Temporal Task and Motion Planning with Metric Time for Multiple Object Navigation

Elisa Tosello, Alessandro Valentini, Andrea Micheli

Fondazione Bruno Kessler, Trento, Italy
etosello@fbk.eu, alvalentini@fbk.eu, amicheli@fbk.eu

Abstract

Integrating metric time into Task And Motion Planning (TAMP) is challenging, especially with simultaneous object motion. Existing work focuses on classical and numeric TAMP, not considering deadlines, motions overlapping in time, and other temporal constraints. In this paper, we fill this gap by formalizing Temporal Task and Motion Planning (TTAMP) for multi-object navigation. We propose a novel interleaved planning technique for this problem, which leverages incremental Satisfiability Modulo Theory to ensure efficient reasoning on deadlines and action duration coupled with a motion planner supporting simultaneous object motion. Geometric data on encountered obstacles prunes unreachable symbolic regions, while temporal bounds limit the geometric search space. For multiple moving objects, our algorithm contextualizes the conflicts learned from the motion planner on overlapping actions so that entire classes of temporal plans are pruned from the search space of the task planner, ensuring the eventual termination of the interplay. We provide a comprehensive benchmark suite and demonstrate the effectiveness of our solver in leveraging these scenarios.

Code — https://github.com/fbk-pso/tampest.git

Introduction

Planning for multiple objects moving simultaneously is highly complex, requiring the integration of *task planning* (what actions to take to achieve a goal), *temporal reasoning* (scheduling activities to meet constraints), and *motion planning* (how to move the system). Consider a fleet of warehouse robots delivering orders within specific time windows. The task planning model defines available actions (e.g., motion and task allocation) and initial and final configurations of robots and items, with the aim to find a course of actions to achieve the goal. Temporal planning includes durations, time constraints (e.g., delivery deadlines), and potential overlaps (parallel execution). The motion planner ensures motion actions feasibility by checking collision avoidance with static and dynamic obstacles (e.g., other robots) while ensuring sufficient time for their physical execution.

Despite advances in Task and Motion Planning (TAMP) (Garrett et al. 2021; Dantam 2020) and Temporal Planning (Shin and Davis 2005; Coles et al. 2021; Valentini, Micheli, and Cimatti 2020), their integration is still unexplored. Current TAMP approaches generate feasible task-motion plans, but overlook action durations and overlaps. Temporal planners manage time constraints but lack motion integration and fail to consider trajectory impacts, like navigating narrow passages to meet deadlines despite moving through open spaces. The challenge lies in combining them, involving durative and overlapping motion actions, thus intertwining temporal and spatial reasoning to manage concurrent, potentially conflicting movements.

To this end, this paper introduces Temporal Task and Motion Planning (TTAMP), formalizes the TTAMP problem of concurrent multi-object navigation, and enhances the open-source modeling tool from (Tosello, Valentini, and Micheli 2024) with metric time and simultaneous motion support.

We designed and implemented a new interleaved TTAMP solver, named T-TAMPEST (Temporal Task and Motion Planning by Encoding into Satisfiability Testing), which leverages incremental Satisfiability Modulo Theory (SMT) (Barrett et al. 2009) to incorporate reasoning on deadlines and actions duration, coupled with a motion planner supporting simultaneous object motion. For motion planning, we use a sampling-based algorithm from the Open Motion Planning Library (OMPL) (Şucan, Moll, and Kavraki 2012), while the collision checker handles object-specific obstacles to accommodate robots with diverse geometries and dynamics. Our solver uses an interleaved approach, where the SMT solver generates candidate plans and a refiner checks motion constraints. Geometric data produced by the motion planner prunes those regions of the symbolic space that are unreachable due to obstacles, as in (Tosello, Valentini, and Micheli 2024). For single-object scenarios, time constraints limit the motion path search to areas reachable within the maximum action duration. The motion planner then supplies the task planner with the actual action duration to aid in state space pruning. For multiple objects, T-TAMPEST incorporates contextual information about the conflicts learned from the motion planner on overlapping actions. If the motion planner fails, new temporal constraints are imposed at the task level, pruning classes of temporal plans and ensuring the end of the interplay.

Finally, we provide a comprehensive benchmark suite for TTAMP and a detailed assessment of T-TAMPEST, showcasing its performance in leveraging such constraints.

Related Work

Integrating temporal planning with motion information remains an open challenge due to the complexity of interleaving task and motion data while accounting for metric time and overlapping actions. Efforts have been made to transition from domain-specific (Srivastava et al. 2014; Garrett, Lozano-Pérez, and Kaelbling 2018; Toussaint 2015) to domain-agnostic (Garrett, Lozano-Perez, and Kaelbling 2020; Tosello, Valentini, and Micheli 2024) TAMP approaches, as well as from classical and numeric planning to temporal planning. For example, PDDL2.1 Level 3 (Fox and Long 2003) supports durative actions, continuous resources, and time windows, with several temporal planners available (Shin and Davis 2005; Coles et al. 2021; Valentini, Micheli, and Cimatti 2020). However, combining motion and time has primarily focused on equipping motion planning problems with deadlines (Ho et al. 2022), rather than extending standard TAMP problems to temporal domains by linking PDDL2.1 Level 3 with motion planning in both space and time. Details follow.

There are motion planners that incorporate specifications given by Linear Temporal Logic (LTL) (Ho et al. 2022; Lahijanian et al. 2016; McMahon and Plaku 2014). (DeCastro and Kress-Gazit 2013), for example, uses LTL for controller synthesis. (Kress-Gazit, Fainekos, and Pappas 2009) generates a hybrid controller that ensures a robot can achieve its task, given its geometric model, admissible environments, and a high-level task in LTL. (Muhayyuddin, Akbari, and Rosell 2017) enhances LTL planning with ontologies for task reasoning and physics-based motion planning for object manipulation, demonstrated with a mobile robot in scenarios requiring object removal to fulfill tasks. LTL, however, cannot express time durations, only the order of events.

Signal Temporal Logic (STL) (Maler and Nickovic 2004) extends LTL to allow time windows. However, STL planners (Raman et al. 2014; Lindemann and Dimarogonas 2017) follow an optimization formulation to incorporate nonlinear constraints and system dynamics that is computationally expensive due to the NP-hardness of mixed integer-linear programs. Although (Takano, Oyama, and Yamakita 2021) presents an optimization-based TAMP approach using STL for robotic pick-and-place tasks, applying STL to systems with complex dynamics remains ineffective. To overcome this limitation, (Saha and Julius 2018) leverages Metric Temporal Logic (MTL) to synthesize control inputs for robotic manipulators, enabling them to execute specific manipulation actions while optimizing performance. (Edelkamp et al. 2018) addresses multigoal motion planning with time windows. It uses temporal planning to sequence goals on a roadmap of the free workspace and pairs this with a motion planner that samples points to expand the motion tree within discrete regions. (Faroni et al. 2024) presents a TAMP approach for optimizing task sequencing and execution under temporal and spatial variability. The framework decouples tasks and actions, where actions represent geometric realizations of tasks. At the task level, timeline-based planning handles temporal constraints and duration variability, while motion planning manages the motion online. All of these approaches are domain specific and lack standardization, failing to effectively integrate temporal planning with metric time and motion planning for feasible task execution, adherence to deadlines and overlapping actions.

In our case, we lift a standard TAMP problem (Dantam 2020) to the temporal case, linking PDDL 2.1 Level 3 (Fox and Long 2003) with motion planning in space and time. We formally define this extension as TTAMP and provide a tool for modeling it. Our solver is domain-agnostic for multi-object navigation, and efficiently uses task and motion data to prune the search space. It also incorporates temporal details on conflicts from overlapping actions, ensuring the eventual termination of the task-motion interplay.

Problem Statement

Consider two mobile robots tasked with delivery items (already on top of them) from one room to another, separated by a closed door, within a specific time window (see Figure 1). We classify robots and door as movable objects as they can change position. The planner must assign destinations and eventually ask for parallel motion to ensure timely deliveries. To reach their destinations, robots must open the door and move across it while synchronizing parallel motions to avoid collisions. This means avoiding fixed obstacles (the walls) and movable ones (the door in its various configurations and the other robot). Obstacles may be static (when one robot remains stationary while the other moves) or dynamic (if in motion). We define this problem as a TTAMP problem, with its formal definition provided below.

Definition 1 *A **Temporal Task and Motion Planning problem** is a tuple $\psi = \langle O, \mathcal{W}, \mathcal{C}, \mathcal{U}, \mathcal{V}, I, \mathcal{A}, \mathcal{G} \rangle$ such that:*

- *O is a set of movable objects, where each object $o \in O$ 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 objects in O. We define $\mathcal{W}_{free}$ as the subset of $\mathcal{W}$ that is free from fixed obstacles.*
- *$\mathcal{U}$ is a map that assigns to each object $o \in O$ a motion model $\mathcal{U}_o$, that is a mathematical representation of the kinematic and dynamic laws that allows the object to evolve within $\mathcal{W}$.*
- *$\mathcal{C}$ is the configuration space, where $\mathcal{C}_o \subseteq \mathcal{C}$ is that subset of $\mathcal{C}$ that represents the joint configurations that $o \in O$ may assume given its motion model. In this context, $occ(o, q) \subseteq \mathcal{W}_{free}$ is the set of points in $\mathcal{W}_{free}$ occupied by o when in configuration $q \in \mathcal{C}_o$.*
- *$\mathcal{V} = \{f_1, .., f_k\}$ 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 $f \in V$ at the start timing.*
- *$\mathcal{A}$ is the set of durative actions $a = \langle dur_a, cond_a, eff_a, mot_a \rangle$ such that:*
  - *dur_a is the duration interval $[l, u]$ for a, with $l \in \mathbb{Q}_{\geq 0}$ and $u \in \mathbb{Q}_{>0}$ being the lower and upper bound on the action duration.*
  - *$cond_a$ is the set $\{cond_{\vdash a}, cond_{\leftrightarrow a}, cond_{\dashv a}\}$ of start, overall, and end conditions. They are boolean combinations of atoms $f = v$, with $f \in \mathcal{V}$ and $v \in Dom(f)$.*

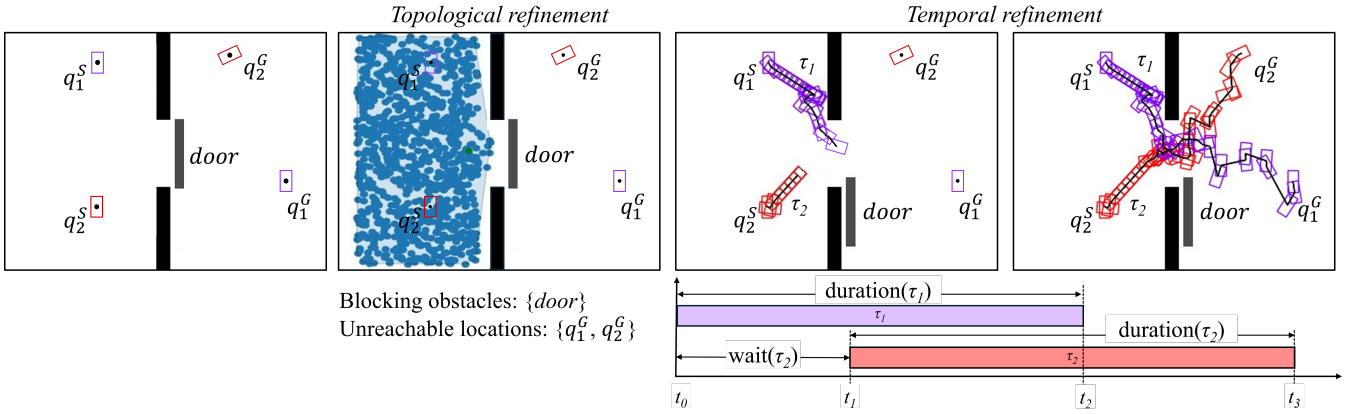

Figure 1: Two robots are tasked with delivering two parcels from q_1^S and q_2^S to q_1^G and q_2^G, respectively. On the first attempt, the motion planner informs the task planner, via *Topological Refinement*, that the door is blocking the two target locations and must be opened. Once the door is opened, *Temporal Refinement* ensures the robots coordinate their timing to pass through the door without collision. To achieve this, the second robot waits for the first to pass through the door before proceeding, allowing both robots to reach their respective destinations efficiently and on time.

  - *eff_a is the set $\{\mathit{eff}_{\vdash a}, \mathit{eff}_{\dashv a}\}$ of start and end effects, each of the form $f := v$, with $f \in \mathcal{V}$, $v \in Dom(f)$.*
  - *mot_a is the motion constraint to be fulfilled during dur_a. It can be either $\perp$ (to indicate that no motion constraint is present) or a tuple $\langle o_a, q_a^S, q_a^G, \mathcal{S}_a, \mathcal{D}_a \rangle$, where $o_a \in \mathcal{O}$ is the involved movable object, while $q_a^S \in \mathcal{C}_{o_a}$ and $q_a^G \in \mathcal{C}_{o_a}$ are its start and goal configurations. $\mathcal{S}_a \subseteq 2^{\mathcal{O} \times \mathcal{C}}$ is a function that maps objects that are stationary during dur_a (static obstacles) to their configurations, which o_a must avoid. $\mathcal{D}_a \subseteq 2^{\mathcal{O} \times \mathcal{C}}$, with $\{o \mid (o, q) \in \mathcal{S}_a \wedge (o, q') \in \mathcal{D}_a\} = \emptyset$, is a function that maps objects possibly moving during dur_a (dynamic obstacles) to their configurations at the beginning of the action, which o_a must avoid.*
- *$\mathcal{G}$ is the goal condition, being a boolean combination of atoms of the form $f = v$, with $f \in \mathcal{V}$ and $v \in Dom(f)$.*

The decisions made over time to solve this problem, including start times and durations, forms a plan. The following definition extends standard temporal plans (Fox and Long 2011) by including trajectories for motion constraints.

Definition 2 *A **time-triggered Task and Motion Planning plan** π is a sequence $\langle \langle t_0, a_0, d_0, \tau_0 \rangle, \ldots, \langle t_n, a_n, d_n, \tau_n \rangle \rangle$, with $a_i \in \mathcal{A}$ the action to be started, $t_i \in Q_{\geq 0}$ its starting time (with $t_i \leq t_{i+1}$), $d_i \in Q_{\geq 0}$ its duration and τ_i is either $\perp$ if a_i has no motion constraint or a trajectory defined as $\tau_i : [0, d_i] \to C_{o_{a_i}}$ s.t. $\tau_i(0) = q_{a_i}^S$ and $\tau_i(d_i) = q_{a_i}^G$. We write $\pi[i]$ for the i-th element of π.*

We omit the full semantics of the task planning problem: π is valid if, starting from the initial state (I) and executing the actions of π, with effects on $\mathcal{V}$ at the specified times and for the chosen durations, all action conditions are satisfied and all goal conditions are satisfied after the last action terminates (Gigante et al. 2022). We focus, instead, on the motion validity of π, considering those action instances in π with motion constraints. Two of these instances are directly motion-parallel if their execution times overlap. Extending this relationship through its transitive closure, we define the resulting relation as motion-parallel. Both relations are symmetric and irreflexive by construction. Formally:

Definition 3 *Given a plan π, two distinct elements $\pi[i]$ and $\pi[j]$ of π are **directly-motion-parallel** if $\mathit{mot}_{a_i} \neq \perp$, $\mathit{mot}_{a_j} \neq \perp$, and $\exists t \in \mathbb{R}. t_i \leq t \leq t_i + d_i \wedge t_j \leq t \leq t_j + d_j$. $\pi[i]$ and $\pi[j]$ are **motion-parallel** (written $\pi[i] || \pi[j]$) if they are directly-motion-parallel or there is a $\pi[k]$ that is motion-parallel with both $\pi[i]$ and $\pi[j]$.*

For a plan to be valid, only dynamic objects in the $\mathcal{D}$ specifications of actions may be in parallel (this can be enforced at the task-planning level using non-overlapping conditions and additional fluents, but here we treat it as a semantic constraint, leaving enforcement to the planner).

Definition 4 *A plan π is **motion-well-formed** if for every $\pi[i] || \pi[j]$, $o_{a_i} \in \mathcal{D}_{a_j}$ and $o_{a_j} \in \mathcal{D}_{a_i}$.*

To define validity, we can now isolate the groups of action instances in a plan that are motion-parallel, because we need to enforce parallel constraints only among these. This is because our model does not impose absolute movable object configuration persistence, it is possible by means of an effect to change the configuration of a movable object in the task planning space. Hence, we need separate checks for each maximal set of action instances that are motion-parallel.

Definition 5 *Given a plan π, a **parallel-motion-group** of π is a maximal subset of motion-parallel elements of π. The set of all parallel-motion-groups of π is denoted as $PMG(\pi)$.*

To compare parallel trajectories in the plan, we extend them by persisting the initial and final configurations.

Definition 6 *Given an element $\pi[i]$ of a plan π, we define*

Algorithm 1: T-Tampest

```
1  procedure SOLVE(ψ, t_p)
2      cache ← ∅                          ▷ Cache for successful motion constraints
3      mc ← ∅                             ▷ Database of new motion constraints
4      while True do
5          π ← SOLVEWITHCONSTRAINTS(ψ, mc)
6          if π ≠ ∅ then
7              found, mc' ← CHECKMOTIONS(π, mc, cache, t_p)
8              if found then
9                  return ⟨π, cache⟩       ▷ Return plan and paths
10             else
11                 mc ← mc ∪ mc'
12         else
13             t_p ← t_p * 2               ▷ Increase motion planning time
14             mc ← ∅                      ▷ Reset the constraints
```

Algorithm 2: Finding a valid plan with motion constraints

```
1  procedure SOLVEWITHCONSTRAINTS(ψ, mc)
2      ψ' ← refine-problem(ψ, mc)
3      π ← task-plan(ψ')
4      return π
```

*its **extended trajectory** as a trajectory $\xi_i(t)$ such that:*

$$\xi_i(t) = \begin{cases} \tau_i(0) & \text{if } t < t_i \\ \tau_i(t - t_0) & \text{if } t_i \leq t \leq t_i + d_i \\ \tau_i(d_i) & \text{if } t > t_i + d_i \end{cases}$$

We can define the plan validity as follows.

Definition 7 *A plan π is **valid** if:*

- *π is motion-well-formed and valid from the task-planning perspective.*
- *For each $G \in PMG(\pi)$, each $\pi[i] \in G$ for each $\langle o, q \rangle \in \mathcal{S}_{a_i}$ and for all $t \in \mathbb{R}_{\geq 0}$, $occ(o, q) \cap occ(o_i, \xi_i(t)) = \emptyset$.*
- *For each $G \in PMG(\pi)$, each $\pi[i] \in G$ for each $\langle o, q \rangle \in \mathcal{D}_{a_i}$ such that there is no $\pi[j] \in G$ with $o_j = o$, and for all $t \in \mathbb{R}_{\geq 0}$, $occ(o, q) \cap occ(o_i, \xi_i(t)) = \emptyset$.*
- *For each $G \in PMG(\pi)$, each distinct pair of elements $\pi[i], \pi[j] \in G$ and for all $t \in \mathbb{R}_{\geq 0}$, $occ(o_i, \xi_i(t)) \cap occ(o_j, \xi_j(t)) = \emptyset$.*

The second constraint ensures action trajectories avoid static obstacles, the third extends this to dynamic obstacles outside the same parallel-motion group as $\pi[i]$, treating them as static, and the fourth ensures non-collision between trajectories of actions within the same parallel-motion group.

T-Tampest

To solve the TTAMP problem ψ, we developed T-TAMPEST (Temporal Task and Motion Planning by Encoding into Satisfiability Testing), which interleaves an incremental SMT solver with a sampling-based motion planner. The SMT solver manages temporal constraints such as deadlines and action durations, ensuring that all temporal requirements are satisfied. The motion planner handles simultaneous object motion, ensuring that the spatial aspects of the plan are feasible. This section outlines T-TAMPEST and explains how

Algorithm 3: Checking motion constraints in a given plan

```
1  procedure CHECKMOTIONS(π, mc, cache, timeout)
2      found ← True                       ▷ Final validity of the plan
3      for each G ∈ PMG(π) do
4          Mot ← {mot_{a_i} | π[i] ∈ G}
5          t_min ← min({t_i | π[i] ∈ G})
6          Delay ← {a_i : t_i − t_min | π[i] ∈ G}
7          if not find(cache, G, Delay) then
8              if {π[i]} = G then          ▷ One object moving
9                  u ← upper-bound(a_i)
10             else
11                 u ← None
12             τ, Σ, Ω ← motion-plan(Mot, Delay, u, timeout)
13             if τ then
14                 cache[(G_≺, Delay)] ← τ
15                 Dur ← ∅                 ▷ Action durations in trajectory
16                 Wait ← ∅                ▷ Action waiting times from t_min
17                 for each π[i] ∈ G do
18                     d' ← duration(τ[a_i])
19                     Dur[a_i] ← d'
20                     Wait[a_i] ← waiting-time(τ[a_i])
21                     if d' > d then
22                         found ← False   ▷ Requiring more time
23                 if not found then
24                     mc[(G_≺, Delay)]←(∅, ∅, Dur, Wait)
25             else
26                 found ← False
27                 mc[(G_≺, Delay)] ← (Σ, Ω, ∅, ∅)
28     return found, mc
```

geometric and temporal constraints are integrated to ensure the successful termination of the planning process.

General framework. In Algorithm 1, we show the pseudo-code of our general framework. The procedure SOLVE iteratively attempts to solve the TTAMP problem ψ. It keeps a *cache* of successfully validated motion constraints and their trajectories (line 2), and a database *mc* of new constraints (line 3). The process begins by calling SOLVEWITHCONSTRAINTS (line 5), which refines ψ based on the current constraints in *mc* and invokes the task planner to generate a potential task plan π (see Algorithm 2). Initially, *mc* is empty, meaning that motion constraints are disregarded and ψ is reduced to a traditional task-planning problem. If a valid plan π is found, SOLVE proceeds to its validation by invoking CHECKMOTIONS (line 7). This function validates the plan against both new constraints and those stored in the cache (see Algorithm 3). Since we use a sample-based motion planning algorithm, which may not terminate if no path exists, we set a timeout t_p to each invocation of the motion planner. If all motion constraints for π are found to be realizable by the motion planner, SOLVE returns the plan π along with the cache of validated motion constraints and associated trajectories (line 9). Otherwise, *mc* is updated with the new constraints *mc'* obtained from CHECKMOTIONS (line 11). If SOLVEWITHCONSTRAINTS fails to find a plan π, it could indicate that the problem is unsolvable or that the motion planner could not find a path within the allotted time. Thus, we double the timeout t_p of the motion planner

(line 13), reset our refinements (line 14), and restart the algorithm. We preserve the *cache* to retain valid motion plans and enhance the efficiency of our algorithm.

Constraints generation. The set *mc* of constraints from CHECKMOTION explains why the motion planner failed to find feasible trajectories for the motion actions in π. It fails either because no collision-free path exists or the path's execution time exceeds the duration set by the task planner. Thus, an element of *mc* has the form (Σ, Ω, *Dur*, *Wait*), where Σ lists obstacles blocking actions a with $mot_a \neq \bot$ in π, and Ω contains the corresponding unreachable configurations. Essentially, we collect the same information as (Tosello, Valentini, and Micheli 2024) but split it by each parallel movable object. For an action a with $mot_a \neq \bot$, *Dur* contains the duration $duration(\tau[a])$ required to execute the trajectory $\tau[a]$, that we assume is the lower bound for the duration of this action in the parallel group it is in. *Wait* collects the time the object involved in the motion $\tau[a]$ has to wait before starting to move (see Figure 1).

In Algorithm 3, CHECKMOTIONS finds the set PMG(π) of parallel-motion-groups in π and iterates over each group *G*. Snap actions in *G* are defined as

$$snaps(G) = \{\langle a_i, start, t_i\rangle \mid \pi[i] \in G\} \cup \{\langle a_i, end, t_i + d_i\rangle \mid \pi[i] \in G\}$$

where each snap action has the form $\langle a_i, x, t\rangle$, with a_i the action, $x \in \{start, end\}$ its start or end, and $t \in \{t_i, t_i + d_i\}$ the timing. *G* is ordered, with its order defined as

$$G_{\preceq} = \langle \langle a, x\rangle \mid \langle a, x, t\rangle \in snaps(G)\rangle$$

such that if $\langle a_i, x\rangle$ precedes $\langle a_j, y\rangle$, then $\langle a_i, x, t_i\rangle$, $\langle a_j, y, t_j\rangle \in snaps(G)$ and $t_i \leq t_j$.

For each group, CHECKMOTIONS extracts the motion constraints imposed by the problem (*Mot*), determines the start time of the earliest action (t_{min}), and calculates the delays (*Delay*) for all actions relative to this reference time (lines 4-6). It then checks if a trajectory for *G* and its delays is already cached. If no trajectory is found and only one object is moving, the algorithm extracts the upper time bound *u* of the action (line 8), limiting the search of the motion planner to areas reachable within this time. For multiple objects moving, *u* is ignored. Indeed, a single object's motion might exceed the action's upper bound not because no trajectory fits within it, but because it is infeasible given the context of search: temporal constraints are dependent on the order $G_{\preceq}$ of parallel actions within each group *G* and the set of delays *Delay* associated with each action relative to the start time of the sequence. We then have to investigate the cause of infeasibility based on this context.

Given the motion constraints (*Mot*), delays (*Delay*), and optionally the upper bound u, the system invokes the motion planner to generate a feasible motion plan τ within the assigned $timeout$ (line 12). If τ exists, it is cached, the algorithm verifies the duration of all motions involved (*Dur*) (line 18), and computes the waiting times (*Wait*) relative to the start of the first motion (line 20). If any motion duration exceeds what was planned by the task planner, the plan is invalid, indicating that the action requires more time than allocated. This invalidity, along with the identified timing issues (*Dur* and *Wait*) is stored in mc, aiding the task planner pruning entire temporal regions of the search space (line 24).

If τ is not found, π is invalid. The information from the motion planner about blocking obstacles Σ and unreachable positions Ω (line 12) is stored in mc and used to refine the task problem by pruning all symbolically defined geometric regions identified as unreachable (line 27). As shown in Algorithm 3, such geometric constraints are globally defined and derived from analyzing each action in isolation, independently of action order, overlap, or duration.

Constrained solving. Once the new *mc* is computed, we re-solve the problem taking into consideration this new information. As mentioned earlier, we distinguish between single-object and multi-object movement scenarios.

In the single-object scenario, only one motion action is executed at a time, and the new constraints are limited to unreachable targets and occluding obstacles, or updated lower bounds for the action duration. We can apply the approach outlined in Algorithm 2: SOLVEWITHCONSTRAINTS refines ψ based on the newly identified constraints stored in *mc* and then invokes the task planner to search for a new plan π. While geometric constraints learning follows (Tosello, Valentini, and Micheli 2024), temporal learning focuses on refining action durations based on the feedback of the motion planner. This refinement is planner-independent and can be applied to any planner utilizing our framework.

In the multi-object case, geometric explanations are handled as before, but temporal constraints depend on the order $G_{\preceq}$ of events and their delays *Delay*. Off-the-shelf temporal planners cannot handle this type of constraints, so we resort to a direct encoding on top of a standard SMT-based planning approach (Panjkovic and Micheli 2023). At each step k, the encoding is augmented by the following formulae.

Given an order of events $G_{\preceq} = \langle e_1, \ldots, e_n\rangle$, we define the start sub-sequence $G^{\vdash}_{\preceq}$ as the list of events in $G_{\preceq}$ that are action starts. We indicate the i-th element of $G^{\vdash}_{\preceq}$ as $G^{\vdash}_{\preceq}[i]$ and the i-th element of $G_{\preceq}$ as $G_{\preceq}[i]$. Slightly changing the notation of (Panjkovic and Micheli 2023), we indicate with $a@i$ the boolean variable deciding if action a is started in step i, $t@i$ indicate the absolute time of step i, and $d_a@j_i$ indicates the duration of action a started at step i.

For each $(G_{\preceq}, Delay) \rightarrow (\emptyset, \emptyset, Dur, Wait) \in mc$, we generate a constraint of the form:

$$\bigwedge_{j_1=1}^{k-|G^{\vdash}_{\preceq}|} (G^{\vdash}_{\preceq}[1]@j_1 \rightarrow \bigwedge_{j_2=j_1+1}^{k-|G^{\vdash}_{\preceq}|+1} (G^{\vdash}_{\preceq}[2]@j_2 \rightarrow \cdots \rightarrow (\bigwedge_{i=2}^{|G_{\preceq}|} (time(G_{\preceq}, i) \geq time(G_{\preceq}, i-1))) \rightarrow \text{LEARNED}) \ldots))$$

where

$$\text{LEARNED} = \bigwedge_{a=G^{\vdash}_{\preceq}[i] \in G^{\vdash}_{\preceq}} ((d_a@j_i \geq Dur[a]) \vee (t@j_i - t@j_1 \geq Delay[a] + Wait[a] + \delta) \vee (t@j_i - t@j_1 \leq Delay[a] - \epsilon))$$

with:

$$time(G_{\preceq}, i) = \begin{cases} t@j_v & \text{if } G_{\preceq}[i] = G^{\vdash}_{\preceq}[v] \\ t@j_v + d_a@j_v & \text{if } G_{\preceq}[i] \text{ is the end of } a = G^{\vdash}_{\preceq}[v] \end{cases}$$

Given the first start $G^{\vdash}_{\preceq}[1]$ at time j_1, the second start $G^{\vdash}_{\preceq}[2]$ at time j_2, and all other start events at their timing, if the task planner wants to use the order $G_{\preceq}$, it should either extend the action durations to the learned values in *Dur[a]*, postpone the starting of the actions to at least $Delay[a] + Wait[a]$ (making it possible to satisfy the duration constraints, in principle), or anticipate the start of the actions. In this formulation, we use two constants δ and ϵ to ensure the progression of the algorithm: it is otherwise possible to find infinitely many candidate plans with the same order differing by smaller and smaller amounts of time. These constraints added to the SMT encoding of (Panjkovic and Micheli 2023), implement a bounded planner capable of exploiting what learned from CHECKMOTIONS to avoid candidate plans that are guaranteed to be infeasible (assuming the motion planner had enough time to find a motion plan).

Theoretical guarantees. Our algorithm is sound: the candidate plans are valid by construction for the task planner, and trajectories found by the motion planner are also valid by construction, adhering to all timing constraints. We believe that the approach is incomplete: temporal planning is undecidable (Gigante et al. 2022), and combining it with motion planning adds further complexity. A thorough investigation of the completeness of our approach is left for future work.

Modeling and Benchmarking

Besides formulating the TTAMP problem of Definition 1 and developing a suitable solver, we extended the modeling tool from (Tosello, Valentini, and Micheli 2024) to support this class of problems and created a set of benchmarks that includes simultaneous durative motion actions. Below is an overview of our implementation and benchmark suite.

Following (Tosello, Valentini, and Micheli 2024), a set of *Movable Objects* adopt specific *Configuration Objects* (e.g., poses) within an *Occupancy Map*. We support fluents that map a *Movable Object* to its *Configuration Object* within the *Occupancy Map*. Besides *Instantaneous Motion Action*s, the *Durative Motion Action* is a durative action from temporal planning enhanced with motion constraints that apply throughout its duration. A motion constraint of an action a has the form $path(o_a, q_a^S, [q_a^G], \{o : q \; \forall (o,q) \in \mathcal{S}_a \cup \mathcal{D}_a\})$, with $\{o \mid (o,q) \in \mathcal{S}_a \wedge (o,q') \in \mathcal{D}_a\} = \emptyset$, as per Definition 1. It constraints the path of the object o_a between q_a^S and $[q_a^G]$ to avoid obstacles in $\mathcal{S}_a$ and $\mathcal{D}_a$. Objects in $\mathcal{S}_a$ are static throughout a, while the pose of those in $\mathcal{D}_a$ is known at the start but changes during a.

We propose five TTAMP benchmarks, based on criteria from (Lagriffoul et al. 2018) and inspired by the temporal domains of the International Planning Competition and (Micheli and Valentini 2021). They task robotic agents with navigating through 2D black-and-white maps, with black areas denoting fixed obstacles. They must avoid or remove moving or movable obstacles under metric time. Agents are modeled as 2D polygons with a Reeds-Shepp motion model and a car-like control model (v, θ), with $v \in \{v_{min}, v_{max}\}$ the vehicle's velocity and $\theta \in \{\theta_{min}, \theta_{max}\}$ its steering angle. Further descriptions follow.

- **Timed doors**. A fleet of *n* robots must navigate through *m* closed doors to reach their destinations. Robots can move between locations, avoiding collisions with fixed and moving obstacles (walls, doors, and other robots), with durations based on robot speed and the euclidean distance to be traveled. They can open doors if in front of them by instantly changing their configurations from closed to open (as if pushing buttons). *x* random locations in the free space expand the task space.
- **Driverlog**. A set of *n* trucks delivers *m* packages to *m* destinations, initially secured by closed doors. Trucks can move if the door ahead is open and a driver is aboard. The *x* drivers can drive trucks, walk between locations, and handle loading and unloading packages. To open a door, a truck must be in front of it. All actions have durations, and driving includes motion constraints to avoid collisions with doors, walls, drivers, and other trucks.
- **Kitting**. A group of *n* robots assembles *x* kits by moving, loading, and unloading *m* materials. The *x* locations storing materials are initially secured by doors that robots must open to access. Actions have durations, with motion actions constrained to avoid collisions. Robots, powered by batteries, can only act if they have sufficient energy, with actions consuming battery power.
- **MAJSP**. This job-shop scheduling problem involves *n* robots transporting *x* products between *m* machines for treatment. Processed products are placed on pallets for collection. Locations with machines are initially secured by doors. Robots navigate these locations, open doors, load and unload items, and transport products for treatment. Actions have durations, motion constraints to avoid obstacles, and each robot has a limited battery.
- **Floortile**. A fleet of *n* robots uses *c* colors to paint *x* floor tiles, *m* of them occupied by movable objects. Robots and objects can move in four directions (up, down, left, right), with motion constraints to avoid collisions with each other. Robots paint one tile at a time using one color, changing spray guns when needed, and can only paint tiles directly in front (up) or behind (down). Moving, painting, and changing color takes time.

Experimental Evaluation

In this section, we evaluate the effectiveness of T-TAMPEST.

As temporal refinements are essential for soundness, we examine how geometric refinements affect performance and coverage by varying CHECKMOTION explanations. *All-Refinements* uses the full algorithm, sending back to the Task Planner information on both unreachable locations and blocking obstacles. *Only-Reachables* sets $\Omega = \{\mathcal{S}_a \mid a \in \pi\}$, giving information only on unreachable locations. *Only-Obstacles* sets $\Sigma = \{q_a^G \mid a \in \pi\}$, returning only blocking obstacles, marking the target as unreachable, and ignoring the state of all other configurations. *No-Refinements* sets $\Sigma = \{q_a^G \mid a \in \pi\}$ and $\Omega = \{\mathcal{S}_a \mid a \in \pi\}$, removing only the violated constraints (i.e., the blocked location). Benchmarks and solvers are available open source (check **Code**).

Planner	**Timed Doors** (tot. 48)		**Driverlog** (tot. 36)		**Kitting** (tot. 108)		**MAJSP** (tot. 36)		**Floortile** (tot. 180)	
	#	$\bar{t}_{run}$ (% t_p)	#	$\bar{t}_{run}$ (% t_p)	#	$\bar{t}_{run}$ (% t_p)	#	$\bar{t}_{run}$ (% t_p)	#	$\bar{t}_{run}$ (% t_p)
T-TAMPEST(*None*)	29	351.09 s (59%)	**12**	**464.75** s (44.93%)	33	516.86 s (38.97%)	14	615.79 s (76.16%)	**64**	**623.23** s (10.93%)
T-TAMPEST(*Only-Reachable*)	31	136.07 s (85.16%)	9	114.64 s (65.06%)	22	894.87 s (31.60%)	15	379.33 s (68.98%)	63	705.84 s (9.06%)
T-TAMPEST(*Only-Obstacles*)	27	299.41 s (88.88%)	11	312.10 s (46.91%)	**35**	**481.73** s (37.92%)	13	560.83 s (68.25%)	60	565.67 s (9.24%)
T-TAMPEST(*All*)	**32**	**134.98** s (84.83%)	10	122.10 s (65.23%)	26	728.07 s (27.80%)	**15**	**352.54** s (63.51%)	53	500.70 s (11.10%)

Table 1: Overall performance of our solver across four configurations: *None* (no geometric refinement), *Only-Reachable* (avoids unreachable targets), *Only Obstacles* (considers only obstacles that occlude targets), and *Full* (the combination of the two). For each domain, we report the number of instances solved (#), average running time on solved instances ($\bar{t}_{run}$), and the percentage of running time dedicated to the motion planner (% t_p).

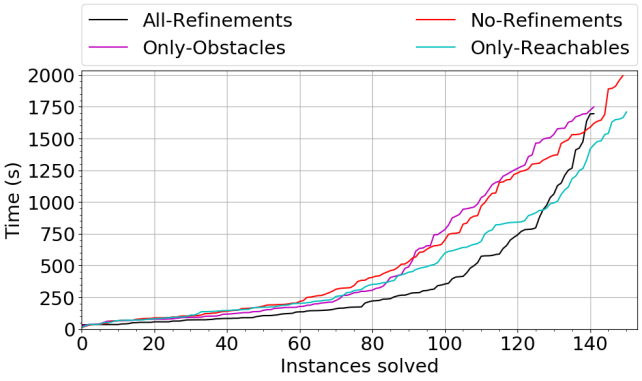

Figure 2: Overall performance across all benchmark instances and planners using different topological refinements.

Our test cases follows.

- **Timed doors**. We consider $n \in [1, 2, 3]$ robots and $m \in [1, 2, 3, 4]$ blocking doors. There are either 0 (x = [(0, 0)]) or 10 extra configurations, distributed as either fully reachable (c = [(10,0)]), fully unreachable (c = [(0, 10)]), or split equally between both (n_c = [(5,5)]).
- **Driverlog**. We evaluate $n \in [1, 2, 3]$ trucks with $m \in [1, 2, 3, 4]$ packages, each located in an area closed by a door, and x = [1, 2, 3] human drivers available.
- **Kitting**. We examine $n \in [1, 2, 3]$ robots assembling $x \in [1, 2, 3]$ kits, with $m \in [1, 2, 3, 4]$ materials to be subdivided and distributed across m locations closed by doors. Robots start with a fully charged battery (100%), with each action consuming 1%.
- **MAJSP**. We analyze $n \in [1, 2, 3]$ robots, $x \in [1, 2, 3]$ pallets, and $m \in [1, 2, 3, 4]$ treatments, located in areas closed by doors. The battery is defined as above.
- **Floortile**. We have $n \in [1, 2, 3]$ robots, $x \in [1, 2, 4, 6, 8]$ tiles to be colored in a 4x4 floor, $m \in [1, 2, 3, 4]$ tiles occupied by obstacles, and $c \in [1, 2, 3]$ available colors.

We set the incremental horizon of our SMT planner to $h_{max} = 100$, use Space-Time Rapidly-exploring Random Tree Star (ST-RRT*) as the motion planner (Grothe et al. 2022), and implement a collision checker to verify if the 2D robot's footprint intersects obstacles. ST-RRT* integrates spatial and temporal aspects, enabling planning in unbounded time spaces through incremental time-bound extensions. It is also optimized for space-time planning, with features like conditional sampling and simplified rewiring.

Results. Figure 2 shows the impact of using topological refinements across all instances and domains, with the x-axis indicating solved instances and the y-axis showing computation time. All approaches solve 140-150 instances, with *No-Refinements* and *Only-Reachables* performing ∼10% better than *All-Refinements* and *Only-Obstacles*. However, *All-Refinements* reduces overall running time by ∼20%. Table 1 supports these findings: while the number of solved instances is similar across approaches, in motion-critical domains like *Timed Doors* ($t_p > 80\%$, with t_p the motion planning time) and *MAJSP* ($t_p > 60\%$), incorporating explanations for unreachable goals and blocking obstacles significantly improves performance, halving the total running time. For *Timed Doors*, T-TAMPEST without refinements (*None*) achieves an overall running time $\bar{t}_{run} = 351.09$ s for $\# = 29$ instances solved. With all refinements enabled (*All*), it improves to $\bar{t}_{run} = 134.98$ s, solving $\# = 32$ instances. For *MAJSP*, T-TAMPEST(*None*) achieves $\bar{t}run = 615.79$ s for $\# = 14$ instances, while T-TAMPEST(*All*) improves to $\bar{t}run = 352.54$ s for $\# = 15$ instances. In domains like *Kitting* and *Floortile*, which involve many non-geometric actions such as color selection or directional painting, our solver excels on simpler instances but struggles to scale due to the SMT component's limitations. As plan size grows, the increasing parameter complexity slows resolution. Despite this, our analysis shows that as the very first TTAMP approach, T-TAMPEST already solves a significant number of instances in complex scenarios.

Conclusion and Future Work

This paper addressed the challenge of Temporal Task and Motion Planning (TTAMP) for simultaneous multi-object navigation with metric time. We formalized the problem and extended an existing modeling tool to handle metric time and concurrent motion. We proposed T-TAMPEST, a novel TTAMP planner that interleaves incremental SMT with sampling-based motion planning to generate plans, refine them based on time constraints, and prune unreachable regions. Our results demonstrate the effectiveness of T-TAMPEST and show that our geometric refinements reduce running time, particularly in motion-critical domains.

Future work includes enhancing our solver with optimization techniques from (Panjkovic and Micheli 2023), extending formalization to manipulation actions, integrating replanning for non-determinism, and testing in simulated and real-world robot scenarios.

Acknowledgments

This work has been partially supported by the AI4Work project funded by the 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

Barrett, C. W.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C. 2009. Satisfiability Modulo Theories. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, 825–885. IOS Press. ISBN 978-1-58603-929-5.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2021. Forward-Chaining Partial-Order Planning. *Proceedings of the International Conference on Automated Planning and Scheduling*, 20(1): 42–49.

Dantam, N. T. 2020. *Task and Motion Planning*, 1–9. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-41610-1.

DeCastro, J. A.; and Kress-Gazit, H. 2013. Guaranteeing reactive high-level behaviors for robots with complex dynamics. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 749–756.

Edelkamp, S.; Lahijanian, M.; Magazzeni, D.; and Plaku, E. 2018. Integrating Temporal Reasoning and Sampling-Based Motion Planning for Multigoal Problems With Dynamics and Time Windows. *IEEE Robotics and Automation Letters*, 3(4): 3473–3480.

Faroni, M.; Umbrico, A.; Beschi, M.; Orlandini, A.; Cesta, A.; and Pedrocchi, N. 2024. Optimal Task and Motion Planning and Execution for Multiagent Systems in Dynamic Environments. *IEEE Transactions on Cybernetics*, 54(6): 3366–3377.

Fox, M.; and Long, D. 2003. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*, 20: 61–124.

Fox, M.; and Long, D. 2011. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *CoRR*, abs/1106.4561.

Garrett, C. R.; Chitnis, R.; Holladay, R.; Kim, B.; Silver, T.; Kaelbling, L. P.; and Lozano-Pérez, T. 2021. Integrated Task and Motion Planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4(1): 265–293.

Garrett, C. R.; Lozano-Perez, T.; and Kaelbling, L. P. 2020. PDDLStream: Integrating Symbolic Planners and Blackbox Samplers. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Garrett, C. R.; Lozano-Pérez, T.; and Kaelbling, L. P. 2018. FFRob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*, 37(1): 104–136.

Gigante, N.; Micheli, A.; Montanari, A.; and Scala, E. 2022. Decidability and complexity of action-based temporal planning over dense time. *Artificial Intelligence*.

Grothe, F.; Hartmann, V. N.; Orthey, A.; and Toussaint, M. 2022. ST-RRT*: Asymptotically-Optimal Bidirectional Motion Planning through Space-Time. In *2022 International Conference on Robotics and Automation (ICRA)*, 3314–3320.

Ho, Q. H.; Ilyes, R. B.; Sunberg, Z. N.; and Lahijanian, M. 2022. Automaton-Guided Control Synthesis for Signal Temporal Logic Specifications. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, 3243–3249.

Kress-Gazit, H.; Fainekos, G. E.; and Pappas, G. J. 2009. Temporal-Logic-Based Reactive Mission and Motion Planning. *IEEE Transactions on Robotics*, 25(6): 1370–1381.

Lagriffoul, F.; Dantam, N. T.; Garrett, C.; Akbari, A.; Srivastava, S.; and Kavraki, L. E. 2018. Platform-Independent Benchmarks for Task and Motion Planning. *IEEE Robotics and Automation Letters*, 3(4): 3765–3772.

Lahijanian, M.; Maly, M. R.; Fried, D.; Kavraki, L. E.; Kress-Gazit, H.; and Vardi, M. Y. 2016. Iterative Temporal Planning in Uncertain Environments With Partial Satisfaction Guarantees. *IEEE Transactions on Robotics*, 32(3): 583–599.

Lindemann, L.; and Dimarogonas, D. V. 2017. Robust Motion Planning employing Signal Temporal Logic. *CoRR*, abs/1703.02075.

Maler, O.; and Nickovic, D. 2004. Monitoring Temporal Properties of Continuous Signals. In Lakhnech, Y.; and Yovine, S., eds., *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, 152–166. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-30206-3.

McMahon, J.; and Plaku, E. 2014. Sampling-based tree search with discrete abstractions for motion planning with dynamics and temporal logic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3726–3733.

Micheli, A.; and Valentini, A. 2021. Synthesis of Search Heuristics for Temporal Planning via Reinforcement Learning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI*, 11895–11902. AAAI Press.

Muhayyuddin; Akbari, A.; and Rosell, J. 2017. Physics-based Motion Planning with Temporal Logic Specifications. *CoRR*, abs/1710.00419.

Panjkovic, S.; and Micheli, A. 2023. Expressive Optimal Temporal Planning via Optimization Modulo Theory. In *AAAI*.

Raman, V.; Donzé, A.; Maasoumy, M.; Murray, R. M.; Sangiovanni-Vincentelli, A.; and Seshia, S. A. 2014. Model predictive control with signal temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, 81–87.

Saha, S.; and Julius, A. A. 2018. Task and Motion Planning for Manipulator Arms With Metric Temporal Logic Specifications. *IEEE Robotics and Automation Letters*, 3(1): 379–386.

Shin, J.-A.; and Davis, E. 2005. Processes and continuous change in a SAT-based planner. *Artificial Intelligence*.

Srivastava, S.; Fang, E.; Riano, L.; Chitnis, R.; Russell, S.; and Abbeel, P. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)*, 639–646. IEEE.

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

Takano, R.; Oyama, H.; and Yamakita, M. 2021. Continuous Optimization-Based Task and Motion Planning with Signal Temporal Logic Specifications for Sequential Manipulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 8409–8415.

Tosello, E.; Valentini, A.; and Micheli, A. 2024. A Meta-Engine Framework for Interleaved Task and Motion Planning using Topological Refinements. arXiv:2408.05795.

Toussaint, M. 2015. 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, 1930–1936. AAAI Press. ISBN 9781577357384.

Valentini, A.; Micheli, A.; and Cimatti, A. 2020. Temporal Planning with Intermediate Conditions and Effects. In *AAAI*.