# Path Planning with Probabilistic Roadmaps and Co-Safe Linear Temporal Logic

Erion Plaku

***Abstract*— Linear Temporal Logic makes it possible to express tasks in terms of propositions, logical connectives, and temporal connectives. This paper shows how to incorporate a subclass of LTL, namely co-safe LTL, into Probabilistic RoadMap (PRM) path planners. PRMs provide an important class of approaches which have been shown to work well for high-dimensional configuration spaces. The proposed Temporal-PRM approach combines the roadmap with a finite automaton representing the co-safe LTL formula $\phi$ and conducts the search over the combined graph. As a result, roadmap connections are reused when needed to find paths that satisfy $\phi$. Experimental validation is provided in simulation by using different scenes, co-safe LTL specifications, a snake-like robot model with numerous degrees-of-freedom, and different sampling strategies.**

## I. Introduction

The objective in path planning has traditionally been to compute a collision-free path from an initial to a goal configuration. Recently, there has been a growing interest in incorporating more sophisticated tasks directly into motion planning [4], [7]–[10], [16], [17], [19]. In this context, LTL has often been used as the discrete logic in which to express the task. LTL makes it possible to express tasks in terms of propositions, logical ($\wedge$ and, $\vee$ or, $\neg$ not), and temporal connectives ($\bigcirc$ Next, $\Diamond$ Eventually, $\Box$ Always, $\cup$ Until). As an illustration, the task of inspecting areas of interest while avoiding collisions can be expressed as

$$\Diamond\pi_{A_1} \wedge \ldots \wedge \Diamond\pi_{A_n} \wedge \Box\neg\pi_{O_1} \wedge \ldots \wedge \Box\neg\pi_{O_m}, \qquad (1)$$

where $\pi_{A_i}$ and $\pi_{O_j}$ denote the propositions "robot inspected area $A_i$," and "robot is in collision with obstacle $O_j$," respectively. As another example, the task of inspecting area $A_1$ before $A_2$ can be expressed as

$$(\neg\pi_{A_2}) \cup (\pi_{A_1} \wedge \Diamond\pi_{A_2}).$$

The problem of planning motions that satisfy an LTL specification $\phi$ is often approached by first using model checking to compute a sequence of propositional assignments $\sigma = \tau_1, \tau_2, \ldots$ that satisfies $\phi$. In a second stage, controllers are used to enable the robot to satisfy the propositions in the order specified by $\sigma$. In this context, discrete logic is used in [2] to combine a set of behavior schemas which use controllers to link motion to abstract actions. This idea has been revisited more recently in [12] 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 model checking to determine the sequence of triangles a point robot needs to visit and relies on a controller to drive the robot between adjacent triangles to carry out the LTL specification.

A limitation of these approaches that treat discrete planning and motion control separately is that it is generally not known in advance which propositional assignments are actually feasible in the continuous world. The LTL specification could present exponentially many alternative discrete solutions, as it is the case in Eq. 1 which leaves it to the planner to determine a feasible order of inspecting the areas. As a result of geometric constraints imposed by obstacle avoidance and the geometric shape of the robot it may be difficult or impossible to carry out certain propositional assignments $\tau_i$. Such constraints limit the applicability of these approaches to specific systems and to specific discrete actions for which controllers are available.

To handle discrete and continuous abstractions, recent work by the author has proposed a two-layered approach that couples the ability of sampling-based motion planning to handle the complexity arising from high-dimensional robotic systems and collision avoidance with the ability of discrete search to take into account LTL specifications [16], [18], [19]. While discrete planning guides sampling-based motion planning, the latter feeds back information to further refine the guide and advance the search toward a solution that satisfies the LTL specification. Other two-layered approaches that utilize discrete search and sampling-based motion planning have also been developed for robot climbing [6], multi-modal motion planning [10], and manipulation planning [15], [21]. These other approaches, however, do not take into account discrete specifications given by LTL.

This paper builds upon the success of combining LTL with sampling-based motion planning [4], [11], [16], [18], [19]. While previous work in this area has relied on growing a tree in the state space of the robot, this paper shows how to incorporate LTL into PRM path planners. In the context of sampling-based motion planning, PRMs provide an important class of approaches which have been shown to work well for high-dimensional configuration spaces. Since LTL-tree approaches were developed for systems with dynamics, they do not capture the connectivity of the configuration space as PRM does. As shown in this paper, rather than extending the tree from one propositional assignment to another (as it is the case in LTL-tree approaches), the roadmap graph is combined with LTL to search for a solution. As a result, roadmap connections are reused when needed to find paths that satisfy LTL specifications. This bears similarities to the distinction in the context of point-to-point path planning

Dept. of Electrical Engineering and Computer Science, Catholic University of America, Washington DC 20064. Email: plaku@cua.edu

between PRM approaches as multi-query methods and tree-approaches as single-query methods.

Since LTL planning is PSPACE-complete [20], as in the LTL-tree approaches, TemporalPRM considers co-safe LTL formulas. Co-safe LTL formulas are satisfied by finite sequences of propositional assignments rather than infinite sequences which satisfy general LTL formulas. Co-safe LTL formulas can be translated into NFAs (Nondeterministic Finite Automata) with at most an exponential increase in size [14]. As recommended in [3], [16], [19], NFAs are converted into DFAs (Deterministic Finite Automata), which are then minimized. A DFA search has a significantly smaller branching factor than an NFA search, since there is exactly one transition that can be followed from each state for each propositional assignment.

The proposed TemporalPRM approach first constructs a roadmap by sampling collision-free configurations and connecting neighboring configurations via local paths. The roadmap is implicitly combined with the automaton representing the co-safe LTL formula and a discrete search is conducted over the combined graph. During the combined discrete search, roadmap vertices are mapped to propositional assignments and roadmap edges are mapped to sequences of propositional assignments. A solution is found at a vertex $v$ if the sequence of edges connecting the initial vertex to $v$, as obtained by the combined discrete search, is mapped to a sequence of propositional assignments that satisfies the co-safe LTL specification. Experimental validation of TemporalPRM is provided in simulation by using different scenes, LTL specifications, a snake-like robot model with numerous degrees-of-freedom, and different sampling strategies.

## II. LTL Specifications

Let $\Pi$ denote a set of propositions. A proposition $\pi_i \in \Pi$ is a problem-specific statement, such as "robot has reached area $A_i$." LTL combines propositions with logical $\neg$ (not), $\wedge$ (and), $\vee$ (or), and temporal connectives $\bigcirc$ (next), $\Diamond$ (eventually), $\Box$ (always), $\cup$ (until). A discrete state $\tau_i \in 2^{\Pi}$ denotes 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 $\sigma = [\tau_i]_{i=1}^{n}$ that satisfies a given LTL formula $\phi$.

### A. LTL Syntax and Semantics [14]

Every $\pi \in \Pi$ is a formula. If $\phi$ and $\psi$ 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 $\sigma$. The notation $\sigma \models \phi$ indicates that $\sigma$ satisfies $\phi$ and is defined as $\sigma \models \pi$ if $\pi \in \Pi$ and $\pi \in \tau_0$; $\sigma \models \neg\phi$ if $\sigma \not\models \phi$; $\sigma \models \phi \wedge \psi$ if $\sigma \models \phi$ and $\sigma \models \psi$; $\sigma \models \bigcirc\phi$ if $\sigma^1 \models \phi$; $\sigma \models \phi \cup \psi$ if $\exists\, k \geq 0$ such that $\sigma^k \models \psi$ and $\forall\, 0 \leq i < k : \sigma^i \models \phi$. The other connectives are defined as $\texttt{false} = \pi \wedge \neg\pi$, $\texttt{true} = \neg\texttt{false}$, $\phi \vee \psi = \neg(\neg\phi \wedge \neg\psi)$, $\Diamond\phi = \texttt{true} \cup \phi$, $\Box\phi = \neg\Diamond\neg\phi$, and $\phi\mathcal{R}\psi \equiv \neg(\neg\phi \cup \neg\psi)$.

As discussed, since LTL planning is PSPACE-complete [20], as in the LTL-tree approaches, TemporalPRM considers co-safe LTL formulas, which are satisfied by finite sequences of discrete states and can be translated into DFAs [14]. A DFA is a tuple $\mathcal{A} = (Z, \Sigma, \delta, z_{\mathit{init}}, \mathit{Accept})$, 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, $z_{\mathit{init}} \in Z$ is the initial state, and $\mathit{Accept} \subseteq Z$ 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 $\sigma$ iff $\mathcal{A}(\sigma, z_{\mathit{init}}) \in \mathit{Accept}$. As a result, $\sigma \models \phi$ when the equivalent automaton $\mathcal{A}$ accepts $\sigma$.

### B. Interpretation over Configuration Space

Each proposition $\pi_i$ is associated with a function $\textsc{holds}_{\pi_i}(c)$ which determines if $\pi_i$ holds at the robot configuration $c$. This interpretation provides a mapping $\textsc{DiscreteState} : \mathcal{C} \to 2^{\Pi}$ from the configuration space $\mathcal{C}$ to the discrete space $2^{\Pi}$, i.e.,

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

Moreover, paths over $\mathcal{C}$ give meaning to temporal connectives. A path over $\mathcal{C}$ is a continuous function $\zeta : [0, T] \to \mathcal{C}$, parametrized by time. As the robot configuration changes according to $\zeta$, the discrete state, obtained by the mapping $\textsc{DiscreteState}$, may also change. At time $t_1 = 0$, $\zeta$ satisfies $\tau_1 = \textsc{DiscreteState}(\zeta(t_1))$. $\zeta$ will continue to satisfy $\tau_1$ until some later time $t_2$, where $\zeta$ may satisfy a different $\tau_2 = \textsc{DiscreteState}(\zeta(t_2))$, $\tau_1 \neq \tau_2$. In this way, $\zeta$ can be broken down into a sequence of discrete states $[\tau_i]_{i=1}^{n}$ and a sequence of time intervals $[t_1, t_2), \ldots, [t_{n-1}, t_n), [t_n, T]$ such that inside the $i$-th time interval $\zeta$ satisfies $\tau_i = \textsc{DiscreteState}(\zeta(t_i))$ and that $\tau_i \neq \tau_{i+1}$. For notational convenience, let $\textsc{DiscreteStates}(\zeta) \stackrel{def}{=} [\tau_i]_{i=1}^{n}$ denote the sequence of discrete states satisfied by $\zeta$. In this way, $\zeta$ is said to satisfy an LTL formula $\phi$ iff $\textsc{DiscreteStates}(\zeta) \models \phi$.

### C. Examples

To facilitate presentation, this section provides examples of co-safe LTL specifications, which are also used in the experiments.

The robot is modeled as a manipulator chain whose base can freely translate and rotate. The reference point of the robot corresponds to the center of the base. Rotational joints connect links to one another. A configuration is considered valid if 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.

The workspaces in which the robot operates, as in the related LTL motion-planning work [4], [8], [9], [13], [16], are populated with polygonal obstacles and propositions. In this way, a proposition $\pi_i$ is associated with a polygon $P_i$, and the function $\textsc{holds}_{\pi_i}(c)$ is true iff the reference point of the robot is inside $P_i$ when the robot is placed in configuration

$c$. In addition, the workspace is triangulated. Fig. 1 shows the workspaces.

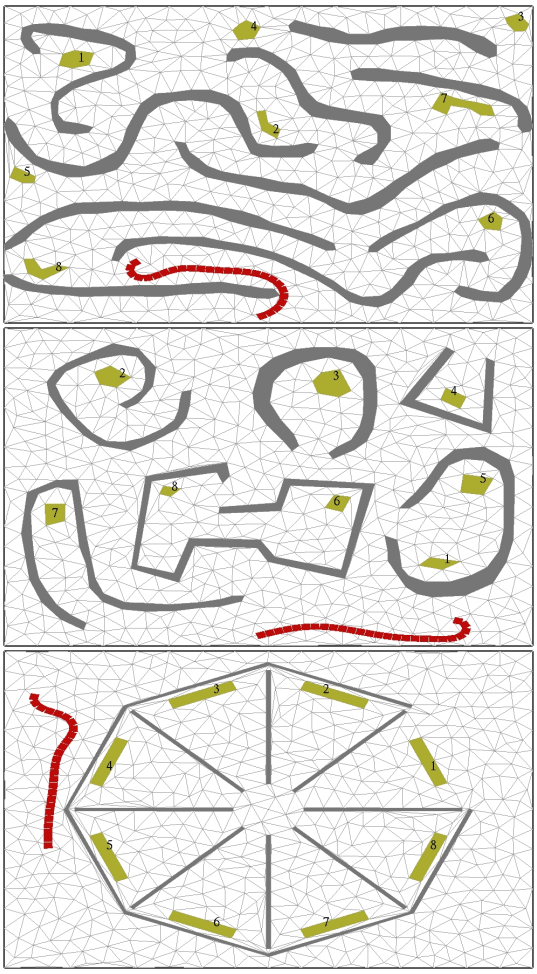

Fig. 1. Workspaces used in the experiments. Obstacles are in gray. Propositions are in a golden color and are numbered $1, 2, \ldots, 8$. The robot is in red and is shown in some initial configuration. The workspace triangulations are also shown.

Co-safe LTL specifications are defined over 8 propositions generated for each workspace, as shown in Fig. 1. Let $\beta = \neg\pi_1 \wedge \ldots \wedge \neg\pi_8$, i.e., $\beta$ is true iff none of the propositions $\pi_1, \ldots, \pi_8$ is true. Note that $\beta$ geometrically corresponds to the area not occupied by $P_1 \cup \ldots \cup P_8$.

The first task is to compute a collision-free configuration-space path $\zeta$ which satisfies $\pi_1, \ldots, \pi_n$ in that order. More specifically, $\zeta$ starts at $\beta$ and satisfies $\beta$ until it satisfies $\pi_1$; then satisfies $\pi_1 \vee \beta$ until it satisfies $\pi_2$; ...; then satisfies $\pi_7 \vee \beta$ until it satisfies $\pi_8$. Formally, the formula is as follows:

$$\phi_1 = \beta \cup (\pi_1 \wedge ((\pi_1 \vee \beta) \cup (\pi_2 \wedge (\ldots (\pi_7 \vee \beta) \cup \pi_8)))).$$

The second task is to compute a collision-free configuration-space path $\zeta$ that eventually satisfies each proposition, i.e.,

$$\phi_2 = \bigwedge_{i=1}^{8} \Diamond(\pi_i).$$

This task presents combinatorially many different possibilities regarding the order in which to satisfy the propositions.

The third task is to compute a collision-free configuration-space path $\zeta$ which eventually satisfies $\pi_i, \pi_j, \pi_k, \pi_i, \pi_j$ with $i, j, k$ all different from each other, i.e.,

$$\phi_3 = \bigvee_{1 \leq i,j,k \leq 8, i \neq j, j \neq k, i \neq k} \Diamond\pi_i \wedge (\Diamond\pi_j \wedge (\Diamond\pi_k \wedge (\Diamond\pi_i \wedge (\Diamond\pi_j)))).$$

This task presents polynomially different possibilities about which propositions to satisfy.

## III. TemporalPRM

Let $c_{init}$ denote the initial robot configuration. Let $\phi$ denote the co-safe LTL specification. The objective of TemporalPRM is to compute in the configuration space a collision-free path $\zeta$ that satisfies $\phi$, i.e., DiscreteStates$(\zeta) \models \phi$. Section III-A and III-B describe the roadmap construction and combined graph search in TemporalPRM and section III-C discusses improvements to the general approach.

### A. Roadmap Construction

TemporalPRM constructs a roadmap by sampling collision-free configurations and connecting neighboring configurations via simple paths, e.g., linear interpolation, geodesic interpolation. Pseudocode is provided in Algo. 1.

As in PRM, TemporalPRM can employ various sampling and connection strategies during roadmap construction. The roadmap is maintained as a graph $R = (V_R, E_R)$. Each roadmap vertex $v \in V_R$ is associated with the corresponding collision-free robot configuration, denoted as $cfg(v)$ (Algo. 1:4). The discrete state obtained by the mapping DiscreteState$(cfg(v))$ is computed and stored as $\tau(v)$ (Algo 1:5) for easy access during the combined search, as explained in Section III-B. The initial vertex, which is associated with the initial robot configuration, is added to the roadmap at the beginning of the roadmap construction.

As in PRM, after generating the vertices, TemporalPRM attempts to connect each vertex $v$ to its $k$ nearest neighbors, which are defined according to a distance metric (Algo. 1:9-11). Each roadmap edge $(v, u) \in E_R$ is associated with the corresponding collision-free configuration-space path that connects $cfg(v)$ and $cfg(u)$, denoted as $cpath(v, u)$ (Algo. 1:12–16). Linear interpolation or some other continuous function can be used to define the configuration-space path between two configurations. As in PRM, TemporalPRM can use incremental or subdivision collision-checking to determine if $cpath(v, u)$ is collision free. During collision checking, TemporalPRM also computes the sequence of discrete states associated with $cpath(v, u)$ (Algo 1:17–20), which is used by the combined search, as explained in Section III-B. If no collision is found, the edge $(v, u)$ is added to the roadmap. Since configuration-space paths are reversible, the edge $(u, v)$ is also added to the roadmap (Algo. 1:22–25).

**Algorithm 1** ROADMAP($R = (V_R, E_R), n, k$)

**Input:** $R = (V_R, E_R)$ – current roadmap;
$n$ – nr of additional vertices; $k$ – nr of neighbors
**Output:** new roadmap graph $R = (V_R, E_R)$

◇ *generate $n$ additional roadmap vertices*
*size* $\leftarrow |V_R|$
**for** $i = 1$ to $n$ **do**
  $v \leftarrow$ CREATEVERTEX()
  *cfg*$(v) \leftarrow$ GENERATEVALIDCFG()
  $\tau(v) \leftarrow$ DISCRETESTATE(*cfg*$(v)$)
  $V_R \leftarrow V_R \cup \{v\}$
**end for**
◇ *generate roadmap edges for the newly added vertices*
**for** $i =$ *size* $+ 1$ to *size* $+ n$ **do**
  $v \leftarrow$ GETVERTEX$(R, i)$
  *neighs* $\leftarrow$ NEIGHBORS$(R, v, k)$
  **for** $u \in$ *neighs* **do** ◇ *generate edge from $v$ to $u$*
    ◇ *compute cpath$(v, u)$ by linear interpolation*
    *steps* $\leftarrow$ *floor*(DIST(*cfg*$(v)$, *cfg*$(u)$)/*StepSize*)
    $\tau \leftarrow \tau(v)$; *valid* $\leftarrow$ `true`
    **for** $i = 1$ to *steps* $- 1$ and *valid* **do**
      $c \leftarrow (1 - i/$*steps*$)$*cfg*$(v) + (i/$*steps*$)$*cfg*$(u)$
      *valid* $\leftarrow$ ISVALIDCFG$(c)$
      **if** *valid* and DISCRETESTATE$(c) \neq \tau$ **then**
        $\tau \leftarrow$ DISCRETESTATE$(c)$;
        *discreteStates*$(v, u)$.*pushback*$(\tau)$
      **end if**
    **end for**
    **if** *valid* **then**
      $E_R \leftarrow E_R \cup \{(v, u), (u, v)\}$
      *discreteStates*$(u, v) \leftarrow$ REVERSE(*discreteStates*$(v, u)$)
    **end if**
  **end for**
**end for**
**return** $R = (V_R, E_R)$

**Algorithm 2** COMBINEDSEARCH($R = (V_R, E_R), \mathcal{A}$)

**Input:** $R = (V_R, E_R)$ – roadmap
$\mathcal{A}$ automaton representing LTL formula $\phi$
**Output:** solution path if found; `null`, otherwise

1: *stack.push*$(\langle v_{init}, z_{init}\rangle)$; *parent*$(\langle v_{init}, z_{init}\rangle) \leftarrow$ *undefined*
2: **while** *stack.empty* $=$ `false` **do**
3:  $\langle v, z\rangle \leftarrow$ *stack.pop*
4:  **if** $z$ is an accepting automaton state **then**
5:  **return** path from $\langle v_{init}, z_{init}\rangle$ to $\langle v, z\rangle$
6:  **end if**
7:  **for** $u \in V_R$ such that $(v, u) \in E_R$ **do**
8:  $z' \leftarrow \mathcal{A}$(*discreteStates*$(v, u), z)$
9:  **if** *parent*$(\langle u, z'\rangle) =$ *undefined* **then**
10:  *stack.push*$(\langle u, z'\rangle)$; *parent*$(\langle u, z'\rangle) \leftarrow \langle v, z\rangle$
11:  **end if**
12:  **end for**
13: **end while**
14: **return** `null`

### B. Combined Search

Consider a sequence of connected roadmap vertices $v_{init} = v_0, v_1, \ldots, v_n$, i.e., $(v_i, v_{i+1}) \in E_R$, $0 \leq i < n$. Let $\zeta$ denote the configuration-space path from *cfg*$(v_{init})$ to *cfg*$(v_n)$ obtained by concatenating the configuration-space paths associated with the roadmap edges $(v_0, v_1), \ldots (v_{n-1}, v_n)$, i.e.,

$$\zeta = \textit{cpath}(v_0, v_1) \circ \ldots \circ \textit{cpath}(v_{n-1}, v_n).$$

Note that $\zeta$ constitutes a solution iff DISCRETESTATES$(\zeta) \models \phi$. While PRM finds a solution by performing a standard graph search over the roadmap $R$, the same cannot be achieved in the case of path planning with co-safe LTL. The reason that a standard graph search over $R$ would not work is that a co-safe LTL specification is satisfied by a configuration-space path and not by a configuration, i.e., there is no goal configuration.

As a result, TemporalPRM conducts the search over an abstract graph $RA = (V_{RA}, E_{RA})$, which is obtained by combining the roadmap $R$ and the automaton $\mathcal{A}$ representing the co-safe LTL formula $\phi$. Each vertex in $V_{RA}$ consists of a pair $\langle v, z\rangle$, where $v$ is a roadmap vertex and $z$ is an automaton state and each edge in $E_{RA}$ is of the form $(\langle v_i, z_i\rangle, \langle v_j, z_j\rangle)$. Note that $RA$ is not computed explicitly; its vertices and edges are rather computed on-the-fly as needed by the graph search. In an implicit formulation, graph search requires as input an initial vertex, a function RAGOAL$(\langle v, z\rangle)$ to determine if the goal is achieved at vertex $\langle v, z\rangle$, and a function RAOUTEDGES$(\langle v, z\rangle)$ to determine the edges coming out of $\langle v, z\rangle$. If a shortest-path is desired, then a function RAEDGECOST$(\langle v', z'\rangle, \langle v'', z''\rangle)$ to determine the cost of the edge $(\langle v', z'\rangle, \langle v'', z''\rangle)$, and, perhaps, a heuristic function RAHEURISTIC$(\langle v, z\rangle)$ to provide an admissible heuristic for the cost from $\langle v, z\rangle$ to a goal vertex are also needed.

To facilitate presentation, we describe how to conduct depth-first-search over the abstract graph $RA$. Pseudocode is given in Algo. 2. Other graph search techniques, such as breadth-first search, Dijkstra's shortest path, A*, etc., can be applied in a similar fashion.

In this implicit formulation, the initial vertex is given by $\langle v_{init}, z_{init}\rangle$, where $v_{init}$ is the roadmap vertex associated with the initial robot configuration and $z_{init}$ is the initial state of the automaton $\mathcal{A}$. The initial vertex $\langle v_{init}, z_{init}\rangle$ is pushed onto the stack and the parent of $\langle v_{init}, z_{init}\rangle$ is set to undefined (Algo. 2:1). While the stack is not empty, the top element is popped, i.e., $\langle v, z\rangle \leftarrow$ *stack.pop* (Algo. 2:3). Note that at this point, there is already a graph path $\langle v_{init}, z_{init}\rangle = \langle v_0, z_0\rangle, \langle v_1, z_1\rangle, \langle v_n, z_n\rangle = \langle v, z\rangle$ from $\langle v_{init}, z_{init}\rangle$ to $\langle v, z\rangle$, which is obtained by following the parent pointers starting from $\langle v, z\rangle$ until reaching $\langle v_{init}, z_{init}\rangle$, and then reversing the sequence. Let $\zeta$ denote the configuration-space path from *cfg*$(v_{init})$ to *cfg*$(v)$, which is obtained by concatenating the configuration-space paths associated with the roadmap edges $(v_0, v_1), (v_1, v_2), \ldots (v_{n-1}, v_n)$. The invariant of the graph search, as it will become apparent in a moment, is that $z$ is

the automaton state obtained by running DISCRETESTATES($\zeta$) on the automaton $\mathcal{A}$, i.e., $z = \mathcal{A}(\text{DISCRETESTATES}(\zeta), z_{init})$. As such, $\zeta$ constitutes a solution to the co-safe LTL specification iff $z$ is an accepting automaton state (Algo. 2:4).

If $z$ is not an accepting automaton state, then the graph search is expanded from $\langle v, z\rangle$ by invoking the function RAOUTEDGES($\langle v, z\rangle$) (Algo. 2:7–12), which we now describe. Consider a roadmap edge $(v, u) \in E_R$. As the robot moves along the configuration-space path *cpath*$(v, u)$ associated with the edge $(v, u)$, the state of the automaton changes to $z'$, which is obtained by running DISCRETESTATES(*cpath*$(v, u)$) on the automaton $\mathcal{A}$ starting from $z$ (Algo. 2:8). Note that DISCRETESTATES(*cpath*$(v, u)$) has already been computed during roadmap construction and stored as *discreteStates*$(v, u)$. If $\langle u, z'\rangle$ has not already been expanded in the graph search, i.e., *parent*$(\langle u, z'\rangle)$ is undefined, then $\langle u, z'\rangle$ is pushed onto the stack and $\langle v, z\rangle$ is marked as its parent (Algo. 2:9–11). The invariant of the combined graph-search holds since the configuration-space path $\zeta'$ from $\langle v_{init}, z_{init}\rangle$ to $\langle u, z'\rangle$ is obtained by concatenating $\zeta$ and *cpath*$(v, u)$, and, DISCRETESTATES($\zeta'$) ends up on $z'$ when run on the automaton $\mathcal{A}$.

### C. Improvements and Implementation Details

*1) Roadmap Cycles:* A common technique to reduce roadmap construction time in PRM is to create a roadmap that does not have any cycles since cycles do not improve roadmap connectivity. A disjoint set data structure is typically used to quickly determine if two roadmap vertices $v, u \in V_R$ belong to the same connected component. Edge collision checking is then attempted only if $v, u$ belong to two different connected components in the roadmap, since, otherwise, the edge $(v, u)$ would create a cycle.

In path planning with co-safe LTL, the order in which roadmap vertices are visited during the graph search over the combined graph $RA$ matters. Not allowing any cycles could in some cases have the adverse effect of increasing the computational time needed to obtain a solution. As an illustration, referring to the third workspace in Fig. 1, if the current roadmap is such that connections to area 2 are made via area 1, then it may be difficult even with additional sampling to obtain a solution to the specification "reach area 2 before area 1."

This, however, does not mean that all cycles are necessary. In fact, the roadmap edge $(v, u)$ would not be needed if the roadmap $R$ already contained a configuration-space path $\zeta$ from *cfg*$(v)$ to *cfg*$(u)$ such that DISCRETESTATES($\zeta$) = DISCRETESTATES(*cpath*$(v, u)$). It is, however, challenging to obtain an implementation of this definition that is faster than actually checking *cpath*$(v, u)$ for collisions. As an approximation to this definition, in TemporalPRM, edge collision checking is not done if DISCRETESTATE(*cfg*$(v)$) = DISCRETESTATE(*cfg*$(u)$) and $v$, $u$ belong to the same connected component when restricted to roadmap vertices that are mapped to the same discrete state as $v$ and $u$. Computationally, this can be done efficiently by using a disjoint set data structure. In particular, when adding a roadmap edge $(v', v'')$, the join operation in the disjoint set is performed iff DISCRETESTATE(*cfg*$(v')$) = DISCRETESTATE(*cfg*$(v'')$). Then, determining whether or not the edge $(v, u)$ should be checked for collision can be accomplished by querying the disjoint set to find if $v$ and $u$ belong to different connected components. The reason for this approximate definition is that since *cfg*$(v)$ and *cfg*$(u)$ are close to each other, it is likely that the discrete state does not change as the robot moves along *cpath*$(v, u)$. As the experimental results show, using this approximation considerably reduces the overall computational time when compared to the alternatives of not allowing any cycles or allowing all cycles.

*2) Enhanced Sampling:* As in PRM, different sampling strategies, e.g., Gaussian [5], obstacle-based [1], can be used in TemporalPRM to improve roadmap connectivity. This paper also implements a workspace-based sampling strategy. In particular, each time a configuration $c$ is added to the roadmap, TemporalPRM also determines the triangle in the workspace decomposition or the polygonal proposition which contains the reference point of the robot when placed in configuration $c$. In this way, TemporalPRM keeps track of the number of roadmap configurations associated with a triangle or a polygonal proposition. During sampling, a triangle or a polygonal proposition, denoted as $R_i$, is selected with probability $p(R_i) = \frac{1}{n_i^2} / \sum_{j=1}^{m} \frac{1}{n_j^2}$, where $n_k$ is the number of roadmap configurations associated with $R_k$ (which could be a triangle or a polygonal proposition). The reference point of the robot is then generated uniformly at random inside the selected triangle or polygonal proposition and the remaining DOFs of the robot are sampled uniformly at random. If the configuration is valid, it is added to the roadmap. Otherwise, a new configuration is sampled according to the above procedure. This workspace-based sampling strategy aims to place robot configurations in sparsely populated regions of the workspace, since sparsely populated regions could indicate narrow passages where additional sampling should take place to improve roadmap connectivity. As the experimental results indicate, the workspace-based sampling reduces the overall computational time of TemporalPRM when compared to uniform sampling and some other common PRM sampling strategies.

## IV. EXPERIMENTS AND RESULTS

Experimental validation is provided in simulation by using different scenes, co-safe LTL specifications, and a snake-like robot model with numerous DOFs, as described in Section II-C. By increasing the number of links, the robot provides challenging test cases for high-dimensional motion-planning problems. In the experiments in this paper, the number of links is varied from 1 to 26, yielding problems with $3, \ldots, 28$ DOFs. The impact of different sampling strategies and roadmap cycles are also studied.

*Measuring the computational efficiency:* The running time of TemporalPRM for each problem instance is obtained as the average of ten different runs. As in PRM, TemporalPRM requires as input the number of roadmap vertices ($n$) and the number of nearest neighbors ($k$). For each run,

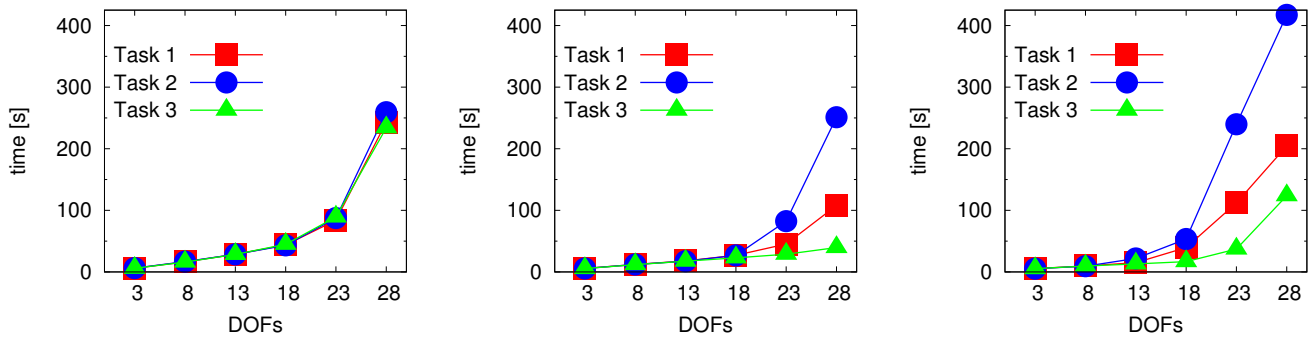


Fig. 2. Results of TemporalPRM over the three scenes and the three formulas when using uniform sampling.

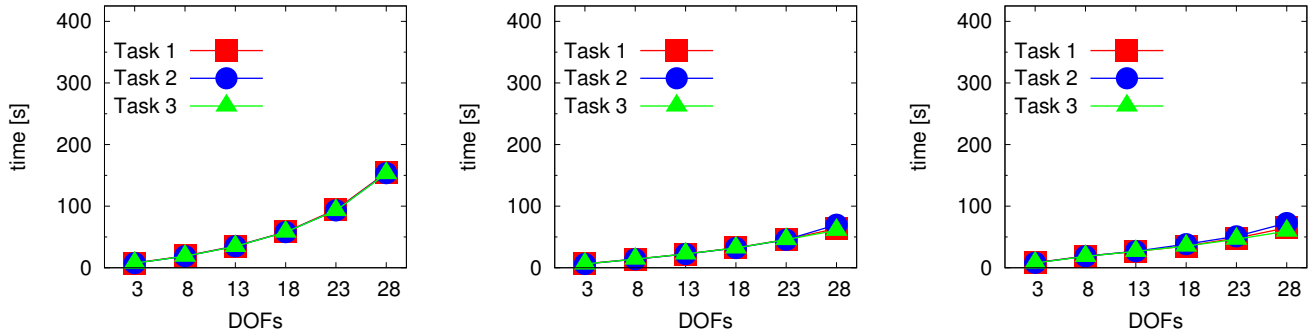


Fig. 3. Results of TemporalPRM over the three scenes and the three formulas when using enhanced sampling (Section III-C.2).

the initial number of roadmap vertices to be generated is set to 40000 and the number of nearest neighbors is set to 300. If a solution is not found, an additional 5000 vertices are added to the roadmap. This process is repeated until a solution is found or an upper bound of 600s on computational time is exceeded. These parameter values were found to work well for the problems considered in this work. In fact, tests revealed that small values of $k$ lead to largely disconnected graphs, while considerably large values of $k$ increase computational time by forcing the planner to check for collisions many edges whose configurations are far from each other, and are thus likely to result in collisions. As with PRM, applying TemporalPRM to new problems may require some experimentation to determine good values for $n$ and $k$. Experiments are run on a Dell machine (CPU: U4100 at 1.30GHz, RAM: 8GB) using Ubuntu 11.10. Code is compiled with GNU g++-4.6.1

### A. Results

*1) Impact of Sampling:* Fig. 2 provides a summary of the results of TemporalPRM over different workspaces and LTL specifications when using uniform sampling. 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. Nevertheless, TemporalPRM is able to efficiently solve all problem instances, even as the number of DOFs is increased to 28.

Fig. 3 provides a summary of the results of TemporalPRM when using enhanced sampling (Section III-C.2). As shown, enhanced sampling reduces the running time of TemporalPRM as it is able to sample more inside narrow passages as defined by sparsely populated regions.

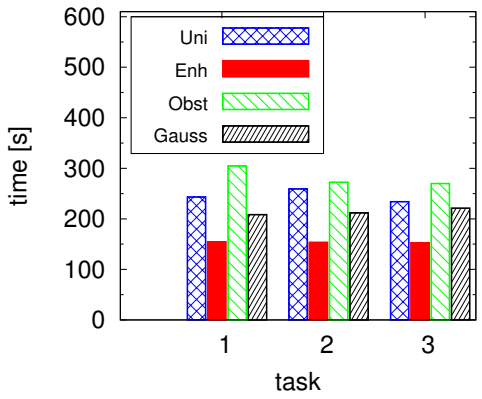


Fig. 4. Impact of different sampling strategies on TemporalPRM. Results are shown for workspace 1 and the robot with 28 DOFs. Similar results are obtained for the other workspaces.

In addition to uniform and enhanced sampling, we also experimented with obstacle-based [1] and Gaussian [5] sampling. A summary of the results is shown in Fig. 4. The computational efficiency in the case of obstacle-based and Gaussian sampling is similar to that of uniform sampling.

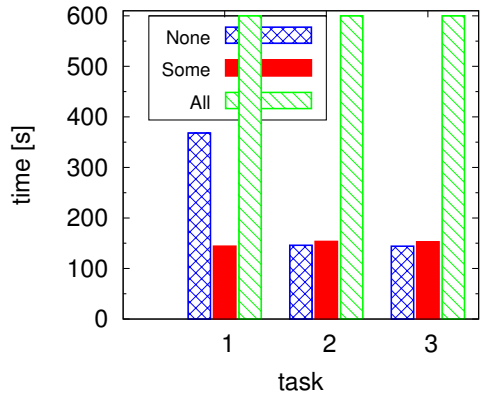


Fig. 5. Impact of roadmap cycles on TemporalPRM. Results are shown for workspace 1 and the robot with 28 DOFs. Similar results are obtained for the other workspaces.

*2) Impact of Roadmap Cycles:* Experiments were also conducted to study the impact of roadmap cycles. Fig. 5 provides a summary of the results. Three different scenarios were considered: (i) no cycles are allowed (ii) some cycles are allowed as described in Section III-C.1; and (iii) all cycles are allowed, i.e., each edge is checked for collision. As expected, when allowing all cycles, TemporalPRM timed out, as the number of edge collision checks is significant. As discussed in Section III-C.1, when not allowing any cycles, there could be some cases where the running time increases rather than decreases since the co-safe LTL specification may require the robot to satisfy the propositions in a certain order that does not conform to the roadmap vertices. Indeed, this is the case with task 1, which requires the robot to satisfy $\pi_1, \ldots, \pi_8$ in that order. When there are considerable alternatives about the order in which to satisfy the propositions, then the performance of TemporalPRM does not suffer when not allowing any cycles. Since not allowing any cycles could lead to wide variations in computational efficiency depending on the co-safe LTL specification, as discussed in Section III-C.1 and validated by the results, a better alternative is to allow only certain cycles that involve edges associated with two different propositions.

## V. Discussion

This paper showed how to incorporate co-safe LTL specifications into PRM path planners. PRMs provide an important class of approaches which have been shown to work well for high-dimensional configuration spaces. The integration with co-safe LTL makes it possible to express tasks in terms of propositions, logical connectives, and temporal connectives. TemporalPRM constructs a roadmap in a similar fashion as PRM. During a second stage, the roadmap is implicitly combined with the co-safe LTL representation and a discrete search is conducted over the combined graph. During the combined discrete search, roadmap vertices are mapped to propositional assignments and roadmap edges are mapped to sequences of propositional assignments. The paper also studied the impact of sampling strategies and roadmap cycles.

## References


[1] N. M. Amato, B. Bayazit, L. Dale, C. Jones, and D. Vallejo, "OBPRM: An obstacle-based PRM for 3d workspaces," in *International Workshop on Algorithmic Foundations of Robotics*, Houston, TX, 1998, pp. 156–168.

[2] R. C. Arkin, "Integrating behavioral, perceptual and world knowledge in reactive navigation," *Robotics and Autonomous Systems*, vol. 6, pp. 105–122, 1990.

[3] R. Armoni, S. Egorov, R. Fraer, D. Korchemny, and M. Vardi, "Efficient LTL compilation for SAT-based model checking," in *Intl Conf on Computer-Aided Design*, San Jose, CA, 2005, pp. 877–884.

[4] A. Bhatia, M. Maly, L. Kavraki, and M. Vardi, "Motion planning with complex goals," *IEEE Robotics Automation Magazine*, vol. 18, pp. 55–64, 2011.

[5] V. Boor, M. H. Overmars, and A. F. van der Stappen, "The Gaussian sampling strategy for probabilistic roadmap planners," in *IEEE International Conference on Robotics and Automation*, Detroit, MI, 1999, pp. 1018–1023.

[6] T. Bretl, "Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem," *International Journal of Robotics Research*, no. 4, pp. 317–342, 2006.

[7] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *International Journal of Robotics Research*, no. 1, pp. 104–126, 2009.

[8] X. C. Ding, M. Kloetzer, Y. Chen, and C. Belta, "Formal methods for automatic deployment of robotic teams," *IEEE Robotics and Automation Magazine*, vol. 18, no. 3, pp. 75–86, 2011.

[9] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic mobile robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.

[10] K. Hauser and V. Ng-Thow-Hing, "Randomized multi-modal motion planning for a humanoid robot manipulation task," *International Journal of Robotics Research*, vol. 30, no. 6, pp. 678–698, 2011.

[11] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic $\mu$-calculus specifications," in *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*. IEEE, 2009, pp. 2222–2229.

[12] H. Kress-Gazit, T. Wongpiromsarn, , and U. Topcu, "Correct, reactive robot control from abstraction and temporal logic specifications," *IEEE Robotics and Automation Magazine on Formal Methods for Robotics and Automation*, vol. 18, no. 3, pp. 65–74, 2011.

[13] H. Kress-Gazit, D. C. Conner, H. Choset, A. Rizzi, and G. J. Pappas, "Courteous cars: Decentralized multi-agent traffic coordination," *Special Issue of the IEEE Robotics and Automation Magazine on Multi-Agent Robotics*, 2007.

[14] O. Kupferman and M. Vardi, "Model checking of safety properties," *Formal methods in System Design*, vol. 19, no. 3, pp. 291–314, 2001.

[15] D. Nieuwenhuisen, A. van der Stappen, and M. Overmars, "An effective framework for path planning amidst movable obstacles," in *International Workshop on Algorithmic Foundations of Robotics*, ser. Springer Tracts in Advanced Robotics, New York, NY, 2006, vol. 47, pp. 87–102.

[16] E. Plaku, "Planning in discrete and continuous spaces: From LTL tasks to robot motions," in *Towards Autonomous Robotic Systems*, Bristol, UK, 2012, p. in press.

[17] E. Plaku and G. D. Hager, "Sampling-based motion and symbolic action planning with geometric and differential constraints," in *IEEE International Conference on Robotics and Automation*, Anchorage, AK, 2010, pp. 5002–5008.

[18] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Falsification of LTL safety properties in hybrid systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science. York, UK: Springer, 2009, vol. 5505, pp. 368–382.

[19] E. Plaku, L. Kavraki, and M. Vardi, "Falsification of ltl safety properties in hybrid systems," *International Journal on Software Tools and Technology Transfer*, pp. 1–16, 2012.

[20] A. Sistla, "Safety, liveness and fairness in temporal logic," *Formal Aspects of Computing*, vol. 6, pp. 495–511, 1994.

[21] M. Stilman and J. Kuffner, "Planning among movable obstacles with artificial constraints," *International Journal of Robotics Research*, vol. 27, no. 12, pp. 1295–1307, 2008.