Anytime Integrated Task and Motion Policies for Stochastic Environments

Naman Shah, Kislay Kumar, Pranav Kamojjhala, Deepak Kala Vasudevan, and Siddharth Srivastava
School of Computing, Informatics, and Decision Systems Engineering,
Arizona State University, Tempe, AZ, USA
{namanshah, kkumar28, pkamojjh, dkalavas, 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 stochastic 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.

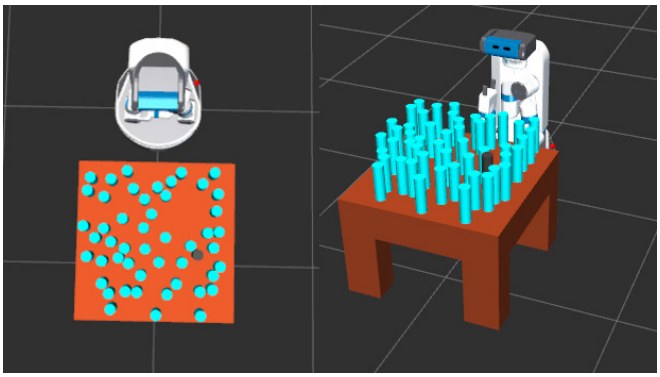

Fig. 1. A stochastic variant of the cluttered table domain where robot needs to pick up the black can, but pickups may fail.

I. Introduction

Recent years have witnessed immense progress in research on integrated task and motion planning [1], [2], [3], [4], [5]. The focus of this research direction is on enabling robots to autonomously solve complex tasks that require high-level planning that goes beyond physical movements, while taking into account low-level constraints such as collisions and dynamic stability. Research in this direction provides several approaches for solving deterministic, fully observable versions of such problems. However, the problem of integrated task and motion planning under uncertainty has been understudied.

In this paper, we consider integrated task and motion planning problems where the robot's actions and the environment are stochastic. This problem is well known to be more difficult computationally because sequential plans are no longer sufficient; solutions take the form of policies that prescribe an action for every state that the robot may encounter during execution. 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 time-consuming, error-prone 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.

In order to address this problem, we utilize finite-state controllers to express task and motion policies as tree-structured contingent policies. A naive approach for computing such a policy would be to first compute a high-level policy using an abstract model of the problem (e.g., a model written in a language such as PPDDL or RDDL[6]), and to then refine each "branch" of the solution policy with motion plans. Such approaches fail because abstract models are lossy and cannot correctly capture physical constraints such as collisions [7], [8], [9]. Furthermore, as the planning horizon increases, computing complete task and motion policies becomes intractable as it requires the computation of exponentially many task and motion "plans."

In order to address these problems, we propose a novel *anytime* approach for computing integrated task and motion policies. This approach continually improves policies while ensuring that situations that are most likely to be encountered in an execution are resolved first. It also provides a running estimate of the probability mass of likely executions covered in the current policy. This estimate can be used to start execution based on the level of risk acceptable in a given application, allowing one to trade-off precomputation-time for on-the-fly replanning if an unhandled situation is encountered. Our experiments indicate the probability of encountering an unresolved contingency drops exponentially as the algorithm proceeds. Our approach generalizes methods for computing solutions for most-likely outcomes during

execution [10], [11] to the problem of integrated task and motion planning by drawing upon approaches for anytime computation in AI planning [12], [13], [14].

The resulting approach is the first *probabilistically complete* algorithm for computing integrated task and motion policies in stochastic environments using a powerful relational representation for specifying input problems. The use of relational representations allows us to easily express problems involving object manipulation, which would be cumbersome if not infeasible in propositional representations. Our approach can be used with arbitrary MDP planners and motion planners. This allows our approach to scale automatically with improvements in those fields.

We model the overall problem as an abstracted Markov decision process (MDP), where each action in the MDP (e.g. pickup) corresponds to an uncountable set of possible lower-level motion planning problems. 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 grasp pose and raise pose, and a motion plan for each occurrence of the pickup action in the computed policy.

We begin with a presentation of the background definitions (II) and our formal framework (III). (IV) describes our overall algorithmic approach, followed by a description of empirical results using the Fetch robot in simulation (V), and a discussion of prior related work (VI).

II. 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.

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),*
	∀$traj$ *intersects(vol(obj ,target_pose),*
	sweptVol(robot, traj)→ Collision(obj_1, traj),
	probabilistic
	0.8 [*RobotAt($config_2$),*
	at(obj_1, target_pose)]
	0.2 [*RobotAt(around_$config_2$),*
	at(obj_1,around_target_pose)]
abstract effect	¬*holding(obj_1),*
	∀$traj$ ⍰ *Collision(obj_1, traj),*
	probabilistic
	0.8 [*RobotAt($config_2$),*
	at(obj_1, target_pose)]
	0.2 [*RobotAt(around_$config_2$),*
	at(obj_1,around_target_pose)]

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

A solution to an MDP is a *policy*, $\pi : S \to A$, 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 [15]. 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 $\pi : S \times \{h_0, h_1, \ldots, h_n\} \to A$ 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.

III. 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.

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 [16], [17] 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 [18] as well as answer set programming [19]. 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.

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 [20], [21].

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.

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

Solutions to task and motion planning problems, like solutions to SSPs, are policies with actions from the concrete model $\mathcal{M}$.

IV. Algorithmic Framework

A. 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 non-deterministic. Variants of our overall approach can be constructed with different implementations of these choices; the versions used in our evaluation are described in IV-B.

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 $a \in \mathcal{M}$.

Our overall algorithm interleaves computation among the processes of (a) *concretizing an abstract policy*, (b) *updating the abstraction to include predicate valuations for a fixed concretization*, and (c) *computing an abstract policy for an updated abstract state*. This is done using the *plan refinement graph* (PRG). 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, the current state of search for concretizations

of all actions $a_j \in [\pi]_u$, and a partial concretization σ for a topological prefix of the policy tree $[\pi]_u$. Each edge *(u,v)* between nodes *u* and *v* in the PRG is labeled with a partial concretization $\sigma_{u,v}$ and the failed preconditions for the first abstract action in a root-to-leaf path in $[\pi]_u$ that doesn't have a feasible refinement under $\sigma_{u,v}$. Recall that this occurs because the abstract model is lossy and doesn't capture precise action semantics. $[\mathcal{M}]_v$ is the version of $[\mathcal{M}]_u$ where the predicates corresponding to the failed preconditions (corresponding to effects with ⑦, created due to the abstraction discussed in Sec. III) have been replaced with their literal versions that are true under $\sigma_{u,v}$.

Alg.1 carries out the interleaved search outlined above as follows. It first initializes the PRG with a single node containing an abstract policy for the abstract SSP (line 1). In every iteration of the main loop it selects a node in the PRG and extracts an unrefined root-to-leaf path from the policy for that node (lines 3-5). It then interleaves the three processes as follows.

a) 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 $c_{i+1} \in a_{i+1}(c_i)$ 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.

b) 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 abstract state in this path whose subsequent action's concretization is infeasible. This abstract state is updated with the true forms of the violated preconditions that hold in this concretization, using symbolic arguments. E.g., *Collision(teapot, $traj\_cup$)*). , The rest of the policy after this abstract state is discarded. A state update is immediately followed by the computation of a new abstract policy.

c) 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.

Several approaches are possible for selecting the PRNode to concretize or update in line 3. We used incremental-broadening depth-first search on the PRGraph in all of our experiments with the breadth bound 5.

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
```

B. Optimizations and Formal Results

We develop the basic algorithm outlined above (Alg. 1) along two major directions: we enhance it to facilitate anytime computation and to improve the search for concretizations of abstract policies.

Anytime computation for task and motion policies: 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 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

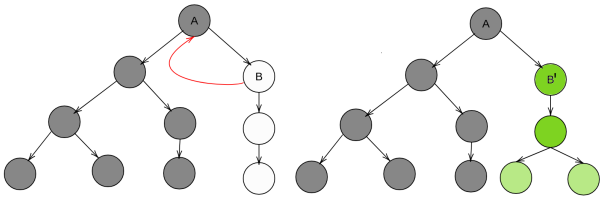

Fig. 3. Figure: Left: Backtracking from node B invalidates the refinement of subtree rooted at A. Right: Replanning from node B which in some cases requires fewer resources.

Environment	% solved	Avg. Time (seconds)
Cluttered-15	95	1093.71
Cluttered-20	79	1144.85
Cluttered-25	74	1392.83
Aircraft Inspection	100	1457.08
Keva-PI	100	989.83
Keva-Tower	100	2327.88
Keva-Twisted-Tower	100	2333.50

Fig. 4. Summary of times taken to solve the test problems. Timeout for cluttered table, aircraft inspection: 2400 seconds, building Keva stuctures: 4000 seconds.

estimated cost of refining that path c. We use a priority queue for the RTL paths with their *p/c* values as the keys.

Search for concretizations: Sample-based backtracking search [22] 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 on line 16). Thm. 2 shows that our algorithm is probabilistically complete.

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.

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.

V. Empirical Evaluation

We implemented the presented framework using an open-source implementation [23] of LAO* [24] as the SSP solver, the OpenRAVE [25] robot simulation system along with 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 diverse and challenging test problems. In practice, fixing the horizon h a priori can render some problems unsolvable. Instead, we implemented a variant that dynamically increases the horizon until the goal is reached with probability $p > 0$. We evaluated our approach on a variety of problems where combined task and motion planning is necessary. The source code, and the experiment videos can be found at `https://aair-lab.github.io/atam_full.html`.

Cluttered Table: In this problem, we have a table cluttered with cans having different probabilities of being crushed when grabbed by the robot. Some cans are delicate and are highly likely to be crushed (with probability set to 0.1, 0.5, 0.9 for different experiments in Fig. 7(a)) when the robot grabs them, incurring a high cost, while others are normal and are less likely to be crushed (with probability set to 0.05). The goal for the robot is to pick up a specified can. We used different number of cans (15, 20, 25), and different random configurations of cans to extensively evaluate the proposed framework. We also used this scenario to evaluate our approach in the real-world (Fig. 5 using the Fetch mobile manipulation robot [26].

Aircraft Inspection: In this problem, a UAV needs to inspect possibly faulty parts of an aircraft. Its goal is to

Fig. 5. Real world experiments with Fetch mobile manipulation robot. The goal here is to pick up the specified bottle avoiding picking up delicate bottles.

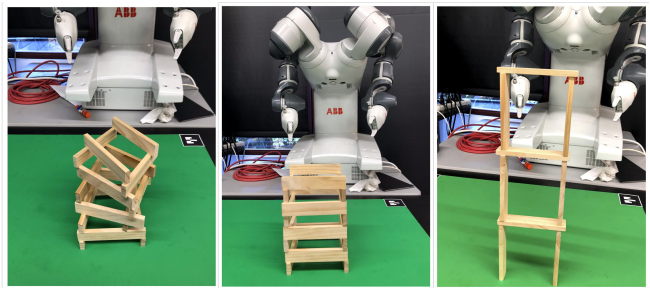

Fig. 6. Structures built using Keva planks. Left: spiral tower; center: square tower; right: stacked π.

locate the fault and notify the supervisor about it. However, its sensors are not accurate and may fail to locate the fault with some probability (probability set to 0.05, 0.1, 0.15 for experiments in Fig. 7(b)) while inspecting the location; it may also drift to another location while flying. Charging stations are available for the UAV to dock and charge itself. All movements use up an amount of battery charge depending on the length of the trajectory, but the high-level planner cannot determine whether the current level of battery is sufficient for an action as it doesn't have access to precise trajectories. This makes it necessary for the high-level to obtain feedback from low-level to solve the problem.

Fig. 7 shows the results for the probability of 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 compute refinements for more outcome with cumulative probability more than 0.8 whereas the entire policy tree refinement takes more than 4600 seconds.

Building structures using Keva planks: In this problem, the ABB YuMi robot needs to build differenTwistedt structures using Keva planks. Keva planks are laser cut wooden planks with uniform geometry. Fig. 6 shows some of the target structures. Planks are placed one at a time by a user after each pickup and placement by the YuMi. Each new plank may be placed at one of a few predefined locations, which adds uncertainty in the planks' initial location. For experiments, two predefined locations were used to place the planks with probability 0.8 for the first location and probability 0.2 for the second location. Starting with a 3D model of the target structure, the YuMi needs to create a task and motion policy for successively picking up and placing planks to build the structure. There are infinitely many configurations in which one plank can be placed on another, but the abstract model blurs out different regions on the plank. The put-down pose generator uses the target structure to concertize each plank's target put-down pose.

A. Analysis of Results

Fig. 7 shows the anytime characteristics of our approach in all of the test domains. 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, and the x-axis shows the time (seconds) required to refine that probability mass.

These empirical results indicate that in all of our test domains, the refined probability mass increases rapidly with time so that about 80% of probable executions are covered within about 30% of the computation time. 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, which can be clearly seen in 7. 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.

Fig. 4 shows the average times taken to concretize entire policy tree for variants of each of our test problems. These values are averages of 50 runs for the cluttered table, 20 runs for aircraft inspection and 5 runs for Keva plan construction.The number of runs for Aircraft inspection problems and Keva plans was kept lower because their runtimes showed negligible variance.

It can be seen in Fig.4 that refining the entire policy tree requires a huge amount of time, but most of the probable executions are refined in less than 40% of time, reinforces the need of an anytime solution in such scenarios.

VI. 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 [7], [27], [28], [8], [29], [30], [31]. Kaelbling and Lozano-Perez [32] 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 [33] use an explicit multigraph to represent the problem for which motion planning refinements are desired. Other approaches [10] address problems where the high-level formulation is deterministic and the low-level is determinized

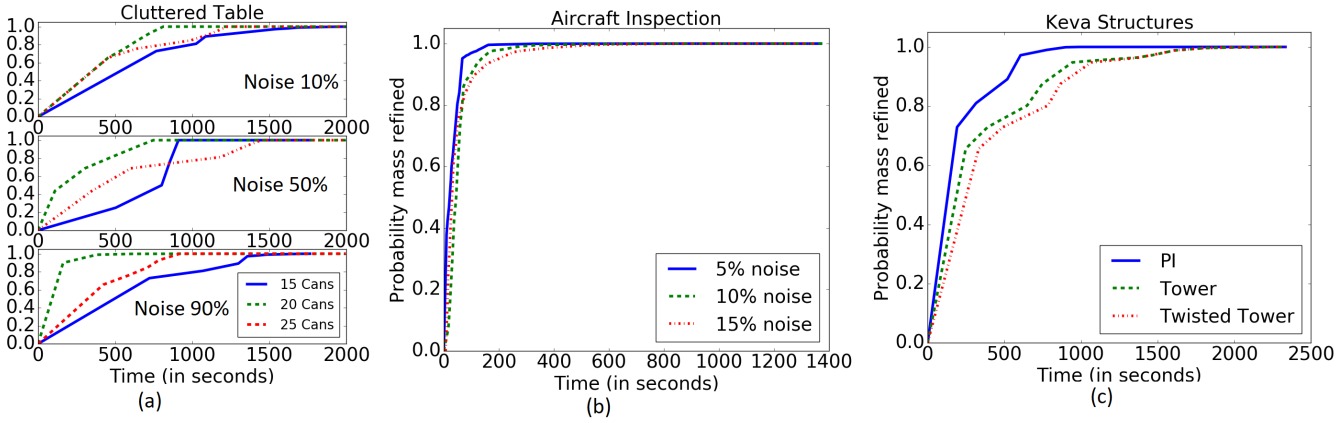

Fig. 7. Anytime performance of ATM-MDP, showing the time in seconds (x-axis) v/s probability mass refined (y-axis).

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 [34] which implements a primitive version of the algorithm presented in the paper.

Principles of abstraction in MDPs have been well studied [35], [36], [37], [38]. However, these approaches assume that the full, unabstracted MDP can be efficiently expressed as a discrete MDP. Marecki et al. [39] 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., [40], [41]) 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. [42].

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 grants IIS 1844325 and IIS 1909370.

REFERENCES

[1] C. R. Garrett, T. Lozano-Prez, and L. P. Kaelbling, "Ffrob: Leveraging symbolic planning for efficient task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 104–136, 2018. [Online]. Available: https://doi.org/10.1177/0278364917739114

[2] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.

[3] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtos, and M. Carreras, "Rosplan: Planning in the robot operating system," in *Twenty-Fifth International Conference on Automated Planning and Scheduling*, 2015.

[4] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Stripstream: Integrating symbolic planners and blackbox samplers," *CoRR*, vol. abs/1802.08705, 2018. [Online]. Available: http://arxiv.org/abs/1802.08705

[5] R. Chitnis, D. Hadfield-Menell, A. Gupta, S. Srivastava, E. Groshev, C. Lin, and P. Abbeel, "Guided search for task and motion plans using learned heuristics," in *Proc. ICRA*, 2016.

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

[7] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *IJRR*, vol. 28, pp. 104–126, 2009.

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

[9] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "A modular approach to task and motion planning with an extensible planner-independent interface layer," in *Proc. ICRA*, 2014.

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

[11] S. Yoon, A. Fern, and R. Givan, "FF-replan: A baseline for probabilistic planning," in *Proc. ICAPS*, 2007.

[12] T. L. Dean and M. S. Boddy, "An analysis of time-dependent planning." in *Proc. AAAI*, 1988.

[13] S. Zilberstein and S. J. Russell, "Anytime sensing, planning and action: A practical model for robot control," in *Proc. IJCAI*, 1993.

[14] T. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson, "Planning under time constraints in stochastic domains," *Artificial Intelligence*, vol. 76, no. 1-2, pp. 35–74, 1995.

[15] D. P. Bertsekas and J. N. Tsitsiklis, "An analysis of stochastic shortest path problems," *Mathematics of Operations Research*, vol. 16, no. 3, pp. 580–595, 1991.

[16] E. F. Codd, "Relational completeness of data base sublanguages," in *Database Systems*, R. R, Ed., 1972.

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

[18] S. Srivastava, N. Immerman, and S. Zilberstein, "A new representation and associated algorithms for generalized planning," *Artificial Intelligence*, vol. 175, no. 2, pp. 615–647, 2011.

[19] Z. G. Saribatur, P. Schüller, and T. Eiter, "Abstraction for non-ground answer set programs," in *Proc. JELIA*, 2019.

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

[21] S. Srivastava, S. Russell, and A. Pinto, "Metaphysics of planning domain descriptions," in *Proc. AAAI*, 2016.

[22] S. Srivastava, X. Cheng, and S. Russell, "First-order open-universe POMDPs: Formulation and algorithms," in *Proc. UAI*, 2014.

[23] L. Pineda, "MDP-Lib," https://github.com/luisenp/mdp-lib, 2014.

[24] E. A. Hansen and S. Zilberstein, "LAO*: A heuristic search algorithm that finds solutions with loops," *Artificial Intelligence*, vol. 129, no. 1-2, pp. 35–62, 2001.

[25] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Carnegie Mellon University, 2010.

[26] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich, "Fetch and freight: Standard platforms for service robot applications," in *Workshop on Autonomous Mobile Service Robots*, 2016.

[27] E. Plaku and G. D. Hager, "Sampling-based motion and symbolic action planning with geometric and differential constraints," in *Proc. ICRA*, 2010.
[28] A. Hertle, C. Dornhege, T. Keller, and B. Nebel, "Planning with semantic attachments: An object-oriented view," in *Proc. ECAI*, 2012.
[29] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "FFrob: An efficient heuristic for task and motion planning," in *Proc. WAFR*, 2015.
[30] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: A constraint-based approach." in *Proc. RSS*, 2016.
[31] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Sampling-based methods for factored task and motion planning," *The International Journal of Robotics Research*, 2018.
[32] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, 2013.
[33] I. A. Şucan and L. E. Kavraki, "Accounting for uncertainty in simultaneous task and motion planning using task motion multigraphs," in *Proc. ICRA*, 2012.
[34] S. Srivastava, N. Desai, R. Freedman, and S. Zilberstein, "An anytime algorithm for task and motion mdps," *arXiv preprint arXiv:1802.05835*, 2018.
[35] J. Hostetler, A. Fern, and T. Dietterich, "State aggregation in monte carlo tree search." in *Proc. AAAI*, 2014.
[36] A. Bai, S. Srivastava, and S. J. Russell, "Markovian state and action abstractions for MDPs via hierarchical MCTS." in *Proc. IJCAI*, 2016.
[37] L. Li, T. J. Walsh, and M. L. Littman, "Towards a unified theory of state abstraction for mdps." in *ISAIM*, 2006.
[38] S. P. Singh, T. Jaakkola, and M. I. Jordan, "Reinforcement learning with soft state aggregation," in *Proc. NIPS*, 1995, pp. 361–368.
[39] J. Marecki, Z. Topol, M. Tambe, *et al.*, "A fast analytical algorithm for mdps with continuous state spaces," in *Proc. AAMAS*, 2006.
[40] M. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," in *Proc. ICLR*, 2016.
[41] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[42] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, "Active model learning and diverse action sampling for task and motion planning," in *Proc. IROS*, 2018.