Planning in Discrete and Continuous Spaces: From LTL Tasks to Robot Motions

Erion Plaku

Dept. of Electrical Engineering and Computer Science
Catholic University of America, Washington DC 20064

Abstract. Enabling robots to accomplish sophisticated tasks requires enhancing their capability to plan at multiple levels of discrete and continuous abstractions. Toward this goal, the proposed approach couples the ability of sampling-based motion planning to handle the complexity arising from high-dimensional robotic systems, nonlinear dynamics, and collision avoidance with the ability of discrete planning to handle discrete specifications. The approach makes it possible to specify tasks via Linear Temporal Logic (LTL) and automatically computes collision-free and dynamically-feasible motions that enable the robot to carry out assigned tasks. While discrete planning guides sampling-based motion planning, the latter feeds back information to further refine the guide and advance the search. Sampling is also used in the discrete space to shorten the length of the discrete plans and to expand the search toward new discrete states. Experiments with high-dimensional dynamical robot models performing various LTL tasks show significant computational speedups over related work.

1 Introduction

The field of robotics nowadays is marked by an emphasis towards increasing the autonomy of robots. A fundamental component of autonomy is the ability of a robot to plan its motions in order to carry out assigned tasks. Whether the task is to search, inspect, navigate, or explore, it generally involves abstractions into discrete, logical actions, where each discrete action often requires substantial continuous motion planning to carry out. These settings require the robots to reason and plan at multiple levels of discrete and continuous abstractions. The coupling of the discrete and the continuous, however, poses significant challenges as discrete and continuous planning have generally been treated separately.

Discrete planning makes the simplifying assumption that both the world in which the robot operates and the actions that it performs are discrete [1]. This has made it possible to efficiently plan sequences of discrete actions to accomplish sophisticated tasks given by various logics, such as LTL, which combines propositions, logical connectives ($\wedge$ and, $\vee$ or, $\neg$ not), and temporal connectives ($\bigcirc$ next, $\Diamond$ eventually, $\Box$ always, $\cup$ until, $\mathcal{R}$ release). As an example, the task of vacuuming all the rooms can be expressed in LTL as

$$\Diamond \pi_{R_1} \wedge \ldots \wedge \Diamond \pi_{R_n}, \tag{1}$$

where π_{R_i} denotes the proposition "robot vacuumed room R_i." As another example, the task "after inspecting a contaminated area, go next to a decontamination station, and then eventually return to a base station" can be expressed as

$$(\neg\pi_{C_1}\wedge\ldots\wedge\neg\pi_{C_n})\,\mathcal{U}\,((\pi_{C_1}\vee\ldots\vee\pi_{C_n})\wedge\bigcirc(\pi_{D_1}\vee\ldots\vee\pi_{D_m})\wedge\Diamond(\pi_{B_1}\vee\ldots\vee\pi_{B_\ell}))\,, \tag{2}$$

where $\pi_{C_i}, \pi_{D_j}, \pi_{B_k}$ denote the propositions robot is at contaminated area C_i, decontamination station D_j, and base station B_k, respectively.

In contrast, motion planning assumes a continuous world since it is essential to compute trajectories that can be followed by the robot in the physical world. As a result, planned motions need not only to avoid collisions but also to satisfy differential constraints imposed by the robot dynamics. Due to the increased complexity, motion planning has generally been limited to simpler tasks, such as planning motions to reach a goal state [2,3]. Tasks arising in navigation, exploration, search-and-rescue, and other domains, however, are far more sophisticated and generally cannot be expressed just in terms of a goal state.

The problem of planning motions that satisfy a task given by a discrete logic such as LTL is often approached by first computing a discrete plan $\sigma = \tau_1, \tau_2, \ldots$ that satisfies the discrete abstraction of the task. Referring to Eq. 2, σ could be obtained by setting $\tau_1 = \{\pi_{C_2}\}$, $\tau_2 = \{\pi_{D_3}\}$, and $\tau_3 = \{\pi_{B_1}\}$, which indicates that the robot should inspect area C_2, then go to D_3, and finally B_1. In a second stage, controllers are used to enable the robot to reach states that satisfy the propositions in the order specified by σ. Discrete search in conjunction with controllers has been widely used in field robotics, for example, to first obtain a sequence of grid cells or waypoints and then closely follow the waypoints [4, 5]. Discrete logic is used in [6] to combine a set of behavior schemas which use controllers to link motion to abstract actions. This idea has been revisited more recently in [7] to synthesize reactive controllers from LTL for car-like systems and in [8] to deploy robotic teams in urban environments. The work in [9] uses LTL to determine the sequence of triangles a point robot needs to visit and relies on a controller to drive the robot between adjacent triangles.

A principal limitation of approaches that treat discrete planning and motion planning separately is their underlying assumption that any discrete plan can be carried out in the continuous world. Note that the task specification could present polynomially or exponentially many different discrete solutions, as it is the case in Eq. 1 and 2. As a result of constraints imposed by obstacles, robot geometry, and motion dynamics, it may be impossible to carry out certain discrete plans. Determining which discrete plan is feasible in the continuous setting is in fact one of the main challenges when incorporating LTL into motion planning. As a result, the applicability of these approaches is limited to specific robots and to specific discrete plans for which controllers are available.

To bridge the gap between the discrete and the continuous, the proposed framework couples the ability of motion planning to handle the complexity arising from high-dimensional robotic systems, nonlinear dynamics, and collision avoidance with the ability of discrete planning to take into account discrete specifications. The framework makes it possible to specify tasks via LTL and

automatically computes collision-free and dynamically-feasible motions that enable the robot to carry out the assigned tasks. Motion planning in the framework is based on state-of-the-art sampling-based approaches which have had significant success in solving challenging motion-planning problems with dynamics (cf. [2,3]). In particular, the framework builds upon related work by the author which uses discrete search and sampling-based motion planning to reduce the planning time when dealing with nonlinear dynamics [10], complex goals [11], and hybrid systems [12]. The framework leverages from sampling-based motion planning the underlying idea of searching for a solution trajectory by selectively sampling and exploring the continuous space of feasible motions. In particular, sampling-based motion planning extends a tree in the continuous state space by adding new trajectories as tree branches, which are obtained by sampling input controls and propagating forward the motion dynamics of the robot. Discrete planning guides the sampling-based motion planner by searching over both the LTL formula representation and a workspace decomposition to provide discrete plans as intermediate sequences of propositional assignments that should be satisfied. In distinction from related work [12, 13], the proposed approach samples the discrete space to shorten the length of the discrete plans and to expand the search toward new propositions that enable the sampling-based motion planner to explore the continuous state space. Experiments with high-dimensional dynamical robot models performing various LTL tasks show significant computational speedups over related work.

2 LTL Specifications

Let Π denote a set of propositions, where each $\pi_i \in \Pi$ corresponds to a Boolean-valued problem-specific statement, such as "robot is in area A_i." LTL combines propositions with logical connectives ($\neg$ not, $\wedge$ and, $\vee$ or), and temporal connectives ($\bigcirc$ next, $\Diamond$ eventually, $\Box$ always, $\cup$ until, $\mathcal{R}$ release). A discrete state $\tau_i \in 2^{\Pi}$ denotes all the propositions that hold true in the world. As the world changes, e.g., as the result of robot actions, the discrete state could also change. LTL planning consists of finding a sequence of discrete states $\sigma = \tau_0, \tau_1, \ldots$ that satisfies a given LTL formula, whose syntax and semantics are defined below.

LTL Syntax and Semantics: Every $\pi \in \Pi$ is a formula. If ϕ and ψ are formulas, then $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\bigcirc\phi$, $\Diamond\phi$, $\Box\phi, \phi \cup \psi$, $\phi \mathcal{R} \psi$ are also formulas. Let $\sigma = \tau_0, \tau_1, \ldots$, where each $\tau_i \in 2^{\Pi}$. Let $\sigma^i = \tau_i, \tau_{i+1}, \ldots$ denote the i-th postfix of σ. The notation $\sigma \models \phi$ indicates that σ satisfies ϕ and is defined as

$\sigma \models \pi$ if $\pi \in \Pi$ and $\pi \in \tau_0$; $\quad$ $\sigma \models \neg\phi$ if $\sigma \not\models \phi$;
$\sigma \models \phi \wedge \psi$ if $\sigma \models \phi$ and $\sigma \models \psi$; $\quad$ $\sigma \models \bigcirc\phi$ if $\sigma^1 \models \phi$;
$\sigma \models \phi \cup \psi$ if $\exists\, k \geq 0$ such that $\sigma^k \models \psi$ and $\forall\, 0 \leq i < k : \sigma^i \models \phi$.

Also, $\texttt{false} = \pi \wedge \neg\pi$, $\texttt{true} = \neg\texttt{false}$, $\phi \vee \psi = \neg(\neg\phi \wedge \neg\psi)$, $\Diamond\phi = \texttt{true} \cup \phi$, $\Box\phi = \neg\Diamond\neg\phi$, and $\phi \mathcal{R} \psi \equiv \neg(\neg\phi \cup \neg\psi)$. More details can be found in [14].

Co-Safe LTL and Automaton Representation: Since LTL planning is PSPACE-complete [15], as in related work [12, 13], this paper considers co-safe LTL. Co-safe LTL formulas are satisfied by finite sequences of discrete states rather than infinite sequences which satisfy general LTL formulas. Most robotic tasks are finite in nature, hence, the use of co-safe LTL does not limit the general applicability of the approaches. Co-safe LTL formulas can be translated into DFAs (Deterministic Finite Automata), which are then minimized [14]. A DFA is a tuple $\mathcal{A} = (Z, \Sigma, \delta, z_{init}, Accept)$, where Z is a finite set of states, $\Sigma = 2^{\Pi}$ is the input alphabet, $\delta : Z \times \Sigma \to Z$ is the transition function, $z_{init} \in Z$ is the initial state, and $Accept \subseteq Z$ is the set of accepting states. The state obtained by running $\mathcal{A}$ on $\sigma = [\tau_i]_{i=1}^{n}$, $\tau_i \in 2^{\Pi}$, starting from the state z is defined as

$$\mathcal{A}([\tau_i]_{i=1}^{n}, z) = \begin{cases} z, & n = 0 \\ \delta(\mathcal{A}([\tau_i]_{i=1}^{n-1}, z), \tau_n), & n > 0. \end{cases}$$

$\mathcal{A}$ accepts σ iff $\mathcal{A}(\sigma, z_{init}) \in Accept$. As a result, $\sigma \models \phi$ when the equivalent automaton $\mathcal{A}$ accepts σ. To facilitate presentation, let *Reject* denote all rejecting states of $\mathcal{A}$, i.e., states that cannot reach an accepting state. Moreover, let $\delta(z)$ denote all the non-rejecting states connected by a single transition from z, i.e.,

$$\delta(z) = \{\delta(z, \tau) : \tau \in 2^{\Pi}\} - Reject.$$

Interpretation of LTL over Continuous Motion Trajectories: The continuous state space $\mathcal{S}$ gives meaning to propositions $\pi_i \in \Pi$. As an example, the proposition π_i "robot is in area A_i" holds iff the robot is actually in A_i. More generally, a function $\textsc{holds}_{\pi_i} : \mathcal{S} \to \{\texttt{true}, \texttt{false}\}$ defines if $\pi_i \in \Pi$ holds at $s \in \mathcal{S}$. This interpretation provides a mapping $\textsc{DiscreteState} : \mathcal{S} \to 2^{\Pi}$ from the continuous state space to the discrete space, i.e.,

$$\textsc{DiscreteState}(s) = \{\pi_i : \pi_i \in \Pi \wedge \textsc{holds}_{\pi_i}(s) = \texttt{true}\}.$$

Moreover, trajectories over $\mathcal{S}$ give meaning to temporal connectives. A trajectory over $\mathcal{S}$ is a continuous function $\zeta : [0, T] \to \mathcal{S}$, parametrized by time. As the continuous state changes according to ζ, the discrete state, obtained by the mapping DiscreteState, may also change. In this way, ζ gives rise to a sequence of discrete states, $\textsc{DiscreteStates}(\zeta) \stackrel{def}{=} [\tau_i]_{i=1}^{n}$, where $\tau_i \neq \tau_{i+1}$. As a result of this mapping, ζ is said to satisfy an LTL formula ϕ iff $\textsc{DiscreteStates}(\zeta) \models \phi$.

The motion dynamics are specified as a set of differential equations $f : \mathcal{S} \times \mathcal{U} \to \dot{\mathcal{S}}$, where $\mathcal{U}$ describes the control inputs that can be applied to the robot (e.g., a car can be controlled by setting the acceleration and the rotational velocity of the steering wheel). The approach can take into account nonlinear dynamics, relying only on a simulator, which, when given a state s, a control u, and a time step dt, computes the new state that results by integrating f.

A dynamically-feasible trajectory $\zeta : [0, T] \to \mathcal{S}$ starting at $s \in \mathcal{S}$ is obtained by computing a control function $\tilde{u} : [0, T] \to \mathcal{U}$ and propagating the dynamics

forward in time through numerical integration of $\dot{\zeta}(h) = f(\zeta(h), \tilde{u}(h))$, i.e.,

$$\zeta(t) = s + \int_0^t f(\zeta(h), \tilde{u}(h))\, dh.$$

Problem Statement: Given $\langle \mathcal{S}, \mathcal{U}, f, s_{init}, \Pi, \phi \rangle$ compute a control function $\tilde{u} : [0, T] \to \mathcal{U}$ such that the resulting dynamically-feasible trajectory $\zeta : [0, T] \to \mathcal{S}$ is collision free and satisfies the LTL specification ϕ, i.e., $\textsc{DiscreteStates}(\zeta) \models \phi$.

Examples: To facilitate presentation, this section provides examples of the robot model, workspaces, and the LTL tasks used in the experiments.

Robot Model: The robot model, as shown in Fig. 1, resembles a snake consisting of several links attached to each other. The continuous state

$$s = (x, y, \theta_0, v, \psi, \theta_1, \theta_2, \ldots, \theta_N),$$

consists of the position $(x, y) \in \mathbb{R}^2$ ($|x| \leq 30, |y| \leq 25$), orientation $\theta_0 \in [-\pi, \pi)$, velocity v ($|v| \leq 2$), and steering-wheel angle ψ ($|\psi| \leq 1rad$) of the head link, and the orientation θ_i ($\theta_i \in [-\pi, \pi), 1 \leq i \leq N$) of each trailer link. The geometry of each link corresponds to a unit box.

The robot is controlled by setting the acceleration a ($|a| \leq 1$) and the rotational velocity ω ($|\omega| \leq 1rad/s$) of the steering wheel. The motion dynamics are modeled as a car pulling trailers (adapted from [16] and [3, pp. 731]), i.e.,

$$\dot{x} = v\cos(\theta_0); \quad \dot{y} = v\sin(\theta_0); \quad \dot{\theta_0} = v\tan(\psi); \quad \dot{v} = a; \quad \dot{\psi} = \omega$$
$$\dot{\theta_i} = \frac{v}{d}(\sin(\theta_{i-1}) - \sin(\theta)) \prod_{j=1}^{i-1} \cos(\theta_{j-1} - \theta_j),$$

where $1 \leq i \leq N$ and $d = 0.05$ is the hitch length. A continuous state s is considered valid iff the robot does not collide with the obstacles and the robot does not self intersect, i.e., non-consecutive links do not collide with each other.

While related work in LTL motion planning [8, 9, 13] has focused on low-dimensional systems, by increasing the number N of trailer links, the robot model provides challenging high-dimensional problems. As a result of no self intersections and constraints imposed by the motion dynamics, the robot has to wiggle its way through narrow passages in order to reach the goal.

Workspaces: The workspaces in which the robot operates, as in the related LTL motion-planning work [8, 9, 12, 13], are populated with polygonal obstacles and propositions, as shown in Fig. 1. In this way, a proposition π_i is associated with a polygon p_i, and the function $\textsc{holds}_{\pi_i}(s)$ is true iff the position component of s is inside p_i. In addition, as in related work, each workspace is triangulated. The polygonal propositions $p_1, \ldots, p_k$ and the triangles $t_1, \ldots, t_m$ give rise to an adjacency region graph $G = (R, E)$, where

$$R = \{p_1, \ldots, p_k, t_1, \ldots, t_m\} \text{ and } E = \{(r_i, r_j) : r_i, r_j \in R \text{ are adjacent}\}.$$

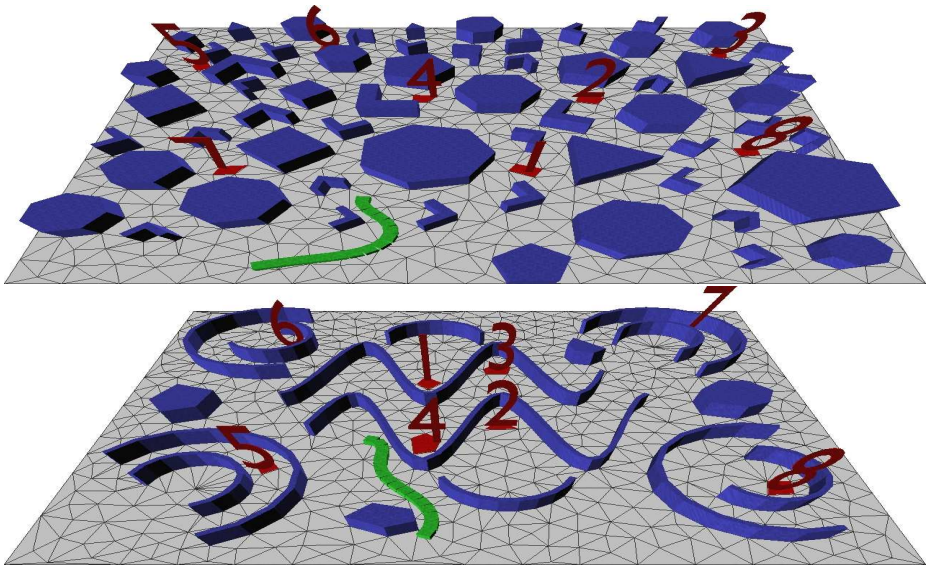

Fig. 1. Workspaces used in the experiments. Obstacles are in blue. Propositions $1, 2, \ldots, 8$ are in red. The robot is in green and is shown in some initial state. The workspace triangulations are also shown. Videos of solutions obtained by the proposed approach can be found at `http://faculty.cua.edu/plaku/ResearchMPAI.html`.

Each region $r \in R$ is labeled by the corresponding discrete state, i.e.,

$$\textsc{DiscreteState}(r) = \begin{cases} \{\pi_\ell\}, & \text{if } r = p_\ell \text{ for some } 1 \leq \ell \leq k, \\ \emptyset, & \text{otherwise.} \end{cases}$$

LTL Tasks: LTL tasks used in the experiments are defined over 8 propositions, as shown in Fig. 1. The first task is to compute a collision-free and dynamically-feasible trajectory ζ which satisfies $\pi_1, \pi_2, \ldots, \pi_8$ in succession, i.e.,

$$\phi_1 = \beta \cup (\pi_1 \wedge ((\pi_1 \vee \beta) \cup (\pi_2 \wedge (\ldots (\pi_7 \vee \beta) \cup \pi_8)))), \quad \text{where } \beta = \wedge_{i=1}^{8} \neg \pi_i.$$

The second task is to compute a collision-free and dynamically-feasible trajectory which eventually satisfies $\pi_i, \pi_j, \pi_k, \pi_i, \pi_j$ with different i, j, k, i.e.,

$$\phi_2 = \bigvee_{1 \leq i,j,k \leq 8, i \neq j, j \neq k, i \neq k} \Diamond \pi_i \wedge (\Diamond \pi_j \wedge (\Diamond \pi_k \wedge (\Diamond \pi_i \wedge (\Diamond \pi_j)))).$$

This task presents polynomially different discrete solutions.

The third task, which presents exponentially many different discrete solutions, is to compute a collision-free and dynamically-feasible trajectory ζ that eventually satisfies each proposition, i.e.,

$$\phi_3 = \bigwedge_{i=1}^{8} \Diamond \pi_i.$$

3 Method

Let $s_{init} \in \mathcal{S}$ denote the initial state of the robot. Let ϕ denote the LTL specification. To compute a collision-free and dynamically-feasible trajectory ζ that satisfies ϕ, the approach couples sampling-based motion planning in the continuous state space $\mathcal{S}$ with discrete search over both the automaton $\mathcal{A}$ representing LTL and the region graph $G = (R, E)$. To facilitate presentation, the general idea is described first followed by details of the approach.

3.1 Coupling Sampling-based Motion Planning and Discrete Search

Sampling-based motion planning uses a tree data structure $\mathcal{T}$ as the basis for conducting the search in the continuous space $\mathcal{S}$. Each vertex $v_i \in \mathcal{T}$ is associated with a collision-free continuous state, denoted as $state(v_i)$. The vertex v_i also keeps track of its parent in $\mathcal{T}$, denoted as $parent(v_i)$, and the collision-free and dynamically-feasible trajectory that connects its parent state to $state(v_i)$, denoted as $ptraj(v_i)$. Initially, $\mathcal{T}$ contains only its root vertex v_1, where $state(v_1)$ corresponds to s_{init}. As the search progresses, $\mathcal{T}$ is extended by adding new vertices v_j and new collision-free and dynamically-feasible trajectories $ptraj(v_j)$. As described in Section 3.3, each $ptraj(v_j)$ is obtained by sampling controls and propagating forward for several steps the motion dynamics of the robot starting from $state(parent(v_j))$ and stopping the propagation if a collision is found.

Let $traj(\mathcal{T}, v_j)$ denote the trajectory from $state(v_1)$ to $state(v_j)$, which is obtained by concatenating the trajectories associated with the tree edges from v_1 to v_j. Then, $traj(\mathcal{T}, v_j)$ satisfies ϕ iff DiscreteStates($traj(\mathcal{T}, v_j)$) $\models \phi$, which indicates that the sequence of discrete states ends up on an accepting state when run on the automaton $\mathcal{A}$ representing ϕ. To make this computation efficient, each vertex v_j is associated with the automaton state, denoted as $z(v_j)$, obtained by running DiscreteStates($traj(\mathcal{T}, v_j)$) on $\mathcal{A}$. The computation of $z(v_j)$ is done incrementally when checking $ptraj(v_j)$ for collisions, as described in Section 3.3. Each vertex v_j is also associated with $region(v_j)$, which denotes the workspace region that contains the position component of $state(v_j)$.

The objective of the discrete layer is to guide sampling-based motion planning. To do so effectively, the discrete layer selects at each iteration an automaton state z_{from} and a region r_{from} from which to expand the search and an automaton state z_{to} toward which to expand the search. The automaton state z_{from} and the region r_{from} are selected from those already associated with the vertices in $\mathcal{T}$. The automaton state z_{to} is selected from those connected by a single automaton transition from z_{from}, i.e., $z_{to} \in \delta(z_{from})$. An additional criterion is that z_{to} should not be associated with the tree vertices, i.e., $\forall v_j \in \mathcal{T} : z(v_j) \neq z_{to}$, so that the search can be expanded toward new automaton states.

The discrete and the continuous layers work in tandem to effectively compute a collision-free and dynamically-feasible trajectory that satisfies ϕ. The search proceeds incrementally until a solution is obtained or an upper bound on computational time is exceeded. Each iteration consists of invoking the discrete layer to select $z_{from}, r_{from}, z_{to}$ and then compute a discrete plan, i.e., a

sequence of automaton states and workspace regions connecting $\langle z_{from}, r_{from}\rangle$ to z_{to}. Sampling-based motion planning is then invoked to expand $\mathcal{T}$ from vertices associated with $\langle z_{from}, r_{from}\rangle$ toward z_{to} while using the discrete plan as a guide. As $\mathcal{T}$ is expanded, new automaton states and workspace regions could be reached by the vertices and trajectories added to $\mathcal{T}$. As a result, the discrete layer could suggest to the continuous layer a different discrete plan for expanding $\mathcal{T}$ in the next iteration. This coupling of discrete planning and sampling-based motion planning, as evidenced by experiments, allows the approach to efficiently compute a collision-free and dynamically-feasible trajectory that satisfies the LTL specification. Pseudocode is given in Algo 1. A more detailed description follows.

Algorithm 1 LTLMotionPlanning($\Pi, \phi, s_{init}, f, t_{max}$)

1: $\mathcal{A} \leftarrow$ automaton(ϕ); $\mathcal{T} \leftarrow$ CreateTree(s_{init}); $G = (R, E) \leftarrow$ region graph
2: **while** time() $< t_{max}$ **and** solved() = `false` **do**
3: $\langle z_{from}, r_{from}, z_{to}\rangle \leftarrow$ DiscreteTarget($\mathcal{T}, \mathcal{A}, G$) ♢ *discrete layer: lines 3–7*
4: $\sigma \leftarrow$ DiscretePlan($\mathcal{A}, G, z_{from}, r_{from}, z_{to}$); $\sigma_{active} \leftarrow \emptyset$
5: **for** $i = |\sigma|$ down to 1 **do**
6: $\langle z, r\rangle \leftarrow \sigma(i)$
7: **if** $|vertices(\mathcal{T}, z, r)| > 0$ **then** $\sigma_{active}.insert(\langle z, r\rangle)$
8: **for** several times **do** ♢ *continuous layer: lines 8–18*
9: $\langle z, r\rangle \leftarrow$ select(σ_{active}); $v \leftarrow$ select($vertices(\mathcal{T}, z, r)$); $u \leftarrow$ SampleControl
10: **for** several times **do** ♢ *extend $\mathcal{T}$ from v by applying control u to state(v)*
11: $dt \leftarrow$ step($state(v), u$); $s_{new} \leftarrow$ IntegrateDiffEqs($f, state(v), u, dt$)
12: $z_{new} \leftarrow \delta(z(v),$ DiscreteState(s_{new})); $r_{new} \leftarrow$ LocateRegion(s_{new})
13: **if** collision(s_{new}) = `true` **or** $(z_{new}) \in Reject$ **then**
14: break for loop of extend $\mathcal{T}$
15: $v_{new} \leftarrow$ AddNewVertex($\mathcal{T}, s_{new}, u, dt, r_{new}, z_{new}, v$)
16: **if** $z_{new} \in Accept$ **then return** $traj(\mathcal{T}, v_{new})$
17: **if** $\langle z_{new}, r_{new}\rangle \notin \sigma_{active}$ **then** $\sigma_{active}.insert(\langle z_{new}, r_{new}\rangle)$
18: $vertices(\mathcal{T}, z_{new}).insert(v_{new})$; $vertices(\mathcal{T}, z_{new}, r_{new}).insert(v_{new})$; $v \leftarrow v_{new}$

3.2 Selecting Discrete Targets and Computing Discrete Plans

DiscreteTarget($\mathcal{T}, \mathcal{A}, G$) selects an automaton state z_{from} and a region r_{from} from which to expand the search and an automaton state z_{to} toward which to expand the search (Algo 1:3). To facilitate the selection, tree vertices are grouped according to their associated automaton states, i.e.,

$$vertices(\mathcal{T}, z) = \{v : v \in \mathcal{T} \wedge z(v) = z\}.$$

These vertices are further grouped according to their associated regions, i.e.,

$$vertices(\mathcal{T}, z, r) = \{v : v \in \mathcal{T} \wedge z(v) = z \wedge region(v) = r\}.$$

Let Γ denote the set of all automaton states that are already associated with the vertices in $\mathcal{T}$, i.e., $\Gamma = \{z : z \in Z \wedge |vertices(\mathcal{T}, z)| > 0\}$. The automaton

state z_{from} is selected from Γ according to the probability distribution $prob(z) = 2^{-d(z)} / \sum_{z' \in \Gamma} 2^{-d(z')}$, where $d(z)$ denotes the minimum number of automaton transitions to reach an accepting state in $\mathcal{A}$ from z. Since the overall objective is to effectively compute a trajectory that satisfies ϕ, the selection of z_{from} is biased toward automaton states that are close to accepting states.

The region r_{from} is selected uniformly at random from those regions associated with $vertices(\mathcal{T}, z_{from})$, i.e., $\{r : r \in R \wedge |vertices(\mathcal{T}, z_{from}, r)| > 0\}$. The random selection is commonly advocated in motion-planning literature [2,3] as it allows the sampling-based motion planner to expand $\mathcal{T}$ from different regions.

To expand the search toward new automaton states, z_{to} is selected among those automaton states not yet associated with the tree vertices, i.e., $z_{to} \notin \Gamma$. To ensure that the discrete plans are not too long, another criterion is that z_{to} should be a non-rejecting automaton state connected by a single transition from z_{from}, i.e., $z_{to} \in \delta(z_{from})$. Taking these criteria into account, z_{to} is selected uniformly at random from the set $\delta(z_{from}) - \Gamma$. In this way, the selection of $z_{from}, r_{from}, z_{to}$ aims to expand the search from tree vertices associated with automaton states that are close to accepting states and toward new automaton states.

DiscretePlan$(\mathcal{A}, G, z_{from}, r_{from}, z_{to})$ computes a sequence of automaton states and regions that connects $\langle z_{from}, r_{from} \rangle$ to z_{to} (Algo 1:4). The discrete search is conducted over an abstract graph, which is obtained by implicitly combining the automaton $\mathcal{A}$ and the region graph $G = (R, E)$. More specifically, the edges coming out of a vertex $\langle z, r \rangle$ in the abstract graph are computed as

$$\text{edges}(\langle z, r \rangle) = \{\langle \delta(z, \text{DiscreteState}(r')), r' \rangle : (r, r') \in E\}.$$

The cost of an edge $(\langle z', r' \rangle, \langle z'', r'' \rangle)$ is defined as the distance from region r' to r''. A vertex $\langle z, r \rangle$ in the abstract graph is considered as a goal vertex iff $z = z_{to}$.

The discrete plan σ from $\langle z_{from}, r_{from} \rangle$ to z_{to} is computed as the shortest path with probability p and as a random path with probability $1 - p$. While shortest paths provide greedy guides to the sampling-based motion planner, random paths allow for exploration of new regions, which provide alternative routes to prevent the sampling-based motion planner from getting stuck. A randomized version of depth-first-search, which visits the out-going vertices in a random order is used to compute random paths. Shortest paths are computed using Dijkstra's algorithm. Note that sampling-based motion planning can expand $\mathcal{T}$ only from those $\langle z, r \rangle \in \sigma$ for which $|vertices(\mathcal{T}, z, r)| > 0$. For this reason, $\langle z, r \rangle \in \sigma$ is added to σ_{active} if $|vertices(\mathcal{T}, z, r)| > 0$. (Algo 1:5-7).

3.3 Expanding the Search Tree in the Continuous State Space

The objective of sampling-based motion planning is to expand $\mathcal{T}$ by adding several collision-free and dynamically-feasible trajectories using σ_{active} as a guide. Each trajectory is generated by first selecting $\langle z, r \rangle$ from σ_{active} uniformly at random (Algo 1:9). A vertex v is then selected uniformly at random from $vertices(\mathcal{T}, z, r)$ and a control input u is sampled uniformly at random (Algo 1:9). Random selections and sampling are commonly advocated in motion-planning literature [2,3]

and are shown to work well for the problems studied in this work. A trajectory ζ is then obtained by integrating for several steps the motion dynamics of the robot when applying the control input u to $state(v)$. To ensure accuracy, this paper uses Runge-Kutta methods with an adaptive integration step (Algo 1:11). Intermediate states s_{new} along ζ are added as new vertices to $\mathcal{T}$. Each new vertex v_{new} is associated with the corresponding region r_{new} and automaton state z_{new} (Algo 1:12). Recall that r_{new} is computed as the region that contains the position component of s_{new}. The automaton state z_{new} is computed as $\delta(z(v), \textsc{DiscreteState}(s_{new}))$, where v is the parent of v_{new}. The integration of ζ stops if s_{new} is in collision or z_{new} is a rejecting automaton state (Algo 1:13-14). Otherwise, if z_{new} is an accepting automaton state, then $traj(\mathcal{T}, v_{new})$ constitutes a collision-free and dynamically-feasible trajectory that satisfies the LTL specification. In such case, the search terminates successfully (Algo 1:16). The motion planner may augment σ_{active}. In fact, $\langle z_{new}, r_{new} \rangle$ is added to σ_{active} if not already there (Algo 1:17). Such additions enable the motion planner to expand the search toward new automaton states and new regions.

4 Experiments and Results

Experimental validation is provided by using different scenes, LTL specifications, and a snake-like robot model with nonlinear dynamics, as described in Section 2. By increasing the number of links, the robot provides challenging test cases for high-dimensional problems. In the experiments, the number of links is varied as $0, 5, 10, 15$ yielding problems with $5, 10, 15, 20$ DOFs. The running time for each problem instance is obtained as the average of thirty different runs. Experiments are run on an Intel Core 2 Duo machine (CPU: 2.40GHz, RAM: 8GB).

Fig. 2 provides a summary of the results. Comparisons to related work [12,13] show significant computational speedups. The speedups are more pronounced in the case of tasks 2 and 3. Recall that task 1 presents only one possible discrete solution, while tasks 2 and 3 represent polynomially and exponentially many different possibilities. As a result, while the automaton for task 1 has linear size, the automata for tasks 2 and 3 have polynomial and exponential size, respectively. Since related work guides motion planning by using complete discrete plans, i.e., from $\langle z(v_{init}), r(v_{init}) \rangle$ to some $z \in Accept$, as the size of the automaton increases, so does the length of the discrete plan and the computational cost to obtain such discrete plans. In distinction, the proposed approach uses short discrete plans from $\langle z_{from}, r_{from} \rangle$ to z_{to}, where z_{to} is connected by only one automaton transition from z_{from}. Moreover, while all the discrete plans in related work start from $\langle z(v_{init}), r(v_{init}) \rangle$, the proposed approach is biased to start discrete plans from automaton states that are close to accepting states.

As expected, the running time increases with the number of DOFs. As more and more links are added to the robot, it becomes increasingly difficult for the robot to move along the narrow passages of the workspaces and satisfy the LTL specifications. Nevertheless, the proposed approach is able to efficiently solve all

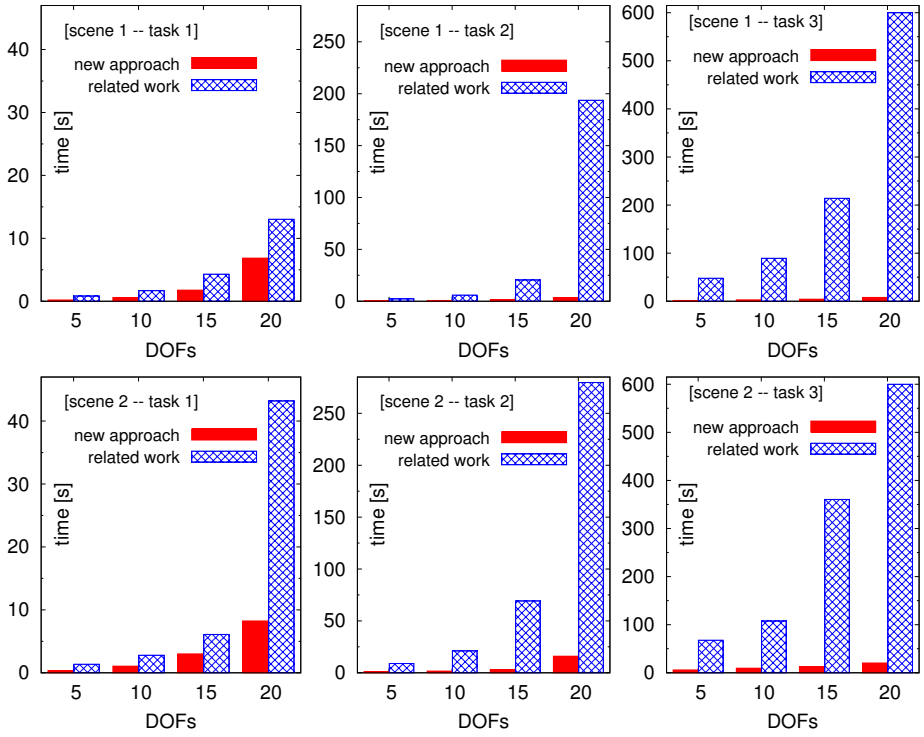

Fig. 2. Comparison of the proposed LTL motion-planning approach to related work [12,13]. Results are shown over the two scenes (Fig. 1) and the three LTL specifications (Section 2) when varying the number of DOFs of the snake-like robot from 5 to 20.

problem instances, even as the number of DOFs is increased to 20, while related work struggles to find solutions.

5 Discussion

Toward the goal of bridging the gap between the discrete and the continuous, this paper showed how to effectively compute collision-free and dynamically-feasible motion trajectories that satisfy task specifications given by LTL. LTL makes it possible to express tasks in terms of propositions, logical connectives, and temporal connectives. The approach coupled the ability of sampling-based motion planning to handle the complexity arising from high-dimensional robotic systems, nonlinear dynamics, and collision avoidance with the ability of discrete planning to take into account discrete specifications. Comparisons to related work show significant computational speedups.

The proposed approach opens up several venues for further research. Improving discrete planning and sampling-based motion planning components and their interplay would further reduce the running time needed to obtain solutions.

Another venue is to obtain robust implementation on physical robotic systems, which requires dealing with uncertainties in the outcome of motion controls and imperfect state information. This could be pursued by combining the proposed approach with recent work on motion planning in belief spaces [17].

References

1. Ghallab, M., Nau, D.S., Traverso, P.: Automated Planning: theory and practice. Morgan Kaufmann (2004)
2. Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L.E., Thrun, S.: Principles of Robot Motion: Theory, Algorithms, and Implementations. MIT Press (2005)
3. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge, MA (2006)
4. Stentz, A.: Optimal and efficient path planning for partially-known environments. In: IEEE Intl. Conf. on Robotics and Automation, San Diego, CA (1994) 3310–3317
5. Tompkins, P., Stentz, A., Wettergreen, D.: Mission-level path planning and re-planning for rover exploration. Robotics and Autonomous Systems **54**(1) (2006) 174 – 183
6. Saffiotti, A., Konolige, K., Ruspini, E.H.: A multivalued logic approach to integrating planning and control. Artificial Intelligence **76**(1-2) (1995) 481–526
7. Kress-Gazit, H., Wongpiromsarn, T., , Topcu, U.: Correct, reactive robot control from abstraction and temporal logic specifications. IEEE Robotics and Automation Magazine **18**(3) (2011) 65–74
8. Ding, X.C., Kloetzer, M., Chen, Y., Belta, C.: Formal methods for automatic deployment of robotic teams. IEEE Robotics and Automation Magazine **18**(3) (2011) 75–86
9. Fainekos, G.E., Girard, A., Kress-Gazit, H., Pappas, G.J.: Temporal logic motion planning for dynamic mobile robots. Automatica **45**(2) (2009) 343–352
10. Plaku, E., Kavraki, L.E., Vardi, M.Y.: Motion planning with dynamics by a synergistic combination of layers of planning. IEEE Trans. on Robotics **26**(3) (2010) 469–482
11. Plaku, E., Hager, G.D.: Sampling-based motion and symbolic action planning with geometric and differential constraints. In: IEEE Intl. Conf. on Robotics and Automation, Anchorage, AK (2010) 5002–5008
12. Plaku, E., Kavraki, L.E., Vardi, M.Y.: Falsification of LTL safety properties in hybrid systems. In: Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Volume 5505 of Lecture Notes in Computer Science. Springer, York, UK (2009) 368–382
13. Bhatia, A., Maly, M., Kavraki, L., Vardi, M.: Motion planning with complex goals. IEEE Robotics Automation Magazine **18** (2011) 55–64
14. Kupferman, O., Vardi, M.: Model checking of safety properties. Formal Methods in System Design **19**(3) (2001) 291–314
15. Sistla, A.: Safety, liveness and fairness in temporal logic. Formal Aspects of Computing **6** (1994) 495–511
16. Laumond, J.: Controllability of a multibody mobile robot. IEEE Trans. on Robotics and Automation **9**(6) (1993) 755–763
17. Van Den Berg, J., Abbeel, P., Goldberg, K.: LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. Intl. J. of Robotics Research **30**(7) (2011) 895–913