Sampling-Based Tree Search with Discrete Abstractions for Motion Planning with Dynamics and Temporal Logic

James McMahon and Erion Plaku

Abstract—This paper presents an efficient approach for planning collision-free, dynamically-feasible, and low-cost motion trajectories that satisfy task specifications given as formulas in a temporal logic, namely Syntactically Co-Safe Linear Temporal Logic (LTL). The planner is geared toward high-dimensional mobile robots with nonlinear dynamics operating in complex environments. The planner incorporates physics-based engines for accurate simulations of rigid-body dynamics.

To obtain computational efficiency and generate low-cost solutions, the planner first imposes a discrete abstraction by combining an automaton representing the LTL formula with a workspace decomposition. The planner then uses the discrete abstraction to induce a partition of a sampling-based motion tree being expanded in the state space into equivalence classes. Each equivalence class captures the progress made toward achieving the temporal logic specifications. Heuristics defined over the abstraction are used to estimate the feasibility of expanding the motion tree from these equivalence classes and reaching an accepting automaton state. Costs are adjusted based on progress made, giving the planner the flexibility to make rapid progress while discovering new ways to expand the search. Comparisons to related work show statistically significant computational speedups and reduced solution costs.

I. INTRODUCTION

A growing number of robotics applications in mobile navigation, manipulation, and game design necessitate reasoning with both discrete actions and continuous motions. In this context, Linear Temporal Logic (LTL) has often been used to express tasks by combining propositions with logical (\(\land\) and, \(\lor\) or, \(\neg\) not) and temporal (\(\bigcirc\) next, \(\diamond\) eventually, \(\bigcup\) until, \(\square\) always) operators. For instance, the task of inspecting several areas in a warehouse can be expressed as

\[
\bigcirc \pi_{A_1} \land \ldots \land \bigcirc \pi_{A_n},
\]

where \(\pi_{A_i}\) denotes the proposition “robot inspected area \(A_j\).” As another example, the task of cleaning areas in a warehouse can be expressed as

\[
\diamond ((\pi_{D_1} \lor \ldots \lor \pi_{D_n}) \implies \bigcirc (\pi_{W_1} \lor \ldots \lor \pi_{W_m})),
\]

where \(\pi_{D_i}\) and \(\pi_{W_j}\) denote the propositions “robot cleaned area \(D_i\)” and “robot is in waste station \(W_j\),” respectively. LTL can also be used to impose partial ordering such as “clean \(D_1, D_2\) before \(D_3, D_4\),” which can be expressed as

\[
\neg \pi_{D_3} \land \neg \pi_{D_4} \lor (\pi_{D_1} \lor \pi_{D_2}) \land \bigcirc (\pi_{D_3} \lor \pi_{D_4}).
\]

Given a robot model, an initial state, a description of the environment, and a specification \(\phi\) in temporal logic, the problem considered in this paper is to plan a collision-free and dynamically-feasible motion trajectory that satisfies \(\phi\). Such problem poses unique computational challenges stemming from (i) robot dynamics and collision avoidance, (ii) temporal constraints, and (iii) intertwined dependencies between motion trajectories and temporal constraints.

Robot dynamics express the relation between input controls and the resulting motions, taking into account the physical properties of the robot. As dynamics for complex systems are often nonlinear and high-dimensional, it makes motion planning quite challenging, especially when the robot has to operate in complex environments [3], [6]. As a result, motion planning has traditionally focused on reachability where the objective is to plan a feasible trajectory to a goal region. In this context, sampling-based motion planning has shown promise (cf. [4], [13]).

Challenges arise at the discrete level, as well, and significant work in AI is devoted to effective task planning (cf. [19]). In this domain, the task is typically broken down into discrete sequences of actions to be executed by the robot.

When considering the combined motion-planning problem with dynamics and LTL specifications, there are additional challenges due to the intertwined dependencies. In fact, a discrete solution that satisfies an LTL formula \(\phi\) may not be feasible due to constraints imposed by obstacles and dynamics. In the other direction, collision-free and dynamically-feasible trajectories may violate constraints imposed by \(\phi\).

Due to the computational challenges of motion planning with dynamics and LTL, research has considered various simplified settings and tradeoffs. A body of work has focused on synthesizing provably-correct controllers from LTL specifications [9], [10], [12], [15]. While this research can handle general LTL, it is limited by the bisimulation assumption, which requires a controller that can ensure dynamically-feasible and collision-free motions between neighboring regions. Ensuring bisimulation is a challenging and open problem for high-dimensional systems with nonlinear dynamics [3], [6], such as those considered in this paper.

The work in [23] creates a sparse random graph (RRG), a variant of RRT [14], to generate paths that satisfy LTL specifications. Similar to the controller synthesis, this work requires a controller that can guarantee steering between any two states, i.e., solve two-value boundary problems [2], [8]. The work in [7] uses a variant of RRG for optimal motion planning with deterministic \(\mu\)-calculus. Due to rewiring of tree branches, this work requires a controller to provide optimal steering between any two states.

This paper proposes an effective approach for motion planning with dynamics and LTL specifications based on a discrete abstraction and sampling-based motion planning.

The authors are with the Dept. of Electrical Engineering and Computer Science, Catholic University of America, Washington DC 20064. J. McMahon is also with the U.S. Naval Research Laboratory, Washington, DC 20375.
The planner is geared toward high-dimensional mobile robots with nonlinear dynamics. The planner incorporates physics-based engines, e.g., ODE, Bullet, PhysX, to model rigid-body dynamics, friction, and nonflat terrains.

Motivated by related work towards a synergistic combination of layers of planning (Syclop and LTLSyclop), this planner introduces several improvements. As in LTLSyclop, the discrete abstraction is obtained by combining an automaton representing the LTL formula $\phi$ with a workspace decomposition. LTLSyclop uses accepting paths in the abstraction to guide the sampling-based expansion of a motion tree in the state space. Due to constraints imposed by obstacles and dynamics, however, LTLSyclop often wastes computational time before realizing that the current guide should be abandoned due to lack of progress.

In this planner, improvements provide better interplay between planning layers, reduced running times and solution costs, incorporation of physics-based engines for accurate simulations of rigid-body dynamics, and simplified implementation. The technical contribution has several key aspects, such as (i) partition of a sampling-based motion tree into equivalence classes as induced by an abstraction; (ii) heuristic costs to estimate the feasibility of reaching an accepting automaton state from each equivalence class; and (iii) multi-objective selection of equivalence classes based on heuristic costs, selection penalties, and trajectory costs in order to promote both computational efficiency and generation of low-cost solutions.

In contrast to LTLSyclop, a key aspect of the planner is the use of the abstraction to induce a partition of the motion tree into equivalence classes. Each equivalence class captures the progress made toward an accepting automaton state. Heuristic costs based on shortest abstract paths evaluate the progress made toward an accepting automaton state from each equivalence class, and (iii) multi-objective selection of equivalence classes based on heuristic costs, selection penalties, and trajectory costs in order to promote both computational efficiency and generation of low-cost solutions.

In an incremental fashion, the planner selects an equivalence class. A multi-objective procedure then expands the motion tree from vertices associated with the selected equivalence class. A multi-objective procedure is employed to select equivalence classes based on heuristic costs, selection penalties, and actual trajectory costs in order to promote expansion toward accepting automaton states while also discovering new ways to expand the search. The proposed approach also incorporates physics-based engines for accurate simulations of rigid-body dynamics, and it is easier to implement as it does not require the complex coverage and free volume estimates used by LTLSyclop.

The efficiency of the planner is shown for nonlinear, high-dimensional, robot models operating in complex environments, and for a variety of temporal logic specifications expressing sequencing, coverage, counting, and partial-ordering tasks. Comparisons to LTLSyclop show statistically significant computational speedups and reduced solution costs. The impact of the workspace decomposition on the overall performance of the planner is also evaluated.

We also note that this paper, as in LTLSyclop [1], [17], considers syntactically co-safe LTL formulas, which use $\neg, \lor, \land, \bigcirc, \Diamond, \Box, \mathcal{U}$ but not $\Box$ when written in positive-normal form ($\neg$ appears only in front of atomic propositions) [21]. Syntactically co-safe LTL can be interpreted over finite discrete traces, which is necessary when planning finite motion trajectories, as it is the case in sampling-based motion planning. By considering co-safe LTL, the proposed planner is able to account for high-dimensional systems, nonlinear dynamics, physics-based interactions, and complex environments. This provides a tradeoff with controller-synthesis and deterministic $\mu$-calculus approaches [7], [9], [10], [12], [23] which can handle general LTL but require a controller which can guarantee bisimulation or steering between states.

II. MATHEMATICAL FRAMEWORK

This section defines motions in the continuous space, LTL representation and interpretation over continuous motion trajectories, and the problem statement.

1) Robot Models and Motion Trajectories: A state $s \in S$, where $S$ is the state space, defines position, orientation, linear and angular velocities, and other components that change as a result of a motion. A state is valid if the robot is not in collision and state values are within bounds, as determined by a function $\texttt{valid} : S \rightarrow \{\top, \bot\}$. A control $u \in U$, where $U$ is the control space, defines external inputs to control the robot. Robot dynamics determine the motion trajectory $\zeta : [0, T] \rightarrow S$ resulting from applying the control function $\hat{u} : [0, T] \rightarrow U$, starting from a state $s \in S$. This relation is encapsulated by a function $s_{\text{new}} \leftarrow \texttt{motion}(s, u, \Delta t)$, where the new state $s_{\text{new}}$ is obtained by applying $u$ to $s$ for a short time duration $\Delta t$. When dynamics are defined by differential equations $f : S \times U \rightarrow \dot{S}$, motion can be implemented, for example, as a Runge-Kutta integrator. More accurate simulations of rigid body dynamics are obtained by using physics-based engines such as Bullet [5] and ODE [22]. Section IV-A.1 provides examples of robot models.

2) LTL Syntax and Semantics: Let $\Pi$ denote the set of propositions. Every $\pi_i \in \Pi$ is a formula. If $\phi$ and $\psi$ are formulas, then so are $\neg \phi$, $\phi \land \psi$, $\phi \lor \psi$, $\bigcirc \phi$, $\Diamond \phi$, $\pi_i \land \phi$, $\pi_i \lor \phi$. Let $\Sigma^\omega$ denote the set of infinite traces over $\Sigma = 2^{\Pi}$. Let $\sigma \in \Sigma^\omega$ and let $\pi^i$ denote the $i$-th postfix of $\sigma$. The notation $\sigma \models \phi$ indicates that $\sigma$ satisfies $\phi$ and is defined as $\sigma \models \pi$ if $\pi \in \Pi$ and $\sigma \models \pi_0$; $\sigma \models \neg \phi$ if $\sigma \not\models \phi$; $\sigma \models \phi \land \psi$ if $\sigma \models \phi$ and $\sigma \models \psi$; $\sigma \models \bigcirc \phi$ if $\sigma^1 \models \phi$; $\sigma \models \phi \lor \psi$ if $\exists k \geq 0: \sigma^k \models \psi$ and $\forall 0 \leq i < k : \sigma^i \models \phi$. Let $\Sigma = \pi \land \neg \pi$; $\bigcirc \phi$ is $\phi \land \bigcirc \phi$; $\Diamond \phi$ is $\Box \phi \lor \neg \Box \phi$. Details can be found in [11].

3) Syntactically Co-Safe LTL: Under some restrictions, LTL formulas can be interpreted over finite traces. In particular, an LTL formula $\phi$ is co-safe if any infinite trace that satisfies $\phi$ has a finite prefix, which, when concatenated with any infinite trace, it still satisfies $\phi$ [21]. Such prefixes are referred to as good prefixes of $\phi$. There are also syntactical restrictions that lead to co-safety: an LTL formula containing

\footnote{Syclop refers to the synergistic framework used for motion planning with dynamics [18]. LTLSyclop refers to the framework when it also incorporates syntactically co-safe LTL. LTLSyclop was introduced in [16] in the context of hybrid systems, and was adapted to motion planning in [1], [17].}
only \( \neg, \lor, \land, \bigcirc, \diamond, \cup \) when written in positive-normal form (\( \neg \) appears only in front of atomic propositions), is co-safe.

4) Automata Representation: The set of all good prefixes characterizing a syntactically co-safe LTL formula \( \phi \) can be represented as a nondeterministic finite automaton (NFA) [11]. Fig. 1 shows an example. An NFA can be determinized to obtain a DFA, which is more amenable to computation.

\[ \begin{align*}
\pi_1 \lor \pi_2 \\
(\pi_1 \land \pi_3) \lor (\pi_2 \land \pi_3) \\
\top \\
\top \\
\top
\end{align*} \]

Fig. 1. NFA representing all the finite good prefixes of the syntactically co-safe LTL formula \( \phi = \top((\pi_1 \lor \pi_2) \land \top(\pi_3)) \).

A DFA is a tuple \( \mathcal{A} = (Z, \Sigma, \delta, z_{init}, \text{Accept}) \), where \( Z \) is a finite set of states, \( \Sigma = 2^\Pi \) is the input alphabet, \( \delta: Z \times \Sigma \rightarrow Z \) is the transition function, \( z_{init} \in Z \) is the initial state, and \( \text{Accept} \subseteq Z \) is the set of accepting states. The extended transition function \( \hat{\delta}: Z \times \Sigma^* \rightarrow Z \) is defined in the usual way, i.e., \( \forall z \in Z, w \in \Sigma^*, \alpha \in \Sigma: \)

\[ \hat{\delta}(z, \epsilon) = z \text{ and } \hat{\delta}(z, \alpha w) = \hat{\delta}(\hat{\delta}(z, w), \alpha). \]

\( \mathcal{A} \) accepts \( \sigma \in \Sigma^* \) iff \( \hat{\delta}(z_{init}, \sigma) \in \text{Accept} \). As a result, \( \sigma \models \phi \) when \( \sigma \) is accepted by the DFA \( \mathcal{A} \) representing \( \phi \).

5) LTL Semantics over Motion Trajectories: As in related work [1], [10], [17], a workspace \( W \) contains several obstacles \( O_1, \ldots, O_t \) and several regions of interest \( P_1, \ldots, P_n \), corresponding to propositions \( \pi_1, \ldots, \pi_n \). Fig. 3 provides an example. A proposition \( \pi_0 \) is associated with the unoccupied workspace area \( P_0 = (W \setminus O) \cup \bigcup_{i=1}^n P_i \), where \( O = \bigcup_{i=1}^t O_i \).

A function \( \text{PROP}: W \rightarrow \Pi \) maps each point \( p \in W \) to the proposition in \( \Pi = \{\pi_0, \pi_1, \ldots, \pi_n\} \) that holds there, i.e.,

\[ \text{PROP}(p) = \pi_i \implies p \in P_i. \]

For convenience, the notation \( \text{PROP}(s) \) is used as shorthand for \( \text{PROP}(\text{position}(s)) \), where \( \text{position}(s) \) denotes the position component of the continuous state \( s \in S \).

The discrete trace of a trajectory \( \zeta: [0, T] \rightarrow S \), denoted by \( \text{TRACE}(\zeta) \), is defined by the sequence of regions \( P_0, P_1, \ldots, P_n \) reached by \( \zeta \). A region \( P_i \) is said to be reached by \( \zeta \) if \( \text{PROP}(\zeta(t)) = \pi_i \) for some \( 0 \leq t \leq |\zeta| \), where \( |\zeta| \) denotes the time duration of \( \zeta \). Fig. 2 shows an example of a trajectory and its trace. As a result of this mapping, \( \zeta \) is collision free and satisfies the syntactically co-safe LTL specification \( \phi \), i.e., \( \text{TRACE}(\zeta) \models \phi \).

III. Method

Pseudocode is given in Algo. 1. The main components of the planner are described below.

A. Discrete Abstraction

The planner uses a simplified abstract version of the problem which ignores the dynamics and treats the robot as a point mass to guide the overall search. The discrete abstraction is obtained by combining the automaton \( \mathcal{A} \) representing the LTL formula \( \phi \) with a workspace decomposition.

1) Workspace Decomposition: The unoccupied workspace area, \( P_0 = (W \setminus O) \cup \bigcup_{i=1}^n P_i \), is decomposed into a number of nonoverlapping (except at the boundary) regions \( W_1, \ldots, W_t \). The physical adjacency among these and the regions \( P_1, \ldots, P_n \) associated with the propositions of interest \( \pi_1, \ldots, \pi_n \) is expressed as a graph \( D = (R, E) \), where \( R = \{W_1, \ldots, W_t, P_1, \ldots, P_n\} \) and \( E = \{r_i, r_j \in R \mid r_i, r_j \in \text{boundary} \} \).

Each \( W_1, \ldots, W_t \) is labeled with \( \pi_0 \), while a region \( P_i \) is labeled with \( \pi_i \). A function \( \text{REGION}: W \rightarrow R \) locates the region that contains the input point, returning an error if the point is inside an obstacle. For convenience, \( \text{REGION}(s) \) with \( s \in S \) is used as shorthand for \( \text{REGION}(\text{position}(s)) \).

As in related work [1], [17], this paper uses triangulations for the workspace decomposition. Triangulations are preferred over grid-based decompositions as they preserve the geometry of the obstacles, i.e., triangles do not intersect.
obstacles. To ensure that the decomposition of $P_0$ does not overlap with obstacles and regions of interest, the obstacles and regions $P_1, \ldots, P_n$ are treated as holes inside the workspace bounding box. The Triangle package [20] is used for the computation of the triangulation. Fig. 3 provides an illustration. The function \texttt{region} runs in polylogarithmic time with respect to the number of triangles.

2) \textit{Product of the LTL Automaton with the Decomposition:} Using standard tools from model checking [11], the LTL formula $\phi$ is converted into a DFA $A$. The discrete abstraction is then defined as $A \times D$. An abstract state $\langle z, r \rangle$ is composed of the automaton state $z \in Z$ and decomposition region $r \in R$. The edges from $\langle z, r \rangle$ are defined as:

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

3) \textit{Heuristic Costs:} Consider a path $\Xi$ in $A \times D$ from $\langle z, r \rangle$ to an accepting automaton state. If the robot can follow $\Xi$, then the resulting trajectory will satisfy $\phi$. Of course, due to constraints imposed by dynamics and obstacles, it may be difficult or even impossible to follow certain parts of $\Xi$. Nevertheless, $\Xi$ can serve as a heuristic to guide the planner as it expands a motion tree in the continuous state space $S$. For this purpose, the heuristic value $h(\langle z, r \rangle)$ is defined as the length of the shortest path in $A \times D$ from $\langle z, r \rangle$ to an accepting automaton state, where the weight of an abstract edge $\langle (z_1, r_1), (z_2, r_2) \rangle$ is defined as $||\text{centroid}(r_1) - \text{centroid}(r_2)||$. Dijkstra’s single-source shortest-path algorithm is used to compute the heuristic costs in a single call. To do this, edges are added from each abstract state $\langle z, r \rangle$ with $z \in \text{Accept}$ to a special abstract state $\langle z_{\text{spec}}, r_{\text{spec}} \rangle$. Dijkstra’s algorithm is then run backwards using $\langle z_{\text{spec}}, r_{\text{spec}} \rangle$ as the source.

To provide a uniform range for different problems, the heuristic costs are normalized in the range $[0, 1]$, i.e.,

$$h(\langle z, r \rangle) = 1 - \frac{(1 - \epsilon)h(\langle z, r \rangle)}{\max_{(z', r') \in A \times D} h(\langle z', r' \rangle)}.$$  

As explained in Section III-C, a small positive value $\epsilon$ rather than zero is chosen for the abstract state with the maximum heuristic value in order to have a nonzero probability of expanding the motion tree from each $\langle z, r \rangle$.

B. Partition of the Motion Tree into Equivalence Classes According to Abstract States

The discrete abstraction is used to induce a partition of the motion tree $T$ into equivalence classes. The motion tree $T$ is represented as a directed acyclic graph whose edges are associated with collision-free and dynamically-feasible trajectories. A vertex in $T$ is associated with the initial state $s_{\text{init}}$ and is referred to as the root of $T$. The details of how $T$ is expanded are provided later in Section III-C.2. For now, let $\text{traj}(T, v)$ denote the trajectory from the root of $T$ to the vertex $v$. Informally, $v_i$ and $v_j$ belong to the same equivalence class iff $\text{traj}(T, v_i)$ and $\text{traj}(T, v_j)$ end up in the same automaton state and same decomposition region, and, thus, provide the same discrete information.

More formally, let $s\text{state}(v), \text{region}(v), z\text{state}(v)$ denote the continuous state, decomposition region, and automaton state associated with the vertex $v$, respectively. Then,

$$s\text{state}(v) = \zeta(\langle z \rangle), \text{ where } \zeta = \text{traj}(T, v) \text{ region}(v) = \text{REGION}(s\text{state}(v)) \text{ zstate}(v) = \delta(z_{\text{init}}, \text{TRACE} \text{(traj}(T, v))))$$  

An abstract state $\langle z, r \rangle$ then induces an equivalence class $\Gamma(z, r)$ consisting of all the vertices that map to $\langle z, r \rangle$, i.e.,

$$\Gamma(z, r) = \{ v : v \in T \land z\text{state}(v) = z \land \text{region}(v) = r \}.$$  

The motion tree $T$ is then partitioned into a number of equivalence classes, i.e.,

$$\Gamma = \{ \Gamma(z, r) : |\Gamma(z, r)| > 0 \}.$$  

The partition, as described next, is used to effectively guide the expansion of $T$ in the continuous state space $S$.

C. Overall Guided Expansion of the Motion Tree

A motion tree $T$ is expanded by adding collision-free and dynamically-feasible trajectories as tree branches. Initially, $T$ contains $s_{\text{init}}$ and $\Gamma$ contains $\Gamma(z_{\text{init}}, \text{REGION}(s_{\text{init}}))$. The overall planner is driven by the following functions:

1) \textbf{SELECTEQUIVALENCECLASS($\Gamma$)} selects an equivalence class $\Gamma(z, r) \in \Gamma$ from which to expand $T$; and

2) \textbf{EXPANDMOTIONTREE($T$, $\Gamma(z, r)$)} expands $T$ by adding a new collision-free and dynamically-feasible trajectory from a vertex associated with $\Gamma(z, r)$.

The planner proceeds iteratively by invoking in succession \textbf{SELECTEQUIVALENCECLASS} and \textbf{EXPANDMOTIONTREE} until a solution is found or an upper bound on time is reached.

1) \textit{Selecting an Equivalence Class:} The selection procedure seeks to promote both rapid expansions toward an accepting automaton state and generation of a low-cost solution trajectory. For the first objective, a set of candidates $\Gamma_{\text{cands}}$ is computed based on the heuristic costs. For the second objective, the candidate in $\Gamma_{\text{cands}}$ with the lowest average trajectory cost is selected for expansion. This two-step procedure, motivated by strategies in multi-objective stochastic search, yielded better results than additive combinations of the heuristic and average trajectory costs.

For the computation of $\Gamma_{\text{cands}}$, a weight is defined based on the normalized heuristic cost as

$$w(\Gamma(z, r)) = \left( \frac{h(\langle z, r \rangle)}{\max_{(z', r') \in A \times D} h(\langle z', r' \rangle)} \right)^\alpha \beta^{n\text{sel}(\Gamma(z, r))},$$  

where $n\text{sel}(\Gamma(z, r))$ is the number of times $\Gamma(z, r)$ has been selected for expansion, $\alpha \geq 1$, and $0 < \beta < 1$. Note that $w(\Gamma(z, r))$ is high when $\langle z, r \rangle$ is close to an accepting automaton state according to paths in the abstraction $A \times D$. The parameter $\alpha$ serves to tune the strength of the heuristic, while $\beta$ serves as a penalty factor to avoid overexploration of $\Gamma(z, r)$ or become stuck when expansions from $\Gamma(z, r)$ are infeasible due to obstacles and dynamics.

The set of candidates, $\Gamma_{\text{cands}}$, includes not only the equivalence class with the maximum weight but also some other equivalence classes with large weights, where the likelihood of inclusion is high when the weight is close to the maximum weight. More specifically, $\Gamma(z, r)$ is included in $\Gamma_{\text{cands}}$ iff

$$w(\Gamma(z, r)) \geq \kappa w(\Gamma_{\text{max}}) \text{ and } \text{RAND} \leq \exp(-\lambda + \frac{w(\Gamma(z, r))}{w(\Gamma_{\text{max}})}),$$  

where $\Gamma_{\text{max}} = \arg\max_{\Gamma(z', r') \in \Gamma} w(\Gamma(z', r'))$, $0 < \kappa \leq 1$, $\lambda \geq 1$, and RAND generates a random number in the
interval (0, 1). Note that $\kappa$ serves as a strict bound to ensure that only equivalence classes with sufficiently high weights are considered. Similar to the temperature in a Metropolis criterion, $\lambda$ serves to adjust the likelihood of inclusion.

To promote the generation of a low-cost trajectory, the motion tree $T$ is then expanded from the equivalence class in $\Gamma_{ends}$ with the minimum average trajectory cost, i.e.,

$$\Gamma(z,r) = \arg\min_{\Gamma(z',r') \in \Gamma_{ends}} \frac{1}{|v_{\text{new}}|} \sum_{v \in \Gamma(z',r')} \cos(\text{TRAJ}(T, v)).$$

As an implementation note, the average trajectory cost is updated each time a vertex is added to $\Gamma(z,r)$.

2) Expanding the Motion Tree: The motion tree $T$ is expanded from $\Gamma(z,r)$ by first selecting a vertex from $\Gamma(z,r)$ and then extending a collision-free and dynamically-feasible trajectory starting from $s\text{state}(v)$ (Algo. 1:b).

Let $z\text{states}(T) = \bigcup_{v \in T} \{z\text{state}(v)\}$ denote the automaton states that have been reached by $T$. The selection of $v$ is a three-step process that seeks to promote expansions toward new automaton states. First, a target proposition $\pi_i$ is selected uniformly at random from $\{\pi_j : \pi_j \in \Pi \land \delta(z, \pi_j) \notin z\text{states}(T)\}$. Next, a target point $p$ is sampled uniformly at random inside one of the regions along the shortest path from $v$ to $\Pi_i$ in the decomposition graph $D = (R, E)$. These shortest paths are computed only once during initialization by running Dijkstra’s algorithm using each $\Pi_i$ as the source.²

The motion tree $T$ is then expanded from the vertex $v$ in $\Gamma(z,r)$ that is closest to $p$, where the distance from $v$ to $p$ is computed as $||\text{position}(s\text{state}(v)), p||$. In this way, the selection strategy promotes expansions from vertices in $\Gamma(z,r)$ that could lead toward new automaton states. If all the automaton states that have an incoming transition from $z$ appear in $z\text{states}(T)$, then the target point $p$ is selected uniformly at random to promote expansions in new directions.

The control input $u$ that is applied to $s\text{state}(v)$ is sampled uniformly at random. This is a common strategy in sampling-based motion planning as random controls promote expansions along different directions [4, 13]. When available, PID controllers that aim to steer $s\text{state}(v)$ toward the target point $p$ can also be used. The planner does not place any requirements on these controllers, so exact steering is not needed. A collision-free and dynamically-feasible trajectory is obtained by applying $u$ starting from $s\text{state}(v)$ and running the motion simulator for several time steps. The new state $s_{\text{new}}$, obtained after each simulation step, is checked for collisions. If a collision is found, the expansion terminates. Otherwise, a new vertex $v_{\text{new}}$ is added to $T$ with $s_{\text{new}}$ as its associated continuous state and $v$ as its parent (Algo. 1:c1-c2). At this time, $z\text{state}(v_{\text{new}})$ and $\text{region}(v_{\text{new}})$ are also computed. Note in particular that $z\text{state}(v_{\text{new}})$ is computed efficiently as $\delta(\text{state}(v), \text{PROP}(s_{\text{new}}))$ (Algo. 1:c3).

The equivalence classes are also updated accordingly. As an implementation note, $\Gamma$ is maintained as a hash map indexed by $\langle z, r \rangle$. If $v_{\text{new}}$ leads to a new abstract state, then the corresponding equivalence class is created and added to $\Gamma$ (Algo. 1:c6). In this way, the planner has the flexibility to expand $T$ from new equivalence classes in future iterations.

IV. EXPERIMENTS AND RESULTS

Experimental validation is provided using nonlinear, high-dimensional, robot models operating in complex environments, required to perform various tasks written as syntactically co-safe LTL formulas. Comparisons to related work show statistically significant improvements both in terms of computational time and solution cost. The impact of the decomposition granularity is also evaluated.

A. Experimental Setup

1) Robot Models and Scenes: A physics-based vehicle model and a snake-like robot model are used for the experiments. The vehicle, based on the model available in Bullet [5], is controlled by setting the engine force and changing the steering angle. The vehicle state consists of position, orientation, linear velocity, steering angle, and angular velocity. The vehicle model provides a challenging system as it is high-dimensional and has nonlinear dynamics. Moreover, as shown in Fig. 3, the vehicle is required to operate in complex environments, moving through narrow passages and over bumpy terrains in order to reach regions of interest in accordance with the LTL specifications.

The snake-like robot is a high-dimensional model with nonlinear dynamics obtained by attaching several trailers to a car that pulls them as it moves. The differential equations are as follows (adapted from [13, pp. 731]):

\[
\begin{align*}
\dot{\theta} &= \frac{1}{I_p} (F_s - \tau - \theta), \\
\dot{\phi} &= \frac{1}{I_p} (\tau - \theta), \\
\dot{x} &= v \cos(\phi), \\
\dot{y} &= v \sin(\phi), \\
\dot{\theta} &= \omega, \\
\dot{\omega} &= \frac{1}{I_p} (\tau - \theta).
\end{align*}
\]

Fig. 3. Robot models and scenes. Regions 1, ..., 8 correspond to propositions $\pi_1, \ldots, \pi_8$. Scene 2 also shows a triangulation. Figures viewed better on-screen or when printed in color. Video attachment shows solutions obtained by the planner.
\[ \dot{x} = v \cos(\theta); \dot{y} = v \sin(\theta); \dot{\theta} = \frac{v}{L} \tan(\psi); \dot{\psi} = a; \dot{\psi} = \omega; \phi_1 = \frac{1}{2}(\sin(\theta_{i-1}) - \sin(\theta_i)) \prod_{j=1}^{i-1} \cos(\theta_{i-1} - \theta_j). \]

The robot state consists of \( (x, y, \theta_0, v, \psi, \theta_1, \ldots, \theta_N) \), where \( x, y, \theta_0, v, \psi \) denote the position, orientation, velocity, and steering angle; \( \theta_i \) is the orientation of the \( i \)-th trailer; \( L \) is the body length of each trailer; and \( N \) is the number of trailers (set to 15 in the experiments, so that \( S \) has 20 dimensions). The robot is controlled by setting \( a \) (acceleration) and \( \omega \) (rotational velocity of \( \psi \)). The hitch length is set to a small value (\( d = 0.01 \)) so that the model resembles a snake.

2) LTL Formulas: Experiments are conducted using LTL formulas to express strict sequencing, sequencing, coverage, partial ordering, and counting tasks, as described below.

In the strict sequencing task, \( \phi^n_{\text{strict-seq}} \) the robot is required to visit the regions of interest \( P_1, \ldots, P_n \) in order passing only through \( P_0 \) in between, i.e.,
\[ \phi^n_{\text{strict-seq}} = \bigcirc(\pi_1 \land \bigcirc(\pi_0 \land \bigcirc(\pi_2 \land \ldots \land \bigcirc \pi_n))). \]

The more relaxed version of the sequencing task, \( \phi^n_{\text{seq}} \), only requires that there are times \( t_1 < t_2 < \ldots < t_n \) such that the robot is in \( P_i \) at time \( t_i \), while allowing the robot to visit any region \( P_1, \ldots, P_n \), in between, i.e.,
\[ \phi^n_{\text{seq}} = \bigcirc(\pi_1 \land \bigcirc(\pi_2 \land \ldots \land \bigcirc(\pi_1 \land \ldots \land \pi_n))). \]

In the coverage task, \( \phi^n_{\text{cov}} \), the order does not matter as long as the robot visits each \( P_1, \ldots, P_n \), i.e.,
\[ \phi^n_{\text{cov}} = \bigcirc(\pi_1 \land \ldots \land \bigcirc \pi_n). \]

In the partial-order task, \( \phi^n_{\text{partial-seq}} \) the robot is required to visit the regions of interest \( P_1, \ldots, P_n \) in order passing only through \( P_0 \) in between, i.e.,
\[ \phi^n_{\text{partial-seq}} = \bigcirc(\pi_1 \land \bigcirc(\pi_0 \land \bigcirc(\pi_2 \land \ldots \land \bigcirc \pi_n))). \]

Experiments were run on an Intel Core i7 (CPU: 1.90GHz, RAM: 4GB) using Ubuntu 13.04 and GNU g++4.7.3.

B. Results

Results comparing LTLsyclop and the proposed planner both in terms of running time and solution costs are shown in Fig. 4 and Fig. 5, respectively. Results on the impact of the decomposition are shown in Fig. 6.

1) Results on Running Time: Fig. 4 shows the results for each LTL formula type, \( \phi^n_{\text{strict-seq}} \phi^n_{\text{seq}} \phi^n_{\text{cov}} \phi^n_{\text{halt}} \phi^n_{\text{count}} \), as a function of \( n \). In addition to the mean and standard deviation, Fig. 4 shows the effect sizes in order to provide a statistical measure on the magnitude of the improvements obtained by the new planner, measured as
\[ \text{Hedges' } \gamma = (\mu_{\text{time,LTLsyclop}} - \mu_{\text{time,new}})/\sigma_p, \]
where \( \mu \) is the sample mean, \( \sigma_p \) is the pooled standard deviation, and new refers to this work. A common convention in statistics is to categorize effect sizes into “small” (less than 0.2), “medium” (around 0.5), and “large” (above 0.8). As shown in Fig. 4, the effect sizes of the approach over LTLsyclop are all statistically large.

Our understanding is that the efficiency of the proposed planner comes from the discrete abstraction \( A \times D \) and the heuristics it uses to guide the expansion of the motion tree \( T \) toward an accepting automaton state. Moreover, by applying a penalty after each selection of \( z_i(r_i) \), the planner reduces the likelihood of overexploration or becoming stuck attempting to expand \( T \) from an infeasible \( z_i(r_i) \). The planner also avoids the dependence on long guides as it is the case with LTLsyclop which tries to follow a discrete path from \( z_{\text{init}}, r_{\text{init}} \) to an accepting automaton state. Due to constraints imposed by obstacles and dynamics, LTLsyclop often wastes computational time before realizing that the current guide should be abandoned due to lack of progress.

2) Results on Solution Cost: Fig 5 shows the results on solution cost, measured as the distance traveled by the robot. LTL-Syclop, due to the shortest-path search used to compute guides, has been shown to produce low-cost solutions. Results show that the proposed planner offers statistically significant reductions in solution cost over LTLsyclop.

3) Impact of Decomposition Granularity: Fig. 6 shows results when varying the decomposition granularity. Different triangulations are obtained by adjusting the average triangle area. The results indicate that the planner remains efficient for a variety of decompositions.

V. DISCUSSION

This paper presented an efficient algorithm for planning collision-free, dynamically-feasible, and low-cost motion
Fig. 4. Results on running time. (top, middle) Mean and standard deviation for scenes 1 and 2. (bottom) Statistical effect size measured as Hedges’ \( g = (\bar{\mu}_{\text{new}} - \bar{\mu}_{\text{LTLSyclop}}) / \sigma_p \), where \( \mu \) and \( \sigma_p \) denote the sample mean and the pooled standard deviation, respectively. Effect sizes less than or equal to 0.2 are typically deemed as “small,” around 0.5 as “medium,” and above 0.8 as large.

Fig. 5. Results on solution cost. (top, middle) Mean and standard deviation for scenes 1 and 2. (bottom) Statistical effect size measured as Hedges’ \( g = (\bar{\mu}_{\text{cost,LTLSyclop}} - \bar{\mu}_{\text{cost,new}}) / \sigma_p \), where \( \mu \) and \( \sigma_p \) denote the sample mean and the pooled standard deviation, respectively. Effect sizes less than or equal to 0.2 are typically deemed as “small,” around 0.5 as “medium,” and above 0.8 as large.
trection that satisfy co-safe LTL formulas. Some key features of the planner include (1) a discrete abstraction obtained by combining the automaton representing the LTL formula with a workspace decomposition; (2) partition of a sampling-based motion tree into equivalence classes as induced by the abstraction; (3) heuristic costs to estimate the feasibility of reaching an accepting automaton state from each equivalence class; (4) multi-objective selection of equivalence classes based on heuristic costs, selection penalties, and trajectory costs in order to promote both computational efficiency and generation of low-cost solutions. The planner is geared toward high-dimensional mobile robots with nonlinear dynamics. It also incorporates physics-based engines for accurate simulations of rigid-body dynamics.

Comparisons to related work showed statistically significant computational speedups and reduced solution costs.

In future work, machine learning could be used to automatically adjust the cost parameters based on progress made during the tree expansion. Abstraction refinement is another venue which could prove beneficial to identify and refine regions in the decomposition where not much progress is being made. We will also pursue applications of the approach in inspection, search-and-rescue missions, and game design.

ACKNOWLEDGMENTS

The work of J. McMahon is supported by the Office of Naval Research, code 32.

REFERENCES


