2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

A Hierarchical Decoupling Approach for Fast Temporal Logic Motion Planning

Ziyang Chen, Zhangli Zhou, Shaochen Wang, and Zhen Kan

***Abstract*— Fast motion planning is of great significance, especially when a timely mission is desired. However, the complexity of motion planning can grow drastically with the increase of environment details and mission complexity. This challenge can be further exacerbated if the tasks are coupled with the desired locations in the environment. To address these issues, this work aims at *fast* motion planning problems with temporal logical specifications. In particular, we develop a hierarchical decoupling framework that consists of three layers: the high-level task planner, the decoupling layer, and the low-level motion planner. The decoupling layer is designed to bridge the high and low layers by providing necessary information exchange. Such a framework enables the decoupling of the task planner and path planner, so that they can run independently, which significantly reduces the search space and enables fast planing in continuous or high-dimension discrete workspaces. In addition, the implicit constraint during task-level planning is taken into account, so that the low-level path planning is guaranteed to satisfy the mission requirements. Numerical simulations demonstrate at least one order of magnitude speed up in terms of computational time over existing methods.**

I. Introduction

The effectiveness (i.e., the satisfaction of complex mission requirements) and efficiency (i.e., fast and timely planning) of motion planning algorithms are critical for mobile robots to perform various applications. To enable these applications, prior works mainly rely on the abstracted environment and robot dynamics. However, the algorithm complexity can grow drastically with the increase of the environment dimension and mission complexity. This challenge can be further exacerbated if the tasks are coupled with the locations in the environment. For instance, consider a search and rescue mission as shown in Fig. 1(a), which requires the robot to equip with search devices by first visiting the tool store and then search for survivors in the damaged areas 1 and 2. Such a mission can be represented by an LTL formula $\Phi = \neg(ap_2 \vee ap_3)U(ap_1 \wedge \neg ap_2 \wedge \neg ap_3)$, where ap_1, ap_2, ap_3 represent the actions of visiting the tool store to load search devices, searching over the damaged areas 1 and 2, respectively. Note that Φ requires the robot not to visit the damaged areas before visiting the tool store.

A comment approach to solving the above motion planning problem is to build a product automaton based on the automaton generated by Φ and the abstracted environment and robot dynamics [1], [2]. A satisfying motion plan can then be found by searching over the product automaton [3], [4] as in Fig. 1(b) or by sampling based methods [5], [6] as in Fig. 1(c). However, with the increase of environment dimension and LTL task complexity, the product automaton can grow drastically, leading to high computational cost. To mitigate this issue, an alternative is to leverage hierarchical approaches [7], in which the task and motion are planned separately, so that complex LTL tasks and high dimensional environments can be efficiently handled. However, a potential issue in most existing hierarchical approaches [8]–[11] is that the task-level planner might overlook the constraints that are implicitly coupled with the motions. For instance, to perform Φ, the task-level planner decides that the tool store needs to be visited, regardless of the detailed path planning, as shown in Fig. 1(d). Motion planners then deal with the low-level path planning towards the tool store in the environment by using off-the-shelf tools, e.g., sampling-based approaches, potential field based approaches, etc, as shown in Fig. 1(e). Apparently, it violates the LTL Φ, as the constraint of not visiting ap_2 and ap_3 before ap_1 is ignored when the task-level planner informs the goals to the motion-level planner.

Motivated by the discussion above, this work aims at *fast* motion planning problems with temporal logical specifications. We are interested in combining the goods of the aforementioned two approaches, i.e., the guarantee of mission satisfaction in automaton-based approaches and the efficiency in handling complex tasks and high dimensional environments in hierarchical approaches. Specifically, we present a hierarchical decoupling framework that consists of three layers: the high-level task planner, the decoupling layer, and the low-level motion planner. The decoupling layer is designed to bridge the high and low layers by providing necessary information exchange. Taking advantage of the hierarchical structure, the task planner and path planner are decoupled and run independently, which significantly reduces the search space and enables fast planing in continuous or high dimensional workspace. Rather than relying on the product automaton, our method directly operates on the nondeterministic Büchi automaton (NBA), and can infer the temporal order of sub-tasks from the automaton, which enables parallel execution of sub-tasks. Compared with most hierarchical approaches, our framework ensures mission completion by taking into account the implicit constraint during task-level planning, so that low-level path planning is guaranteed to satisfy the mission requirements. Numerical simulations demonstrate at least one order of magnitude speed up in terms of computational time over existing methods such as the abstraction-free algorithms.

Z. Chen, Z. Zhou, S. Wang, and Z. Kan (Corresponding Author) are with the Department of Automation at the University of Science and Technology of China, Hefei, Anhui, China, 230026.

This work was supported in part by the National Natural Science Foundation of China under Grant U2013601 and Grant 62173314.

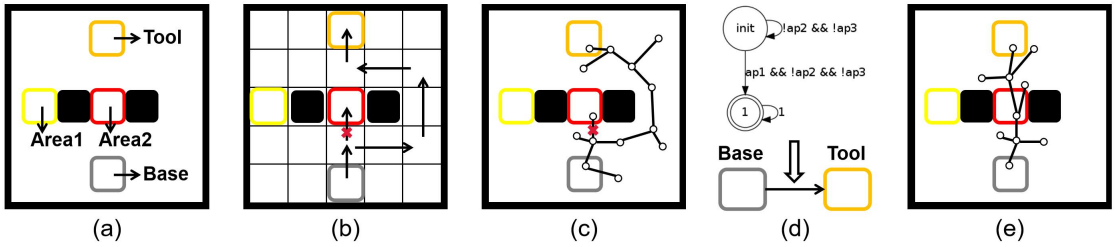


Fig. 1. (a) The environment with base station, tool store, damaged houses (i.e, area 1 and 2), and unaccessible areas (black blocks). The robot starts from the base and is tasked with the mission Φ. (b) The planned path using the product automaton based approach when the environment is grid. (c) When considering a continuous environment, the product automaton based approach is integrated with sampling methods to plan the path of the robot. (d) The generated Büchi automaton when using hierarchical approaches. The task-level planner indicates the robot should move from base to the tool store. (e) The realized low-level path using sampling methods.

II. Related Works

Due to the rich expressivity, linear temporal logic (LTL) has been increasingly used to describe complex robotic tasks [1]–[3], [12]–[16]. In these works, the motion planning depends on graph search techniques over the product automaton. Since the product automaton can grow exponentially large with the increase of the LTL task and the abstracted environment and transition systems, the planning can be time consuming and even intractable. To reduce the computational complexity, sampling-based planning algorithms were developed to trade strong completeness guarantees for efficient and scalable path planning. In [17] and [18], rapidly-exploring random trees (RRT) and rapidly-exploring random graph (RRG) were exploited, respectively, for the motion planning with temporal goals. To avoid the construction of the product automaton, efficient sampling-based algorithms were developed in [5] for discrete transitions systems and global temporal logic tasks. Common in the aforementioned approaches is that a discrete abstraction of the environment and the transition system is required, which is computationally expensive to construct. The work of [6] presents an abstraction-free method that incrementally builds trees for efficient motion planning. Other related works include [19] and [20]. However, the aforementioned methods still need to reduce the sampling step size and sample many points in uninteresting areas.

Hierarchical task and motion planning has also been investigated in the literature. In [8], the hierarchy is exploited for the manipulation planning and the atomic propositions are defined in the object space. In [17], a geometry-based multi-layered synergistic approach was developed for motion planning problems. In [9], an extensible planner-independent interface layer was developed to facilitate the combination of high-level task planning and low-level motion planning. In [10], the task and path are planned independently for multiple robots to operate reactively in an unknown environment. In [11], long-term temporal logic goals with short-term reactive requirements were considered. In these works the tasks are coupled with the underlying environment; however, the low-level path planning in [9]–[11], [17] can overlook such constraints, resulting in that the realized path might violate the specified missions. As an exception, the modified automaton is adopted in [21] to avoid the violation of the task-level constraints. In [22], the task is defined in the object space and the map, which can still potentially violate the mission requirements. Therefore, without taking into account the coupled constraint of task and locations, the mission completion cannot be fully guaranteed.

III. Preliminaries

Linear temporal logic is employed in this work to specify the task specifications. The syntax of an LTL formula Φ is defined over a set of atomic propositions AP as $\Phi := true|ap|\neg| \wedge |X|U$, where $true$ is the Boolean value, $ap \in AP$ is an atomic proposition, $\neg$ (negation) and $\wedge$ (conjunction) are standard Boolean operators, X (next) and U (until) are temporal operators. Let $\Theta = 2^{AP}$ denote the alphabet, where 2^{AP} represents the power set of AP. The semantics of an LTL formula are defined over an infinite sequence $\boldsymbol{\pi} = \pi_0\pi_1 \ldots$ with $\pi_i \in 2^{AP}$ for all $i \geq 0$. Denote by $\boldsymbol{\pi} \models \Phi$ if the word $\boldsymbol{\pi}$ satisfies Φ. Detailed descriptions of the syntax and semantics of LTL can be found in [23].

An LTL formula can be translated to a nondeterministic Büchi automaton (NBA) using existing tools, such as LTL2STAR [24]. The NBA is defined as $B = \{S, s_0, \Theta, \Delta, S_F\}$, where S is a finite set of states, $s_0 \in S$ is the initial state , $\Delta : S \times S \to 2^{\Theta}$ is the set of alphabets that enables the state transition (s, s') , $s, s' \in S$, and $S_F \subseteq S$ is the set of accepting states. We denote by $s \xrightarrow{\pi} s'$ if $\pi \in \Delta(s, s')$. Given an input sequence $\boldsymbol{\pi} = \pi_0\pi_1 \ldots$, the generated trajectory from s_0 over B is $\boldsymbol{s} = s_0 s_1 \ldots$ with $s_i \xrightarrow{\pi_{i+1}} s_{i+1}$ for all $i \geq 0$. If π can generate at least one run $\boldsymbol{s}$ that intersects the accepting states S_F infinitely many times, π is called an accepting run of B. In general, an accepting run can be written in the prefix–suffix structure, where the prefix part starting from an initial state and ending at an accepting state is traversed only once and the suffix part, a cyclic path of accepting states, is traversed infinitely often. Throughout this work, let B_Φ denote the NBA generated by the LTL formula Φ.

IV. Problem Formulation

A. Environment

Consider a continuous and bounded workspace M containing N disconnected labeled areas $D_1, D_2..., D_N$. The rest

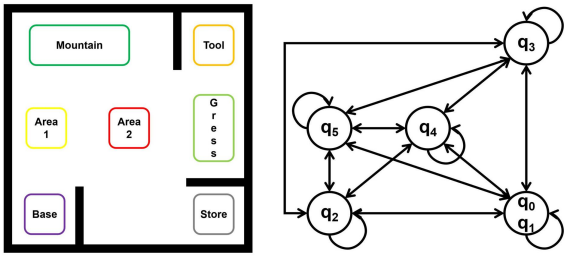


Fig. 2. (a) The workspace of rescue robot. The Store, Base, Tool, House 1 and 2 are labeled areas (i.e., $D_1 \dots D_5$ in M), the thick black lines indicate obstacles, and the rest white areas are the free-motion area (i.e., D_0). It is assumed that Mountain and Grass are not accessible to the robot. (b) The abstract map T corresponding to the workspace M, where q_i corresponds to D_i in M. To reduce the planning complexity, only labeled areas are considered in the abstract map. The mountain and grass will be taken care in the low-level motion planning.

area in M, denoted by D_0, is referred to the free-motion region that robot can move freely. Let $p = (x, y) \in \mathbb{R}^2$ be a position in M and $LA : M \to AP$ be a function that maps p to an atomic proposition in AP. For instance, $LA(p) = ap_j$, $\forall p \in D_i$, indicates that $ap_j \in AP$ can be executed in $D_i \in M$. To facilitate fast planning in M, we construct an abstract map T.

Definition 1. The abstract map is defined as $T = \{Q, q_0, \Sigma, AP, L, LM\}$, where Q is a finite set of abstract map states, $q_0 \in Q$ is the initial state, $\Sigma : Q \times Q \to \mathbb{R}_{\geq 0}$ maps the state transition to a non-negative cost, $L : Q \to AP$ is a labeling function that indicates the atomic proposition associated with the state in Q, and $LM : Q \to M$ is a labeling function that maps $q \in Q$ to a position $p \in M$.

By Def. 1, an abstract map T can be constructed as a directed graph containing nodes representing the labeled areas and the cost Σ representing the edge weight. In this work, the cost of transition (q_i, q_j) is defined as the inter-distance from region $LM(q_i)$ to $LM(q_j)$, i.e., $\Sigma(q_i, q_j) = \|LM(q_i) - LM(q_j)\|$.

Example 1. Consider a search and rescue mission as shown in Fig. 2. Let $AP = \{ap_1, ap_2, ap_3, ap_4, ap_5\}$ be a set of atomic tasks, where ap_1, ap_2, and ap_3 represent the mission of visiting Store (e.g., get supplies), Base (e.g., task assignment), and Tool (e.g., equip with search device), respectively. And ap_4 and ap_5 represent the mission of searching for survivors in House 1 and 2, respectively. Since Mountain and Grass are not involved in the task, they are not considered when constructing the abstract map T, but they will be considered as obstacles in the low-level planning. Therefore, the search and rescue mission of the robot can be described as an LTL formula $\Phi = ((\neg(ap_2))Uap_1) \wedge (Fap_2) \wedge (\neg(ap_4 \vee ap_5)U(ap_3 \wedge \neg ap_4 \wedge \neg ap_5)) \wedge GFap_4 \wedge GFap_5$, which requires the robot to visit Store, Base, and Tool sequentially and then keep searching for survivors in House 1 and 2. During the mission operation, the robot is required to avoid Mountain, Grass and other obstacles.

B. Planning Model

Given the workspace M and the Büchi automaton B_Φ, the robot task plan is defined as a tuple $\Pi = (\boldsymbol{s}, \boldsymbol{\varepsilon}, \boldsymbol{\pi})$, where $\boldsymbol{s} = s_0 s_1 \dots$ is the trajectory of automaton states, $\boldsymbol{\varepsilon} = \varepsilon_0 \varepsilon_1 \varepsilon_2 \dots$ is the sequence of positions in M with ε_0 indicating the empty position, $\boldsymbol{\pi} = \pi_0 \pi_1 \pi_2 \dots$ is sequence of atomic propositions with π_0 indicating $true$. Specifically, for $i \geq 0$, s_{i+1} is the automaton state after π_{i+1} is applied to s_i and $\varepsilon_i \in M$ is the position in the workspace where π_i is executed. By defining $\Pi_i = (s_i, \varepsilon_i, \pi_i)$, the task plan can be rewritten as $\Pi = \Pi_0 \Pi_1 \Pi_2, \dots$, which consists of a series of planning tuples that satisfy $\pi_{i+1} \in \Delta(s_i, s_{i+1})$ and $\pi_i = LA(\varepsilon_i)$ for $i \geq 0$. If Π satisfies the LTL formula Φ in the workspace M, it is denoted as $\Pi \models (\Phi, M)$. Given the task plan Π, Let $P = p_0 p_1 p_2 \dots$ be the generated path of the robot in M, where p_0 is the robot initial position. Apparently, P has to be consistent with $\boldsymbol{\varepsilon}$ and realizes the transitions from ε_i to ε_{i+1}.

The satisfying trajectory of an LTL generally has the structure of plan prefix and plan suffix [23]. However, when the plan prefix and plan suffix are individually designed and optimized, it is possible that the end of the optimal plan prefix is not the same as the start of the plan suffix, so that the plan prefix and plan suffix cannot be smoothly connected. Hence, in this work we introduce a finite trajectory Π_{tra}, namely plan transition, that starts from the end of plan prefix and ends at the start of the plan suffix.

Definition 2. The planning Π is in the form of $\Pi = \Pi_{pre} \Pi_{tra} \Pi_{suf} \Pi_{suf}, \dots$, where Π_{pre} and Π_{suf} are finite prefix and finite cyclic suffix, respectively, and Π_{tra} is the finite transition that connects Π_{pre} and Π_{suf}.

Since $[\Pi_{pre}, \Pi_{tra}, \Pi_{suf}] \models (\Phi, M)$ also indicates that $[\Pi_{pre}, \Pi_{tra}, \Pi_{suf}, \Pi_{suf}, \dots] \models (\Phi, M)$, we only need to determine Π_{pre}, Π_{tra}, Π_{suf} in Π, and the path planning P_{pre}, P_{tra}, and P_{suf} of the robot can be determined accordingly.

As discussed in Sec. I, when using hierarchical approaches, the high-level planner needs to determine not only "where to go", but also the implicit motion constraints (e.g., "where not to go") during each stage of the task planning. Hence, we define the barriers BAR to indicate such constraints to the robot.

Definition 3. The barriers in Π are defined as: $BAR = bar_0 bar_1, \dots$, where $bar_i \in 2^{AP}$, $i = 0, 1, \dots$, indicates the banned actions (as well as the associated areas) during the transition from Π_i to Π_{i+1}. .

Based on the defined $[\Pi_{pre}, \Pi_{tra}, \Pi_{suf}]$ and BAR, the robot motion planning can be stated as follows.

Problem 1. Given an LTL task Φ and a continuous and bounded workspace M, the goal is to obtain a *fast* motion planning for a mobile robot to satisfy Φ.

V. Planning Algorithm

This section presents a hierarchical decoupling framework to address Problem 1. As shown in Fig. 3, the general idea

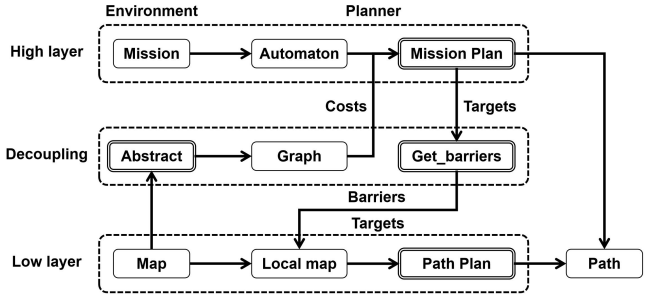


Fig. 3. The hierarchical decoupling framework. It constructs by high layer, low layer, and decoupling layer. The high layer is for the mission planning. The low layer is for the path planning. The decoupling layer is to abstract real environment and guides the plan in the real environment.

is to divide the planning problem into a task planner (i.e., the high layer) and a path planner (i.e., the low level). The task planner determines the targets that the robot needs to sequentially arrive at (i.e., where to go) and the barriers that the robots should avoid during motion (i.e., where not to go), while the path planner realizes the navigation to the targets in the real environment. Unlike prior works, the task planner and path planner are decoupled in this work, so that they can run independently for fast and effective planning. To enable such a decoupling, the decoupling layer is designed, which bridges the high and low layers by providing necessary information exchange. In particular, by interacting with the decoupling layer, cost efficient destinations can be determined in the high layer, which in turn determines the barriers to guide the path planning in the low layer.

Algorithm 1: Hierarchical decoupling framework

Input: LTL formula Φ, workspace M
Output: The path planning P_{pre}, P_{tra}, P_{suf}
1 Generate the Büchi automaton B_Φ by Φ;
2 Construct the abstract map T from M;
3 Search for the task plan $\Pi_{pre}, \Pi_{tra}, \Pi_{sub}$ and barriers BAR using the high level planning (Alg. 2);
4 Initialize the path plan $P_{pre} = [p_0]$, $P_{tra} = []$, $P_{suf} = []$;
5 **for** $\Pi_i \in [\Pi_{pre}, \Pi_{tra}, \Pi_{suf}]$ **do**
6 Obtain the trajectory P_i from Π_i to Π_{i+1} according to M and BAR using the low level planning (Alg. 4);
7 Add P_i to P_{pre}, P_{tra}, P_{suf};
8 **end**
9 Return P_{pre}, P_{tra}, P_{suf}

An overview of the developed hierarchical decoupling approach is shown in Alg. 1. Given the LTL task Φ and the workspace M, the Büchi automaton B_Φ and the abstract map T are first constructed. For formula Φ, the related atomic propositions in AP and their corresponding areas in M are selected. Considering the related proposition ap_j and one area of interest D_i, we construct q_i that satisfies $LM(q_i) \in D_i$ and $L(q_i) = LA(LM(q_i)) = ap_j$. Then, we can construct Σ by the distance of each map state tuple. The construction of T can be found in Example 1. Then, B_Φ and T are used to obtain the planning $\Pi_{pre}, \Pi_{tra}, \Pi_{suf}$ and the barriers BAR. Low-level path planing P_{pre}, P_{tra}, and P_{suf} are generated accordingly.

A. High-level Planning

The goal of task planning is to divide the mission into high-level organized sub-tasks (e.g., a set of destinations/action to be visited/performed sequentially) while abstracting out low-level planning details. Such a decomposition can significantly reduce the computational complexity; however, necessary constraints can be ignored, especially when the tasks are implicitly coupled with the environment. For instance, the task planner may only inform the robot the next destination while the robot is implicitly constrained to not bypass certain areas before arriving at the destination. Such implicit constraints need to be appropriately taken into account. Hence, the task planner needs to determine not only the destinations, but also the barriers.

Algorithm 2: High-level planning

Input: the abstract map T and Büchi automaton B_Φ
Output: Π_{pre},Π_{tra},Π_{suf},BAR
1 **for** s *in* S_F **do**
2 $\Pi_{test}(s) = Sampling(s_0, s, T, B_\Phi)$;
3 **end**
4 Select the plan prefix with the least cost from $\{\Pi_{test}(s)\}$ as Π_{pre};
5 Set $\Pi_{pre}(end) = (s_{ep}, \varepsilon_{ep}, \pi_{ep})$;
6 Initialize $S_{F_f} = \emptyset$;
7 **for** s *in* S_F **do**
8 $\Pi_{test}(s) = Sampling(s_{ep}, s, T, B_\Phi)$;
9 **if** $\Pi_{test}(s)$ *exists* **then**
10 add s to S_{F_f};
11 **end**
12 **end**
13 **for** s *in* S_{F_f} **do**
14 $\Pi_{test}(s) = Sampling(s, s, T, B)$;
15 **end**
16 Select the plan suffix with the least cost as Π_{suf};
17 Set $\Pi_{suf}(end) = (s_{es}, \varepsilon_{es}, \pi_{es})$;
18 Determine $\Pi_{tra} = Sampling(s_{ep}, s_{es}, T, B_\Phi)$;
19 Determine $BAR = Get\_barriers([\Pi_{pre}, \Pi_{tra}, \Pi_{suf}])$;
20 Return Π_{pre},Π_{tra},Π_{suf}, BAR;

The task planning is outlined in Alg. 2. First, based on the abstract map T and Büchi automaton B_Φ, the plan prefix with the least cost is obtained (lines 1-4). Since the sampling method in [5] can obtain a satisfying trajectory from an initial state to an accepting state in S_F, it is employed to sample the states from T and B_Φ for feasible state transitions and then obtain a set of prefix parts, denoted by $\{\Pi_{test}(s)\}$ that start from the initial state s_0 to all accepting states $s \in S_F$. Let the cost be defined as $J(\Pi) = \sum_{i=0}^{|\Pi|} \Sigma(||\varepsilon_{i+1} - \varepsilon_i||_2)$, which is consistent with the definition of cost in [5]. Based on the defined cost, the plan with the least cost is selected from $\{\Pi_{test}(s)\}$ as the prefix path Π_{pre}. Let $\Pi_{pre}\,(end)$ denote the last state in the prefix part Π_{pre} and denote by $\Pi_{pre}(end) = (s_{ep}, \varepsilon_{ep}, \pi_{ep})$ its entries.

After determining Π_{pre}, lines 5-18 presents how a feasible cyclic suffix part is obtained. Let S_{F_f} denote the set of starting and end states of cyclic suffix parts that are feasible for the plan prefix Π_{pre}. To construct S_{F_f}, the sampling

method is used to explore all reachable states in S_F from s_{ep}. If there exists a feasible trajectory from s_{ep} to an accepting state $s \in S_F$, then s is one of the feasible starting states of the suffix plan and will be added to the set S_{F_f} (lines 5-12). According to S_{F_f}, we can continue to explore all feasible cyclic suffix parts. Let $s_{es} \in S_F$ be the starting and end state of plan suffix and all suffix plans that end at s_{es} can be enumerated using the sampling method. We then select one with the least cost as the cyclic suffix plan, denoted by Π_{suf}. By setting the starting point as s_{ep} and the ending point as s_{es}, we can use the sampling method to obtain the shortest transition plan Π_{tra} to connect Π_{pre} and Π_{suf} (line 13-18). The idea behind this design is to optimize Π_{pre} and Π_{suf} individually without requiring that the end state of Π_{pre} is the same as the starting state of Π_{suf}, which gives more freedom to tune the system performance.

Algorithm 3: Get_barriers

Input: Π,B
Output: BAR
1 Initialize $BAR = \emptyset$;
2 **for** $(s_i, \varepsilon_i, \pi_i)$ *in* Π **do**
3 **if** $\pi_i = \pi_0$ **then**
4 add $bar_0 = \emptyset$ to BAR and continue;
5 **end**
6 Set $bar_i = \emptyset$;
7 **for** *ap in* AP/π_i **do**
8 **if** $ap \wedge \pi_i$ *is not in* $\Delta(s_{i-1}, s_i)$ **then**
9 add ap to bar_i
10 **end**
11 **end**
12 add bar_i to BAR
13 **end**
14 Return BAR;

Last, $Get\_barriers$ function is designed to obtain $BAR = bar_1 bar_2 \ldots$, where bar_i indicates the constraints (e.g., banned actions in this work) that should be taken into account during the transition from Π_{i-1} to Π_i, $i > 0$. As shown in Alg. 3, for each $\Pi_i = (s_i, \varepsilon_i, \pi_i) \in \Pi$, if $\pi_i \wedge ap$ is not in $\Delta(s_{i-1}, s_i)$ for any $ap \in AP$, it indicates that ap is not executable when agent executes π_i. That is, the labeled area corresponding to the action ap should not be accessible to the robot. Therefore, ap is added to bar_i, which will then be used to update BAR.

For instance, since ap_4 or ap_5 are not allowed during the transitions from $ap_0 \rightarrow ap_1 \rightarrow ap_2 \rightarrow ap_3$, $\Delta(s_0, s_1), \Delta(s_1, s_2), \Delta(s_2, s_3)$ will not have any word that contains ap_4 or ap_5. Therefore, we can obtain $BAR = bar_0 \ldots bar_5$ with $bar_0 = \emptyset$, $bar_1 = bar_2 = bar_3 = \{ap_4, ap_5\}$, and $bar_4 = bar_5 = \emptyset$.

B. Low-level planning

After determining the task-level destinations and the associated constraints, the next step is to realize the navigation towards the destinations. Inspired by [25], RRT and receding horizon control (RHC) are integrated in this work to deal with the motion planning. Let P_i denote the path segment that connects ε_i and ε_{i+1}, i.e., $P_i = \varepsilon_i p_1 \ldots p_n \varepsilon_{i+1}$ where $p_j \in M$, $j = 1, \ldots n$. Alg. 4 outlines how P_i is obtained. Let d_s denote the sampling step size (i.e., the maximum distance between p_j and p_{j+1} in P_i) and r_s denote the radius of sampling area. We first sample a set of random points PT_{temp} from the sampling area centered at ε_i with radius r_s. We then select the point p_{target} from PT_{temp} that is closest to ε_{i+1} and does not belong to BAR_i (Lines 3-4). RRT is then employed to determine a path PH from the current point $p_{current}$ to the target point p_{target} (lines 5-6). Following the idea of RHC, the first point in PH is used to update P_i (lines 7-8). Repeat the process above until ε_{i+1} is reached (lines 9-11).

Algorithm 4: Low-level planning

Input: ε_i, ε_{i+1}, BAR, M
Output: path segment P_i
1 Initialize $P_i = \varepsilon_i$;
2 **while** *1* **do**
3 Sample a set of points PT_{temp};
4 Find $p_{target} \in PT_{temp}$ such that $\|p_{target} - \varepsilon_{i+1}\|$ is minimized and $p_{target} \notin BAR_i$;
5 Set $p_{current}$ as the end point of P_i;
6 Find a path $PH = RRT(p_{current}, p_{target}, BAR_i, d_s)$;
7 Select $p = PH(1)$;
8 Add p to the end of P_i;
9 **if** $|p - \varepsilon_{i+1}| < d_s$ **then**
10 break out
11 **end**
12 **end**
13 Return P_i;

C. Complexity analysis

Different from conventional sampling methods [6], our approach is a layered sampling in the sense that we only sample the automaton states and the abstract map in the high-level task planner while, in the low-level motion planner, only points in the workspace M are sampled to realize point-to-point navigation, which significantly improve the sampling efficiency. Let X denote the set of nodes in point to point navigation. The size of T is $|Q|$ and the size of B_Φ is $|S|$. The length of planning scheme satisfies $|[\Pi_{pre}, \Pi_{tra}, \Pi_{sub}]| < 3 \times |S|$. For the method in [6], the space complexity can be expressed as $3 \times |S| \times |X|$ and the time complexity is $|X|^2 \times (3 \times |S|)^2$. For the sampling method with hierarchical decoupling framework, the space complexity is $3 \times |S| \times |Q| + |X|$ and the time complexity is $|X|^2 + |Q|^2 \times (3 \times |S|)^2$. Since $|Q|$ is much smaller than $|X|$, the space complexity and time complexity of layered sampling are much smaller than conventional sampling methods. Besides, the layered map sampling has much lower dependence on map resolution and thus it is applicable to a more refined map.

VI. Numerical Simulations

A. Algorithm Complexity

In this section, our approach is compared with the traditional map-based sampling algorithm and the layered sam-

pling algorithm by searching over the the same map to show its advantages in the time and space complexity. Specifically, we consider the task Φ in Example 1.

The map-based sampling algorithm from [11] and the automaton and map based sampling (referred to as double sampling) from [6] are compared with the layered map search and layered double sampling adapted from our hierarchical decoupling framework. The step size in these sampling algorithms is set as 1 and the number of search points is approximately 100. The results are shown as Table I. which clearly show that the time and space complexity of the layered sampling algorithm are greatly reduced.

TABLE I

SOLUTION TIME FOR PATH PLANNING UNDER TASK Φ

Method	Time(/s)	Complexity
Map sampling	5.38	360000
Double sampling	0.395	1800
Layered map sampling	0.0182	900+100
Layered double sampling	0.0107	90+100

B. Sensitivity to Map Resolution

With the decrease of the sampling step size or the increase of the map size, the map resolution and the search space will increase, which affect the solution complexity. In this section, more comparisons are carried out to show that our approach is not only fast but also insensitive to the map resolution. We continue with the task and map in Example 1 and compare the map-based sampling [11] with our hierarchical decoupling approach to show how the planning speed varies with different sampling step. The results are listed in Table II, which record the solution time and the time ratio. The time ratio is the ratio between the solution time under different steps and the solution time when $step = 1$. Apparently, our approach is not sensitive to the map resolution, as its calculation time increases slowly with the increase of map resolution. Hence, our approach can be used for complex task planning in a high-resolution map. In contrast, traditional methods show strong sensitivity to the map resolution.

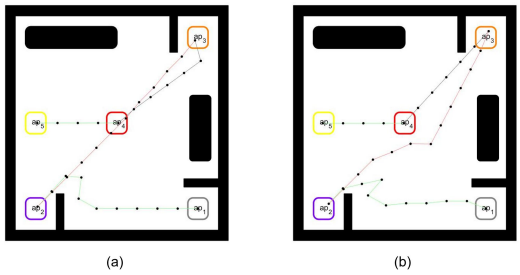
TABLE II

SOLUTION TIME WITHOUT HIERARCHICAL DECOUPLING FRAMEWORK

Method	Step size d_s	Time(/s)	Time ratio
Vasile et al. [11]	1	5.38	1.00
	0.3	48.5	9.01
	0.1	545	101
Our	1	0.0182	1.00
	0.3	0.0227	1.24
	0.1	0.0316	1.73
	0.03	0.0496	2.73
	0.01	0.125	6.87

C. Security

This section shows the security guarantee of the algorithm compared with ordinary layered architecture. Fig. 4 (a) shows the planned path using the the original layered framework, while Fig. 4 (b) shows the planned path using our approach.



(a) (b)

Fig. 4. The map contains five areas, which correspond to five atomic propositions ap_1, ap_2, ap_3, ap_4, ap_5 respectively. The starting point is in area 1. In (a), the original layered framework is used and the planned path from ap_2 to ap_3 is through ap_4. In (b), the layered framework in this paper is used and the planned path from ap_2 to ap_3 avoids ap_4.

It can be found that our layered framework adequately reflects the intent of the mission and the original layered framework can not transfer the information "where not to go" to the low-level planner. Therefore, our framework has a better performance on mission based on LTL formula.

VII. EXPERIMENT

Experiments were performed on a mobile robot, turtlebot3 burger, to verify the developed method. The workspace is similar as the Fig. 2(a) and the formula Φ is the same as Example 1. Given a Büchi automaton, the robot can obtain the plan within 0.06s. The agent will visit the blue, pink, orange, red, and yellow areas in order. And it will avoid entering the red area until reaching the orange area. Finally, the agent will cycle through the yellow and red regions to complete ap_4 and ap_5. The details of experiment can be found in `https://www.youtube.com/watch?v=9jzamf1TlyE`.

(a) (b)

Fig. 5. The map contains five areas, which correspond to five atomic propositions ap_1, ap_2, ap_3, ap_4, ap_5 respectively. In (a), the agent is going to the orange area for ap_3 from the pink area with ap_2 while avoiding the red area with ap_4 and the yellow area with ap_5. In (b), the agent continually visits the yellow area with ap_5 and the red area with ap_4.

VIII. CONCLUSIONS

In this work, a hierarchical decoupling framework is developed. It allows the the task planner and path planner to run independently, which significantly reduces the search space and enables fast motion planing. Since the current work focuses on motion planning of a single robot, additional research will consider extension to task and motion planning for homogeneous/heterogeneous multi-robot systems.

REFERENCES

[1] S. L. Smith, J. Tumova, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *Int. J. Robotics Res.*, vol. 30, no. 14, pp. 1695–1708, 2011.

[2] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Trans. Robot.*, vol. 25, no. 6, pp. 1370–1381, 2009.

[3] A. Jones, M. Schwager, and C. Belta, "Information-guided persistent monitoring under temporal logic constraints," in *Proc. Am. Control Conf.* IEEE, 2015, pp. 1911–1916.

[4] M. Guo and D. V. Dimarogonas, "Multi-agent plan reconfiguration under local LTL specifications," *Int. J. Robot. Res.*, vol. 34, no. 2, pp. 218–235, 2015.

[5] Y. Kantaros and M. M. Zavlanos, "Stylus*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems," *Int. J. Robot. Res.*, vol. 39, no. 7, pp. 812–836, 2020.

[6] X. Luo, Y. Kantaros, and M. M. Zavlanos, "An abstraction-free method for multirobot temporal logic optimal control synthesis," *IEEE Trans. Robot.*, vol. 37, no. 5, pp. 1487–1507, 2021.

[7] L. Kaelbling and T. Lozano-Perez, "Hierarchical task and motion planning inthe now," in *IEEE Int. Conf. Robot. Autom.*, 2010.

[8] T. Lozano-Pérez and L. P. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst.* IEEE, 2014, pp. 3684–3691.

[9] 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 *IEEE Int. Conf. Robot. Autom.* IEEE, 2014, pp. 639–646.

[10] Y. Kantaros, M. Malencia, V. Kumar, and G. J. Pappas, "Reactive temporal logic planning for multiple robots in unknown environments," in *IEEE Int. Conf. Robot. Autom.* IEEE, 2020, pp. 11 479–11 485.

[11] C. I. Vasile, X. Li, and C. Belta, "Reactive sampling-based path planning with temporal logic specifications," *Int. J. Robot. Res.*, p. 0278364920918919, 2020.

[12] M. Cai, S. Xiao, Z. Li, and Z. Kan, "Optimal probabilistic motion planning with potential infeasible ltl constraints," *IEEE Trans. Autom. Control*, vol. 68, no. 1, pp. 301–316, 2023.

[13] M. Cai, M. Hasanbeig, S. Xiao, A. Abate, and Z. Kan, "Modular deep reinforcement learning for continuous motion planning with temporal logic," *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 7973–7980, 2021.

[14] A. Ulusoy, S. L. Smith, and C. Belta, "Optimal multi-robot path planning with ltl constraints: guaranteeing correctness through synchronization," in *Distributed Autonomous Robotic Systems*. Springer, 2014, pp. 337–351.

[15] M. Guo, C. P. Bechlioulis, K. J. Kyriakopoulos, and D. V. Dimarogonas, "Hybrid control of multiagent systems with contingent temporal tasks and prescribed formation constraints," *IEEE Trans. Control Network Syst.*, vol. 4, no. 4, pp. 781–792, 2017.

[16] M. Cai, K. Leahy, Z. Serlin, and C.-I. Vasile, "Probabilistic coordination of heterogeneous teams from capability temporal logic specifications," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 1190–1197, 2021.

[17] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Sampling-based motion planning with temporal goals," in *IEEE Int. Conf. Robot. Autom.* IEEE, 2010, pp. 2689–2696.

[18] O. Arslan and P. Tsiotras, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *IEEE Int. Conf. Robot. Autom.* IEEE, 2013, pp. 2421–2428.

[19] C. I. Vasile and C. Belta, "Sampling-based temporal logic path planning," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst.* IEEE, 2013, pp. 4817–4822.

[20] K. He, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "Towards manipulation planning with temporal logic specifications," in *IEEE Int. Conf. Robot. Autom.* IEEE, 2015, pp. 346–352.

[21] X. Luo and M. M. Zavlanos, "Temporal logic task allocation in heterogeneous multirobot systems," *IEEE Trans. Robot.*, pp. 1–20, 2022.

[22] V. Vasilopoulos, Y. Kantaros, G. J. Pappas, and D. E. Koditschek, "Reactive planning for mobile manipulation tasks in unexplored semantic environments," in *Proc. Int. Conf. Robot. Autom.* Xi'an, China: IEEE, 2021, pp. 6385–6392.

[23] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[24] P. Gastin and D. Oddoux, "Fast LTL to Büchi automata translation," in *Int. Conf. Comput. Aided Verif.* Springer, 2001, pp. 53–65.

[25] M. Cai, H. Peng, Z. Li, H. Gao, and Z. Kan, "Receding horizon control based motion planning with partially infeasible LTL constrains," *IEEE Control Syst. Lett.*, vol. 5, no. 4, pp. 1279–1284, 2020.