Adaptive Envelope MDPs for Relational Equivalence-based Planning

Natalia H. Gardiol and Leslie Pack Kaelbling

MIT CSAIL
Cambridge, MA 02139
{nhg|lpk}@csail.mit.edu

Abstract. We describe a method to use 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 unadulterated 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, then, 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 [**?**]. Anytime algorithms can operate either off-line (working until a specified time limit) or by interleaving refinement with execution.

Consider the problem of going to the airport. There are many features that might exist in 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 simple: first, find a plan; then, elaborate the plan. One early technique in this vein was proposed by Dean *et al.* [1], who introduced the idea of Envelope MDPs, used in an algorithm called Plexus. Given a planning problem represented as an atomic-state MDP [2], 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 trouble, we might take our shoes into consideration from the outset.

Our technique will start with the basic idea mentioned above, but with additional steps for efficiently handling 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 adding new states to an existing envelope, the Relational Envelope-based Planning approach (REBP), provides a framework for also adding new *dimensions* to the plan.

2 Compactly modeling large problems

A well-specified 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 in the world, 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) [3], which extends the classical STRIPS language [**?**,**?**] to probabilistic domains. We will use the term “rule” or “operator” when we mean an unground rule such as it appears in the domain description, and we will use “action” to denote a ground instance of a rule.

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. [4]

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:

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

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

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:

$$\forall B.\mathit{type}(B,\mathit{block}) \wedge on(B,\mathit{table}), \text{ or}$$

$$\exists B1.\mathit{type}(B1,\mathit{block}) \wedge \exists B2.\mathit{type}(B2,\mathit{block}) \wedge B1 \neq B2 \wedge on(B1,B2) \wedge on(B2,\mathit{table}).$$

Unless the context is ambiguous, we will leave future type specifications implicit in the text.

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.

Similarly, we define a relational MDP (RMDP) as a tuple $\langle \mathcal{P}, \mathcal{Z}, \mathcal{O}, \mathcal{T}, R \rangle$, thereby defining an analogous 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: The set of ground actions, likewise, is induced, by the set of rules $\mathcal{Z}$ and the objects in the world.

Transition Dynamics: For the transition dynamics, we use a compact set of rules $\mathcal{Z}$ as given in the domain description. A rule applies 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 $R(s)$.

Given these foundations, we can now begin to put together our approach.

3 Equivalence-based planning

The first requirement for an envelope MDP approach in the relational case is a method for finding the initial envelope of states. Equivalence-based planning [5] was originally developed as a way of speeding up planning by identifying equivalence classes of actions in relational MDPs. The algorithm uses a graph isomorphism-based definition of equivalence among states. It works by representing each class of states and actions by a single canonical exemplar and planning only within the space of exemplars, rather than in the original, ground state and action spaces. This technique produces a provably complete planning algorithm and can keep the number of effective actions from increasing combinatorially in large domains.

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 (properties) that apply to them. Two actions applicable in a given state s are defined to belong to be equivalent if they transition to equivalent successor states.

In this paper, we extend this idea by allowing the representation 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.

Specifically, we begin the planning process by first making a deterministic approximation of the original planning problem. Operators are assumed to deterministically produce their most likely outcome. Next, we identify the minimal set of predicates, or basis, necessary to solve a relaxed version of the problem. We mean "relaxed" in the

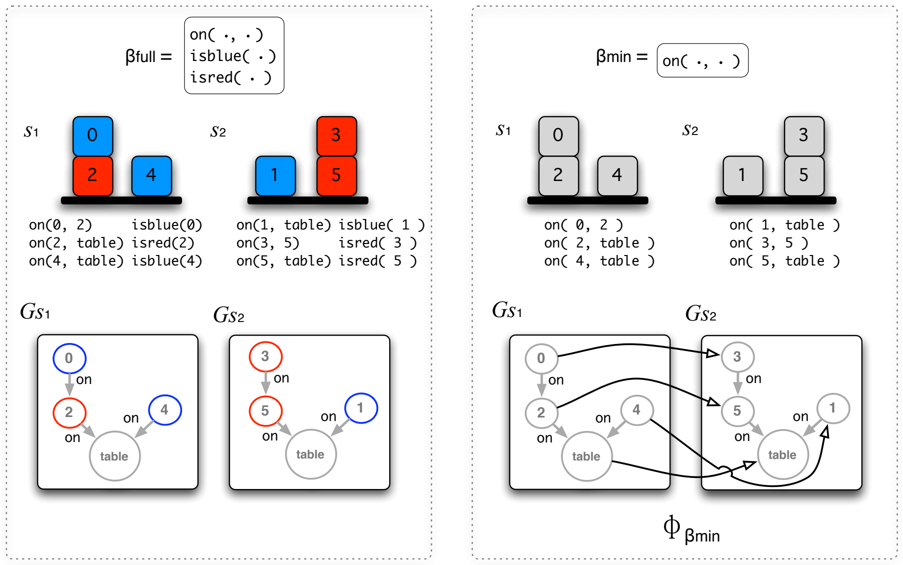

Fig. 1. In the original set of predicates representing the problem, β_{full}, states s_1 and s_2 are distinct. But in the reduced basis β_{min}, without colors, an isomorphism Φ can be found between the state relation graphs G_{s1} and G_{s2}.

sense of the well-known Fast-Forward heuristic (FF) [**?**], which is the heuristic in use 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 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 1 for an example of formulating a ground state with different bases.

The forward-search algorithm for REBP with a minimal basis set is given in Algorithm 1. By extending the machinery of the envelope-deliberation framework, which we will see shortly, β can be later refined and augmented to improve policy robustness.

4 Computing an abstract envelope

Now we will use the output of the deterministic planning process to bootstrap an MDP and directly address uncertainty in the domain. The output of the planning phase is a sequence of canonical actions, which corresponds to 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.

```
Input: Initial state $s_0$, goal condition $g$, set of rules $Z$
Output: Sequence of actions from $s_0$ to a goal state
1. Find a minimal representational basis, $\beta$
2. Find canonical initial state, $\tilde{s_0}$
2. 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: REBP forward-search algorithm. $\mathcal{A}'$ denotes the set of *canonical* actions, each of which represents an equivalence class of actions.

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 in our value estimate.

Because each state in our MDP is an abstract state, we must allow for the possibility that the dynamics driving the transition between abstract states may differ depending on which underlying ground states are participating in the transition. Instead of a scalar probability, then, we represent each transition probability as an interval. Interval MDPs, and the corresponding Interval Value Iteration algorithm, were first shown by Givan *et al.* [6, 7].

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, $\mathcal{A}$, consists of 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 consistent with the above canonical state.

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 $\beta$
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 combined search space over all the ground states is combinatorially much larger. Without ground information around in some form, it will be impossible to determine how to modify the basis set later.

While each MDP state keeps around its underlying ground state for this purpose, 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

Now that we have a set of states $\mathcal{Q}$, we need to determine the possible transitions between the states in $\mathcal{Q}$. 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, 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.[1] *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 2 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. The deterministic planning process returns a sequence of actions: $pickup(3, table)$, $put(3, 1)$. The abstract envelope MDP has been

[1] 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.

computed from this plan to produce the set of states in the figure, The various parts of the AMDP are labeled, and the collection S_i of each abstract state contains only a single element at this point.

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

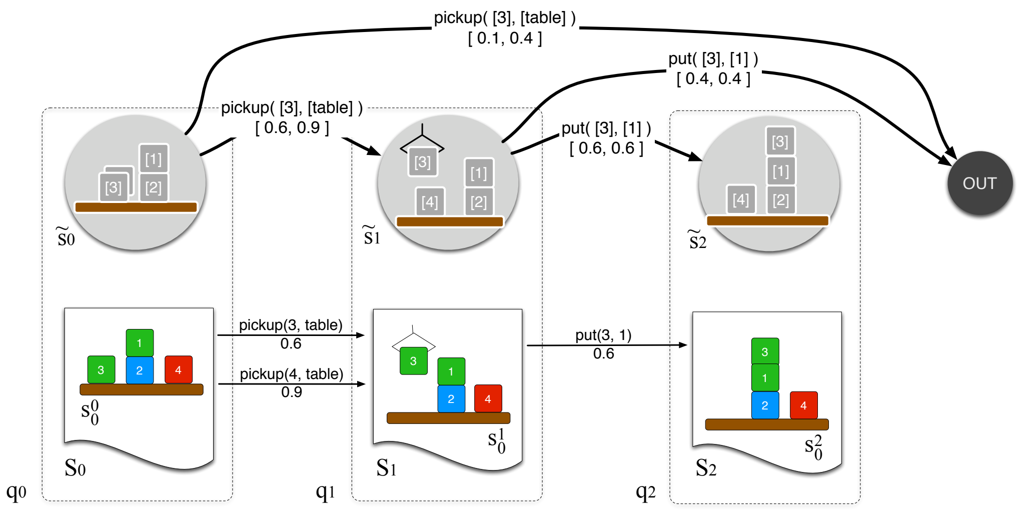

Fig. 2. The abstract MDP after computing the nominal transition probabilities. For simplicity, we only show a subset of the possible actions. For example, the action $pickup([1], [2])$, which is applicable in state $\tilde{s}_0$, would have a transition to OUT with probability $[1.0, 1.0]$ since there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

5 Changing the representation

At this point, we have constructed our MDP with an abstract state space and with probabilities represented as intervals. We may like, however, to add a predicate, or set of predicates, into the basis β in order to tighten these intervals and lessen the uncertainty in our 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.

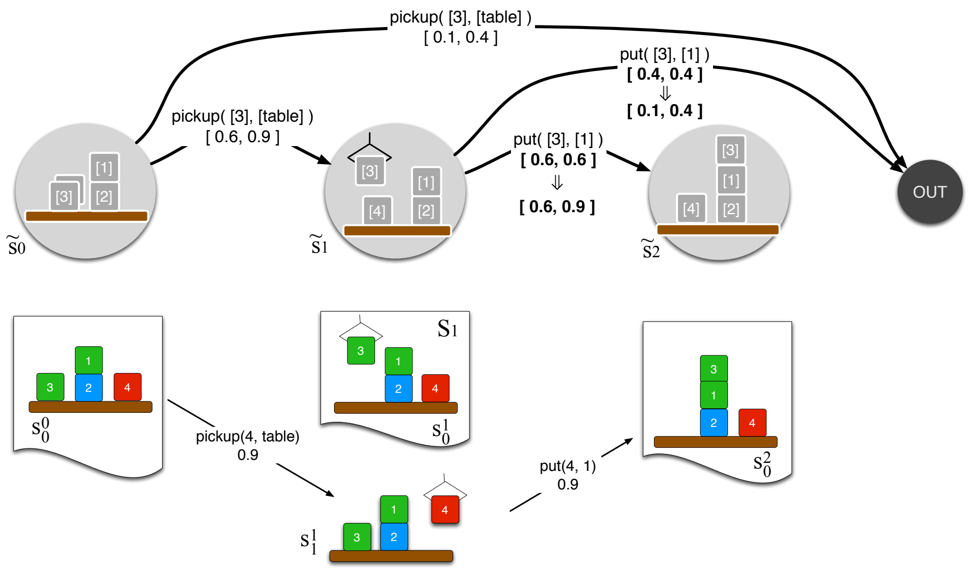

Fig. 3. The abstract MDP after a round of sampling to update transition interval estimates. State $s' = s_1^1$ was sampled from s_0^0, and it yielded a transition probability to $\tilde{s}_2$, outside of the current estimate of P_{120} (assuming the action $put([3],[1])$ has index 0 for $\tilde{s}_1$).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

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 our 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.

The basic mechanism is to add a procedure, *proposeBasis(*r, β*)*, which takes as an argument a rule r and the current basis β, and returns a list of candidate predicate sets to be added to the basis. What does it mean for an operator to propose a predicate set? Consider the $pick-up$ 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 principal place in the algorithm in which to refine the representation is as a part of the existing envelope-refinement loop. In this loop, we keep a sorted list of the transitions in our MDP. Currently, we sort transitions in descending order by width of the interval; i.e., the maximally uncertain transitions are at the top of the list.[2] Then, when we need to suggest a refinement, we start with the transition at the top of the list and request the associated operator's proposal.

The second opportunity comes when we reach a representational "failure" point. In the process of sampling from actions that were not originally in our optimistic plan, and, thus, made no contribution the original choice of basis, computing their effects might have unexpected results. This becomes obvious when we produce an outcome state that has no applicable actions. We call this a "failure" point, and we deal with it as follows. First, we remove, as much as possible, the trajectory that leads to this state. We do this by iterating backward from the failed state until we either reach the initial state, or, a state that has more than one incoming transition. At this point, we re-route that single outgoing transition to the OUT state. We set a flag that disallows any future sampling from that action. Then, starting from the offending state, we work our way backwards through the transitions until we find a transition with that has a non-empty predicate set to propose. If we do find one, we add this proposal to the list of candidate predicate sets. Then, the next time the MDP considers a new proposal, it selects from this list.

Once we have a proposal to try, we initialize a *new* MDP using the original plan and the *new* basis. Then, the regular phases of policy improvement and envelope expansion happen for both of them in parallel. We can add as many parallel MDPs as desired. In our current implementation, we set the limit at n=5 interval MDPs in parallel. The reason for keeping a number of MDPs in parallel is that the basis-refinement algorithm is greedy. The first time a proposal is requested from the first abstract MDP 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 MDP traks its proposals and does not make 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 MDPs 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 MDP with the highest policy value. The general REBP algorithm 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-valued policy with as compact a model as possible. We will look at the various ways of combining the techniques described in

[2] 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, $\beta$
Let plan P = REBPForwardSearch( $s_0$, $g$, $Z$ )
begin Initialize envelope MDP $M$ with $P$ and $\beta$ :
    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, $\beta'$, corresponding to the sequence
            of actions between $q'$ and $q_0$
            if $\beta'$ 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 $\beta'$ from the top transition
            if $\beta'$ not empty then append $\beta$ to end of list $l_i$

        Do interval value iteration in $M_i$ until convergence
        if $l_i$ not empty then
            Select a basis $\beta'$ from the list
            Construct a new MDP $M'$ with plan $P$ and basis $\beta'$.
            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.

this work, and we'll try to identify 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. [3] The world consists of a map (of varying size; six territories in the b 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

[3] 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 challenge problem [8]. While no longer taking place in a real-time system, we call this planning domain the MadRTS domain to signal this origin.

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 [4].

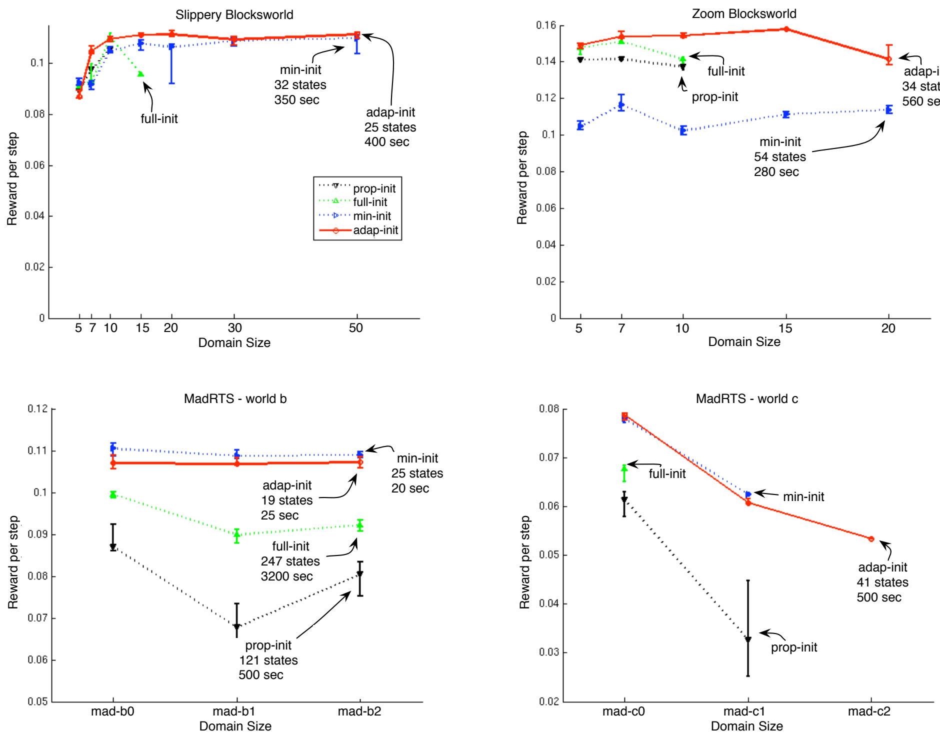

Fig. 4. Comparison of accumulated reward for the initial-plan-based algorithms.

Figure 4 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-

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

Fig. 5. Full numerical results corresponding to Figure 4.

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.
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.* [1], our work is perhaps most closely related to that of Baum and Nicholson [9], 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 [10] 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 [11] and Knoblock [12]. THese approaches produce a hierarchy of "weakenings" from the ground problem up. Following explicitly in this vein is work by Armano *et al.* [13], 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 interval MDP. 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.

References

1. Dean, T., Kaelbling, L.P., Kirman, J., Nicholson, A.: Planning under time constraints in stochastic domains. Artificial Intelligence **76** (1995)
2. Puterman, M.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons (1994)
3. Younes, H., Littman, M.: PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, Carnegie Mellon (2004)
4. Gardiol, N.H.: Relational Envelope-based Planning. PhD thesis, MIT, Cambridge, MA (2007) MIT-CSAIL-TR-2007-061.
5. Gardiol, N.H., Kaelbling, L.P.: Action-space partitioning for planning. In: National Conference on Artificial Intelligence (AAAI), Vancouver, Canada (2007)
6. Givan, R., Leach, S., Dean, T.: Bounded parameter Markov decision processes. In: Proceedings of the European Conference on Planning (ECP-97). (1997)
7. Givan, R., Leach, S., Dean, T.: Bounded parameter Markov decision processes. Artificial Intelligence (2000)
8. Molineaux, M., Aha, D.W.: TIELT: A testbed for gaming environments. In: National Conference on Artificial Intelligence (AAAI). (2005)
9. Baum, J., Nicholson, A.: Dynamic non-uniform abstractions for approximate planing in large structured stochastic domains. In: 5th Pacific Rim International Conference on Artificial Intelligence. (1998)
10. Lane, T., Kaelbling, L.P.: Nearly deterministic abstractions of markov decision processes. In: 18th National Conference on Artificial Intelligence (AAAI-2002). (2002)
11. Sacerdoti, E.D.: Planning in a hierarchy of abstraction spaces. Artificial Intelligence **5** (1974) 115–135
12. Knoblock, C.A.: Automatically generating abstractions for planning. Artificial Intelligence **68** (1994)
13. Armano, G., Cherchi, G., Vargiu, E.: Generating abstractions from static domain analysis. In: WOA 2003 (Dagli Oggetti agli Agenti, Sistemi Intelligenti e Computazione Pervasiva). (2003)