ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA241015

A Meta-Engine Framework for Interleaved Task and Motion Planning Using Topological Refinements

Elisa Tosello[a,*], Alessandro Valentini[a] and Andrea Micheli[a]

[a]Fondazione Bruno Kessler, Trento, Italy

Abstract. Task And Motion Planning (TAMP) is the problem of finding a solution to an automated planning problem that includes discrete actions executable by low-level continuous motions. This field is gaining increasing interest within the robotics community as it significantly enhances robot's autonomy in real-world applications. Many solutions and formulations exist, but no clear standard representation has emerged. In this paper, we propose a general and open-source framework for modeling and benchmarking TAMP problems. Moreover, we introduce an innovative meta-technique to solve TAMP problems involving moving agents and multiple task-state-dependent obstacles. This approach enables using any off-the-shelf task planner and motion planner while leveraging a geometric analysis of the motion planner's search space to prune the task planner's exploration, enhancing its efficiency. We also show how to specialize this meta-engine for the case of an incremental SMT-based planner. We demonstrate the effectiveness of our approach across benchmark problems of increasing complexity, where robots must navigate environments with movable obstacles. Finally, we integrate state-of-the-art TAMP algorithms into our framework and compare their performance with our achievements.

1 Introduction

Task And Motion Planning (TAMP) is the problem of finding high-level plans to accomplish assigned tasks (task planning), as well as the motions needed to execute these plans (motion planning). Consider a warehouse robot collecting items and placing them in bins for shipment. At the task level, it determines the sequence of actions needed, such as collecting items and navigating. At the motion level, it plans the movements considering obstacles. Merely sequencing task and motion planning may lead to ineffective solutions, with the robot possibly moving directly toward the goal, ignoring obstacles. In contrast, integrating these components effectively allows the robot's plan to adapt dynamically. For instance, if a pallet blocks an aisle, the robot will try to move it before proceeding further.

A wide range of solutions and formulations exist, but no clear standard representation has emerged [19]. In this paper, we propose a formalization and implementation for modeling TAMP problems related to navigation tasks with multiple movable objects, which is independent of specific planners and languages. We also offer an open-source modeling tool within the open-source Unified Planning (UP) library[1] to facilitate the integration of setups and planners for evaluation and comparison. To validate our approach, we provide an exhaustive benchmarks suite aligned with existing TAMP evaluation criteria [19], addressing challenges like infeasible task actions, large task spaces, and balancing task complexity with motion execution.

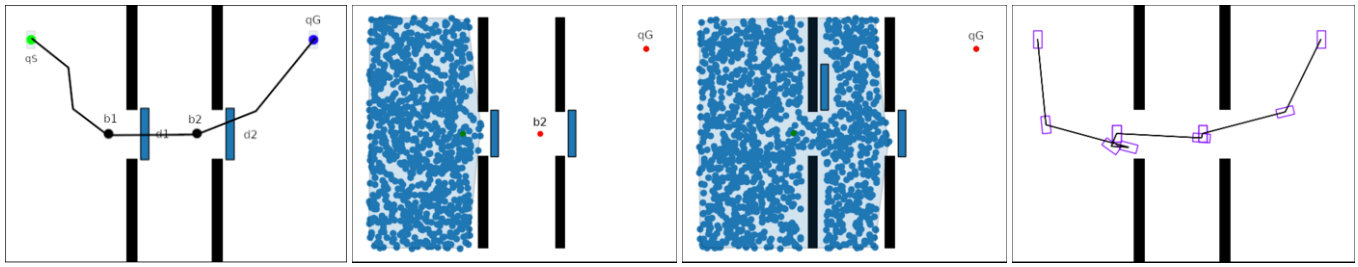
Furthermore, we integrate into our framework a planning technique tailored to this class of problems that allows to combine off-the-shelf automated planners with off-the-shelf motion planners. We exploit the *Meta-Engine* feature of the UP library to instantiate our framework with any task planner available through the library. Then, we use the Open Motion Planning Library (OMPL) [28] to plan motions (but any other solver could be exploited). Our approach fits into the category of interleaved TAMP [12, 5]: a Benders Decomposition [2] of the TAMP problem where the task planner decides a candidate plan disregarding the motion constraints. Then, the motion planner refines the plan by adding the motion details. If it fails, it analyzes the reason for the failure and derives an explanation that the task planner can use to prune its search for new plans. In this sense, the core of our approach is what we call *topological refinement*: we approximate the area explored by the motion planner, derive the encountered obstacles, and exploit them to formulate new constraints that we add at the task level. This refinement allows us to prune entire symbolic space regions rather than just the immediate unrealizable action, as typically done in traditional TAMP approaches.

One drawback of off-the-shelf automated planners is the need to restart the task planning search with each new constraint learned. Hence, we also present TAMPEST (Task And Motion Planning by Encoding into Satisfiability Testing): a simple but effective algorithm based on Satisfiability Modulo Theory (SMT) [1] that leverages the incrementality of modern SMT solvers to avoid restarts.

Finally, we integrate PDDLSTREAM [11], a solver increasingly used in TAMP, into our framework, making it one of the solvers supported by the UP library. Our TAMP formulation, compatible with any UP-supported solver, is automatically converted to PDDLStream's format, allowing it to solve TAMP problems without customization. This integration demonstrates our ability to separate problem formulation from the solving algorithm, empowering users to compare various solvers using identical problem formulations and input data. We include a thorough experimental assessment, comparing various task planners, sampling-based motion planners, and benchmarking against PDDLSTREAM. We show TAMPEST's effectiveness and efficiency, particularly with topological refinements.

The paper is organized as follows. Section 2 formalizes our TAMP problem, and Section 3 introduces our meta-engine approach, with its SMT-based specialization in Section 4. Section 5 details our benchmarks, Section 6 reviews related work, and Section 7 discusses experiments and results. Finally, in Section 8 we draw our conclusions.

* Corresponding Author. Email: etosello@fbk.eu
[1] Available at https://github.com/aiplan4eu/unified-planning



(a) (b) (c) (d)

Figure 1: The *Doors* domain. (1a) A robot r has to move from q_S to q_G, passing through doors $\{d_1, d_2\}$, initially closed. Each door opens when its button is pushed. (1b) Initially, r tries to reach q_G but finds d_1 closed, shown by blue dots indicating sampled configurations. (1c) After opening d_1, it tries to reach q_G again but finds d_2 closed. (1d) Only after opening both d_1 and d_2, r finds a collision-free path from q_S to q_G.

2 Problem Statement

In this Section, we formalize a TAMP problem with mobile agents moving within a workspace populated by task-dependent obstacles. As a motivating example, consider a robot tasked with navigating an office with sliding doors controlled by button presses (see Figure 1). To reach its destination, the robot has to find a sequence of actions to move and open doors. Simultaneously, it must physically execute these actions, finding motion primitives that ensure collision-free movement. Upon pressing the button, it must be aware of the change in door configuration so it can pass through and reach the target. A formal definition of this class of problems follows.

Definition 1. *A (ground)* ***Task And Motion Planning problem*** *is a tuple* $\psi = \langle \mathcal{R}, \mathcal{W}, \mathcal{C}, \mathcal{U}, \mathcal{V}, I, \mathcal{A}, \mathcal{G} \rangle$ *such that:*

- $\mathcal{R}$ *is a set of mobile agents, where each agent* r *is characterized by a certain geometric model.*
- $\mathcal{W} \subseteq \mathbb{R}^N$ *(N = 2 or N = 3) is the workspace, that is the physical volume of all end point positions reachable by the robots in* $\mathcal{R}$*. We define* $\mathcal{W}_f$ *as the subset of* $\mathcal{W}$ *free from fixed obstacles.*
- $\mathcal{U}$ *is a map that assigns to each agent* $r \in \mathcal{R}$ *a motion model* $\mathcal{U}_r$*, that is a mathematical representation of the kinematic and dynamic laws that allows the agent to evolve within* $\mathcal{W}$*.*
- $\mathcal{C}$ *is the configuration space, where* $\mathcal{C}_r \subseteq \mathcal{C}$ *is that subset of* $\mathcal{C}$ *that represents the joint configurations that* $r \in \mathcal{R}$ *may assume given its motion model. In this context,* $occ(r, q) \subseteq \mathcal{W}_f$ *is the set of points in* $\mathcal{W}_f$ *occupied by* r *when in configuration* $q \in \mathcal{C}_r$*.*
- $\mathcal{V} = \{f_1, .., f_k\}$ *is a finite set of variables (or fluents)* $f \in \mathcal{V}$*, each with a finite or infinite domain* $Dom(f)$*.*
- I *is the initial task state, which assigns a value* $I(f) \in Dom(f)$ *to each* $f \in V$*.*
- $\mathcal{A}$ *is a set of actions* $a = \langle \mathcal{P}, \mathcal{E}, \mathcal{M} \rangle$ *such that:*
  - $\mathcal{P}$ *is a set of preconditions* $pre \in \mathcal{P}$*, with* pre *a Boolean combination of atoms* $f = v$*, with* $f \in \mathcal{V}$ *and* $v \in Dom(f)$*.*
  - $\mathcal{E}$ *is a set of effects* $eff \in \mathcal{E}$ *each of the form* $f := v$ *with* $f \in \mathcal{V}$ *and* $v \in Dom(f)$*.*
  - $\mathcal{M}$ *is a (possibly empty) set of motion constraints of the form* $\langle r, q_S, q_G, O \rangle$*, where* $r \in \mathcal{R}$ *is the agent performing a,* $q_S \in \mathcal{C}_r$ *is its start configuration,* $q_G \in \mathcal{C}_r$ *is the target configuration, and* $O \subseteq 2^{\mathcal{R} \times \mathcal{C}}$ *is a function associating the other movable agents, which r must avoid, to the configurations they occupy.*
- $\mathcal{G}$ *is the goal condition, represented as a Boolean combination of atoms of the form* $f = v$*, with* $f \in \mathcal{V}$ *and* $v \in Dom(f)$*.*

Focusing on the semantics of the problem, a state S is a total assignment of values to the fluents such that $S(f) \in Dom(f)$ for all $f \in \mathcal{V}$. An action is applicable in a state S if its preconditions are satisfied by substituting each fluent f appearing in the Boolean combination with $S(f)$ and if all the motion constraints $\mathcal{M}$ are satisfiable.

A motion constraint $\langle r, q_S, q_G, O \rangle$ is satisfied if there exists a collision-free path τ: [0,1] $\rightarrow \mathcal{C}_r$ that moves r from $\tau(0) = q_S$ to $\tau(1) = q_G$. τ must be compliant with the motion model $\mathcal{U}_r$, must reside in $\mathcal{W}_f$, i.e., $\forall t \in [0,1]$. $occ(r, \tau(t)) \subseteq \mathcal{W}_f$, and must be collision-free with the obstacles listed on O, i.e., $\forall t \in [0,1]$. $\forall \langle r', q' \rangle \in O$. $occ(r, \tau(t)) \cap occ(r', q') = \emptyset$.

The successor of S, once applied $a = \langle \mathcal{P}, \mathcal{E}, \mathcal{M} \rangle$, is $a(S)$ where:

$$a(S)(f) = \begin{cases} v & \text{if } \langle f := v \rangle \in \mathcal{E} \\ S(f) & \text{otherwise} \end{cases}$$

The plan π solving ψ is a sequence $\langle a_0, \ldots, a_n \rangle$ of actions such that a_0 is applicable in I, each action a_i is applicable in $a_{i-1}(a_{i-2}(\cdots(a_0(I))))$, and the final state satisfies $\mathcal{G}$.

In our example, r navigates a deterministic and fully observable 2D map with fixed obstacles (walls). Thus, $\mathcal{W}_f$ includes all points on the map not occupied by walls. The robot uses a ReedsShepp-type motion model [23], with configuration (x, y, θ), where (x, y) are Cartesian coordinates and θ is the orientation angle. A motion constraint $\langle r, q_S, q_G, O \rangle$ is satisfied if we find a path from q_S to q_G while avoiding doors in O, which open when their button is pressed.

Definition 1 is a ground formalization of the TAMP problem we tackle. For the sake of brevity, we only formalize the syntax and semantics of the ground representation. In practical modeling, we adopt a lifted representation, as is customary in the planning community. Our peculiarity is to consider movable agents and configurations of interest as objects of the problem, allowing fluents to have subsets of configurations as domains. This is useful for specifying goals for the agents and expressions evaluating movable agents or configurations. If e_r is an expression evaluating a movable agent and e_q is an expression evaluating a configuration, a motion constraint will have the form $\langle e_r, e_{q_S}, e_{q_G}, O_l \rangle$, with O_l a set of pairs of the form $\langle e_r, e_q \rangle$. The semantics is given by grounding: we assess the expressions within the lifted motion constraint in the state where the action starts, and we obtain the ground motion constraint of Definition 1.

3 Meta-Engine Framework

To efficiently solve the TAMP problem ψ, we developed a meta-engine framework that interleaves an off-the-shelf task planner and an off-the-shelf motion planner provided as inputs. The task planner generates a candidate plan without considering motion constraints. We then check this plan for motion feasibility. If unfeasible, we extract data from the search space of the motion planner on unrealizable

Algorithm 1 Our Meta-Engine Framework

```
1  procedure METASOLVE(ψ, timeout)
2      cache ← ∅                                  ▷ Cache of successful motion constraints
3      ψ' ← ψ                                     ▷ ψ' is the abstracted problem
4      while True do
5          π ← task-plan(ψ')                      ▷ Call task planner on ψ'
6          if π ≠ ∅ then
7              found, mc ← CHECKMOTIONS(π, cache, timeout)
8              if found then
9                  return ⟨π, cache⟩              ▷ π is a valid plan, cache has the paths
10             else
11                 ψ' ← REFINEPROBLEM(ψ', mc)
12         else
13             timeout ← timeout*2                ▷ Increase motion planner timeout
14             ψ' ← ψ                             ▷ Reset the refinements
```

Algorithm 2 Checking motion constraints in a given plan

```
1  procedure CHECKMOTIONS(π, cache, timeout)
2      found ← True                               ▷ Final validity of the plan
3      mc ← ∅                                     ▷ Set of unsat motion constraints and learned info
4      for each a = ⟨P, E, M⟩ ∈ π do
5          for each c = ⟨r, q_S, q_G, O⟩ ∈ M do
6              if ∄⟨c, τ⟩ ∈ cache then            ▷ cache stores past solutions
7                  τ, ⟨σ, ω⟩ ← motion-plan(c, timeout)
8                  if τ ≠ ∅ then                  ▷ If a path τ is found
9                      cache ← cache ∪ {⟨c, τ⟩}   ▷ Save path τ in cache
10                 else                           ▷ If no path was found
11                     found ← False              ▷ π cannot be validated
12                     mc ← mc ∪ {⟨r, q_S, σ, ω⟩} ▷ Learn info
13     return found, mc
```

constraints, refine the task problem, and restart the process. In this Section, we detail this general schema and the refinement process.

As reported in Algorithm 1, the task planner searches for a plan π that is valid for the problem ψ while disregarding the motion constraints (line 5). By excluding the motion aspect, the problem is reduced to a traditional task-planning problem. If a valid plan is found, the function CHECKMOTIONS checks all the motion constraints of all the actions involved in the plan (line 7). Since many motion planning algorithms are sample-based and do not guarantee termination if a path does not exist, we set a *timeout* to each invocation of the motion planner. The algorithm keeps a *cache* which stores each motion constraint successfully checked and its trajectory τ. If all the motion constraints of all the actions of π are found to be realizable by the motion planner, then the plan is returned together with the *cache* (line 9). If at least one motion constraint cannot be solved, we refine ψ' (see *Topological Refinements*). If no candidate plan is found, the problem is either unsolvable or a previous plan was feasible, but the motion planner failed to find a path within the given time. Thus, we double the *timeout* of the motion planner, reset our refinements, and restart the algorithm (line 13). The *cache* is not reset, preserving any valid motion plan and improving the efficiency of the algorithm.

In Algorithm 2, CHECKMOTIONS generates a value *found* which is True if the plan is valid. Otherwise, it returns a set *mc* of motion constraint explanations $\langle r, q_S, \sigma, \omega\rangle$, where $r \in \mathcal{R}$, $q_S \in \mathcal{C}_r$, $\sigma \subseteq \mathcal{C}_r$ and $\omega \in 2^{\mathcal{R}\times\mathcal{C}}$. This means the motion planner did not find a path for r from q_S to any target in σ due to obstacles r' in c', with $\langle r', c'\rangle \in \omega$. We explain how this data is computed and used for refinement.

Topological refinements. It is crucial for the performance of our technique that CHECKMOTIONS can provide the explanations *mc* for the unsatisfied motion constraints. If the constraint $\langle r, q_S, q_G, O\rangle$ is infeasible, it means the target q_G is blocked by fixed or movable obstacles (or we did not give enough time to the motion planner, but this is handled as discussed above). In the first case, no valid plan exists for *r*. In the second case, some obstacles in *O* prevent r from reaching the target and must be moved. In our motivating example, this means a closed door is blocking the robot from reaching its destination.

If the constraint $\langle r, q_S, q_G, O\rangle$ is invalid, we find the convex hull

$$\mathcal{H}(q_S) = \{\sum_{j=1}^{K} \lambda_j p_j \mid \bigwedge_{j=1}^{K} \lambda_j \geq 0 \wedge \sum_{j=1}^{K} \lambda_j = 1\}$$

of the points $\{p_1, \ldots, p_k\}$ sampled by the motion planner from q_S.

Let X be the set $\{q_1, \ldots, q_m\} \subseteq \mathcal{C}_r$ of *interesting* configurations that the agent may assume, i.e., the motion constraints' configurations involving r for the ground case or the objects of type *Configuration* for the lifted case. We check which configurations yield an occupancy that does not belong to $\mathcal{H}(q_S)$. The idea is that $\mathcal{H}(q_S)$ is an approximation of the positions that the agent can reach and we want to compute the set of interesting locations that are unreachable from the specified starting configuration q_S. We call the resulting set σ and we define it formally as $\{q \in X \mid occ(r, q) \not\subseteq \mathcal{H}(q_S)\}$.

The second element of the explanation concerns the blocking movable obstacles. Not all the obstacles in O block the agent from reaching its goal, hence we isolate the obstacles that prevent the motion planner from computing a feasible path connecting q_S to σ. We call this set $\omega \subseteq O$. This set can be efficiently computed by keeping track of the collisions analyzed by the motion planner: if a collision happens in a point $p \in occ(r', q')$ with $\langle r', q'\rangle \in O$, we add the element $\langle r', q'\rangle$ to ω. The intuition is that obstacles we do not collide with do not hinder finding a valid plan, offering no useful information for pruning the task planner's search space. Hence, they can be omitted.

CHECKMOTIONS collects all the conflicts in μ and uses this data to refine the problem (line 11). The idea is to prevent the task planner from using actions that are not feasible because of the explanations in μ. We present here two refinements, one for the grounded problem of Definition 1 and a more practical one for the lifted case.

In the grounded refinement, we remove any actions with motion constraints that conflict with explanations in *mc*, thereby refining the set of actions. Formally, given $\psi = \langle \mathcal{R}, \mathcal{W}, \mathcal{C}, \mathcal{U}, \mathcal{V}, I, \mathcal{A}, \mathcal{G}\rangle$, we return $\psi' = \langle \mathcal{R}, \mathcal{W}, \mathcal{C}, \mathcal{U}, \mathcal{V}, I, \mathcal{A}', \mathcal{G}\rangle$ with $\mathcal{A}'$ defined as:

$$\{a = \langle \mathcal{P}, \mathcal{E}, \mathcal{M}\rangle \in \mathcal{A} \mid \nexists m = \langle r, q_S, q_G, O\rangle \in \mathcal{M}. \langle r, q_S, \sigma, \omega\rangle \in mc \wedge (q_G \in \sigma \vee \omega \subseteq O)\}$$

This prevents the execution of actions with known unrealizable constraints (with the given *timeout*).

The lifted case is similar, but requires the addition of preconditions to eliminate all the groundings that would conflict with the learned explanations. For each action a in the lifted TAMP problem, we add the following precondition for each lifted motion constraint $m = \langle e_r, e_{q_S}, e_{q_G}, \mathcal{O}^l\rangle$ of a and for each explanation $\langle r, q_S, \sigma, \omega\rangle \in mc$:

$$e_r \neq r \vee e_{q_S} \neq q_S \vee \bigwedge_{q\in\sigma} e_{q_G} \neq q \vee \bigvee_{\langle r', c'\rangle \in \omega} \bigwedge_{\langle e_r, e_c\rangle \in O^l} ((e_r \neq r') \vee (e_c \neq c'))$$

which informally means that m of a is consistent with the explanation if any of the following conditions are met: i) e_r does not evaluate to r; ii) e_{q_S} does not evaluate to q_S; iii) the destination e_{q_G} does not evaluate to any element of σ; iv) there exists an obstacle in ω that has a different configuration or doesn't exist in this constraint.

Theoretical Guarantees. Many motion planners exist and can be leveraged by our meta-engine. In our case, we exploit the Rapidly exploring Random Tree (RRT) algorithm [20] and its Lazy version. Our

Algorithm 3 Tampest

```
1  procedure SOLVE(ψ, h_max, timeout)
2      cache ← ∅
3      while True do
4          ⟨h, mc⟩ ← ⟨1, ∅⟩
5          solver ← SMT-solver()
6          solver.add-assertion(initial-step(ψ))
7          while h ≤ h_max do
8              f, l ← incremental-step(ψ, h)
9              solver.add-assertion(f)
10             if mc ≠ ∅ then
11                 solver.add-assertion(get-lemmas(mc, h))
12             solver.push()
13             solver.add-assertion(l)
14             while solver.solve() do
15                 π ← get-plan(solver.get-model())
16                 found, mc' ← CHECKMOTIONS(π, cache, timeout)
17                 if found then
18                     return ⟨π, cache⟩
19                 else
20                     solver.pop()
21                     for each i ∈ {1, ..., h} do
22                         solver.add-assertion(get-lemmas(mc', i))
23                     mc ← mc ∪ mc'
24                     solver.push()
25                     solver.add-assertion(l)
26             solver.pop()
27             h ← h + 1
28         timeout ← timeout * 2
```

proposal becomes *probabilistic complete* assuming the task planner is complete, because the probability of finding a solution tends to 1 as the *timeout* given to the motion planner to compute a plan tends to infinity. We also assume that when a motion from q_S to q_G fails, q_G is always unreachable, preventing to enter an infinite loop as the *interesting* configuration set is finite.

4 SMT-based Specialization

We tailored our framework to leverage the incremental solution capabilities of SMT-based solvers. Such solvers maintain a stack of constraints (called assertions), enabling efficient repeated satisfiability checks as constraints are pushed onto or popped from the constraint stack. This feature eliminates the need for restarting the planning routine upon failure to find a valid plan, enhancing overall scalability.

Our approach is called TAMPEST and it iterates between task and motion planning while progressively increasing the search depth until finding a valid plan or reaching the maximum step horizon h_{max}.

As shown in Algorithm 3, the general schema is that of the meta-engine in Algorithm 1, with the outer while loop serving for the refinement of the motion planner *timeout*, the learned explanations *mc*, and the horizon h. The inner loop is the focal point of the approach. We encode the task part of ψ as an SMT planning problem, analogously to many SATPlan-like approaches [17, 24], and we add to our SMT solver the assertions relative to the initial state, which hold at step 0 (line 6). At each step $h \leq h_{max}$, we generate and add the assertions f and l (lines 9-13). As in [5], f asserts that a selected action implies its preconditions and effects, the state remains the same unless changed by an action effect, and only one subset of non-mutex actions is taken at time. Assertion l, instead, characterizes the goal. The solver searches for a valid plan π, which means finding a satisfying assignment for the asserted logical formulae (line 15). If a model exists, we check the motion feasibility of π via CHECKMOTIONS, possibly exploiting the cache (line 16). If all constraints are satisfied, we return the plan and the paths (line 18). Otherwise, we pop the solver and add the logical lemmas representing the topological refinements *mc*'. We use the same logical formulation used for the lifted refinement in the meta-engine encoding the preconditions as an SMT formula instantiated at all the symbolic times $i \in \{1, \ldots, h\}$. Once this data is added, we push the solver, re-add the goal, and try to find a solution again (lines 20:25). Every time we enlarge the encoding bound, we permanently add the lemmas for all the explanations in *mc* at h, ensuring their validity across all encoding steps (line 11).

5 Modeling and Benchmarking

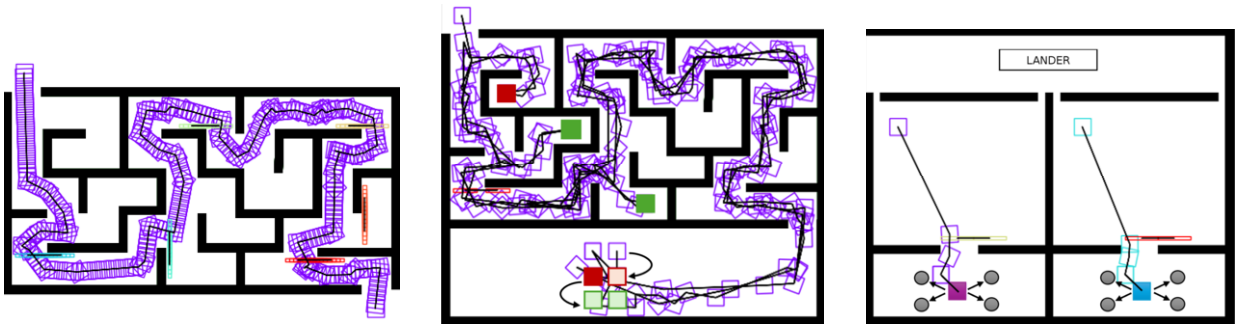
Besides formulating the TAMP problem of Definition 1 and defining suitable TAMP solvers, we developed a comprehensive open-source framework for modeling and benchmarking these problems. An overview of the key components of this implementation follows, along with a description of the benchmark suite we designed.

UP[1] is an open-source, planner-agnostic planning library that collects planning tools and algorithms to model, manipulate, and solve classical, numerical, temporal, and other complex tasks, such as multi-agent assignments. To enable the modeling of TAMP problems, we extended the TAMP modeling of the UP adding obstacle avoidance. Besides preconditions and effects, motion actions include motion constraints of the form *path(r, q_S, [q_G], {o : q_o ∀ o ∈ $\mathcal{O}$})*, i.e., there $\exists \pi : [q_S, [q_G]] \rightarrow \mathcal{C}_r$ for $r \in \mathcal{R}$ and $\{o : q_o \; \forall \; o \in \mathcal{O}\}$, as in Definition 1. Non-fixed objects are defined as *Movable Objects* with a geometric and motion model. Their configurations are *Configuration Objects* with a value in the form provided by the motion model of the agent (e.g., (*x*, *y*, *yaw*) in SE(2)). The workspace is an *Occupancy Map* collecting all useful data for motion planning and collision avoidance with fixed obstacles, such as the 2D image or 3D mesh of the operating environment and its reference system. We allow fluents that accept as input a *Movable Object* and output its current *Configuration Object* within the *Occupancy Map.* As for all the tools of the UP library, this extension is independent of the planning language and planner available to define and solve this problem.

With this extension, we offer a set of benchmarks that task robotic agents with Navigating Among Movable Obstacles (NAMO) [27], i.e., moving through a workspace while removing or avoiding movable obstacles. As in [19], we assume the search space is i) *geometric*: motion planning focuses only on finding feasible object poses based on the geometric constraints of the world; ii) *fully observable*: the initial state is completely known both geometrically and semantically; iii) *deterministic*: world state changes exclusively result from planned actions, and object motions precisely adhere to the output of the motion planner. We consider the following evaluation criteria:

- **Infeasible task actions**. Some task actions are impossible due to obstructing obstacles that prevent feasible motion plans.
- **Large task spaces**. The task planning problem requires substantial search effort.
- **Motion/Task Trade-off**. The problem can be solved with fewer steps if the right obstacles are moved.
- **Non-monotonicity**. Some objects need to be moved more than once for achieving the goal.
- **Non-geometric actions**. Some actions, like perception, change the discrete state but not the robot configuration.

The description of our benchmarks follows. For each domain, we offer a comprehensive setup, ensuring faithful replication in both 2D and 3D environments. This approach guarantees reliable assessment of solver performance, even within complex search spaces. In 2D



(a) (b) (c)

Figure 2: 2D setups of our benchmarks. (2a) *Maze* with $n_d = 5$ doors. (2b) *Delivery* with $n_d = 1$ door and $n_p = 4$ parcels, one delivered. (2c) *Rovers* with $n_d = 2$ rovers (and doors) collecting rock and soil samples, and taking images of $n_c = 4$ configurations around the samples.

Criteria	Doors	Maze	Delivery	Rover
Infeasible task actions	x	x	x	x
Large task spaces	x	x	x	x
Motion/Task trade-off		x	x	x
Non-monotonicity			x	x
Non-geometric actions				x

Table 1: Criteria evaluated by each benchmark problem.

scenarios, movable objects are polygons and the robot navigates using a ReedsShepp path within a black-and-white map, where black represents areas occupied by fixed obstacles. In 3D, objects are 3D rigid bodies and move according to an SE(3) motion model. Due to limited space, we will discuss only the benchmarks deemed paradigmatic according to the outlined evaluation criteria (see Table 1):

- **Doors**. One robot needs to navigate through n_d closed doors to reach a final destination, using the {*move*, *open*} action set (see Figure 1). *move* enables the robot to navigate from a start to a goal location and incorporates a motion constraint that avoids collisions with movable and static obstacles (doors and walls). *open* allows the robot to open a door when positioned in front of it, like pushing a button. Once the button is pushed, the door configuration changes instantaneously from closed to open. n_c extra configurations are randomly sampled in the free space: they do not aid in achieving the goal, they merely expand the task space. Thus, even if the problem is simple, the optimal plan contains $2n_d + 1$ steps while the worst-case scenario needs $2n_d + n_c + 1$ steps to take the robot from start to goal while opening all the doors and visiting all extra locations (*large task space*). Closed doors make some locations unreachable (*infeasible task actions*).
- **Maze**. A robot must navigate out of a maze while visiting n_c randomly distributed configurations (see Figure 2a). n_d doors block various passages, not all leading to exit or target locations. Their motion model requires the motion planner to compute opening paths. Actions are {*move*, *open*}. Again, we are exploring a *large task space* equipped with *infeasible task actions*. Moreover, we should find a good *motion/task trade-off* to efficiently solve the problem: while opening all doors and reaching the assigned targets is valid, only opening necessary doors yields efficiency.
- **Delivery**. Inspired by the *delivery* domain of IPC, *Maze* locations become parcels with no geometry and motion model. They are distinguished by colors and must be arranged into rows by color, each row delivered before the next (see Figure 2b). Actions are {*move*, *open*, *load*, *unload*}, where *load* involves collecting a parcel and placing it atop an agent. *unload* enables the agent to remove an item from its cargo and deposit it at a specified location (*large task space*). The robot has a fixed capacity (*numerical problem*), and can unload packages only when positioned in front of the unloading location, though some parcels are already at their stations. n_d doors block n_d passages, some of which are useful to reach the unloading area (*unfeasible task actions*). The layout of the unloading area and the presence of obstructing doors influence the *motion/task trade-off*. Parcels initially at unloading stations enable assessment of *non-monotonicity*: if a parcel blocks the unloading of other items, it must be temporally relocated.
- **Rovers**. We reproduce the *rover* domain of IPC to demonstrate the generality of our approach (see Figure 2c). n_d rovers must collect rock and soil samples, separated from each robot by a door. Then, they must calibrate their cameras, photograph n_c objectives located around each sample without occlusions, and send the results back to a lander. Due to obstacles that limit the reachability of parts of the workspace, one rover must be utilized for each sample and the objectives around it. Actions are {*move*, *open*, *calibrate*, *sample rock*, *sample soil*, *send analysis*, *drop*, *take image*, *send image*}, and some of them change only the discrete state and not the configuration space (*non-geometric actions*).

6 Related Work

Many planners exist that combine symbolic and geometric search. As an example, the aSyMov planner [3] interleaves a FF-based task planner with lazily-expanded roadmaps. However, this approach is impractical when action plans are valid in the symbolic space but infeasible in the geometric one. To address this issue, many approaches have been developed over the years. For instance, Dornhege *et al.* [7] add semantic attachments to the definition of the task, and they call the motion planner after each action to check both its geometric and semantic feasibility. Other strategies, as discussed in [16, 26, 10, 29, 9], are tailored to specific classes of manipulation problems, limiting their adaptability to new domains, such as those introduced in this paper, without significant engineering effort. They lack a modular, domain-agnostic problem description language with clear semantics. In this regard, ROSPlan [4] offers tools for AI Planning in a ROS system, while PDDLStream enhances this capability by integrating symbolic planners and black-box samplers by extending Planning Domain Definition Language (PDDL) [13] with streams: declarative specifications of sampling procedures that link in a black-box way the symbolic representation of constraints with their sample-based counterparts. In TAMP, they are used to map the existence of collision-free paths with the functions checking their validity. Our formulation is less general as it is tailored specifically towards TAMP problems. However, this targeted approach allows us to exploit the motion planner's output to prune large regions of the task search space, significantly reducing the computational overhead.

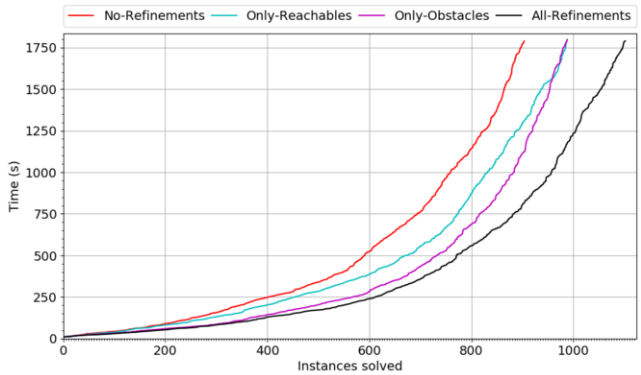


Figure 3: Overall performance on all benchmark instances and all planners when exploiting different topological refinements.

Indeed, calling the motion planner after each symbolic call is time-consuming, particularly when dealing with geometrically unfeasible states [18]. To enhance efficiency, the geometric search is typically limited to candidate symbolic plans. Srivastava *et al.* [26], for example, interface a task planner with an optimization-based motion planner and use a heuristic to remove occluding objects. Dantam *et al.*. [6] propose TMKit: an incremental SMT solver that incrementally generates symbolic plans and call the motion planner for validation. They all suffer from long processing time, solve problems consisting of a limited number of actions and, given their focus on manipulation tasks, handle a limited quantity of manipulable objects. Some approaches exist that tries to overcome these limitations. Similar to TMKit, our SMT specialization employs an incremental approach to generate a valid symbolic plan. Initially, it assumes the validity of all motion actions within the plan. Once a task plan is established, it invokes the motion planner to verify feasibility. For any unfeasible motion action, we generate topological refinements on the geometric space. These refinements are leveraged at the task level, enhancing efficiency and allowing for plans with many actions.

7 Experimental Evaluation

In this Section, we evaluate our meta-engine framework across various task and motion planners. We assess the effectiveness of its SMT-based specialization and quantify improvements from topological refinements. Moreover, we compare our framework with PDDLSTREAM, highlighting our ability to integrate existing solvers and the superior performance of our proposal. Benchmarks and solvers are available open source[2]. Our test cases follows:

- **Doors**. We feature $n_d \in [1, 2, 4, \ldots, 10]$ closed doors that must all be open to reach the final destination. Additionally, either 0 (n_c = [(0, 0)]) or 10 extra configurations are randomly distributed in the reachable space (n_c = [(10,0)]), the initially unreachable space (n_c = [(0, 10)]), or equally split between the two (n_c = [(5,5)]).
- **Maze**. We increase domain complexity by introducing $n_d \in [1, 2, 3, \ldots, 10]$ doors in a maze, where not all block the final destination. Extra-configurations become $n_c \in [0, 1, 2, 3, \ldots, 10]$ mandatory inspection targets randomly placed within the maze.
- **Delivery**. We sample $n_d \in [1, 2, 4, \ldots, 10]$ closed doors, not all obstructing the target, and $n_r + n_g \in [0, 1, 2, 3, \ldots, 8]$ red and green parcels. Colors are randomly sampled among available parcels. Parcels must be delivered in two rows, with at most 4 red parcels placed in the front and 4 green parcels in the back. $d_r \leq 3$ red parcels and $d_g \leq 3$ green parcels are already in their delivery spots, eventually blocking the reachability of the unloading locations behind them, that means $n_c = x = [(n_r, n_g, d_r, d_g)]$. The robot's load capacity n_l ranges from 1 to 4.
- **Rovers**. We involve $n_d \in [2, 4, 6, 810]$ robots (and doors). Each robot analyzes either one soil or one rock sample, each one situated one closed door away from the robot. We design $n_c \in [0, 1, 2, 3, 4]$ objectives to be photographed around each sample.

We tested *Maze* and *Rover* domains in both 2D and 3D setups, while *Doors* and *Delivery* tests were limited to their 2D implementations. Indeed, these setups closely resemble those of the former domains.

We instantiate our *Meta-Engine* with FAST-DOWNWARD [15], the Expressive Numeric Heuristic Search Planner (ENHSP) [25], and TAMER [30], and evaluate their performance compared with TAMPEST (with h_{max} = 100), where the last three can solve numerical problems such as our *Delivery* domain. We combine each solver with the RRT [20] and LAZYRRT motion planners (with *timeout* = 3s). In 2D scenarios, we implement an ad-hoc collision checker that verifies the feasibility of a pose of the robot pose by ensuring that its footprint does not intersect obstacles. In 3D, we exploit the Flexible Collision Library [22]. Finally, we study our refinement schema by disabling some of the explanations computed by CHECKMOTIONS. We set the topological refinements $mc' = \{\langle \sigma, \omega \rangle\}$ as follows. *All-Refinements* is the full algorithm as described in the previous sections. *Only-Reachables* assumes $\omega = O$, disabling the analysis of the obstacles with which the agent collided, but retaining the analysis of the unreachable points. *Only-Obstacles* forces $\sigma = \{q_G\}$, retaining the obstacles analysis but disabling the unreachable configurations one, to only remove the target location. *No-Refinements* forces $\sigma = \{q_G\}$ and $\omega = O$, removing only the violated constraint.

Focusing on PDDLSTREAM, we explore its *incremental*, *focused*, *binding*, and *adaptive* variants equipped with FAST-DOWNWARD [14], as provided by default. To enable them to solve our benchmarks, we convert the motion constraints into streams, mapped with functions that certificates the existence of paths. We employ the same motion planners and collision checkers as before.

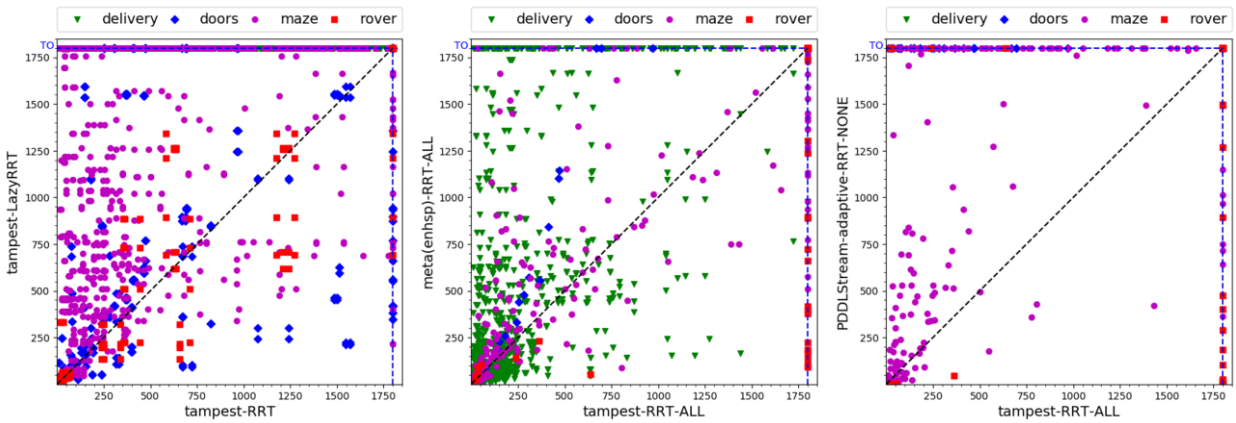
We set a global timeout of 1800 s, a memory limit of 10 GB, and ran tests on an Intel Xeon CPU 6226R @2.9GHz.

Results. In Figure 3, we show the impact of leveraging topological refinements across all instances of all domains. The x-axis denotes the number of solved instances, while the y-axis represents computational time. Utilizing *All-Refinements* increases the number of solved instances by roughly 20% compared to single refinements and 30% compared to none, also reducing computational time. Some instances have numerous obstacles obstructing large portions of the workspace, highlighting the usefulness of leveraging topological refinements, especially in scenarios with a high number of *infeasible task actions*.

Focusing on the motion planner, RRT outperforms its lazy version, which performs collision checking only at the end. Indeed, our setups feature many obstacles, causing LAZYRRT to add a significant number of validation steps during collision checking. Figure 4a proves this statement when using TAMPEST, especially for the *Delivery case*, where LAZYRRT timeouts in all cases.

In Table 2, we compare planners across all domains, once selected RRT. All PDDLSTREAM variants exhibit lower performance compared to other algorithms, with *adaptive* showing the best results, followed by *incremental*. Indeed, *incremental* generates all streams in advance and then searches for a plan, while *adaptive* first finds a plan, checks its motion validity, and dynamically adjusts its search strategy based on the progress. Their lower performance may stem from

[2] Available at https://github.com/fbk-pso/tampest.git



(a) (b) (c)

Figure 4: Computational time (in seconds) to solve the problem by domain (with timeout TO = 1800 s). (4a) RRT vs. LAZYRRT when using TAMPEST, regardless of the refinement exploited. (4b) TAMPEST vs. *Meta*(ENHPS), both with RRT and *All-Refinements*. (4c) TAMPEST vs. PDDLSTREAM-*adaptive*, both with RRT and TAMPEST with *All-Refinements*.

Planner	**Doors** (tot. 24)		**Maze** (tot. 220)		**Delivery** (tot. 525)		**Rover** (tot. 50)	
PDDLSTREAM-*binding*	2		17		-		0	
PDDLSTREAM-*focused*	0		17		-		0	
PDDLSTREAM-*incremental*	6		43		-		1	
PDDLSTREAM-*adaptive*	1		65		-		1	
Meta(FAST-DOWNWARD)	4	3	52	67	-		7	9
Meta(TAMER)	4	7	51	56	376	376	12	11
Meta(ENHSP)	13	21	121	154	287	287	18	**27**
TAMPEST	17	**24**	126	**164**	415	**422**	12	14

Table 2: Overall performance of all planners on all benchmarks when combined with RRT (left column with *No-Refinement*, right column with *All-Refinements*). All PDDLSTREAM variants are equipped with FAST-DOWNWARD, as provided by default by this framework.

the use of FAST-DOWNWARD, which also impacts our *Meta*(FAST-DOWNWARD). *Meta*(TAMER) has good performance in the numeric case, while *Meta*(ENHSP) and TAMPEST demonstrate the highest success rates: they can manage *large task spaces* more effectively, achieving a good *trade-off* between motion and task. In Figure 4b, we better compare the quality and quantity of the solutions proposed by these two algorithms, each equipped with RRT and all refinements. Our proposal performs particularly well in the *Maze* (magenta dots) and *Delivery* (green triangles) domains, i.e., it can face effectively also *non-monotonic* scenarios. When adding *non-geometric* actions as in the *Rover* domain (red squares), instead, our solver excels with simpler instances but faces scalability issues as plan size increases, similar to other SAT/SMT-based planners. Processing more parameters leads to longer resolution times. In detail, when combined with RRT and all refinements in a 2D setup (best case), it fully solves the *Doors* domain. In the *Maze* domain, performance decreases with 8 or more doors, requiring about 20 actions for 8 doors and failing with 10. In *Delivery* scenarios, scalability issues arise with 4 red and 4 green parcels if 3 block others and the robot's capacity is below 4 (requiring at least 20 actions). For *Rovers*, the performance decreases with 4 robots sampling rocks or soil and photographing at least one sample, needing at least 40 actions. Finally, in Figure 4c we compare PDDLSTREAM's *adaptive* variant with TAMPEST (both with RRT and TAMPEST with all refinements). The former consistently times out, even when our approach easily finds solutions. This stands out notably in the *Maze* domain (magenta dots).

8 Conclusion and Future Work

In this paper, we provided a detailed representation of a multi-agent TAMP scenario with one agent moving at a time and multiple task-dependent obstacles. Our contributions include a general problem formulation and semantic definition, supported by an open-source library for modeling and benchmarking. We also introduced a novel meta-engine framework for combining off-the-shelf task and motion planners to solve complex scenarios. We proposed using geometric context to generate topological refinements and prune the task planner's search space. Additionally, we demonstrated how this meta-engine can be adapted for an incremental SMT-based task planner, named TAMPEST. We compared TAMPEST with existing planners interleaved with sample-based motion planners, with and without topological refinements. SMT's incremental nature accelerates problem resolution, while topological refinements decrease the time required to find a valid plan. Finally, we integrated PDDLStream enabling direct comparison of solvers on the same input data: TAMPEST outperforms PDDLStream, especially when using topological refinements.

In future work, we will include metric time and address scenarios with multiple agents moving simultaneously [21, 8]. We will also integrate replanning mechanisms to handle non-determinism.

Acknowledgments

This work has been partially supported by the AI4Work project funded by EU Horizon 2020 research and innovation program under GA n. 101135990, the STEP-RL project funded by the European Research Council under GA n. 101115870, and by the Interconnected Nord-Est Innovation Ecosystem (iNEST) funded by the European Union Next-GenerationEU (Piano Nazionale di Ripresa e Resilienza (PNRR) – mission 4 component 2, investment 1.5 – D.D. 1058 23/06/2022, ECS00000043).

References

[1] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009. ISBN 978-1-58603-929-5. URL http://dblp.uni-trier.de/db/series/faia/faia185.html#BarrettSST09.

[2] J. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962. URL http://eudml.org/doc/131533.

[3] S. Cambon, R. Alami, and F. Gravot. A hybrid approach to intricate motion, manipulation and task planning. *The International Journal of Robotics Research*, 28(1):104–126, 2009.

[4] 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. *Proceedings of the International Conference on Automated Planning and Scheduling*, 25(1):333–341, Apr. 2015. doi: 10.1609/icaps.v25i1.13699. URL https://ojs.aaai.org/index.php/ICAPS/article/view/13699.

[5] N. T. Dantam. *Task and Motion Planning*, pages 1–9. Springer Berlin Heidelberg, Berlin, Heidelberg, 2020. ISBN 978-3-642-41610-1. doi: 10.1007/978-3-642-41610-1_176-1. URL https://doi.org/10.1007/978-3-642-41610-1_176-1.

[6] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki. Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and Systems*, 2016. doi: 10.15607/RSS.2016.XII.002.

[7] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel. Semantic attachments for domain-independent planning systems. In *Towards Service Robots for Everyday Environments*, pages 99–115. Springer, 2012.

[8] S. Edelkamp, M. Lahijanian, D. Magazzeni, and E. Plaku. Integrating temporal reasoning and sampling-based motion planning for multi-goal problems with dynamics and time windows. *IEEE Robotics and Automation Letters*, 3(4):3473–3480, 2018. doi: 10.1109/LRA.2018.2853642.

[9] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling. Backward-forward search for manipulation planning. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6366–6373, 2015. doi: 10.1109/IROS.2015.7354287.

[10] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling. Ffrob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*, 37(1):104–136, 2018. doi: 10.1177/0278364917739114. URL https://doi.org/10.1177/0278364917739114.

[11] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling. Pddlstream: Integrating symbolic planners and blackbox samplers. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2020. URL https://arxiv.org/abs/1802.08705.

[12] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4(1):265–293, 2021. doi: 10.1146/annurev-control-091420-084139.

[13] P. Haslum, N. Lipovetzky, D. Magazzeni, and C. Muise. *An Introduction to the Planning Domain Definition Language*, pages 1–169. Number 2 in Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2 edition, Jan. 2019. doi: 10.2200/S00900ED2V01Y201902AIM042.

[14] M. Helmert. The fast downward planning system. *J. Artif. Int. Res.*, 26 (1):191–246, jul 2006. ISSN 1076-9757.

[15] M. Helmert. The fast downward planning system. *J. Artif. Int. Res.*, 26 (1):191–246, jul 2006. ISSN 1076-9757.

[16] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*, pages 1470–1477, 2011. doi: 10.1109/ICRA.2011.5980391.

[17] H. A. Kautz and B. Selman. Planning as satisfiability. In B. Neumann, editor, *10th European Conference on Artificial Intelligence, ECAI 92, Vienna, Austria, August 3-7, 1992. Proceedings*, pages 359–363. John Wiley and Sons, 1992.

[18] S. Kiesel, T. Gu, and W. Ruml. An effort bias for sampling-based motion planning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2864–2871, 2017. doi: 10.1109/IROS.2017.8206118.

[19] F. Lagriffoul, N. T. Dantam, C. Garrett, A. Akbari, S. Srivastava, and L. E. Kavraki. Platform-independent benchmarks for task and motion planning. *IEEE Robotics and Automation Letters*, 3(4):3765–3772, 2018. doi: 10.1109/LRA.2018.2856701.

[20] S. M. LaValle et al. Rapidly-exploring random trees: A new tool for path planning. *Research Report*, 1998.

[21] D. Le and E. Plaku. Multi-robot motion planning with dynamics via coordinated sampling-based expansion guided by multi-agent search. *IEEE Robotics and Automation Letters*, 4(2):1868–1875, 2019. doi: 10.1109/LRA.2019.2898087.

[22] J. Pan, S. Chitta, and D. Manocha. Fcl: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, pages 3859–3866, 2012. doi: 10.1109/ICRA.2012.6225337.

[23] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 145(2):367 – 393, 1990.

[24] J. Rintanen. Planning and SAT. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 483–504. IOS Press, 2009. doi: 10.3233/978-1-58603-929-5-483. URL https://doi.org/10.3233/978-1-58603-929-5-483.

[25] E. Scala, P. Haslum, and S. Thiébaux. Heuristics for numeric planning via subgoaling. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, page 3228–3234. AAAI Press, 2016. ISBN 9781577357704.

[26] 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)*, pages 639–646. IEEE, 2014.

[27] M. Stilman and J. Kuffner. Navigation among movable obstacles: real-time reasoning in complex environments. In *4th IEEE/RAS International Conference on Humanoid Robots, 2004.*, volume 1, pages 322–341 Vol. 1, 2004. doi: 10.1109/ICHR.2004.1442130.

[28] I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. doi: 10.1109/MRA.2012.2205651. https://ompl.kavrakilab.org.

[29] M. Toussaint. Logic-geometric programming: an optimization-based approach to combined task and motion planning. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, page 1930–1936. AAAI Press, 2015. ISBN 9781577357384.

[30] A. Valentini, A. Micheli, and A. Cimatti. Temporal planning with intermediate conditions and effects. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 9975–9982. AAAI Press, 2020. doi: 10.1609/AAAI.V34I06.6553. URL https://doi.org/10.1609/aaai.v34i06.6553.