A Unified Sampling-Based Approach to Integrated Task and Motion Planning

Wil Thomason and Ross A. Knepper *

Department of Computer Science
Cornell University
Ithaca, NY 14850
{wbthomason, rak}@cs.cornell.edu

Abstract. We present a novel method for performing integrated task and motion planning (TMP) by adapting any off-the-shelf sampling-based motion planning algorithm to simultaneously solve for a symbolically and geometrically feasible plan using a single motion planner invocation. The core insight of our technique is an embedding of symbolic state into continuous space, coupled with a novel means of automatically deriving a function guiding a planner to regions of continuous space where symbolic actions can be executed. Our technique makes few assumptions and offers a great degree of flexibility and generality compared to state of the art planners. We describe our technique and offer a proof of probabilistic completeness along with empirical evaluation of our technique on manipulation benchmark problems.

1 Introduction

Generalized robot autonomy requires robots to plan solutions for complex and varied tasks. Most robot problems — for example, manipulation problems — require interaction with the physical world; as such, they involve both a *continuous* geometric component and a *discrete* symbolic component abstracting some part of the problem or world state. Traditionally, these two aspects of the problem are studied independently; motion planners solve geometric problems, whereas task planners solve symbolic problems. If the planners are invoked serially, then the task plan may inform the geometric problem specification. A more holistic planning approach, **integrated task and motion planning** (TMP), seeks to unify the planning problem using information from each component to ease the solution of the other.

The qualitative differences between the task and motion planning problems complicate their effective integration. A task planner operates on a high-level abstraction of the world and thus a valid task plan may have no corresponding valid motion plan if the abstraction is too coarse. A task planning abstraction fine enough to capture the geometric details of the world to the same degree as a motion planner may be intractable for the task planner. Mitigating this problem by re-running the task planner naively

*This material is based upon work supported by the National Science Foundation under Grant No. 1646417 and by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program. We are grateful for this support.

following a motion planner failure is also likely to fail, as most task planners produce only a single plan for a given problem instance. Most TMP approaches focus on creating a better interface between disparate task and motion planners.

This interface often uses the motion planner as a source of constraints and/or a validator for task plans [9, 15, 24, 31]. Using a motion planner as a validator may require repeated executions of both planners, which rapidly becomes expensive. Treating a motion planner as a source of constraints (which is compatible with the validator approach) may be more efficient but is often limited by the manual selection of a restricted set of motion constraints to integrate in the task layer.

We present a novel approach to TMP that fully embeds a symbolic abstraction of a planning problem into the problem's continuous representation. This fused representation (Section 3.2) allows a sampling-based motion planner to solve both problems simultaneously and efficiently in a single invocation. To make this process efficient, we also contribute methods for: a factorization of the sampling problem to avoid a dimensional explosion (Section 3.2), a continuous semantics for logical formulae in configuration space (Section 3.3), and the use of standard task-planning heuristics as sampling biases (Section 3.4), which we build into a planner-agnostic sampling algorithm (Section 3.5) for solving TMP problems. Finally, we contribute a proof of concept implementation (Section 3.6) of our technique and evaluate it (Section 5) on a realistic benchmark.

2 Related Work

TMP is an active field of research with many existing approaches. Most of these approaches represent task-planning problems using STRIPS-style [11] domains and actions defined in terms of *Boolean predicates*: representations of symbolic state as Boolean functions applied to environment symbols. We also use this specification format.

Similarly to the "semantic attachments" of Dornhege et al. [10] or the factorized samplers of Garrett, Lozano-Pérez, and Kaelbling [13] (as well as related concepts in other prior work [12, 19]), we rely on user-provided executable predicate semantics implementations to test what predicates a given state satisfies. In contrast to these approaches, our predicate semantics are simple tests on a state's value, do not invoke a motion planner, and are often portable between problems. Further, we use our predicate semantics not only to test known states but also to automatically derive a navigation function to find states satisfying a given predicate.

Prior work has applied sampling-based motion planners to symbolic planning and TMP. Cambon, Alami, and Gravot [7] coupled probabilistic roadmaps (PRM [20]) with symbolic search for transit and transfer paths over a manipulation graph to solve TMP problems, Burfoot, Pineau, and Dudek [6] use rapidly exploring random trees (RRT [27]) to solve STRIPS-style planning problems, and Branicky et al. [5] apply RRTs to planning for hybrid systems (requiring a fusion of continuous and discrete spaces).

Work in multi-modal planning [15–17, 30] and manipulation [2, 3] fuses discrete and continuous configuration spaces. The "modes" of this literature correspond closely to our notion of symbolic state. This work generally assumes the existence of (sometimes precomputed) domain-specific samplers for constraint manifold intersections and uses fixed, specific motion planning algorithms. In contrast, our work contributes an automatic

derivation for manifold-intersection samplers (Section 3.3) and a sampler compatible with *any* sampling-based motion planning algorithm (Section 3.5).

The method of Plaku and Hager [29] is perhaps closest to ours. It also uses a symbolic planner to guide the growth of a random motion planning tree toward a goal. Our approach differs in its handling of purely discrete state (i.e. discrete state without a reasonable continuous representation) and uses a different, improved method of guiding the search and sampling in constraint-satisfying regions.

Approaches to finding constraint-satisfying states are diverse. Dantam et al. [9] use the constraint stack of a satisfiability modulo theories (SMT) solver to efficiently search possible task plans with knowledge of some geometric constraints. Garrett, Lozano-Pérez, and Kaelbling [13] directly sample in feasible regions for actions, whereas Lagriffoul et al. [23] propagate geometric constraints to find such regions, and Toussaint et al. [33, 34] optimize a manually specified differentiable representation of symbolic action constraints. Our approach introduces a continuous semantics for logical formulae specifying the valid execution region of an action and optimizes this automatically derived potential function to locate a state within the region. This approach is also closely related to work in constrained motion planning [21] and formal methods for control synthesis [35].

3 Approach

Our approach is simple: we perform normal motion planning in a space including both the geometric and symbolic state for a problem (Section 3.2). A valid motion plan in this space will by construction obey all geometric, kinematic, and symbolic constraints; as such, it solves both the task and motion parts of a TMP problem. Motion planning in this space is possible with off-the-shelf sampling-based planning algorithms using our novel sampling algorithm (Section 3.5). This sampler guides a planning algorithm toward important regions of the fused geometric/symbolic space without breaking the planner's other properties, such as completeness and exploration bias.

3.1 Geometric and Symbolic Predicates

We consider task-planning problems specified by a *domain* consisting of a set of predicates (Boolean-valued functions) and actions conditioned and operating on these predicates [11, 28]. The "semantics" of a predicate — what it represents in the real world — is determined solely by its use in the definition of the symbolic actions and problem instance. In the context of TMP, we can partition these predicates into two types. *Geometric* predicates can be interpreted in terms of the continuous state space, such as whether one object is on top of another. *Symbolic* predicates have a purely symbolic meaning, such as whether a light is on or off. Each grounding (assignment of ground atoms to arguments) of each symbolic predicate corresponds to one "bit" of symbolic state. Similarly, for each grounding of each geometric predicate there is a corresponding subset of continuous state space where the predicate is true. We call this subset the *predicate region* for a given predicate. Formally:

Definition 1. Predicates and Predicate Regions
A Boolean-valued ***predicate*** *is a function* $P_i : \mathcal{C} \to \mathbb{Z}_2$*, where* $\mathcal{C}$ *is some configuration*

*space (we define a specific $\mathcal{C}$ below). We define the **predicate region** for P_i to be $T_{P_i} = \{q \in \mathcal{C} \colon P_i(q) = \top\} \subseteq \mathcal{C}$ — that is, the subset of $\mathcal{C}$ where P_i is true. In general, T_{P_i} may be of a lower dimension than $\mathcal{C}$. Given a formula F_i involving one or more predicates, we similarly define a **formula region** T_{F_i}. If the formula is a precondition for an action, we typically refer to its formula region as a **precondition region**.*

3.2 Composite Space

The hybrid systems, multi-modal planning, and manipulation planning literature commonly combines discrete modes and continuous degrees of freedom into a single planning space [1, 14, 16, 17, 25]. Similarly, we construct a *composite space* in which the "modes" correspond to unique settings of the bits of symbolic state of a TMP problem and the continuous state consists of both robot configuration and movable object pose. Formally,

Definition 2. Composite Space
The composite space for a TMP problem is

$$\mathcal{C} = \mathcal{C}_{Robot} \bigtimes_{M} SE(3) \bigtimes_{B} \mathbb{Z}_2,$$

where M is the number of movable objects, B is bits of symbolic state, $\mathcal{C}_{Robot}$ is the robot configuration space, and $\mathbb{Z}_2$ is the two-element field. Although we use Boolean symbolic state here, our definitions can be extended to arbitrary symbolic state.

This space may be high-dimensional for even moderate numbers of movable objects and bits of discrete state. Three key insights make planning in composite space tractable: (1) we can factor the composite space to reduce the effective dimensionality for planning; (2) we can avoid sampling in the full object pose space; (3) we can sample within regions that satisfy geometric predicates by introducing an "unsatisfaction semantics". This semantics provides a potential function we can optimize to sample configurations that satisfy arbitrary combinations of predicates. We detail these insights below.

Factoring Composite Space: To decrease the effective dimensionality of a composite space, we can factor it based on unique settings of its bits of symbolic state. This factorization is best understood through a helpful metaphor. By definition, a symbolic predicate describes world state that is not captured by the continuous part of composite space; it is akin to an additional fact added to the world state[1]. In other words, a particular bit of symbolic state is either true or false for the entire continuous configuration space at a given point in composite space. Thus, we can think about different combinations of values for symbolic predicates as encoding distinct "universes"; continuous configuration spaces where different symbolic facts are true. This factorization permits us to consider a single symbolic state at a time as an otherwise ordinary continuous configuration space, and only need to consider reachable symbolic states (i.e. the combinations of symbolic predicate values attainable by taking some sequence of symbolic actions from the initial symbolic state) rather than directly sampling symbolic states.

[1] The distinction between facts captured or not captured by the configuration space is largely a choice of granularity in representation.

Sampling Object Configurations: We assume quasi-static dynamics (i.e. objects move only when manipulated) to further reduce the difficulty of the composite configuration sampling problem. This assumption, which holds for many realistic manipulation problems, allows us to avoid sampling in the full object pose space. Instead, we can track known stable poses of objects (e.g. by adding to a set of valid poses when we choose an action that sets a held object down) and sample from this set. If we have further information about valid stable poses for objects and stable surfaces in the scene, we can sample poses more aggressively, including new poses that match the given template. Our technique neither assumes nor relies on the availability of this information.

3.3 Unsatisfaction Semantics

To plan effectively in composite space, we need to efficiently sample states in precondition regions for symbolic actions with effects that move us closer to the symbolic goal. Accomplishing this without either an explicit representation of the regions where a geometric predicate holds or a predicate-specific sampler is difficult; all previous work that we are aware of relies on one of these two approaches [4, 13]. We cannot rely on simple uniform sampling of the entire composite space because (1) many geometric predicates correspond to lower-dimensional manifolds (and thus have measure zero in the ambient space) and (2) even geometric predicates that are full-dimensional may be sparsely distributed in the composite space. For a pathological example of this, consider a predicate which is true whenever each of its arguments is an integer. The corresponding predicate region is not a manifold (each of the points where it is true has no open neighborhood), and the integers have measure zero in the reals.

We solve this problem by introducing a continuous semantics for geometric predicates which tells us how far a given state is from the nearest state at which the predicate is true, or how "unsatisfied" it is at that state. A geometric predicate (or formula of geometric predicates) interpreted under this semantics provides a potential function allowing us to project uniform random states from the ambient space directly onto the nearest state in the predicate region. This projection requires computing the **minimal ϵ-weakening** of a predicate at a point.

Definition 3. ϵ-Weakening
Take P_i to be a particular predicate defined solely as a formula of the comparison operators $\{=, <, \leqslant, >, \geqslant\} \subset \mathbb{R} \times \mathbb{R} \to \mathbb{Z}_2$, arithmetic operators $\{-, +\} \subset \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, and the Boolean operators $\wedge, \vee, \neg$[2]. Then the ϵ-weakening *of P_i is defined as $\mathcal{W}(P_i, \epsilon) : \mathcal{C} \to \mathbb{R}$; that is, P_i evaluated with each of the aforementioned comparison operators replaced by its ϵ-equivalent such that (for a, b real-valued arithmetic expressions):*

$$a =_\epsilon b := |a - b| \leqslant \epsilon$$
$$a <_\epsilon b := a < b + \epsilon$$
$$a \leqslant_\epsilon b := a \leqslant b + \epsilon$$
$$a >_\epsilon b := a > b - \epsilon$$
$$a \geqslant_\epsilon b := a \geqslant b - \epsilon$$

[2] Although we restrict ourselves to this sufficient set of operators in this paper, ϵ-weakening can be extended to allow a broader operator set.

*We define the **minimal** ϵ**-weakening** of P_i at a point $q \in \mathcal{C}$ to be $\mathcal{W}_{\min}(P_i, \epsilon)$ for*

$$\epsilon = \arg\min_{\epsilon'} \left(P_{i'_\epsilon}(q) = \top \right)$$

We analytically solve for the minimal ϵ for a point $q \in \mathcal{C}$ by refining our ϵ-weakened operators and adding real-valued definitions for the logical operators $\wedge$ and $\vee$ ($\neg$ is implemented as a syntactic transformation, e.g. flipping $\leqslant$ for $>$, etc.):

$$\begin{aligned}
a =_\epsilon b &\implies \epsilon = |a - b| \\
a <_\epsilon b &\implies \epsilon = a - b \\
a \leqslant_\epsilon b &\implies \epsilon = a - b \\
a >_\epsilon b &\implies \epsilon = b - a \\
a \geqslant_\epsilon b &\implies \epsilon = b - a \\
a \wedge_\epsilon b &:= \sqrt{\max(a, 0)^2 + \max(b, 0)^2} \\
a \vee_\epsilon b &:= \min(a, b)
\end{aligned}$$

We clamp ϵ to $\mathbb{R}^+$, so that $\mathcal{W}_{\min}(P_i, \epsilon)(q)$ defines the shortest distance from a point $q \in \mathcal{C}$ to a point $q' \in \mathcal{C}$ such that $P_i(q') = \top$. We can use $\mathcal{W}_{\min}(P_i, \epsilon)$ as a potential function leading to the predicate region for P_i and use any technique for following potential functions to find a state in said predicate region. For our proof-of-concept implementation, we use forward-mode automatic differentiation with gradient descent to find a sample in the truth region for any given predicate formula.

This semantics has an important subtlety: if a geometric predicate is combined with other geometric predicates via $\wedge_\epsilon$ or $\vee_\epsilon$ in a formula, then its definition must be restricted to using the above set of operators, conditionals, and iteration in order for the unsatisfaction value of the formula to be correct. Geometric predicates used in isolation in a formula may use *any* operators, etc., so long as (a) they return the result of one of $<, \leqslant, >, \geqslant, =$ and (b) they compute a potential function. This stems from the ϵ-weakening of $\wedge$ and $\vee$; in future work we seek to remove this restriction. In practice, we find the set of allowable predicate functions even under this restriction to be sufficient.

Predicate regions have little required structure. Since $\mathcal{W}_{\min}(P_i, \epsilon)$ gives the distance from a state to the nearest state in the predicate region, $\nabla \mathcal{W}_{\min}(P_i, \epsilon)$ may be discontinuous or have local minima at which P_i is not true. To mitigate these issues, we use smooth approximations of $\min$ and $\max$ in our implementation and standard techniques for escaping local minima. When gradient descent finishes, we test the resulting state q against the ordinary Boolean version of P_i to ensure that it is valid. If it is not (i.e. $P_i(q) = \texttt{false}$), we repeat gradient descent from a new uniform random sample in $\mathcal{C}$.

3.4 Heuristic Guidance

We need to be able to pick precondition regions as subgoals to efficiently plan in composite space. In other words, we require a means of selecting symbolic actions which will accomplish the goal. We handle this by defining the following minimal interface to a task-planning heuristic:

Definition 4. Heuristic interface
A heuristic $\mathbb{H}(q)$ *must return a list of actions* $\mathbf{A}$ *for which the symbolic part of state* q *satisfies the symbolic part of the precondition of each* $a \in \mathbf{A}$. $\mathbb{H}(q)$ *may optionally sort* $\mathbf{A}$ *by* $\texttt{Priority}(a)$ *for each* $a \in \mathbf{A}$.

Many heuristics and other methods of action suggestion satisfy this interface. In our proof of concept implementation, we use the "helpful actions" heuristic from the FF planner [18], chosen for its simplicity. The heuristic provides long-horizon symbolic planning guidance; the quality of this guidance depends on the particular heuristic chosen. The meaning of `Priority` depends on the particular heuristic; intuitively, it estimates how important the action is to the overall solution.

We make no further assumptions on the heuristic. This means that we cannot assume the heuristic has any knowledge of the geometry of the scene, the kinematics of the robot, or any other factors that inform the physical feasibility of a symbolic action. This information is critical for solving TMP problems efficiently. As such, we define the *scaled priority* of a symbolic action to be:

$$\texttt{P}(a) = \frac{\texttt{Priority}(a)}{1 + \texttt{Fail}(a) + \gamma * \texttt{Success}(a)}$$

where $\texttt{Priority}(a)$ is the heuristic priority of action a, $1 \leqslant \gamma \in \mathbb{R}$ is a constant scaling factor, and $\texttt{Fail}(a)$ and $\texttt{Success}(a)$ are counts of the number of attempts to use a that have failed (i.e. we could not find a state satisfying the precondition, or the state satisfying the precondition was not added to the planner's data structure) and succeeded (i.e. we found a state satisfying the action precondition and added it to the planner's data structure), respectively. Intuitively, if an action keeps failing, it is less likely to be part of a valid plan and should be deprioritized; if an action succeeds, then it should also be deprioritized to avoid using the same actions redundantly.

3.5 Composite Space Sampling

We combine the tools from the previous sections into a sampling algorithm, shown in Algorithm 1. Any sampling-based motion planner can use this algorithm as its sampler to solve TMP problems in composite space (with the caveat that bidirectional planners require further adaptation to plan in composite space, where "distance" is directional).

We use the heuristic of Section 3.4 to choose useful actions to attempt and use ϵ-weakening to efficiently sample in the precondition regions for these actions. At a high level, our algorithm treats these precondition regions as sub-goals for a sampling-based motion planner. By planning between sub-goals in sequence, we are able to construct a chain of motion plans which corresponds to a complete TMP solution. The call to the heuristic on Line 18 of Algorithm 1 has one important subtlety: we request viable actions from the heuristic for a particular symbolic state only *after* we have reached it; as such, we avoid unnecessary task-planning effort by never forcing the heuristic to do work for an impossible symbolic state.

[3] Using a biased coin ensures that we sometimes grow the planning structure "normally" within single symbolic configurations. β may be thought of as trading off between the motion planning problem and the symbolic planning problem.

Algorithm 1: Composite Space Sampling

Data: Task specification, scene description, predicate semantics implementations
Sampler Context : For $c \in \mathcal{C}$ and $u \in U$: a log of symbolic actions (if any) taken in c; a set of metadata associated with u including valid object poses, etc.
Output : Sampled state in $\mathcal{C}$

```
// Flip a coin which returns True with probability β^3
if BiasedCoin() is True then
    return NormalSample()
else
    return HeuristicSample()
end
function NormalSample()
    u ← UniformRandom(U);                          // Symbolic state
    p ← UniformRandom(ValidPoses(u));              // Known valid pose
    r ← UniformRandom(C_Robot);                    // Robot configuration
    return MakeConfiguration(r, p, u);             // Form composite config
end
function HeuristicSample()
    repeat
        c' ← NormalSample();
        // Choose a valid action for the sampled symbolic
           state: First, get the set of actions.
        // This corresponds to ℍ(q) of Definition 4
        h ← PrioritizedActions(c');
        // Sample an action according to its scaled priority
        a ← PrioritySample(h);
        f ← Precondition(a);
        // Solve for a precondition-satisfying state using a
           black-box optimizer
        c ← Solve(c', f);
    until Satisfies(c, Precondition(a));
    UpdateInfo(c.u, GetPoses(c));   // Add to c.u viable pose set
    UpdateLog(c, a);    // Log use of a at c for plan execution
    return c;
end
```

The calls `UpdateInfo` and `UpdateLog` update planner metadata. `UpdateLog` logs associations between configurations and symbolic actions. This log is used to instruct the robot to run the controller for a specific symbolic action at the correct state when executing a plan. `UpdateInfo` updates a planner data structure tracking the valid object pose sets associated with a symbolic state as well as symbolic state connectivity. This information is used during sampling and distance computation.

3.6 Implementation

We have implemented a proof of concept of our algorithm in C++. Fig. 1 shows our system architecture. We use OMPL [32] for motion planning (with our custom sampler and composite configuration space as described in Section 3) and Bullet [8] for geometric

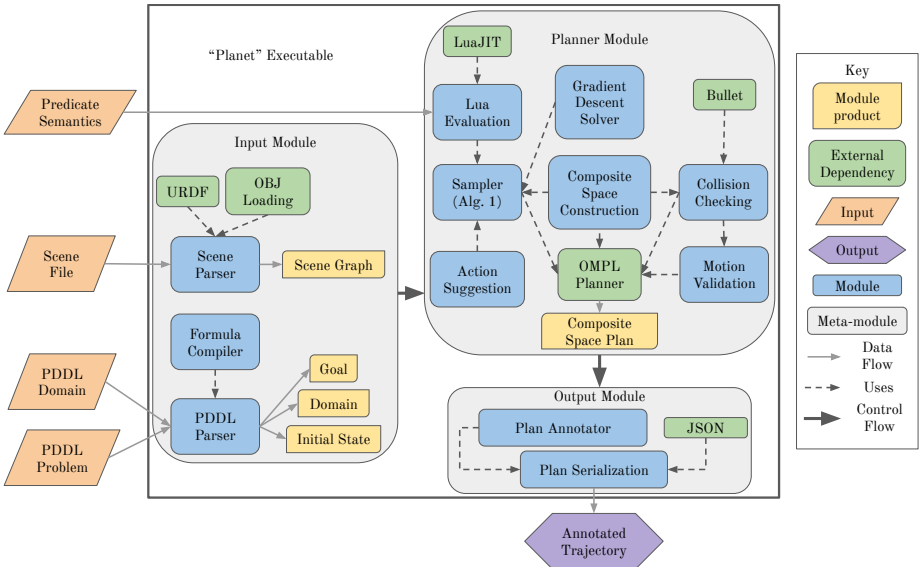


Fig. 1: The architecture of our proof of concept implementation. We take as input a PDDL domain and problem, as well as a file describing the robot and workspace and Lua semantics for a subset of the PDDL predicates. We use these inputs to construct a composite space for the problem and run a standard OMPL [32] motion planner in this space with our sampling algorithm (Algorithm 1). The resulting composite state plan is transformed into a JSON representation of a robot pose trajectory annotated with optional actions (e.g. grasping) to take at each point in the trajectory. This trajectory can be executed by a standard controller for the robot.

collision checking. We use OMPL's implementation of RRT [27] as our sampling-based motion planner. We use PDDL extended with keywords to signify kinematic and geometric predicates to specify symbolic domains and Lua functions to specify predicate semantics; Listing 1 shows an example predicate definition in Lua. We chose Lua for its speed (via LuaJit), easy interoperability with C++, and the availability of high-quality scientific computing libraries for automatic differentiation. In particular, we use SciLua for automatic differentiation.

```
function above(x, y)
  return And(Le(x.pz - y.pz, 0.15), Ge(x.pz - y.pz, 0.0))
end
```

Listing 1: The Lua implementation of a predicate describing one object being above another.

We specify problems as a PDDL domain and problem instance coupled with a scene file in the format described in Lagriffoul et al. [22]. We use this format for the geometric part of our problem specifications to make it easier to benchmark our planner against future planners also using this format. Source code for our planner is available at https://github.com/cornell-rpal/planet.

4 Analysis

The bias β of the coin flip used in Algorithm 1 to choose between "normal" and heuristic sampling is tunable and may have different optimal values in different domains. Both normal and heuristic sampling are necessary for efficient performance; heuristic sampling is more expensive and only returns states in action precondition regions, whereas the more efficient and completeness-preserving "normal" sampling may be unable to find some precondition regions and thus cannot grow the tree between symbolic states.

The minimal heuristic interface defined by Definition 4 allows us to swap out one action suggestion method for another without having to change our technique at all. For instance, the proof of concept implementation discussed in Section 3.6 implements the FF [18] "useful actions" heuristic, but we could equally easily use a neural network policy trained on a particular planning domain, a more sophisticated planning graph analysis heuristic, or even some combination of multiple heuristics or a full task planner.

4.1 Probabilistic Completeness

Algorithm 1 preserves the probabilistic completeness of the motion planning algorithm using it as a sampler (so long as `BiasedCoin` returns both `True` and `False` a non-zero fraction β of the time and the action suggestion method eventually returns all feasible actions for a given state). The proof is straightforward; we sketch it here:

Proof. As stated above, assume that `BiasedCoin` returns `True` with probability $0 < \beta < 1$. Take $\mathbb{H}$ to be the action suggestion method used by Line 18 of Algorithm 1 and further assume that for any state $q \in \mathcal{C}$,

$$\bigcup_{n \to \infty} \mathbb{H}(q) = \texttt{ViableActions}(q)$$

where $\texttt{ViableActions}(q)$ is the set of all actions which have precondition regions in the symbolic state of q, and $\bigcup_{n \to \infty} \mathbb{H}(q)$ is the union of the sets of actions suggested by $\mathbb{H}$ over n invocations of $\mathbb{H}$ for q as n goes to ∞[4]. In other words, $\mathbb{H}$ always eventually suggests all viable actions for a state $q \in \mathcal{C}$.

Let i be the number of sampled states and S the set of samples. As $i \to \infty$, also $n \to \infty$, and thus $\forall q \in S$, $\mathbb{H}$ eventually suggests every viable action from q. Thus, given Algorithm 1, as $i \to \infty$, every viable action a for a sampled $q \in \mathcal{C}$ will be suggested and attempted infinitely often. Each invocation of the black-box solver starts from a uniform random configuration; as the "cost" optimized by the solver (distance to a precondition region) is convex, this means that as $i \to \infty$ dispersion in the precondition region [26] will go to zero (we effectively project from a ball around the precondition region into the precondition region). Performing a viable action a from q introduces symbolic states and object poses caused by applying the effect of a to q. Thus, as $i \to \infty$, we will eventually find precondition-satisfying samples arbitrarily close to

[4] For deterministic, stateless $\mathbb{H}$, the set of actions returned will be the same on every invocation. If $\mathbb{H}$ tracks the set of actions already suggested or is nondeterministic, then the set of actions returned may differ on subsequent invocations of $\mathbb{H}$ for q.

every precondition-satisfying state, and thus we will eventually discover every "relevant" object pose set and symbolic state.

Finally, as $i \to \infty$, `NormalSample` will be invoked infinitely many times. As `NormalSample` amounts to uniform random sampling of robot configurations and known object pose sets and symbolic states, and as we have established, every object pose set and symbolic state which could be part of a solution to the TMP problem will eventually be discovered, it follows that `NormalSample` will eventually sample within a ε-neighborhood of every state which could be a part of a solution to the TMP problem. Hence, if the planning algorithm using the sampler defined by Algorithm 1 is probabilistically complete using a "normal" sampler alone, it will remain probabilistically complete when solving TMP problems using Algorithm 1 as its sampler. □

5 Evaluation

We evaluate our proof of concept implementation on two benchmark problems from the set proposed by Lagriffoul et al. [22]. We modify the "clutter sorting" problem by adding our required annotations of kinematic and discrete predicates and providing an implementation of the predicate semantics. We show results on this variant of the problem as well as a variant with the red blocks removed from the scene. We include this second variant to show that our system can handle obstacles like the red blocks when they are present and also improve its performance in their absence.

We have chosen to use the problem format proposed by Lagriffoul et al. [22] for our evaluation in an effort to make it easier to evaluate future TMP systems against ours (i.e. by using what is intended as a standard specification format and set of benchmark problems for the field).

5.1 Scenario Description

Our variant of the clutter problem of Lagriffoul et al. [22] involves sorting two groups of blocks into specific zones located out of robot reach from each other. This problem demonstrates scalability with the number of objects in the scene as well as the capability to handle infeasible task actions in a large task space. The problem, shown in Fig. 2, specifies a Willow Garage PR2 robot and places the blocks on a set of four tables.

5.2 Experimental Results

We have run our implementation on clutter problem instances ranging in size (the number of objects to be moved) from one to eight, for ten trials of each instance size. The objects to be moved are split evenly among blue and green blocks; if the total number is odd, there is one more blue block than green block. There are always either zero (the blue line in Figs. 3a and 3b) or fourteen (the orange line in Figs. 3a and 3b) red blocks present in the environment. Figure 3a shows the number of states in the solution, and Fig. 3b shows the planning time in seconds. We plot the mean time for each instance size. The error bars show one standard deviation from each mean.

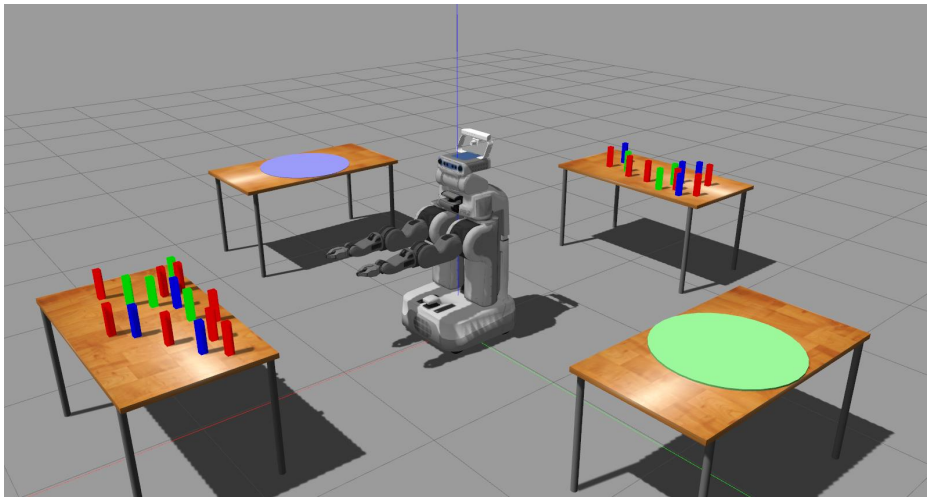
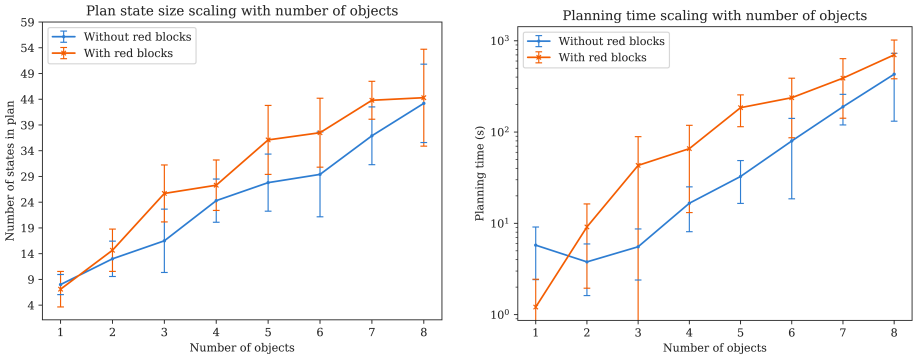


Fig. 2: Our simulation environment for the benchmark problem discussed in Section 5.1. The PR2 must find a plan to move the green blocks to the table with the green circle and the blue blocks to the table with the blue circle.

Fairly benchmarking TMP planners is challenging. In addition to the usual problems with reporting wall-clock time (different computer speeds, implementation languages, levels of engineering "polish" and optimization), the results of raw performance comparisons depend heavily on choices made in the problem specification. For example, the specification makes choices about whether variables should be discretized or left in the continuum, how the semantics of the problem are implemented as predicates, and even what state is relevant to the problem. These factors (and more) may affect the performance of different TMP planners in different ways. Further, wall clock time may not capture precomputation or other planning effort conducted offline; our technique specifically avoids this sort of offline work. This field is still maturing, and it has taken only first steps at standarizing the TMP problem [22]. The community still does not have consensus on a good set of performance metrics to compare TMP planners. For instance, how do generality and flexibility trade off against speed? How does the complexity of writing specifications weigh in? We nevertheless report wall-clock time despite its problems because it remains a widely accepted metric in the community.

Our implementation performs competitively with the state of the art; our evaluation problem is similar to that of Dantam et al. [9] and requires a similar planning time for the same object counts (note that our planning time includes both motion and task planning, whereas the planning time reported by Dantam et al. [9] is only for motion planning).

The average plan size in number of states scales linearly with the number of objects. For the clutter problem, the minimum number of states in a solution plan also scales linearly with the number of objects — if an object is added to the problem with a random position, then in expectation a constant number of states must be added to handle moving that object. This shows that while the plans our implementation produces are not optimal in length, they are only suboptimal by a "constant" factor (technically, by a random variable independent of problem instance size).



(a) (b)

Fig. 3: Performance results on the "Clutter" problem

Our planning time increases exponentially as object count increases. This is not uncommon in TMP planners solving similar problems, due to the combinatorial nature of the task space. This poor performance may be exacerbated by the particular heuristic we chose for our implementation. This heuristic [18] is naive and has no knowledge of global plan feasibility or motion feasibility. As such, as object count increases, if this heuristic cannot rank the priorities of the larger set of actions viable in a given state, then our planner will tend to explore the actions broadly rather than following a globally-aware sequence of actions. A more sophisticated heuristic to guide the action search could correct this deficiency. It is also worth noting the large variance in planning times for the larger problem instances. Much of this variance can be attributed to our use of RRT, which has high variance in performance (we do not use random restarts).

We also ran our planner on a variant of the "Kitchen" problem from Lagriffoul et al. [22] (we remove the radishes, which are intended to make manipulation actions more difficult). We can solve the provided instance of this problem in an average of 304.6 seconds over five trials. We were unable to solve the full kitchen problem in reasonable time, necessitating this simplification. We believe that our naive heuristic is unable to give useful priority values for actions in the large task space of the full kitchen problem: by investigating a histogram of the symbolic states our planner visits, we see that it makes significant progress (i.e. it visits symbolic states which accomplish subgoals of the full problem), but explores too broadly — that is, it doesn't commit to a single task plan and see it through to completion, but explores many possible task plans. We believe that using a different heuristic, in particular a heuristic with some notion of global progress toward the goal, would mitigate this problem.

The broad exploration exhibited by our implementation is a potential blessing. By exploring task plans in parallel, the planner avoids overcommitting to a single plan and is robust to task plan branches becoming nonviable without needing to backtrack. For this same reason, we believe that our technique would also be useful in a nondeterminstic execution context, unlike the current deterministic assumption. Upon action failure, we can reuse the planning data structure our technique has grown during the original planning phase to pick up replanning without needing to start over from the initial state.

6 Conclusion

This paper contributes a novel, efficient approach to solving integrated task and motion planning problems. Our approach breaks the Gordian knot of TMP by simultaneously searching for a task solution and motion solution in a composite space of symbolic and geometric state. We are able to do so efficiently by using a novel continuous semantics for symbolic predicates describing how "unsatisfied" they are at a given state to search for states which satisfy the preconditions of symbolic actions. Further, our approach is both general and flexible — we can reuse definitions of predicate semantics across domains, search for solutions with different properties by changing predicate semantics, take advantage of action suggestion methods tailored to problem domains without changing our heuristic interface, and make use of any sampling-based planning algorithm simply by using our sampling algorithm.

We present our algorithm and results based on analysis and experimentation with a proof of concept implementation of our sampling algorithm on established benchmarks [22] and show that we perform competitively and can solve realistic problems.

6.1 Limitations

Our technique makes a number of assumptions which may limit its applicability. Although our technique could theoretically be expanded to a multi-robot context, we assume the presence of only a single robot. We also do not consider dynamic constraints and assume a quasi-static, deterministic, fully-observable environment. Additionally, our technique makes no independent guarantees of optimality and is reliant on whatever solution optimization the particular planning algorithm using our sampler provides.

6.2 Future Work

Our technique has the potential to be embarassingly parallel — multiple instances of our sampler can operate largely independently in parallel, and this parallel potential could be improved by creating independent samplers for each explored symbolic state.

We also hope that our technique's agnosticism to the specific heuristic and sampling-based planning algorithm used will permit investigation of the fundamental relationship between task planning and motion planning to gain insight into techniques for faster, more robust, flexible TMP solving.

References

[1] R. Alami, T. Siméon, and J.-P. Laumond. "A Geometrical Approach to Planning Manipulation Tasks". In: *Proceedings International Symposium on Robotics Research*. 1989, pp. 113–119.

[2] Jennifer Barry, Kaijen Hsiao, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. "Manipulation with Multiple Action Types". In: International Symposium on Experimental Robotics (ISER). Vol. 88. Springer International Publishing, 2013, pp. 531–545.

[3] Jennifer Barry, Leslie Pack Kaelbling, and Tomas Lozano-Perez. "A Hierarchical Approach to Manipulation with Diverse Actions". In: 2013 IEEE International Conference on Robotics and Automation (ICRA). IEEE, May 2013, pp. 1799–1806.

[4] Dmitry Berenson, Siddhartha S. Srinivasa, and James Kuffner. "Task Space Regions: A Framework for Pose-Constrained Manipulation Planning". In: *International Journal of Robotics Research* 30.12 (2011), pp. 1435–1460.

[5] M.S. Branicky, M.M. Curtiss, J.A. Levine, and S.B. Morgan. "RRTs for Nonlinear, Discrete, and Hybrid Planning and Control". In: *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*. Vol. 1. IEEE, 2003, pp. 657–663.

[6] Daniel Burfoot, Joelle Pineau, and Gregory Dudek. "RRT-Plan: A Randomized Algorithm for STRIPS Planning". In: *Proceedings of the Sixteenth International Conference on International Conference on Automated Planning and Scheduling*. ICAPS'06. AAAI Press, 2006, pp. 362–365.

[7] Stéphane Cambon, Rachid Alami, and Fabien Gravot. "A Hybrid Approach to Intricate Motion, Manipulation and Task Planning". In: *International Journal of Robotics Research* 28.1 (Jan. 1, 2009), pp. 104–126.

[8] Erwin Coumans et al. *Bullet Physics Library*. 2013.

[9] Neil T. Dantam, Zachary K. Kingston, Swarat Chaudhuri, and Lydia E. Kavraki. "Incremental Task and Motion Planning: A Constraint-Based Approach". In: *Robotics: Science and Systems XII*. 2016.

[10] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. "Semantic Attachments for Domain-Independent Planning Systems". In: *Towards Service Robots for Everyday Environments*. Vol. 76. Springer Berlin Heidelberg, 2012, pp. 99–115.

[11] Richard E. Fikes and Nils J. Nilsson. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". In: *Artificial Intelligence* 2.3-4 (Dec. 1971), pp. 189–208.

[12] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. "FFRob: An Efficient Heuristic for Task and Motion Planning". In: *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*. Springer, Cham, 2014, pp. 179–195.

[13] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. "Sampling-Based Methods for Factored Task and Motion Planning". In: *The International Journal of Robotics Research* 37.13-14 (2018), pp. 1796–1825.

[14] Robert L Grossman, Anil Nerode, Anders P Ravn, and Hans Rischel. "Hybrid Systems". Vol. 736. Springer, 1993.

[15] Kris Hauser and Jean-Claude Latombe. "Integrating Task and PRM Motion Planning: Dealing with Many Infeasible Motion Planning Queries". In: *ICAPS '09 Workshop on Bridging the Gap between Task and Motion Planning*. ICAPS'09. 2009.

[16] Kris Hauser and Jean-Claude Latombe. "Multi-Modal Motion Planning in Non-Expansive Spaces". In: *The International Journal of Robotics Research* 29.7 (June 2010), pp. 897–915.

[17] Kris Hauser and Victor Ng-Thow-Hing. "Randomized Multi-Modal Motion Planning for a Humanoid Robot Manipulation Task". In: *The International Journal of Robotics Research* 30.6 (May 2011), pp. 678–698.

[18] Jörg Hoffmann and Bernhard Nebel. "The FF Planning System: Fast Plan Generation through Heuristic Search". In: *Journal of Artificial Intelligence Research* 14 (2001), pp. 263–312.

[19] Leslie Pack Kaelbling and Tomás Lozano-Pérez Tomas. "Hierarchical Task and Motion Planning in the Now". In: *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 1470–1477.

[20] L. E. Kavraki, P. Svestka, J. Latombe, and M. H. Overmars. "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces". In: *IEEE Transactions on Robotics and Automation* 12.4 (Aug. 1996), pp. 566–580.

[21] Zachary Kingston, Mark Moll, and Lydia E. Kavraki. "Sampling-Based Methods for Motion Planning with Constraints". In: *Annual Review of Control, Robotics, and Autonomous Systems* 1.1 (2018), pp. 159–185.

[22] Fabien Lagriffoul, Neil T. Dantam, Caelan Garrett, Aliakbar Akbari, Siddharth Srivastava, and Lydia E. Kavraki. "Platform-Independent Benchmarks for Task and Motion Planning". In: *IEEE Robotics and Automation Letters* 3.4 (Oct. 2018), pp. 3765–3772.

[23] Fabien Lagriffoul, Dimitar Dimitrov, Julien Bidot, Alessandro Saffiotti, and Lars Karlsson. "Efficiently Combining Task and Motion Planning Using Geometric Constraints". In: *The International Journal of Robotics Research* 33.14 (2014), pp. 1726–1747.

[24] Fabien Lagriffoul, Dimitar Dimitrov, Alessandro Saffiotti, and Lars Karlsson. "Constraint Propagation on Interval Bounds for Dealing with Geometric Backtracking". In: *IEEE International Conference on Intelligent Robots and Systems*. 2012, pp. 957–964.

[25] S. M. LaValle. "Planning Algorithms". Cambridge University Press, 2006.

[26] Steven M. LaValle. "From Dynamic Programming to RRTs: Algorithmic Design of Feasible Trajectories". In: *Control Problems in Robotics*. Vol. 4. Springer Berlin Heidelberg, 2003, pp. 19–37.

[27] Steven M. Lavalle. *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. Tech. rep. 98-11. Iowa State University, 1998.

[28] Drew McDermott. "PDDL - The Planning Domain Definition Language". In: AIPS Planning Competition. 1998.

[29] Erion Plaku and Gregory D. Hager. "Sampling-Based Motion and Symbolic Action Planning with Geometric and Differential Constraints". In: *Proceedings - IEEE International Conference on Robotics and Automation*. 2010, pp. 5002–5008.

[30] P. S. Schmitt, W. Neubauer, W. Feiten, K. M. Wurm, G. V. Wichert, and W. Burgard. "Optimal, Sampling-Based Manipulation Planning". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. May 2017, pp. 3426–3432.

[31] 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: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. May 2014, pp. 639–646.

[32] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. "The Open Motion Planning Library". In: *IEEE Robotics & Automation Magazine* 19.4 (Dec. 2012), pp. 72–82.

[33] Marc Toussaint. "Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning". In: International Joint Conference on Artificial Intelligence. 2015, p. 7.

[34] Marc Toussaint, Kelsey Allen, Kevin Smith, and Joshua Tenenbaum. "Differentiable Physics and Stable Modes for Tool-Use and Manipulation Planning". In: *Robotics: Science and Systems XIV*. Robotics: Science and Systems Foundation, June 26, 2018.

[35] Cristian-Ioan Vasile, Vasumathi Raman, and Sertac Karaman. "Sampling-Based Synthesis of Maximally-Satisfying Controllers for Temporal Logic Specifications". In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017, pp. 3840–3847.