *shortest* relaxed plan. As a result, depending on the order of the search, this is a potentially exponential operation. We discuss these ramifications next.

8.1.1 Impact of action commutativity

In logistics-style domains, there is a greater potential for actions that could be executed in parallel, or, irrespective of order. For example, in a sequential plan, it may not matter which we do first: either hoist a crate at the first depot, or move a truck towards the depot. While our equivalence-class analysis eliminates such permutations among objects of the same class (and thereby realizes considerable efficiency gains), it does not do so for structurally distinct objects, nor does it eliminate permutations in order among parallelizable action sequences. As a result, the heuristic computation in these domains, because it seeks to optimize for the shortest path, becomes more costly.

This issue is discussed by Haslum and Geffner (2000) and Korf (1998), who suggest a solution based on imposing a fixed ordering on such actions. If the REBP approach is to be extended efficiently into a greater variety of domains, this is one issue that must be addressed.

8.1.2 Other admissible heuristics

There are other admissible heuristics that may be more efficient to compute than our adaptation of the FF heuristic. For instance, Haslum and Geffner describe a family of heuristics that trades off informativeness with efficiency of computation (2000). Edelkamp (2001) surveys this approach and others but finds that, in heuristic search planning, the heuristics are either not admissible, or, admissible but too weak. Edelkamp proposes by contrast an approach based on pattern databases(Edelkamp, 2001; Korf, 1998). Pattern databases are pre-computed tables of distances between abstractions of states. Edelkamp's approach is appealing in that it finds optimal plans if possible, and approximates the optimal solution in more challenging planning problems (in propositional, deterministic settings); however, space consumption grows rapidly — for example, searching for solutions in benchmark blocks-world domains of more than 13 blocks grinds to a halt the various optimal, general planners studied.

Thus, how to improve the efficiency of the heuristic for a larger variety of domains while preserving its informativeness is a key open question. It is also important to recognize that no one heuristic is best suited for all types of domains — some may be more effective in logistics-style domains, others for puzzle-style domains, and so on.

8.1.3 Considering non-optimal planning

The impact of REBP seems largest in the area of optimal planning, in which the shortest solution must be found, since REBP provides a way to reduce branching factor without losing optimality.

However, in Edelkamp's study (2001), as well as in our own experience and that of other researchers, FF's approach to finding approximate (non-optimal) solutions via hill-climbing is a time-effective approach in large problem instances. The experiments presented in this work were constructed to illustrate the properties of our algorithm specifically when planning difficulty is a function of increased problem size and not of increased solution length; but, there are vast numbers of planning problems out there who are not necessarily guaranteed to scale in this way.

One way to move in this direction may well be to use object equivalence classes in conjunction with the non-admissible FF heuristic. This would produce a plan faster, but it could be a longer solution. However, after discovering this initial solution, REBP can then invoke the anytime envelope expansion phase, which may proceed to discover a shorter solution given more computational resources. This is a tactic we have not yet explored.

8.2 More aggressive approximations

The purpose of minimizing the set of predicates used to represent the planning problem was to force more objects into the same equivalence class and, thus, approximate the original planning problem with a smaller one. This approach turned out to be effective in our experiments, but it is only a rudimentary start based on a simple syntactic analysis of the goal sentence and action preconditions. It may be profitable to investigate other ways of calculating approximate representations. Learning may play an important role here, as discussed in a later section.

Furthermore, as problems increase in size and complexity, it will be necessary to consider approximations to the isomorphism-based equivalence we have developed for objects. For example, approximate graph isomorphism is an idea investigated in a recent paper by Fox *et al.* (2007). Additionally, at the risk of including an even more complex problem into the inner loop of our algorithm, it may be possible to find a more generalizable notion of equivalence in considering isomorphism or approximate isomorphism over *substructures* of graphs rather than whole graphs. Finally, one might consider other types of distance metrics on graphs, such as kernel functions of structured data (Vishwanathan & Smola, 2004; Haussler, 1999; Gartner, Driessens, & Ramon, 2003; Gartner, Lloyd, & Flach, 2004). While a distance function on states might put more actions into the same equivalence class, it is less obvious how to use it to produce object equivalence classes, or whether it would be compatible with the incremental computation of object equivalence classes we have described.

8.3 Improving the envelope expansion

In the original paper on the Plexus algorithm (1995), Dean *et al.* describe various techniques for estimating the value of incorporating a new state into the envelope MDP. If a candidate state is not expected to improve the expected value much, then it is not added to the envelope. Our implementation is simpler: all candidate states are accepted during envelope expansion. Thus, there is nothing to stop the envelope from growing, in the limit, to the size of the abstract state space. Obviously, this is a concern in large domains, and it would be beneficial to incorporate some kind of value analysis to this step.

Furthermore, now that we are dealing with relational domains — instead of the atomic-state domains of the original Plexus algorithm — it may be that a factored or hierarchical approach to the the envelope and basis expansion would be worth considering. For example, it may make little sense to search for refinements between trucks, routes, and crate-contents all at the same time, as is currently done.

The basis expansion, in particular, may benefit from a sensitivity analysis. In the current implementation, we base our search for the refinement of the predicate set based on the width of the transition probability intervals. However, if this wide interval is in a part of

the state space with relatively low risk or reward, why bother refining it? It would be more worthwhile to refine instead those intervals with the greatest impact on the value estimate.

Finally, there is the question of how to most efficiently compute the transition probability intervals. We have chosen a method based on sampling from the underlying state space, which, while having the advantage of simplicity and fidelity to the state space in question, comes with a computational cost. It may be worth considering computing these intervals analytically, by syntactic analysis of the rule schemas.

8.4 The role of learning

Learning is absent from REBP, but there are many aspects which might benefit from it. For example, the idea of reducing the number of actions under consideration by minimizing the basis predicate set brings to mind the idea of *affordances* (Gibson, 1977); that is, the types of actions that are possible to effect on an object. There is the idea that people are able to restrict the number of affordances they consider for an object as a function of their motives (Bernedo-Schneider, 2006). Would it be possible to learn to "see" objects as equivalent, given the role that they play in eventually achieving a goal? This may yield more adaptive approaches than our current idea of computing a minimal basis set of predicates before beginning to plan.

8.5 Completeness, correctness, convergence, and complexity

Finally, what can we say about the computational characteristics of the full REBP algorithm?

We have theoretical guarantees on the planning side as long as we use the full predicate set. What happens when we use a reduced predicate set? The computation of the reduced predicate set simply guarantees that we will be able to solve a relaxed plan, but it guarantees nothing about being able to solve the non-relaxed version. A mechanism for dealing with possible failures of this type will need to be investigated.

This approximation of the state space also affects the AMDP computation. What are the convergence properties for an abstract, interval envelope Markov decision process? It seems plausible to expect so, but, we have not proved whether Dean and Givan's analysis for bounded-parameter MDPs (Givan et al., 1997) holds this case.

A thorough complexity analysis of the algorithm is also open. On the forward-search side: the algorithm is, in the worst case, exponential in the branching factor. On the AMDP side, the greatest bottleneck is in testing for state equivalence when doing the sampling to improve the transition probability interval estimates. However, computing isomorphisms is not a limiting bottleneck in practice (Miyazaki, 1997). What we have observed is that, if an isomorphism is not present, the computation tends to fail quickly. Thus, we only seem to pay the full cost of the computation in the case where the cost is able to be offset by its long-term benefits. Being able to characterize and guarantee this observation, however, is important and open future work.

Nonetheless, it seems hard to avoid computational complexity when dealing with inherently hard problems such as planning, which is at least PSPACE-complete even in the propositional, deterministic case (Bylander, 1994). The best we can hope for is not to let unnecessary complexity get the better of us. The work described here is one step towards that goal.

References

Armano, G., Cherchi, G., & Vargiu, E. (2003). Generating abstractions from static domain analysis. In *WOA 2003 (Dagli Oggetti agli Agenti, Sistemi Intelligenti e Computazione Pervasiva)*.

Baum, J., & Nicholson, A. (1998). Dynamic non-uniform abstractions for approximate planing in large structured stochastic domains. In *5th Pacific Rim Int'l Conference on Artificial Intelligence*.

Bernedo-Schneider, G. (2006). Cognitive modeling of motivations for artificial agents. In Rome, E., Doherty, P., Dorffner, G., & Hertzberg, J. (Eds.), *Towards Affordance-Based Robot Control*, No. 06231 in Dagstuhl Seminar Proceedings, Abstracts Collection. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Dagstuhl, Germany.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, *69*.

Dean, T., Kaelbling, L. P., Kirman, J., & Nicholson, A. (1995). Planning under time constraints in stochastic domains. *Artificial Intelligence*, *76*.

Dean, T., & Boddy, M. (1988). An analysis of time-dependent planning. In *AAAI-88*.

Edelkamp, S. (2001). Plannning with pattern databases. In *European Conference on Planning*.

Fikes, R. E., & Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189–208.

Fox, M., Long, D., & Porteous, J. (2007). Discovering near symmetry in graphs. In *Proceedings of AAAI*.

Gardiol, N. H. (2007). *Relational Envelope-based Planning*. Ph.D. thesis, MIT, Cambridge, MA. MIT-CSAIL-TR-2007-061.

Gardiol, N. H., & Kaelbling, L. P. (2007). Action-space partitioning for planning. In *National Conference on Artificial Intelligence (AAAI)* Vancouver, Canada.

Gartner, T., Driessens, K., & Ramon, J. (2003). Graph kernels and gaussian processes for relational reinforcement learning. In *Thirteenth International Conference on Inductive Logic Programming (ILP-2003)*.

Gartner, T., Lloyd, J. W., & Flach, P. A. (2004). Kernels and distances for structured data. *Machine Learning*, *3*(57).

Gibson, J. J. (1977). The theory of affordances. In Shaw, R., & Bransford, J. (Eds.), *Perceiving, Acting, and Knowing*. Lawrence Erlbaum Associates.

Givan, R., Leach, S., & Dean, T. (1997). Bounded parameter Markov decision processes. In *Proceedings of the European Conference on Planning*.

Givan, R., Leach, S., & Dean, T. (2000). Bounded parameter Markov decision processes. *Artificial Intelligence*.

Haslum, P., & Geffner, H. (2000). Admissible heuristics for optimal planning. In *AIPS*.

Haussler, D. (1999). Convolution kernels on discrete structures. Tech. rep. UCSC-CRL-99-10, University of California at Santa Cruz.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *JAIR*, *14*.

Knoblock, C. A. (1994). Automatically generating abstractions for planning. *AIJ*, *68*.

Korf, R. (1998). Finding optimal solutions to Rubik's cube using pattern databases. In *AAAI*.

Kushmerick, N., Hanks, S., & Weld, D. (1995). Algorithm for probabilistic planning. *Artificial Intelligence*, *76*.

Lane, T., & Kaelbling, L. P. (2002). Nearly deterministic abstractions of markov decision processes. In *18th AAAI*.

Majercik, S. M., & Littman, M. L. (2003). Contingent planning under uncertainty via stochastic satisfiability. *Artificial Intelligence*, *147*.

Miyazaki, T. (1997). The complexity of McKay's canonical labeling algorithm. *Groups and Computation II*, *28*.

Molineaux, M., & Aha, D. W. (2005). TIELT: A testbed for gaming environments. In *National Conference on Artificial Intelligence (AAAI)*.

Puterman, M. (1994). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *AIJ*, *5*, 115–135.

Vishwanathan, S., & Smola, A. (2004). Fast kernels for string and tree matching. In Schölkopf, B., Tsuda, K., & Vert, J. (Eds.), *Kernel Methods in Computational Biology*. MIT Press.

Younes, H., & Littman, M. (2004). PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Tech. rep. CMU-CS-04-167, Carnegie Mellon.