Anytime Integrated Task and Motion Policies for Stochastic Environments

Naman Shah and Siddharth Srivastava
School of Computing, Informatics, and Decision Systems Engineering,
Arizona State University, Tempe, AZ, USA
{namanshah,siddharths}@asu.edu

Abstract

In order to solve complex, long-horizon tasks, intelligent robots need to be able to carry out high-level, abstract planning and reasoning in conjunction with motion planning. However, abstract models are typically lossy and plans or policies computed using them are often unexecutable in practice. These problems are aggravated in more realistic situations with stochastic dynamics, where the robot needs to reason about, and plan for multiple possible contingencies. We present a new approach for integrated task and motion planning in such settings. In contrast to prior work in this direction, we show that our approach can effectively compute integrated task and motion policies with branching structure encoding agent behaviors for various possible contingencies. We prove that our algorithm is probabilistically complete and can compute feasible solution policies in an anytime fashion so that the probability of encountering an unresolved contingency decreases over time. Empirical results on a set of challenging problems show the utility and scope of our methods.

1 Introduction

In order to solve complex tasks, autonomous robots need to be able to compute high-level strategies consistent with low-level constraints such as object geometries, the robot's own joint limits, stability constraints, etc. This becomes more complex when the environment or the robot's actions are stochastic. For instance, consider the problem where a robot needs to pick up a can (black) from a cluttered table (Fig. 1). In order to achieve this objective, the robot needs to consider multiple contingencies, e.g., what if the can slips? What if it tumbles and rolls off when it is placed?

This situation, and the need to efficiently manage contingencies is representative of many real-world situations. In order to safely accomplish tasks such as diffusing IEDs, operating live machinery, or assisting emergency response personnel, it is desirable to pre-compute contingent plans, or policies in order to reduce the need for on-the-fly replanning. These situations require the computation of high-level strategies that can be realized with physical movements motion plans for each high-level action.

One of the main challenges in addressing computing such policies is that in general, their size grows exponentially with the solution depth. Naive approaches that first compute the policy and then refine each "branch" are computationally intractable as many branches may have no feasible motion plans. Furthermore, refining exponentially many paths to the goal would be computationally intractable. We believe this is one of the reasons for the absence of solution approaches for computing task and motion policies.

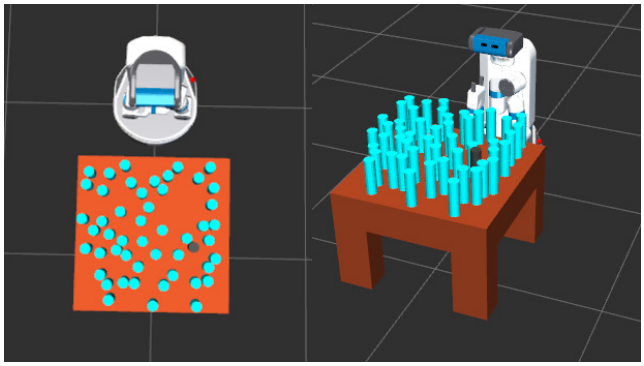

Figure 1: A stochastic variant of the cluttered table domain where robot needs to pick up the black can, but pickups may fail.

In this paper, we present the first *probabilistically complete* algorithm for computing integrated task and motion policies in stochastic environments using a relational input representation. We model the overall problem as an abstract Markov decision process (MDP), where each action in the MDP corresponds to a set of lower-level (motion planning) problems. High-level action outcomes may be stochastic due to unmodeled properties (e.g. the weight of a can and its coefficient of friction against the gripper material), or due to the dynamic nature of the domain (e.g., actions of other persons in the environment).

The overall problem is to compute a policy for the MDP along with "refinements" that select, for each action in the policy, a specific motion planning problem and its solution. E.g., the "high-level" action for picking up a can (Fig. 1) corresponds to infinitely many motion planning problems, each defined by a target specific grasping pose of the gripper and target pose for the can after it is picked up. The refinement process would thus need to associate a specific pair of target poses and a motion plan for each occurrence of the pickup action in the computed policy.

Our approach uses off-the-shelf MDP planners with off-the-shelf motion planners. The use of relational representations allows us to easily express problems involving object manipulation, which would be cumbersome if not infeasible in propositional representations. The use of off-the-shelf MDP planners and motion planners allows our approach to scale automatically with improvements in those fields. We show that our approach has a desirable *anytime* property that makes it possible to tune the amount of precomputation carried out, thereby alleviating the computational challenges discussed above. Our experiments indicate the probability of encountering an unresolved contingency drops exponentially as the algorithm proceeds.

We begin with a presentation of the background definitions (§2) and our formal framework (§3). §4 describes our overall algorithmic approach, followed by a description of empirical results using the Fetch robot in simulation (§5), and a discussion of other related work (§6).

2 Background

A fully observable, deterministic *task planning problem* is a tuple $\langle A, s_0, g\rangle$, where A is a set of propositional actions that are parameterized and defined by preconditions and effects, s_0 is an initial state of the domain, and g is the goal condition which is also a set of propositions. A sequence of actions $a_0, ..., a_n$ executed starting from s_0 will generate a state sequence $s_1, ..., s_{n+1}$, where $s_{i+1} = a_i(s_i)$ is the result of executing a_i in s_i. Solving the task planning problem is to find out the sequence of actions s_i which satisfies the preconditions of a_i for $i = 0, ..., n$ and s_{n+1} satisfies g.

A *motion planning problem* is a tuple $\langle C, f, p_0, p_t\rangle$, where C is the space of possible configurations or poses of a robot, f is a boolean function which determines whether or not a $pose$ is in a collision and p_0, $p_t \in C$ are the initial and final poses. A trajectory is a a sequence of way-points (joint values). A collision-free motion plan solving a motion planning problem is a trajectory in C from p_0 to p_t such that f is false for any pose in the trajectory.

A *Markov decision process* (MDP) is defined as a tuple (S, A, T, R, γ) where S is a set of states; A is a set of possible of actions; $T(s, a, s') = P(s'|s, a)$ for $s, s' \in S, a \in A$; $R(s, a, s')$ is a reward function for $s, s' \in S, a \in A$; γ is the discount factor.

A Solution to an MDP is a *policy*, $\pi : S \rightarrow A$, which maps each state to an action. We are more specifically interested in a subclass of MDPs that have absorbing states, $\gamma = 1$ and a finite horizon. Such MDPs are known as stochastic shortest path (SSP) problems (Bertsekas and Tsitsiklis 1991). An SSP can be defined as a tuple $(S, A, T, C, \gamma = 1, H, S_0, G)$ where S,A,T are as described as above. In addition to that, $C(s, a)$ is the cost for action $a \in A$ in state $s \in S$; H is the length of horizon; S_0 is the initial state; G is the set of absorbing or goal states;

A Solution to an SSP is a policy π of the form $\pi : S \times \{h_0, h_1, \ldots, h_n\} \rightarrow A$ which maps all the states and time steps at which they are encountered to an action. The optimal policy π^* is a policy which reaches the goal state with the least expected cumulative cost. In general, policies for SSPs are not stationary as the horizon is finite. Dynamic programming algorithms such as value iteration or policy iteration can be used to compute these policies. Value iteration can be defined as:

$$V^0(s) = C(s) \quad (1)$$

$$V^i(s) = min_a \sum_{s'} T(s, a, s')R(s) + V^{i-1}(s') \quad (2)$$

$$\pi^i(s) = argmin_a \sum_{s'} T(s, a, s')R(s) + V^{i-1}(s') \quad (3)$$

Non-stationary policies for finite-horizon SSPs can be represented as finite-state machines (FSMs). Given an upper bound on the time horizon, any policy over a finite state and action set can be unrolled into a tree-structured FSM.

Several representations have been developed for efficiently representing the MDPs, such as *Relational Dynamic Influence Diagram Language* (RDDL) (Sanner 2010), *Probabilistic Planning Domain Definition Language* (PPDDL) (Younes and Littman 2004). These languages separate an MDP domain, which consists of parameterized actions, functions, and predicates, from an MDP Problem, which expresses the objects, an initial state and a goal that needs to be achieved. Without loss of generality, we use PPDDL to represent SSPs in this paper.

Place(obj_1, $config_1$, $config_2$, $target\_pose$, $traj_1$)

precon	*RobotAt($config_1$)*, *holding(obj_1)*, *IsMP($traj_1$, $config_1$, $config_2$)*, *IsPlacementConfig(obj_1, $config_2$, $target\_pose$)*, ∀ obj' ¬ *Collision(obj', $traj_1$)*
concrete effect	¬*holding(obj_1)*, *RobotAt($config_2$)*, *at(obj_1, $target\_pose$)* ∀$traj$ *intersects(vol(obj ,target_pose), sweptVol(robot, traj)→ Collision(obj_1, traj)*
abstract effect	¬*holding(obj_1),RobotAt($config_2$)*, ∀$traj$ ⍰ *Collision(obj_1, traj)*

Figure 2: Concrete (above) and abstract (below) effects of a one-handed robot's action for placing an object.

3 Formal Framework

We introduce our formalization with an example.

Example 1. *Consider the specification of a robot's action of placing an item in the refrigerator. In practice, low-level accurate models of such actions may be expressed as generative models, or simulators, as was the case in our experiments. We show a declarative version in Fig. 2 to help identify the nature of abstract representations needed for expressing abstractions of such models. For readability, we use a convention where preconditions are comma-separated conjunctive lists and universal quantifiers represent conjunctions over the quantified variables.*

An accurate description of this action (Fig. 2) requires action arguments representing the object to be picked up (obj_1), the initial and final robot configurations ($config_1$, $config_2$), the $target\_pose$ of the object, and the motion planning trajectory $traj_1$ to be used. These arguments represent the choices to be made when placing an object. The

preconditions of *Place* capture the conditions that $traj_1$ is a collision-free motion plan or trajectory for moving from $config_1$ to $config_2$, and that $config_2$ corresponds to the object being at the target pose (such that opening the gripper would leave it at the target pose; we ignore the third configuration with an open gripper for ease in exposition). The concrete effect of *Place* states that the robot is no longer holding the object, the robot is in $config_2$ and that the object is in collision with all robot trajectories whose swept volume intersects with the object's volume at the target pose. The *intersects* predicate is static as it operates on volumes, while *Collision* can change with the state.

Intuitively, our approach replaces the domains of a subset of action arguments with singleton symbolic values that can be instantiated with values from their real domains to obtain the concrete actions. E.g., the possible robot configurations $config_2$ for placing an object obj are represented by the symbol $config\_obj$. Action effects on predicates over symbolic values can no longer be determined precisely; their values are assigned by the planning algorithm. E.g., it is not possible to determine at this level of abstraction which motion planning trajectories would get obstructed as a result of the placement action. Such predicates are annotated in the set of effects with the symbol ⍰, denoting imprecision due to abstraction (see the abstract effect in Fig. 2). The resulting model is a sound abstraction (Srivastava et al. 2014; Srivastava, Russell, and Pinto 2016).

Abstraction Framework In order to formalize such abstractions we first introduce some notation. We denote states as logical models or structures. We use the term *logical structures* or *structures* to distinguish the concept from SDM models. A structure S, of vocabulary $\mathcal{V}$, consists of a universe $\mathcal{U}$, along with a function f^S over $\mathcal{U}$ for every relation symbol f in $\mathcal{V}$ and an element $c^S \in \mathcal{U}$ for every constant symbol c in $\mathcal{V}$. We denote the value of a term or formula φ in a structure S as $[\![\varphi]\!]_S$. These values are either True, False, or elements of the universe of S. We also extend this notation so that $[\![f]\!]_S$ denotes the interpretation of the function f in S. We consider Boolean relations as a special case of functions.

We formalize abstractions by building on the notion of first-order queries (Codd 1972; Immerman 1998) that map structures over one vocabulary to structures over another vocabulary. In general, a first-order query α from V_ℓ to V_h defines functions in $\alpha(S_\ell)$ using interpretations of V_ℓ-formulas in S_ℓ: $[\![f]\!]_{\alpha(S_\ell)}(o_1, \ldots o_n) = o_m$ iff $[\![\varphi^\alpha_f(o_1, \ldots o_n, o_m)]\!]_{S_\ell} = \textit{True}$, where φ^α_f is a formula in the vocabulary V_ℓ.

In this notation, *function abstractions* or *predicate abstractions* are first-order queries where $V_h \subset V_\ell$; the predicates in V_h are defined as identical to their counterparts in V_ℓ. Such abstractions reduce the number of properties being modeled. *Entity abstractions*, on the other hand, reduce the number of entities being modeled. Such abstractions have been used for efficient generalized planning (Srivastava, Immerman, and Zilberstein 2011) as well as answer set programming (Saribatur, Schüller, and Eiter 2019). Let $\mathcal{U}_\ell$ ($\mathcal{U}_h$) be the universe of S_ℓ (S_h) such that $|\mathcal{U}_h| \leq |\mathcal{U}_\ell|$. We define entity abstractions using an auxiliary representation function $\rho : \mathcal{U}_h \to 2^{\mathcal{U}_\ell}$. Informally, ρ maps each element $\tilde{o}$ of $\mathcal{U}_h$ to the subset of $\mathcal{U}_\ell$ that $\tilde{o}$ represents. E.g., $\rho(\textit{Kitchen}) = \{\textit{loc} : \wedge_i\ \textit{loc} \cdot \textit{BoundaryVector}_i < 0\}$ where the kitchen has a polygonal boundary. An entity abstraction α_ρ using the representation ρ is defined as $[\![f]\!]_{\alpha_\rho(S_\ell)}(\tilde{o}_1, \ldots \tilde{o}_n) = \tilde{o}_m$ iff $\exists o_1, \ldots o_n, o_m$ such that $o_i \in \rho(\tilde{o}_i)$ and $[\![\varphi^{\alpha_\rho}_f(o_1, \ldots o_n, o_m)]\!]_{S_\ell} = \textit{True}$. We omit the subscript ρ when it is clear from context.

Let S be the set of abstract states generated when an abstraction function α is applied on a set of concrete states X. For any $s \in S$, the *concretization function* $\Gamma_\alpha(s) = \{x \in X : \alpha(x) = s\}$ denotes the set of concrete states *represented by the abstract state* s. For a set $C \subseteq X$, $[C]_\alpha$ denotes the smallest set of abstract states representing C. Generating the complete concretization of an abstract state can be computationally intractable, especially in cases where the concrete state space is continuous and the abstract state space is discrete. In such situations, the concretization operation can be implemented as a *generator* that incrementally samples elements from an abstract state's concretization.

Formally, our approach carries out a entity abstraction to yield compact, imprecise yet sound action descriptions (Srivastava, Russell, and Pinto 2016). The entity abstraction is notable in using a *dynamic representation function*. E.g., for the action in Fig. 2, $\rho(config\_obj) = \{config_1 : \varphi_{pre}(config, obj)\}$, where φ_{pre} is precondition for *Place* with existential quantifiers for the other continuous arguments. ρ is dynamic in the sense that the set of poses represented by $config\_obj$ varies with the state because the set of collision free trajectories depends on the state.

Definition 1. *A stochastic task and motion planning problem $\langle M, c_o, \alpha, [M]\rangle$ is defined using a concrete SSP model M and its abstraction $[M]$ obtained using a composition of function and entity abstractions, denoted as α.*

Solutions to task and motion planning problems, like solutions to SSPs, are policies with actions from M.

4 Algorithmic Framework

4.1 Overall Approach

We now describe our approach for computing task and motion policies as defined above. For clarity, we begin by describing certain choices in the algorithm as nondeterministic. Variants of our overall approach can be constructed with different implementations of these choices; the versions used in our evaluation are described in §4.2.

Recall that abstract grounded actions $[a] \in [\mathcal{M}]$ (e.g., *Place(cup, config1_cup, config2_cup, target_pose_cup, traj1_cup)*) have symbolic arguments that can be instantiated to yield concrete grounded actions $a \in \mathcal{M}$. If the argument instantiation satisfies the preconditions of a in a concrete state c, $\mathcal{M}$ can be used to compute the concrete effects of a on c. This process requires that it should be possible to evaluate each predicate instantiation in a low-level state.

Of course, doing this evaluation during the search for a plan can be prohibitively expensive: one would have to compute all possible instantiations of symbolic action arguments

and then use $\mathcal{M}$ to generate the next possible states. The whole purpose of abstraction is to avoid such operations since exploring the space of all possible argument instantiations and carrying out action propagation in $\mathcal{M}$ for each instantiation is computationally intractable, particularly if $\mathcal{M}$ is an arbitrary simulator.

Instead, we interleave computation among the processes of (a) *concretizing an abstract policy*, (b) *update abstraction for a fixed concretization*, and (c) *computing an abstract policy for an updated state*. This is done using the *plan refinement graph* (PRG), a graph that stores the different models, their corresponding abstract policies and partial refinements. Every node *u* in the PRG represents an abstract model $[\mathcal{M}]_u$, an abstract policy $[\pi]_u$ in the form of a tree whose vertices represent states and edges represent action applications, a concretization for a subset of action occurrences in $[\pi]_u$, and the current state of search for concretizations of all actions $a_j \in [\pi]_u$. Every edge *(u,v)* between nodes *u* and *v* in the PRG represents a failure reason φ for a particular concretization σ for $[\pi]_u$; $[\mathcal{M}]_v$ is the version of $[\mathcal{M}]_u$ updated with the failed preconditions corresponding to *(u,v)*.

Alg.1 carries out the interleaved search outlined above as follows. It first initializes the PRG with node containing an abstract policy for the given SSP (line 1), and then selects a node in the PRG and extracts an unrefined root-to-leaf path from the policy for that node (lines 3-5).

Concretization of an available policy Lines 7-13 search for a concretization (refinement) of the partial path by instantiating its symbolic action arguments (including the action refinement to use, e.g. $traj_1$) with values from their original non-symbolic domains, to obtain a feasible concrete policy $\{\pi_i\}$ using a motion planner with $\mathcal{M}$. However, it is possible that $[\pi]$ admits no feasible concretization because every instantiation of the symbolic arguments violates the preconditions of some action in π_i. A concretization $c_0, a_1, c_1, a_2, c_2 \ldots, a_k, c_k$ of the path $[s_0], [a]_1, [s]_1, [a]_2, [s]_2, \ldots, [a]_k, [s]_k$ is feasible starting with a concrete initial state c_0 iff $c_{i+1} \in a_{i+1}(c_i)\Gamma([s]_{i+1})$ for $i = 0, \ldots, k-1$. E.g., an infeasible path would have the robot placing a cup on the table in the concretete state c_0, when every possible motion plan for doing so may be in collision with other objects.

Update abstraction for a fixed concretization Lines 16-20 fix a concretization for the partially refined path selected on line 6, and identify the earliest state in this path whose subsequent action's concretization is infeasible. This state is updated with the true forms of the violated preconditions that hold in this concretization, using symbolic arguments. Discard the plan suffix after this state. E.g., *Collision(*$teapot, traj\_cup$*)*). A state update is immediately followed by the *computation of a new abstract policy* the computation of a new abstract policy.

Computation of a new abstract policy Lines 21-22 compute a new policy with the updated information computed under (b). The SSP solver is invoked to compute a new policy from the updated state; its solution policy is unrolled as a tree of bounded depth and appended to the partially refined path. This allows the time horizon of the policy to be increased dynamically.

In our implementation the *Compute* variable on line 6 is set to either *Concretization* or *UpdateAbstraction* with probability 0.5. The *explore* parameter on line 9 needs to be set with non-zero probability for a formal guarantee of completeness, although in our experiments it was set to False.

Algorithm 1: ATM-MDP Algorithm

Input: model $[\mathcal{M}]$, domain $\mathcal{D}$, problem $\mathcal{P}$, SSP Solver *SSP*, motionPlanner *MP*
Output: anytime, contingent task and motion policy

```
1  Initialize PRG with a node with an abstract policy [π] for P
     computed by SSP;
2  while solution of desired quality not found do
3      PRNode ← GetPRNode();
4      [π] ← GetAbstractPolicy([M], PRNode, D, P, SSP);
5      path_to_refine ← GetUnRefinedPath([π]);
6      Compute ← NDChoice{Concretization,
         UpdateAbstraction};
7      if Compute = Concretization then
8          while [π] has an unrefined path and resource limit is
             not reached do
9              if explore // non-deterministic
10             then
11                 replace a suffix of partial_path with a
                     random action;
12             end
13             search for a feasible concretization of
                 path_to_refine;
14         end
15     end
16     if Compute = UpdateAbstraction then
17         partial_path ← GetUnrefinedSuffix(PRNode,
             path_to_refine);
18         σ ← ConcretizeLastUnrefinedAction([π]);
19         failure_reason ← GetFailedPrecondition([π], σ );
20         updated_state ← UpdateState([π], failure_reason);
21         [π'] ← merge([π], solve(updated_state, G, [M]));
22         generate_new_pr_node([π'], [M]);
23     end
24 end
```

4.2 Optimizations and Formal Results

We discuss two major optimizations of Alg. 1 below.

Selecting the path to refine The main computational challenge for the algorithm is that the number of root-to-leaf (RTL) paths grows exponentially with the time horizon. Waiting for a complete refinement results in wasting a lot of time as the probability of encountering that situation has a very low probability for most of the paths. Each RTL path has a certain probability of being encountered; refining it incurs a computational cost. The optimal selection of the paths to refine within a fixed computational budget can be reduced to the knapsack problem. Unfortunately, however, we do not

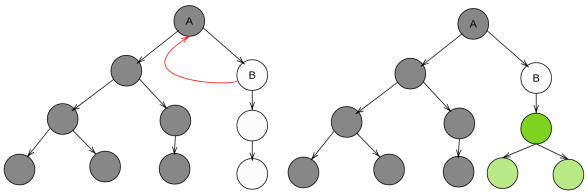

Figure 3: Figure: Left: Backtracking from node B invalidates the refinement of subtree rooted at A. Right: Replanning from node B which in some cases requires fewer resources.

know the precise computational costs required to refine a path. Furthermore, the knapsack problem is NP-hard. However, we can compute provably good approximate solutions to this problem using a greedy approach: we prioritize the selection of a path to refine based on the probability of the encountering that path *p* and the estimated cost of refining that path *c*. We use a priority queue for the RTL paths with their *p/c* values as the keys.

Theorem 1. *Let t be the time since the start of the algorithm at which the refinement of any root-to-leaf path is completed. If path costs are accurate and constant then the total probability of unrefined paths at time t is at most $1 - opt(t)/2$, where $opt(t)$ is the best possible refinement (in terms of the probability of outcomes covered) that could have been achieved in time t.*

The proof follows from the fact that the greedy algorithm achieves a 2-approximation for the knapsack problem. In practice, we estimate the cost as $\hat{c}$, the product of measures of the true domains of each the symbolic argument in the given RTL. Since, $\hat{c} \geq c$ modulo constant factors, the priority queue never can only underestimate the relative value of refining a path, and the algorithm's coverage of high-probability contingencies will be closer to optimal than the bound suggested in the theorem above. This optimization gives a user the option of starting execution when a desired value of the probability of covered contingencies has been reached.

Search for concretizations Sample-based backtracking search (Srivastava et al. 2014) for the concretizations of symbolic variables suffers from a few limitations in stochastic settings that are not present in deterministic settings. Fig. 3 illustrates the problem. In this figure, grey nodes represent actions in the policy tree that have already been refined; the refinement for *B* is being computed. White nodes represent the nodes that still require refinement. If backtracking search changes the concretization for B's parent (Fig. 3, left) it will invalidate the refinements made for the entire subtree of that node. Instead, it may be better to compute an entirely new policy for B (effectively jumping to the UpdateAbstraction mode of computation (line 16). We implement an optimization where the algorithm chooses between this alternative and backtracking to the parent node with probability 0.5.

Numerous additional optimizations could be used to further improve the performance of this approach in future work. In particular, better strategies and/or statistical learning could be used in place of the probabilistic choices in the search for concretizations and in the selection of the mode of computation (line 7). Thm. 2 shows that our algorithm is probabilistically complete.

Theorem 2. *If there exists a proper policy which reaches the goal within horizon* h *with probability* p*, and has feasible low-level refinement, then Alg. 1 will find it with probability* 1.0 *in the limit of infinite samples.*

Proof. Let π_p be the proper policy. Consider a policy π in the PRG; let k denote the minimum depth up to which π_p and π match. k will be used as a *measure of correctness*. When π's PRG node is selected, suppose we try to refine one of the child nodes of depth $k+1$ in the partial path that had the k-length prefix consistent with the solution. The algorithm selects the correct child action with non-zero probability under the *explore* steps (line 11), and then generates a plan to reach the goal from the resultant state. The finite number of discrete actions and the fixed horizon ensures that at in time bounded in expectation, ATM-MDP will generate a policy with the measure of correctness $k+1$. Once the algorithm finds the policy with the measure of correctness *h*, it stores it in the PRG and is guaranteed to find feasible refinements with probability one if the measure of these refinements under the probability-density of the generators is non-zero. □

5 Empirical Evaluation

We implemented the algorithms presented in previous sections using an implementation of LAO* (Hansen and Zilberstein 2001) as the SSP solver, the OpenRAVE (Diankov 2010) system to model and visualize test environment and its collision checkers and BiRRT implementation for Motion Planning. Since there are no common benchmarks for evaluating stochastic task and motion planning problems, we evaluated our algorithm on three test problems geared towards evaluating the systems performance on a range of scenarios. In practice, coming up with an exact number for horizon h is not possible. To overcome that, we implemented a variant which dynamically increases the horizon until the goal is reached with probability $p > 0$.

Cluttered Table In this problem the Fetch robot needs to pick up a specific object from a cluttered table. The target object can be obstructed by different objects which need to picked and placed at different locations to reach the final object. The actions available to the robot are to pick up an object, place an object, and to move around the table. Generators for the concretization of the actions include generating the grasping poses, put-down poses and target base poses. The action of picking up an object succeeds with probability 0.8; the object falls back onto the table with probability 0.2. We increase the number of cans on the table to increase the complexity of this problem. Fig. 4 shows the results for the time taken to solve 100 randomly generated instances for three configurations of the environment with 15, 20, and 25 number of cans with horizon initially kept to 6.

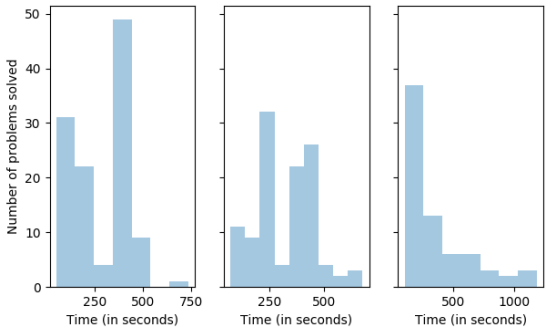

Figure 4: Time taken to compute abstract policies with complete motion planning refinements for randomly generated problems (left: cluttered domain with 15 cans, center: cluttered domain with 20 cans, right: cluttered domain with 25 cans).

Aircraft Inspection In this problem, a UAV needs to inspect an airplane. The available actions are to fly to a certain region, go to the charging station and charge, and to inspect a certain component. While trying to fly from one location to other location, the UAV may drift to a different region with probability 0.05. Each operation consumes a certain amount of battery, but this cannot be computed at the high-level since the high-level abstraction cannot reason with trajectories. There is a battery recharge station which can be reached from anywhere in the environment on reserve power. Generators for this problem include sampling target locations for move actions as well as the waypoints used to envelope a component for the inspection action. The algorithm needs to come up with the sequence of actions to examine the required parts with valid non-colliding trajectories while keeping sufficient battery at each time step. Fig. 5 shows the results for probability reaching the goal refined with the percentage of nodes in policy tree refined. Empirical evaluations show that it takes less than 1% of nodes refined and less than 100 seconds to whereas entire policy tree refinement takes more than 4600 seconds.

Empirical evaluation of anytime performance Fig. 5 shows the anytime characteristics of our approach in all of the test domains. The x-axis shows the percentage of nodes that have been evaluated, refined and potentially replaced with updated policies that permit low-level plans. The y-axis shows the probability with which the policy available at any time during the algorithm's computation will be able to handle all possible execution-time outcomes.

These empirical results indicate that in all of our test domains the refined probability mass increases exponentially with the percentage of nodes refined. This is desirable because most of the possible execution time outcomes are handled by the task and motion policy with only 20-40% of the computation. Such an approach would allow users to determine the amount of computation to invest in prior to execution, based on the acceptable levels of risk.

6 Other Related Work

There has been a renewed interest in integrated task and motion planning algorithms. Most research in this direction has been focused on deterministic environments (Cambon, Alami, and Gravot 2009; Plaku and Hager 2010; Hertle et al. 2012; Kaelbling and Lozano-Pérez 2011; Garrett, Lozano-Pérez, and Kaelbling 2015; Dantam et al. 2016; Garrett, Lozano-Pérez, and Kaelbling 2018). Kaelbling and Lozano-Perez (Kaelbling and Lozano-Pérez 2013) consider a partially observable formulation of the problem. Their approach utilizes regression modules on belief fluents to develop a regression-based solution algorithm. While they address the more general class of partially observable problems, their approach follows a process of online, incremental discretization and does not address the computation of branching policies, which is the focus of this paper. Sucan and Kavraki (Şucan and Kavraki 2012) use an explicit multigraph to represent the problem for which motion planning refinements are desired. Other approaches (Hadfield-Menell et al. 2015) address problems where the high-level formulation is deterministic and the low-level is determinized using most likely observations. Our approach uses a compact, relational representation; it employs abstraction to bridge MDP solvers and motion planners and solves the overall problem in anytime fashion. The closest work is done by (Srivastava et al. 2018) which implements a primitive version of the algorithm presented in the paper.

Principles of abstraction in MDPs have been well studied (Hostetler, Fern, and Dietterich 2014; Bai, Srivastava, and Russell 2016; Li, Walsh, and Littman 2006; Singh, Jaakkola, and Jordan 1995). However, these approaches assume that the full, unabstracted MDP can be efficiently expressed as a discrete MDP. Marecki et al. (Marecki et al. 2006) consider continuous time MDPs with finite sets of states and actions. In contrast, our focus is on MDPs with high-dimensional, uncountable state and action spaces. Recent work on deep reinforcement learning (e.g., (Hausknecht and Stone 2016; Mnih et al. 2015)) presents approaches for using deep neural networks in conjunction with reinforcement learning to solve short-horizon MDPs with continuous state spaces. These approaches can be used as primitives in a complementary fashion with task and motion planning algorithms, as illustrated in recent promising work by Wang et al. (Wang et al. 2018).

ACKNOWLEDGMENTS

We thank Midhun Pookkottil Madhusoodanan for help with an initial implementation of the presented algorithms. This work was supported in part by the NSF under grant IIS 1844325.

References

Bai, A.; Srivastava, S.; and Russell, S. J. 2016. Markovian state and action abstractions for MDPs via hierarchical MCTS. In *Proc. IJCAI*.

Bertsekas, D. P., and Tsitsiklis, J. N. 1991. An analysis of stochastic shortest path problems. *Mathematics of Operations Research* 16(3):580–595.

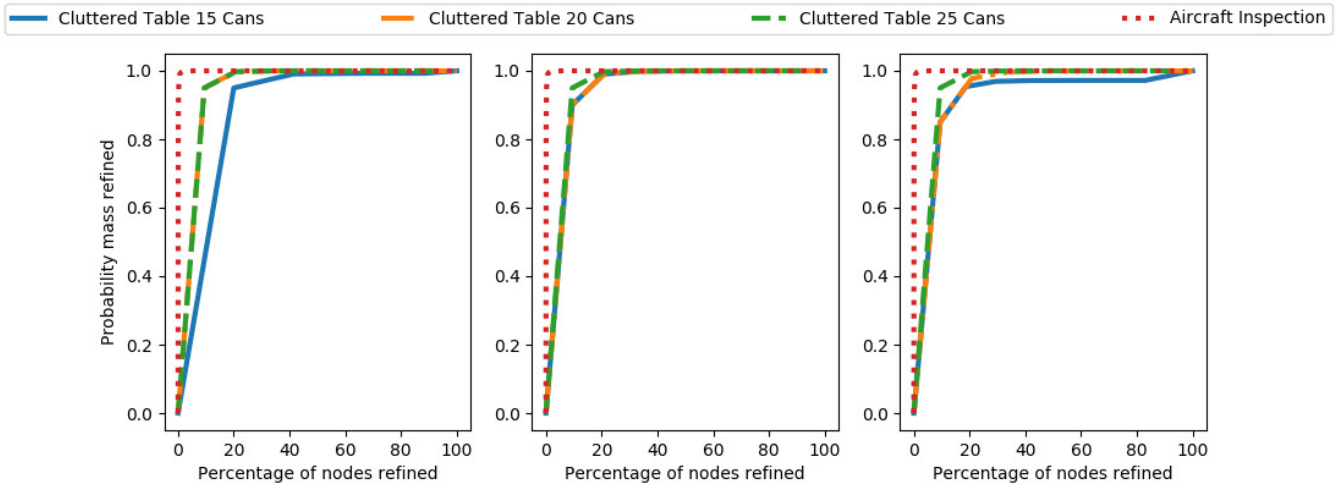

Figure 5: Anytime performance of ATM-MDP, showing the percentage of nodes refined (x-axis) v/s probability mass refined (y-axis).

Cambon, S.; Alami, R.; and Gravot, F. 2009. A hybrid approach to intricate motion, manipulation and task planning. *IJRR* 28:104–126.

Codd, E. F. 1972. Relational completeness of data base sublanguages. In R, R., ed., *Database Systems*.

Dantam, N. T.; Kingston, Z. K.; Chaudhuri, S.; and Kavraki, L. E. 2016. Incremental task and motion planning: A constraint-based approach. In *Proc. RSS*.

Diankov, R. 2010. *Automated Construction of Robotic Manipulation Programs*. Ph.D. Dissertation, Carnegie Mellon University.

Garrett, C. R.; Lozano-Pérez, T.; and Kaelbling, L. P. 2015. FFrob: An efficient heuristic for task and motion planning. In *Proc. WAFR*.

Garrett, C. R.; Lozano-Pérez, T.; and Kaelbling, L. P. 2018. Sampling-based methods for factored task and motion planning. *The International Journal of Robotics Research*.

Hadfield-Menell, D.; Groshev, E.; Chitnis, R.; and Abbeel, P. 2015. Modular task and motion planning in belief space. In *Proc. IROS*.

Hansen, E. A., and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129(1-2):35–62.

Hausknecht, M., and Stone, P. 2016. Deep reinforcement learning in parameterized action space. In *Proc. ICLR*.

Hertle, A.; Dornhege, C.; Keller, T.; and Nebel, B. 2012. Planning with semantic attachments: An object-oriented view. In *Proc. ECAI*.

Hostetler, J.; Fern, A.; and Dietterich, T. 2014. State aggregation in monte carlo tree search. In *Proc. AAAI*.

Immerman, N. 1998. *Descriptive complexity*. Springer Science & Business Media.

Kaelbling, L. P., and Lozano-Pérez, T. 2011. Hierarchical task and motion planning in the now. In *Proc. ICRA*.

Kaelbling, L. P., and Lozano-Pérez, T. 2013. Integrated task and motion planning in belief space. *The International Journal of Robotics Research* 32(9-10):1194–1227.

Li, L.; Walsh, T. J.; and Littman, M. L. 2006. Towards a unified theory of state abstraction for mdps. In *ISAIM*.

Marecki, J.; Topol, Z.; Tambe, M.; et al. 2006. A fast analytical algorithm for mdps with continuous state spaces. In *Proc. AAMAS*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Plaku, E., and Hager, G. D. 2010. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *Proc. ICRA*.

Sanner, S. 2010. Relational dynamic influence diagram language (rddl): Language description. `http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf`.

Saribatur, Z. G.; Schüller, P.; and Eiter, T. 2019. Abstraction for non-ground answer set programs. In *Proc. JELIA*.

Singh, S. P.; Jaakkola, T.; and Jordan, M. I. 1995. Reinforcement learning with soft state aggregation. In *Proc. NIPS*, 361–368.

Srivastava, S.; Fang, E.; Riano, L.; Chitnis, R.; Russell, S.; and Abbeel, P. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *Proc. ICRA*.

Srivastava, S.; Desai, N.; Freedman, R.; and Zilberstein, S. 2018. An anytime algorithm for task and motion mdps. *arXiv preprint arXiv:1802.05835*.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2011. A new representation and associated algorithms for generalized planning. *Artificial Intelligence* 175(2):615–647.

Srivastava, S.; Russell, S.; and Pinto, A. 2016. Metaphysics of planning domain descriptions. In *Proc. AAAI*.

Şucan, I. A., and Kavraki, L. E. 2012. Accounting for uncertainty in simultaneous task and motion planning using task motion multigraphs. In *Proc. ICRA*.

Wang, Z.; Garrett, C. R.; Kaelbling, L. P.; and Lozano-Pérez, T. 2018. Active model learning and diverse action sampling for task and motion planning. In *Proc. IROS*.

Younes, H. L., and Littman, M. L. 2004. PPDDL 1.0: An extension to pddl for expressing planning domains with probabilistic effects. *Technical Report CMU-CS-04-162*.