Cooperative Sampling-Based Motion Planning under Signal Temporal Logic Specifications

Mayank Sewlia[1], Christos K. Verginis[2], and Dimos V. Dimarogonas[1]

Abstract— **We develop a cooperative sampling-based motion planning algorithm for two autonomous agents under coupled tasks expressed as signal temporal logic constraints. The algorithm builds incrementally two spatio-temporal trees, one for each agent, by sampling points in an extended space, which consists of a compact subset of the time domain and the physical space of the agents. The trees are built by checking if newly sampled points form edges in time and space that satisfy certain parts of the coupled task. Therefore, the constructed trees represent time-varying trajectories in the agents' state space that satisfy the task. The algorithm is distributed in the sense that the agents build their trees individually by communicating with each other. The proposed algorithm inherits the properties of probabilistic completeness and computational efficiency of the original sampling-based procedures.**

I. Introduction

Planning of autonomous agents subject to tasks encoded as temporal logic specifications has attracted a great deal of attention during the last two decades. Expressing tasks as temporal logic constraints offers a great degree of versatility, since such constraints can efficiently describe a large variety of complex planning objectives, as opposed to simple point-to-point navigation. A special form of temporal logic, namely signal temporal logic, offers the incorporation of spatial and time specifications for autonomous agents, providing a rich variety of tasks [1].

There exist numerous works that consider planning under temporal logic specifications. Such specifications include both qualitative properties, such as linear temporal logic (LTL) [2], and quantitative spatio-temporal properties, such as metric temporal logic (MTL) [3]–[6], metric interval temporal logic (MITL) [7]–[10] or signal temporal logic (STL) [11]–[18]. MTL and MITL usually specify tasks over finite-state spaces, requiring the abstraction of the underlying continuous-time and -state systems to discrete ones. However, such abstractions bring several drawbacks, such as loss of information and risk of state explosion. Using STL avoids the aforementioned drawbacks and allows one to express specifications in continuous time and space.

Several works consider the multi-agent motion-planning planning problem under STL tasks [17], [19]–[23]. The work [17], [19] develops distributed feedback control algorithms based on funnel control and barrier functions to guide robots toward satisfying the underlying STL tasks. The limitations of continuous spaces, however, such as the inability to explore the entire time and state spaces, leads to the consideration of only specific STL fragments, losing the full expressivity of STL. Works that consider the entire STL fragment adopt centralized solutions based on mixed-integer linear programming (MILP) [20]–[23]. However, MILP programs can be computationally demanding and yield long running times. Furthermore, centralized algorithms do not scale with the number of agents and are sensitive to faults, since a single computer unit plans the actions of all agents. The general motion-planning problem, even for a two-dimensional configuration space with rectangular obstacles is PSPACE-hard [24], thus, a common approach is sampling-based planners. In such planners, the free space is modeled using a graph with nodes and edges. The motion planning problem then reduces to a graph-search problem where a collision free path is found thereon. By constructing this graph, a multidimensional space search is reduced to finding a path in the graph. This paper develops a distributed algorithm that employs a sampling procedure, inheriting its efficiency and probabilistic completeness properties.

This paper addresses the cooperative motion planning problem of two autonomous agents, which evolve continuously in time and space, to deliver coupled tasks expressed as signal temporal logic specifications. Our contribution with respect to the related literature is the development of a computationally-efficient algorithm that solves, in a distributed manner, the cooperative motion-planning problem under the entire fragment of signal temporal logic for continuous-time and -state systems, without resorting to discretization techniques. Among many places, coupled autonomous-agents find application in warehouses where a load is too heavy for a single manipulator and thus require cooperation between two manipulators to move the load. Multiple-agents can then form teams of coupled-agents to move multiple loads at once.

Inspired by standard sampling-based motion-planning techniques, we develop a cooperative time-augmented sampling-based algorithm. In particular, the algorithm builds incrementally two *spatio-temporal* trees, one for each agent, by sampling points in the *coupled* state *and* time domain of the agents. The algorithm checks if the new edges to be added to the trees satisfy certain parts of the given task. In that way, the resulting spatio-temporal trees represent time-varying trajectories in the agents' state space that satisfy

*This work was supported by the ERC CoG LEAFHOUND, the Swedish Research Council (VR), the Knut och Alice Wallenberg Foundation (KAW) and the H2020 European Project CANOPIES.

[1]M. Sewlia and D. V. Dimarogonas are with Division of Decision and Control, School of EECS, KTH Royal Institute of Technology, 100 44 Stockholm, Sweden. {sewlia, dimos}@kth.se

[2]Christos K. Verginis is with Division of Signals and Systems, Department of Electrical Engineering, Uppsala University, Uppsala, Sweden. christos.verginis@angstrom.uu.se

the given spatio-temporal task. Finally, the algorithm can be executed in a distributed manner since the agents build their own trees by communicating with each other. The proposed algorithm is relevant to the approach of [25], which introduces a time-augmented sampling-based algorithm for single-agent systems under STL tasks. Similarly, [18] proposes a sampling-based algorithm for single-agent systems, encoding a temporal-logic task in a cost function to be minimized, without sampling in the time domain.

The rest of the paper is organized as follows. Section II describes the problem considered, Section III provides the proposed algorithm, and Section IV presents simulation results. Finally, Section V concludes the paper.

II. Preliminaries and Problem Formulation

A. Notations

The set of natural numbers is denoted by $\mathbb{N}$ and the set of real numbers by $\mathbb{R}$. With $n \in \mathbb{N}$, $\mathbb{R}^n$ is the set of n-coordinate real-valued vectors and $\mathbb{R}^n_+$ is the set of real n-vector with non-negative elements. The cardinality of a set A is denoted by $|A|$. If $a \in \mathbb{R}$ and $[b,c] \in \mathbb{R}^2$, the Kronecker sum $a \oplus [b,c] = [a+b, a+c] \in \mathbb{R}^2$. We further define the boolean set as $\mathbb{B} = \{\top, \bot\}$ (True, False).

B. Signal Temporal Logic

Let $\mathbf{x} : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ be a continuous-time signal. Signal temporal logic [26] is a predicate-based logic with the following syntax:

$$\varphi = \top \mid \mu \mid \neg\varphi \mid \mathcal{G}_{[a,b]}\varphi \mid \mathcal{F}_{[a,b]}\varphi \mid \varphi_1 \mathcal{U}_{[a,b]} \varphi_2 \mid \varphi_1 \wedge \varphi_2 \quad (1)$$

where φ_1, φ_2 are STL formulas and $\mathcal{U}_{[a,b]}$ encodes the *until* operator, with $0 \leq a \leq b < \infty$; μ is a predicate of the form $\mu : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{B}$ defined via a predicate function $h : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ as

$$\mu = \begin{cases} \top & h(\mathbf{x}, t) \geq 0 \\ \bot & h(\mathbf{x}, t) < 0 \end{cases}. \quad (2)$$

We consider time bounded temporal operators. The satisfaction relation $(\mathbf{x}, t) \models \varphi$ indicates that signal $\mathbf{x}$ satisfies φ at time t and is defined recursively as follows:

$$\begin{aligned}
(\mathbf{x}, t) \models \mu &\Leftrightarrow h(\mathbf{x}, t) \geq 0 \\
(\mathbf{x}, t) \models \neg\varphi &\Leftrightarrow \neg((\mathbf{x}, t) \models \varphi) \\
(\mathbf{x}, t) \models \varphi_1 \wedge \varphi_2 &\Leftrightarrow (\mathbf{x}, t) \models \varphi_2 \wedge (\mathbf{x}, t) \models \varphi_2 \\
(\mathbf{x}, t) \models \varphi_1 \mathcal{U}_{[a,b]} \varphi_2 &\Leftrightarrow \exists t_1 \in [t+a, t+b] \text{ s.t. } (\mathbf{x}, t_1) \models \varphi_2 \\
&\quad \wedge \forall t_2 \in [t, t_1], (\mathbf{x}, t_2) \models \varphi_1.
\end{aligned}$$

We also define the operators *disjunction*, *eventually*, and *always* as $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\mathcal{F}_{[a,b]}\varphi \equiv \top\mathcal{U}_{[a,b]}\varphi$, and $\mathcal{G}_{[a,b]}\varphi \equiv \neg\mathcal{F}_{[a,b]}\neg\varphi$, respectively. The satisfaction relation $(\mathbf{x}, t) \models \varphi$ can also be quantified as follows:

$$\begin{aligned}
\rho(h(\mathbf{x}, t) \geq 0, \mathbf{x}, t) &= h(\mathbf{x}(t), t) \\
\rho(\neg\varphi, \mathbf{x}, t) &= -\rho(\varphi, \mathbf{x}, t) \\
\rho(\varphi_1 \wedge \varphi_2, \mathbf{x}, t) &= \min(\rho(\varphi_1, \mathbf{x}, t), \rho(\varphi_2, \mathbf{x}, t)) \\
\rho(\mathcal{G}_I\varphi, \mathbf{x}, t) &= \inf_{t_1 \in t+I} \rho(\varphi, \mathbf{x}, t_1) \\
\rho(\mathcal{F}_I\varphi, \mathbf{x}, t) &= \sup_{t_1 \in t+I} \rho(\varphi, \mathbf{x}, t_1) \\
\rho(\varphi_1 \mathcal{U}_I \varphi_2, \mathbf{x}, t) &= \sup_{t_1 = t+I} \min\Big(\rho(\varphi_2, \mathbf{x}, t_1), \\
&\qquad \inf_{t_2 \in (t, t_1)} \rho(\varphi_1, \mathbf{x}, t_2)\Big)
\end{aligned}$$

A signal $\mathbf{x}$ satisfies an STL formula φ at time t if and only if $\rho(\varphi, \mathbf{x}, t) \geq 0$.

C. STL Parse Tree

An STL formula defined recursively over the form (1) can be represented as a tree, we name it as an *STL parse tree*. An STL parse tree is constructed as follows:

- each node is either a temporal operator $\{\mathcal{G}_I, \mathcal{F}_I\}$, a logical operator $\{\vee, \wedge, \neg\}$ or a predicate $\{\mu\}$ where $I \subset \mathbb{R}$ is a closed interval,
- each node, apart from predicate nodes, is accompanied by a satisfaction variable $\tau \in \{+1, -1\}$; such nodes are termed *set* nodes,
- a root node has no parent node and a leaf node has no child note. The leaf nodes constitute the predicate nodes of the tree.

A path is a formula from a root node to a leaf node and we see that set of all paths constitutes the tree. A sub-path (or a sub-formula) is a path from a set node to a leaf node. Each set node is accompanied by a satisfaction variable $\tau \in \{+1, -1\}$ and each leaf node is accompanied by a predicate variable $\pi = \mu$. A signal $\mathbf{x}$ satisfies a sub-path if $\tau = +1$ corresponding to the set node where the path begins. An analogous tree of satisfaction and predicate variables can be drawn, called *satisfaction variable tree*. The satisfaction variable tree borrows the same tree structure as the STL parse tree. Each set node from the STL parse tree maps uniquely to a satisfaction variable τ_i and each leaf node maps uniquely to a predicate variable π_i, where i is an enumeration of the nodes in the satisfaction variable tree.

The STL parse tree and the satisfaction variable tree for the STL formula

$$\varphi = \mathcal{F}_{I_1}\Big(\mu_1 \vee \mathcal{G}_{I_2}(\mu_2)\Big) \wedge \mathcal{G}_{I_3}\mathcal{F}_{I_4}(\mu_3) \wedge \mathcal{G}_{I_5}(\mu_4). \quad (3)$$

are shown in Figure 1. Suppose $\tau_2 = +1 \implies \mathbf{x} \models \mathcal{F}_{I_1}\Big(\mu_1 \vee \mathcal{G}_{I_2}(\mu_2)\Big)$. And if $\tau_7 = +1 \implies \mathbf{x} \models \mathcal{G}_{I_5}(\mu_4)$. We then have the following equivalence,

$$(\mathbf{x}, t) \models \varphi \Leftrightarrow \rho(\varphi, \mathbf{x}, t) \geq 0 \Leftrightarrow \tau(\text{root}) = +1. \quad (4)$$

Before we proceed to the problem formulation, we need the following definition:

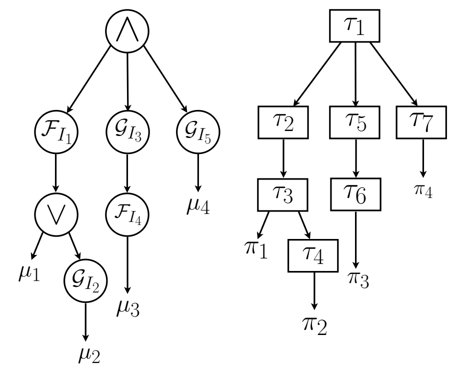


Fig. 1: STL parse tree and satisfaction variable tree for the formula in (3).

Definition 1 ([27]): The time horizon 'th(φ)' of an STL formula φ is recursively defined as,

$$\mathrm{th}(\varphi) = \begin{cases} 0, & \text{if } \varphi = \mu \\ \mathrm{th}(\varphi_1), & \text{if } \varphi = \neg\varphi_1 \\ \max\{\mathrm{th}(\varphi_1), \mathrm{th}(\varphi_2)\}, & \text{if } \varphi = \varphi_1 \wedge \varphi_2 \\ b + \max\{\mathrm{th}(\varphi_1), \mathrm{th}(\varphi_2)\}, & \text{if } \varphi = \varphi_1 \mathcal{U}_{[a,b]} \varphi_2. \end{cases} \tag{5}$$

D. Problem Formulation

In this paper, we consider the motion-planning problem of two autonomous agents subject to a coupled task expressed as an STL formula φ of the form (1). We assume that the agents evolve in $\mathbb{R}^n$, with $n \in \mathbb{N}$, characterized by the state trajectories $\mathbf{x}^i : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$, for $i \in \{1, 2\}$. More specifically, the problem we consider is the design of time-varying trajectories $y_i : [0, \infty) \to \mathbb{R}^n$, starting at the initial configurations $y_i(0) = \mathbf{x}^i(0)$, $i \in \{1, 2\}$, that satisfy a user-defined coupled STL task φ, i.e., $(y, 0) \models \varphi$. with $y := [y_1^\top, y_2^\top]^\top$.

We currently do not focus on dynamics of the agent or the explicit control design that guarantees tracking of the trajectory $y_i(t)$; we assume that the agents are endowed with a control algorithm that allows such tracking. Instead, we focus on the planning problem, i.e., the derivation of continuous time-varying trajectories that satisfy φ. Unlike previous works in the related literature (e.g., [16]), we consider the entire fragment of STL and develop a distributed and efficient sampling-based algorithm.

III. Sampling-Based Planning under STL

This section provides the proposed solution to the co-operative STL planning problem. The solution consists of a variation of the standard Rapidly-exploring Random Tree (RRT) algorithm [28] and it derives a time-varying trajectory $y(t)$ that satisfies φ. The key element is the incorporation of time in the sampling process, leading to a *time-augmented* sampling method. Compared to the original sampling-based algorithms, we replace the obstacle collision-checking procedure with a procedure that checks whether a specific fragment of the constructed tree violates the given STL formula. The aforementioned properties allow the direct derivation of a time-varying trajectory $y(t)$ that satisfies the given STL task encoded in φ. This problem poses additional challenges since it involves determining satisfaction of φ, which is defined over a continuous signal, using only point-wise data. We propose the following algorithm, called STLcoRRT, to address these challenges.

The STLcoRRT algorithm comprises two main components, namely the sampling-based procedure to build a tree, and the STL verification procedure that decides the satisfaction of a STL formula. Each sample corresponds to a point $z = (t, \mathbf{x}) \in \mathcal{Z} \subset \mathbb{R}_+ \times \mathbb{R}^n$, which the STL verification procedure decides if it is feasible with respect to the STL formula φ; if yes, it approves it and adds it as a vertex to the tree. Every added edge is verified against a formula and a path traversing any edge ensures the satisfaction of φ by construction. The termination of the STLcoRRT algorithm guarantees the generation of a trajectory that satisfies the formula.

In what follows, we introduce first the sampling-based procedure in Section III-A and then the overall STLcoRRT algorithm in Section III-B.

A. Sampling-based Procedure

The traditional RRT algorithm samples a point $\mathbf{x}_{\text{samp}}$ from $\mathbb{R}^n$, finds its nearest neighbour $\mathbf{x}_{\text{nearest}}$ from the existing tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, and draws an edge of predefined length *step* to create a new vertex $\mathbf{x}_{\text{new}}$. The inputs to the RRT algorithm are a starting vertex $\mathbf{x}(0)$, representing the initial condition of a path, a goal region, signifying the termination of the algorithm (once $\mathbf{x}_{\text{new}}$ is sampled in the goal region), and a hyper-parameter *step* representing the edge length. An edge from $\mathbf{x}_{\text{nearest}}$ to $\mathbf{x}_{\text{new}}$ is added to $\mathcal{E}$ in the direction of $\mathbf{x}_{\text{samp}}$. The Euclidean distance between $\mathbf{x}_{\text{new}}$ and $\mathbf{x}_{\text{nearest}}$ is user defined and denoted by *step*. The STLcoRRT algorithm, as opposed to the traditional RRT algorithm, generates a spatio-temporal tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ the set of edges. Each vertex in the set $\mathcal{V} = \{z_1, z_2, \dots\}$ is given by $z_i = (t_i, \mathbf{x}_i) \in \mathcal{Z} \subset \mathbb{R}_+ \times \mathbb{R}^n$. By sampling in time, the termination of STLcoRRT generates a trajectory in space and time while RRT only returns a path in space. The continuous sample space $\mathcal{Z}$ is predefined based on the range of variables $\mathbf{x}_i$ and the time horizon of the STL formula as defined in Definition 1. We assume the states $\mathbf{x}_i$ are sampled from a closed set. Another distinction is that in STLcoRRT, a vertex $z_{\text{new}} = (t_{\text{new}}, \mathbf{x}_{\text{new}})$ is sampled if and only if $t_{\text{new}} > t_{\text{nearest}}$ and subsequently any z_{new} mentioned henceforth is considered to be ahead in time with respect to z_{nearest}. Along with this restriction, every eligible z_{new} goes through a STL verification process deciding if the vertex could be a satisfactory point in the final trajectory satisfying $y(t) \models \varphi$.

B. Overall Algorithm

To solve the coupled motion planning problem for two cooperating agents, agent i and agent j, we partition the algorithm into three modules. From the point of view of agent i, the first module is about sampling in a time region where agent j has already explored, i.e. Algorithm 3: `PastAgent`. The second module is sampling in a time region where agent

j is yet to explore, i.e. Algorithm 2: `FutureAgent`. And the final module is verification of the STL formula based on the sampled points, i.e. Algorithm 4: `EdgeApproval`. For the discussions below, we use index i to refer to an agent that is currently running the pseudo-code and index j for the other agent. We now describe the algorithm and modules in detail.

1) Algorithm 1 - STLcoRRT: The main body of STLcoRRT starts with the initialisation of the tree $\mathcal{T}_i$ (not to be confused with the STL parse tree and the satisfaction variable tree) with the initial conditions $z_0^i = (0, \mathbf{x}^i(0))$ (line 1-5). In the pseudo-code, we denote the time component of a sample $z^i = (t^i, \mathbf{x}^i)$ as $z^i.t^i$ i.e., $t^i := z^i.t^i$. Agent i then requests the updated tree of agent j (line 8). The sampling takes place between lines 10-12 where the functions `Sample`, `Nearest` and `Steer` are standard and borrowed from the original RRT algorithm [28];

- `Sample` samples a random point $z^i_{\text{samp}} = (t^i_{\text{samp}}, \mathbf{x}^i_{\text{samp}})$ in $\mathcal{Z}$,
- `Nearest` finds the nearest node $z^i_{\text{nearest}} = (t^i_{\text{nearest}}, \mathbf{x}^i_{\text{nearest}})$ with respect to the Euclidean distance in $\mathcal{T}_i$ to the sampled point z^i_{samp}, and,
- `Steer` finds a node $z^i_{\text{new}} = (t^i_{\text{new}}, \mathbf{x}^i_{\text{new}})$ which is *step* units apart from z^i_{nearest} in the direction of z^i_{samp}.

The condition in line 9 imposes every sample to be sampled ahead in time compared to its nearest neighbour. This guarantees every path in the tree to only move forward in time. Each agent samples points sequentially and builds its tree after communicating with the other agent. We distinguish two scenarios in this sampling procedure: a new sample $z^i_{\text{new}} = (t^i_{\text{new}}, \mathbf{x}^i_{\text{new}})$ of one agent is at a point in time t^i_{new} when (1) the other agent j hasn't built a tree yet i.e. $t^i_{\text{new}} > \max\{t^j_k\},\ k \in \{1, \dots, |\mathcal{V}_j|\}$ or (2) the other agent j has existing edges passing through this time instant t^i_{new} i.e $t^i_{\text{new}} < \max\{t^j_k\},\ k \in \{1, \dots, |\mathcal{V}_j|\}$. In the former case, each agent uses the function `FutureAgent` and in the latter case each agent uses the function `PastAgent` to decide the satisfiability of φ. This process is repeated until $z^i \in \mathcal{Z}^i_{\text{goal}}$ and terminates by finding the trajectory $path_i$. The goal region is $\mathcal{Z}^i_{\text{goal}} = \{z^i \in \mathcal{Z} \mid t^i \geq \text{th}(\varphi)\}$. Both the agents simultaneously run STLcoRRT and communicate with the other agent after every edge addition. We further need the following definition

Definition 2 (Vertex Incidence): A vertex z^j_{new} is said to be incident on an edge $\{z^i_{\text{nearest}}, z^i_{\text{new}}\}$ if and only if $t^i_{\text{nearest}} < t^j_{\text{new}} < t^i_{\text{new}}$.

Before we proceed with the discussion of the rest of the modules, we focus on how to evaluate an STL formula φ using sampled points. Recall that STL is a predicate-based logic and predicates are defined over predicate functions which are functions of states of both agents. To evaluate if a sampled point z^i_{new} could be a potential point in the final satisfiable path for formula φ, we require points of both agent states $z^i_{\text{new}} = (t_{\text{new}}, \mathbf{x}^i_{\text{new}})$ and $z^j_{\text{new}} = (t_{\text{new}}, \mathbf{x}^j_{\text{new}})$ at the same time instant t_{new}. The pair $\{z^i_{\text{new}}, z^j_{\text{new}}\}$ is used to evaluate the predicate and therein the formula φ. Suppose the edge resultant from adding z^i_{new} to $\mathcal{T}_i$ is $\{z^i_{\text{nearest}}, z^i_{\text{new}}\}$. As

Algorithm 1: STLcoRRT: Agent i perspective

```
   Input: $\varphi$, $z_0^i$, $step$, $L$, $\mathcal{Z}^i_{\text{goal}}$, and $\mathcal{Z}$
   Output: $\mathcal{T}_i$ and $path_i$
1  $\mathcal{V}_i \leftarrow \{z_0^i\}$;
2  $\mathcal{E}_i \leftarrow \emptyset$;
3  $\mathcal{T}_i \leftarrow (\mathcal{V}_i, \mathcal{E}_i)$;
4  $t_0 \leftarrow 0$;
5  $path_i \leftarrow \emptyset$;
6  $\tau \leftarrow \{-1\}$;
7  while $z^i \notin \mathcal{Z}^i_{goal}$ do
8      $\mathcal{T}_j \leftarrow$ Agent $j$;
9      while $z^i_{new}.t < z^i_{nearest}.t$ do
10         $z^i_{\text{samp}} \leftarrow$ Sample($\mathcal{Z}$);
11         $z^i_{\text{nearest}} \leftarrow$ Nearest($\mathcal{T}_i$, $z^i_{samp}$);
12         $z^i_{\text{new}} \leftarrow$ Steer($z^i_{nearest}$, $z^i_{samp}$, $step$);
13     if $z^i_{new}.t > \max(\mathcal{V}_j.t)$ then
14         $\mathcal{T}_i, \mathcal{T}_j \leftarrow$ FutureAgent ($\varphi$, $step$, $L$, $z^i_{\text{new}}$,
             $z^i_{\text{nearest}}$, $\tau$, $\mathcal{T}_i$, $\mathcal{T}_j$);
15     else
16         $\mathcal{T}_i \leftarrow$ PastAgent ($\varphi$, $L$, $z^i_{\text{new}}$, $z^i_{\text{nearest}}$, $\tau$, $\mathcal{T}_i$,
             $\mathcal{T}_j$);
17 $path_i \leftarrow path_i \cup$ PlotPath ($\mathcal{T}_i$)
```

in obstacle avoidance problems, where an edge connecting two non-collision vertices can still intersect an obstacle, we follow the same principle and evaluate points on the edge to ensure such points do not violate φ. We pick $L \in \mathbb{N}$ points on the edge $\{z^i_{\text{nearest}}, z^i_{\text{new}}\}$ as

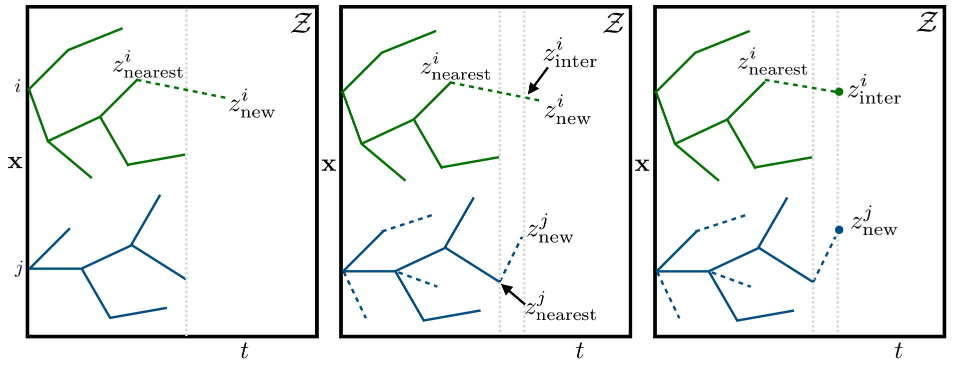
$$z^i = z^i_{\text{nearest}} + l(z^i_{\text{new}} - z^i_{\text{nearest}}),$$

where $l = 0, \frac{1}{L-1}, \frac{2}{L-1}, \dots, \frac{L-2}{L-1}, 1$. Each z^i corresponds to z^j via $z^i.t = z^j.t$ (discussed in Section III-B.2 and III-B.3 below) over which we evaluate φ. The hyper-parameter L specifies the discretisation of the edge.

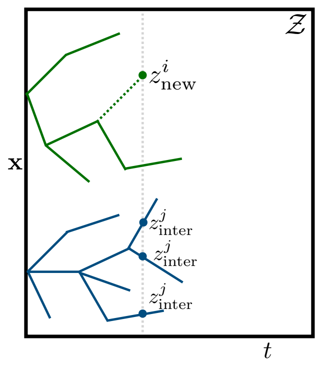
2) Function 2 - `FutureAgent`: This function is called when agent i samples a point $z^i_{\text{new}} = (t^i_{\text{new}}, \mathbf{x}^i_{\text{new}})$ where $t^i_{\text{new}} > t^j_k$ for all $k \in (1, 2, \dots, |\mathcal{V}_j|)$ as seen in Figure 2a (left). It starts by temporarily adding the sampled vertex z^i_{new} to the tree of agent i (lines 1-2). Next, it communicates this $\{z^i_{\text{nearest}}, z^i_{\text{new}}\}$ to agent j and agent j samples indefinitely until a sampled point lies incident to the temporarily added edge of agent i, lines 3-6 and seen in Figure 2a (center). Then, the algorithm linearly interpolates in line 7 to obtain the vertex $z^i_{\text{inter}} = (t^j_{\text{new}}, \mathbf{x}^i_{\text{inter}})$ as seen in Figure 2a (right). This is obtained by solving for $\mathbf{x}^i_{\text{inter}}$ element-wise as the solution of,

$$\mathbf{x}^i_{\text{inter}} = \Big(\frac{\mathbf{x}^i_{\text{nearest}} - \mathbf{x}^i_{\text{new}}}{t^i_{\text{nearest}} - t^i_{\text{new}}}\Big)(t^j_{\text{new}} - t^i_{\text{new}}) + \mathbf{x}^i_{\text{new}}.$$

We define a function `Interpolate` that solves the aforementioned computation. We can now sample points over the edge $\{z^i_{\text{nearest}}, z^i_{\text{inter}}\}$ and all the corresponding interpolated points from $\mathcal{T}_j$ to evaluate the STL formula through the `EdgeApproval` function, depicted in Function 4 and discussed below. By $L \in \mathbb{N}$ (see Section III-B.1) we chose the



(a) `FutureAgent` illustration (i) Left: Agent i adds an edge where Agent j has no incident edges (lines 1-2), (ii) Center: Agent j samples until it finds a vertex incident to Agent i's new edge (lines 3-6), (iii) Right: Agent i interpolates to create a vertex pain $\{z^i_{\text{inter}}, z^j_{\text{new}}\}$ (line 7).



(b) `PastAgent` illustration: Agent i samples z^i_{new} and interpolates in line 7 to find all incident Agent j samples.

Fig. 2: Illustration of `FutureAgent` and `PastAgent`.

number of equally spaced points on the edge $\{z^i_{\text{nearest}}, z^i_{\text{inter}}\}$ in the evaluation of φ. The new vertex z^i_{new} of agent i is deleted if it does not satisfy the STL formula (lines 13-14).

Function 2: FutureAgent

```
Input: $\varphi$, $step$, $L$, $z^i_{\text{new}}$, $z^i_{\text{nearest}}$, $\tau$, $\mathcal{T}_i$, and $\mathcal{T}_j$
Output: $\mathcal{T}_i$ and $\mathcal{T}_j$
1  $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup z^i_{\text{new}}$;
2  $\mathcal{E}_i \leftarrow \mathcal{E}_i \cup \{z^i_{\text{nearest}}, z^i_{\text{new}}\}$;
3  while $z^i_{new}.t < z^i_{nearest}.t$ or $z^j_{new}.t \notin (z^i_{nearest}.t, z^i_{new}.t)$
     do
4      $z^j_{\text{samp}} \leftarrow$ Sample($\mathcal{Z}$);
5      $z^j_{\text{nearest}} \leftarrow$ Nearest($\mathcal{T}_j, z^j_{samp}$);
6      $z^j_{\text{new}} \leftarrow$ Steer($z^j_{nearest}, z^j_{samp}, step$);
7  $z^i_{\text{inter}} \leftarrow$ Interpolate($\{z^i_{nearest}, z^i_{new}\}, z^j_{new}.t$);
8  for $l \leftarrow 0$ to $L$ by $1/(L-1)$ do
9      $z^i = z^i_{\text{nearest}} + l(z^i_{\text{inter}} - z^i_{\text{nearest}})$;
10     $z^j \leftarrow$ Interpolate($\{z^j_{nearest}, z^j_{new}\}, z^i.t$);
11     $\pi, \tau \leftarrow$ EdgeApproval($\varphi, z^i, z^j, \tau$);
12     if $!\pi$ then
13         $\mathcal{V}_i \leftarrow \mathcal{V}_i \backslash z^i_{\text{new}}$;
14         $\mathcal{E}_i \leftarrow \mathcal{E}_i \backslash \{z^i_{\text{nearest}}, z^i_{\text{new}}\}$;
15         return $\mathcal{T}_i$ and $\mathcal{T}_j$
16 $\mathcal{V}_j \leftarrow \mathcal{V}_j \cup z^j_{\text{new}}$;
17 $\mathcal{E}_j \leftarrow \mathcal{E}_j \cup \{z^j_{\text{nearest}}, z^j_{\text{new}}\}$;
18 $\mathcal{V}_i \leftarrow \mathcal{V}_i \backslash z^i_{\text{new}}$;
19 $\mathcal{E}_i \leftarrow \mathcal{E}_i \backslash \{z^i_{\text{nearest}}, z^i_{\text{new}}\}$;
20 $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup z^i_{\text{inter}}$;
21 $\mathcal{E}_i \leftarrow \mathcal{E}_i \cup \{z^i_{\text{nearest}}, z^i_{\text{inter}}\}$;
```

3) Function 3 - `PastAgent`: This function, depicted in Function 3, is called when an agent i samples a point $z^i_{\text{new}} = (t^i_{\text{new}}, x^i_{\text{new}})$ and when there is at least one edge in $\mathcal{T}_j$ that is being incident to z^i_{new}; an illustration is shown in Figure 2b; `PastAgent` begins by evaluating z^j_{inter} from the incident edges using linear interpolation as seen in lines 1-9 . In line 7, as in the case of `FutureAgent`, we chose L samples over the edge $\{z^i_{\text{nearest}}, z^i_{\text{new}}\}$ and find the corresponding interpolated z^j_{inter}'s from $\mathcal{T}_j$ to evaluate `EdgeApproval`. If all the pairs result in $\pi = \top$, then the node z^i_{new} and the edge $\{z^i_{\text{nearest}}, z^i_{\text{new}}\}$ are added to the tree $\mathcal{T}_i$, see lines 5-11.

Function 3: PastAgent

```
Input: $\varphi$, $L$, $z^i_{\text{new}}$, $z^i_{\text{nearest}}$, $\tau$, $\mathcal{T}_i$, and $\mathcal{T}_j$
Output: $\mathcal{T}_i$
1  foreach $\{z^j_m, z^j_{m+1}\} \in \mathcal{E}_j$ do
2      $t^j_m \leftarrow z^j_m.t$;
3      $t^j_{m+1} \leftarrow z^j_{m+1}.t$;
4      if $t^j_m < t^i_{new} < t^j_{m+1}$ then
5          for $l \leftarrow 0$ to $L$ by $1/(L-1)$ do
6              $z^i = z^i_{\text{nearest}} + l(z^i_{\text{new}} - z^i_{\text{nearest}})$;
7              $z^j \leftarrow$ Interpolate($\{z^j_m, z^j_{m+1}\}, z^i.t$);
8              $\pi, \tau \leftarrow$ EdgeApproval($\varphi, z^i, z^j, \tau$);
9              if $!\pi$ then return $\mathcal{T}_i$;
10 $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup z^i_{\text{new}}$;
11 $\mathcal{E}_i \leftarrow \mathcal{E}_i \cup \{z^i_{\text{nearest}}, z^i_{\text{new}}\}$;
```

4) Encoding STL formula into the algorithm via Function 4 (`EdgeApproval`): Function 4 is recursively defined and takes as input an STL formula of the form (1), a pair of points $\{z^i, z^j\}$ from the trees of both agents, and returns a truth value π to determine if the vertices satisfy the STL formula. Along with π, it returns a variable $\tau(\varphi)$ taking values in the set $\{-1, +1\}$ indicating satisfaction of the STL sub-formula φ. The satisfaction of an STL formula cannot be determined by an individual point $z^i_{\text{new}} = (t^i_{\text{new}}, x^i_{\text{new}})$. However, a point z^i_{new} along with z^j_{new} can momentarily determine the satisfaction at that particular instant in time, since the formula consists of both temporal and spatial variables. Next, we present a set of rules for adding a a pair of sampled points $z^i_{\text{new}}, z^j_{\text{new}}$ to the tree based on the STL formula φ in (1).

- $\varphi = \top$: Any sampled point z^i_{new} is added to the tree.
- $\varphi = \mu$: A sampled point z^i_{new} is added to the tree, if and only if $\mu = \top$ with,

$$\mu = \begin{cases} \top & h(\mathbf{x}^i_{\text{new}}, \mathbf{x}^j_{\text{new}}, t_{\text{new}}) \geq 0 \\ \bot & h(\mathbf{x}^i_{\text{new}}, \mathbf{x}^j_{\text{new}}, t_{\text{new}}) < 0 \end{cases},$$

where the predicate function $h(\mathbf{x}^i_{\text{new}}, \mathbf{x}^j_{\text{new}}, t_{\text{new}})$ may encapsulate coupled constraints on states of agents $i,\ j$.

- $\neg\varphi$: A sampled point z^i_{new} is added to the tree if and only if it is not added to the tree for φ.
- $\varphi = \varphi_1 \vee \varphi_2$: A sampled point z^i_{new} is added to the tree if it is added to the tree for φ_1 or φ_2 in the sense presented above. But the point-wise evaluation of an STL formula does not track history of satisfaction of parts of the formula. For example, consider the STL formula $\varphi = \varphi_1 \vee \varphi_2 = (\mathcal{G}_{[1,2]}\mu_1 \wedge \mathcal{F}_{[4,5]}\mu_2) \vee (\mathcal{G}_{[1,2]}\mu_3 \wedge \mathcal{F}_{[4,5]}\mu_4)$ and notice that `EdgeApproval` will make no distinction between the said formula and $(\mathcal{G}_{[1,2]}\mu_1 \wedge \mathcal{F}_{[4,5]}\mu_4) \vee (\mathcal{G}_{[1,2]}\mu_3 \wedge \mathcal{F}_{[4,5]}\mu_2)$. To avoid such a scenario, the algorithm is designed to work as follows: If, in the construction of the tree, a sampled point satisfies any sub-formula of φ_1, then we deactivate φ_2 and if a sampled point satisfies any sub-formula of φ_2, then we deactivate φ_1. In the pseudo-code, if a child node of φ_1, denoted by $\varphi_1 >$, satisfies $\tau(\varphi_1 >) = +1$, then $\varphi_1 \vee \varphi_2 = \varphi_1 \vee \bot$. And if a child node of φ_2, denoted by $\varphi_2 >$, satisfies $\tau(\varphi_2 >) = +1$, then $\varphi_1 \vee \varphi_2 = \bot \vee \varphi_2$. In the example presented above, if $z((0,2),\mathbf{x}) \models \mathcal{G}_{[1,2]}\mu_1$, then $\varphi = (\mathcal{G}_{[1,2]}\mu_1 \wedge \mathcal{F}_{[4,5]}\mu_2) \vee \bot$ and if $z((0,2),\mathbf{x}) \models \mathcal{G}_{[1,2]}\mu_3$, then $\varphi = \bot \vee (\mathcal{G}_{[1,2]}\mu_3 \wedge \mathcal{F}_{[4,5]}\mu_4)$. It may appear that by formulating disjunction as presented above we deliberately make a choice of satisfying either φ_1 or φ_2. The algorithm is designed such that φ_1 or φ_2 is satisfied depending on whether a sub-formula from φ_1 or φ_2 is satisfied prior. The choice of sub-formula from φ_1 or φ_2 is completely random and depends on the evolution of the tree. Thus the choice of φ_1 or φ_2 is still random.
- $\varphi = \varphi_1 \wedge \varphi_2$: A sampled point z^i_{new} is added to the tree if it is added to the tree for φ_1 and φ_2.
- $\varphi = \mathcal{F}_I\varphi$: In the case of an eventually operator, it is possible that a single sample z^i_{new} could satisfy $\mathcal{F}_I\varphi$. And it is also possible that such a point might not be a part of the final trajectory. Thus, to ensure this satisfying point is included in our final trajectory we first extract and store the path connecting z^i_o and z^i_{new}. The trees of both agents are then reset and only the nodes z^i_{new} and z^j_{new} are added to $\mathcal{V}_i$ and $\mathcal{V}_j$ respectively. By doing this, any final trajectory will pass through z^i_{new} as it is the only sample connecting the prior tree and the new tree, lines 21-26. For $I = [a,b]$, if $z^i_{\text{new}}.t < b$ the sampled point z^i_{new} is added to the tree. Any sampled point where $z^i_{\text{new}}.t > b$ is not added to the tree, this stops the tree from growing beyond $z^i_{\text{new}}.t > b$ until we find a satisfying point. Once a point is sampled that satisfies φ, we remove the hold on tree expansion i.e. the tree is allowed to expand beyond $z^i_{\text{new}}.t > b$. Once $\tau(\varphi) = +1$, then this implies that $\tau(\mathcal{F}_I\varphi) = +1$ for the sample z^i_{new}. Note that if $\varphi = \mu$, then $\tau(\varphi) = +1 \Leftrightarrow \mu = \top$.
- $\varphi = \mathcal{G}_I\varphi$: If $z^i_{\text{new}}.t \in [a,b]$, the sampled point z^i_{new} is added to the tree if $\tau(\varphi) = +1$. If $t_{\text{new}} \notin [a,b]$ then z^i_{new} is added to the tree. This means the algorithm does not proceed if it does not satisfy $\mathcal{G}_I\varphi$. Again note $\tau(\varphi) = +1 \Leftrightarrow \mu = \top$ if $\varphi = \mu$.
- $\varphi = \varphi_1 \mathcal{U}_I \varphi_2$: The until operator is treated as $\mathcal{G}_{[z_{\text{new}}.t, z_{\text{new}}.t]}\varphi_1$ until $\tau(\mathcal{F}_{[z_{\text{new}}.t, z_{\text{new}}.t]}\varphi_1) = +1$. The interval is a singleton set since the satisfaction instance of φ_2 cannot be assessed apriori.

Function 4: EdgeApproval

```
Input: $\varphi$, $z^i_{\text{new}}$, $z^j_{\text{new}}$, $\tau$
Output: $\pi, \tau$
switch $\varphi$ do
    case $\top$ do
        return $\top, \tau$
    case $\mu$ do
        if $\mu$ then return $\top, \tau$;
        else return $\bot, \tau$;
    case $\neg\varphi$ do
        return $\neg$EA $(\varphi, \tau)$
    case $\varphi_1 \vee \varphi_2$ do
        if $z^i_{new}.t \in t_0 \oplus I$ and EA$(\varphi_1 >, \tau) = (\cdot, +1)$ then
            return EA $(\varphi_1, \tau) \vee$ EA $(\bot, \tau)$;
        else if $z^i_{new}.t \in t_0 \oplus I$ and EA$(\varphi_2 >, \tau) = (\cdot, +1)$ then
            return EA $(\bot, \tau) \vee$ EA $(\varphi_2, \tau)$;
        else
            return EA $(\varphi_1, \tau) \vee$ EA $(\varphi_2, \tau)$ ;
    case $\varphi_1 \wedge \varphi_2$ do
        return EA $(\varphi_1, \tau) \wedge$ EA $(\varphi_2, \tau)$
    case $\mathcal{F}_I\varphi$ do
        if $z^i_{new}.t \in t_0 \oplus I$ and EA$(\varphi, \tau) = (\top, \cdot)$ then
            $\tau \leftarrow +1$ ;
            $path_i \leftarrow$ PlotPath $(\mathcal{T}_i)$;
            $path_j \leftarrow$ PlotPath $(\mathcal{T}_j)$;
            $\mathcal{V}_i \leftarrow \emptyset$, $\mathcal{E}_i \leftarrow \emptyset$;
            $\mathcal{V}_j \leftarrow \emptyset$, $\mathcal{E}_j \leftarrow \emptyset$;
            $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup z^i_{\text{new}}$;
            $\mathcal{V}_j \leftarrow \mathcal{V}_j \cup z^j_{\text{new}}$;
            return EA$(\varphi, \tau)$ ;
        else if $z^i_{new}.t \in t_0 \oplus I$ and EA$(\varphi, \tau) = (\bot, \cdot)$ then
            return $\neg$ EA$(\varphi, \tau)$ ;
        else
            return EA $(\top, \tau)$ ;
    case $\mathcal{G}_I\varphi$ do
        if $z^i_{new}.t \in t_0 \oplus I$ then
            return EA$(\varphi, \tau$ )
        else
            if $z^i_{new}.t > I.2$ then $\tau \leftarrow +1$;
            return EA$(\top, \tau)$
    case $\varphi_1 \mathcal{U}_I \varphi_2$ do
        while EA $(\mathcal{F}_I\varphi_2, \tau) \neq (\cdot, +1)$ do
            return EA $(\mathcal{G}_{[z^i_{\text{new}}.t, z^i_{\text{new}}.t]}\varphi_1, \tau)$
        return EA $(\mathcal{F}_{[z^i_{\text{new}}.t, z^i_{\text{new}}.t]}\varphi_2, \tau)$
/* EdgeApproval is abbreviated as EA for
   readability purposes                          */
```

The function `EdgeApproval` is recursively defined over (1) and hence covers the entire STL formula. Below we see an example of how `EdgeApproval` treats nested temporal operators.

Example 1: Let $\varphi = \mathcal{G}_{[0,20]}\mathcal{F}_{[1,3]}\mu$. Formula φ requires μ to hold at least every two seconds in the interval $[1,23]s$.

Recalling EdgeApproval(φ), once $\mu = \top$ in the interval $[1,3]s$, line 20 in Function 4 returns $\tau(\mathcal{F}_{[1,3]}\mu) = +1$ and the initial conditions are reset to $z_o \leftarrow z_{\text{new}}$. This ensures $t_o \leftarrow z_{\text{new}}.t$ and the new interval is $[1 + z_{\text{new}}.t, 3 + z_{\text{new}}.t]$. The next case called is $\mathcal{G}_I\varphi$ which returns $\mathcal{G}_I(\top)$ in line 34 and EdgeApproval returns $(\top, +1)$. This process repeats until the entire interval $[0, 23]s$ is covered.

Once the algorithm terminates i.e. when the tree reaches the designated goal region, a path $(y_1(t), y_2(t))$ is generated by PlotPath as done in the standard RRT [28]. The paths $(y_1(t), y_2(t))$ satisfy the STL formula by construction since every node in the tree satisfies the STL formula via the EdgeApproval function. The generated trajectories can be verified using the robust semantics presented in Section II-B by evaluating for $\rho(\varphi, \mathbf{x}, t)$. STLcoRRT, unlike monitoring algorithms, generates trajectories that satisfy the STL formula by construction. The inter-dependency of coupled tasks between agents poses a significant challenge in solving the problem in a distributed way while covering the entire STL formula rather than a fragment of it. We believe that the coupled-agents approach treated here is a first step towards solving the more general multi-agent motion planning under STL task problem.

Complexity, Completeness and Scalability: The time complexity is the sum of individual time complexities of the functions in STLcoRRT, i.e. $\mathcal{C}_{\text{STLcoRRT}}(N) = \mathcal{C}_{\text{Sample}}(N) + \mathcal{C}_{\text{Nearest}}(N) + \mathcal{C}_{\text{Steer}}(N) + \mathcal{C}_{\text{FutureAgent}}(N) + \mathcal{C}_{\text{PastAgent}}(N)$. The time complexity of $\mathcal{C}_{\text{Sample}}(N) + \mathcal{C}_{\text{Nearest}}(N) + \mathcal{C}_{\text{Steer}}(N)$ is $\mathcal{O}(N) + \mathcal{O}(N) + \mathcal{O}(N) \approx \mathcal{O}(N)$ [29]. Each sampled vertex belongs to either FutureAgent or PastAgent. Thus the time complexity of FutureAgent and PastAgent is the worst case time complexity of either one of them. The time complexity of FutureAgent is $\mathcal{C}_{\text{FutureAgent}}(N) = \mathcal{C}_{\text{Interpolate}}(N) + \mathcal{C}_{\text{Interpolate}}(LN) + \mathcal{C}_{\text{EdgeApproval}}(N) = \mathcal{O}(N) + \mathcal{O}(LN) + \mathcal{O}(N \cdot \log(N)) \approx \mathcal{O}(N \cdot \log(N))$. Similarly $\mathcal{C}_{\text{PastAgent}}(N) \approx \mathcal{O}(N \cdot \log(N))$. Thus the worst case time complexity of FutureAgent and PastAgent is still $\mathcal{O}(N \cdot \log(N))$. Finally, the time complexity of $\mathcal{C}_{\text{STLcoRRT}}(N) \approx \mathcal{O}(N \cdot \log(N))$. The space complexity for individual agents is the size of the stored tree and is thus $\mathcal{O}(N)$ where N is the number of vertices. The STLcoRRT algorithm inherits the probabilistic completeness property from the standard sampling algorithm RRT. The interpolation of temporarily added edges in FutureAgent ensures such completeness since non-feasible vertices aren't added to the tree. As with sampling based methods, we can take various steps to speed up the algorithm such as:

- Changing the *step* to a larger value for high dimensions;
- A maximum number of nodes can be specified in Algorithm 1 by changing Line 7 to a for loop;
- An iteration limit can be imposed on the second condition in Line 3 of Function 2. Since in some cases it could be too narrow to sample in $(z^{\text{nearest}}.t, z^i_{\text{new}}.t)$.

IV. Simulations

This section presents some simulation examples demonstrating the functioning of the STLcoRRT algorithm. We start with simulating examples from the fragment $\varphi = \mu|\mathcal{G}_{[a,b]}\mu|\mathcal{F}_{[a,b]}\mu|\varphi_1 \wedge \varphi_2$. All simulations are performed on an eight core 1.9 GHz Intel-core i7 CPU with 16GB of RAM. Figure 3a simulates the case when $\varphi = |x_1 - x_2| > 5$, requiring the agents to be 5 units apart at all times. Agent 1 (in blue) and Agent 2 (in green) build their trees ensuring that every added vertex is atleast 5 units away from all incident vertices of the other agent such that any final trajectory will satisfy φ. Figure 3b simulates the formula $\varphi = \mathcal{G}_{[4,6]}|x_1 - x_2| < 2$ requiring in the interval $[4, 6]$ units, every added vertex is less than two units apart with respect to the other agent. The trees are built satisfying this constraint in the interval $[4, 6]$. In Figure 3c, we simulate the formula $\varphi = \mathcal{F}_{[a,b]}|x_1 - x_2| > 8$ requiring the agents to be 8 units apart at any one instant in the interval $[4, 6]$. All sampled vertices $(z^i_{\text{new}}, z^j_{\text{new}})$ are added to the tree until $|x_1 - x_2| > 8$ becomes true, indicated by red crosses in Figure 3c. Once the condition $|x_1 - x_2| > 8$ holds, the corresponding vertices are set as new starting vertices for the STLcoRRT algorithm and are thus included in the final path. Figure 3d requires $x_1 < 1$ in the interval $[2, 8]$ and $|x_1 - x_2| < 2$ in the interval $[4, 6]$. This is evident from the figure as Agent 1 samples only in the set $\{x_1 < 1\}$ in the interval $[2, 8]$ and Agent 2 comes closer satisfying $|x_1 - x_2| < 2$ in the interval $[4, 6]$.

Next, we present a disjunction example, let $\varphi = \mathcal{F}_{[4,6]}|x_1 - x_2| > 8 \vee \mathcal{G}_{[6,8]}|x_1 - x_2| < 2$. In Figure 4a (Left), we see that Algorithm 1 chose to satisfy the subformula $\mathcal{F}_{[4,6]}|x_1 - x_2| > 8$ while in other run $\mathcal{G}_{[6,8]}|x_1 - x_2| < 2$ was satisfied as seen in Figure 4a (Right). For the case of a nested formula, $\varphi = \mathcal{G}_{[0,10]}\mathcal{F}_{[1,3]}|x_1 - x_2| > 8$, which dictates a task to *'satisfy* $|x_1 - x_2| > 8$ *every 2 seconds in the interval* $[1, 13]$*'*. As seen in Figure 4b, each red cross represents the satisfaction of $\mathcal{F}_{[\star^1, \star^2]}$ where $[\star^1, \star^2]$ are set to $[1, 3]$ at $t = 0$ and re-evaluated after every satisfaction of the eventually.

V. Conclusion

This paper presents an efficient sampling-based algorithm for cooperative motion planning between two agents under STL specifications. We augment the traditional RRT method to generate spatio-temporal trees through which we enforce space and time constraints specified by the STL. The proposed algorithm is distributed as agents independently build their trees by communicating with each other. For future work, we will consider robust STL symantics and extend the existing framework to multiple cooperating agents.

References

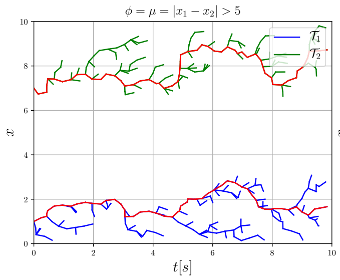
[1] A. Donzé, "On signal temporal logic," *International Conference on Runtime Verification*, pp. 382–383, 2013.

[2] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for mobile robots," *IEEE International Conference on Robotics and Automation*, pp. 2020–2025, 2005.

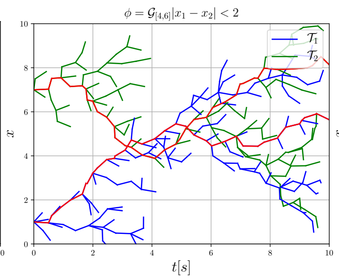
[3] J. Ouaknine and J. Worrell, "On the decidability of metric temporal logic," *IEEE Symposium on Logic in Computer Science (LICS'05)*, pp. 188–197, 2005.

[4] P. Bouyer, F. Laroussinie, N. Markey, J. Ouaknine, and J. Worrell, "Timed temporal logics," *Models, Algorithms, Logics and Tools*, pp. 211–230, 2017.

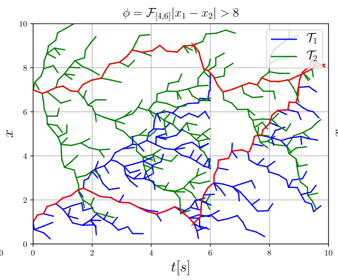
[5] C. K. Verginis, C. Vrohidis, C. P. Bechlioulis, K. J. Kyriakopoulos, and D. V. Dimarogonas, "Reconfigurable motion planning and control in obstacle cluttered environments under timed temporal tasks," *International Conference on Robotics and Automation*, 2019.



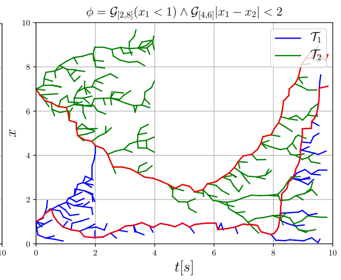
(a) $\varphi = |x_1 - x_2| > 5$



(b) $\varphi = \mathcal{G}_{[4,6]}(|x_1 - x_2| < 2)$

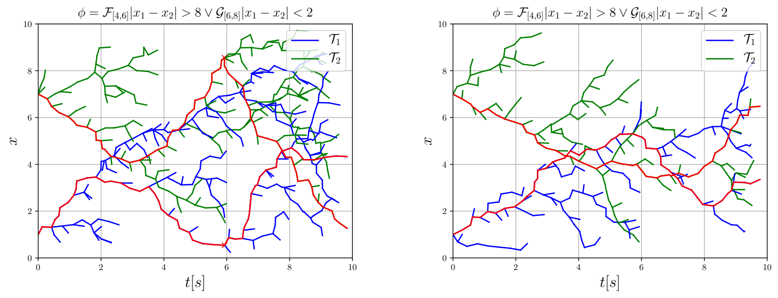


(c) $\varphi = \mathcal{F}_{[4,6]}(|x_1 - x_2| > 8)$

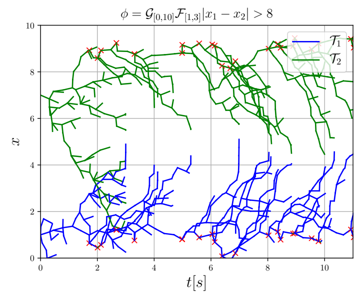


(d) $\varphi = \mathcal{G}_{[2,8]}(x_1 < 1) \wedge \mathcal{G}_{[4,6]}(|x_1 - x_2| < 2)$

Fig. 3: Trajectories generated by Algorithm 1 for a single predicate, always, eventually and conjunction operators.



(a) $\varphi = \mathcal{F}_{[4,6]}|x_1 - x_2| > 8 \vee \mathcal{G}_{[6,8]}|x_1 - x_2| < 2$



(b) $\varphi = \mathcal{G}_{[0,10]}\mathcal{F}_{[1,3]}|x_1 - x_2| > 8$

Fig. 4: Trajectories generated by Algorithm 1 for disjunction operator and nested formula.

[6] S. Karaman and E. Frazzoli, "Vehicle routing problem with metric temporal logic specifications," *IEEE conference on decision and control*, pp. 3953–3958, 2008.

[7] C. K. Verginis and D. V. Dimarogonas, "Timed abstractions for distributed cooperative manipulation," *Autonomous Robots*, vol. 42, no. 4, pp. 781–799, Apr. 2018.

[8] F. Fotiadis, C. K. Verginis, K. G. Vamvoudakis, and U. Topcu, "Assured learning-based optimal control subject to timed temporal logic constraints," *IEEE conference on decision and control*, 2021.

[9] A. Nikou, D. Boskos, J. Tumova, and D. V. Dimarogonas, "On the timed temporal logic planning of coupled multi-agent systems," *Automatica*, vol. 97, pp. 339–345, 2018.

[10] C. N. Mavridis, C. Vrohidis, J. S. Baras, and K. J. Kyriakopoulos, "Robot navigation under mitl constraints using time-dependent vector field based control," *IEEE Conference on Decision and Control (CDC)*, pp. 232–237, 2019.

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

[12] Y. V. Pant, H. Abbas, R. A. Quaye, and R. Mangharam, "Fly-by-logic: Control of multi-drone fleets with temporal logic objectives," *ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 186–197, 2018.

[13] N. Mehdipour, C.-I. Vasile, and C. Belta, "Arithmetic-geometric mean robustness for control from signal temporal logic specifications," *American Control Conference (ACC)*, pp. 1690–1695, 2019.

[14] L. Lindemann and D. V. Dimarogonas, "Efficient automata-based planning and control under spatio-temporal logic specifications," *American Control Conference (ACC)*, pp. 4707–4714, 2020.

[15] G. Yang, C. Belta, and R. Tron, "Continuous-time signal temporal logic planning with control barrier functions," *American Control Conference (ACC)*, pp. 4612–4618, 2020.

[16] L. Lindemann, C. K. Verginis, and D. V. Dimarogonas, "Prescribed performance control for signal temporal logic specifications," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, Dec. 2017, pp. 2997–3002.

[17] L. Lindemann and D. V. Dimarogonas, "Control barrier functions for multi-agent systems under conflicting local signal temporal logic tasks," *IEEE control systems letters*, vol. 3, no. 3, pp. 757–762, 2019.

[18] J. Karlsson, F. S. Barbosa, and J. Tumova, "Sampling-based motion planning with temporal logic missions and spatial preferences," vol. 53, no. 2, pp. 15 537–15 543, 2020.

[19] L. Lindemann and D. V. Dimarogonas, "Feedback control strategies for multi-agent systems under a fragment of signal temporal logic tasks," *Automatica*, vol. 106, pp. 284–293, 2019.

[20] D. Gundana and H. Kress-Gazit, "Event-based signal temporal logic synthesis for single and multi-robot tasks," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3687–3694, 2021.

[21] D. Sun, J. Chen, S. Mitra, and C. Fan, "Multi-agent motion planning from signal temporal logic specifications," *arXiv preprint arXiv:2201.05247*, 2022.

[22] A. T. Buyukkocak, D. Aksaray, and Y. Yazıcıoğlu, "Planning of heterogeneous multi-agent systems under signal temporal logic specifications with integral predicates," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1375–1382, 2021.

[23] I. Haghighi, S. Sadraddini, and C. Belta, "Robotic swarm control from spatio-temporal specifications," *IEEE Conference on Decision and Control (CDC)*, pp. 5708–5713, 2016.

[24] J. Hopcroft, J. Schwartz, and M. Sharir, "On the complexity of motion planning for multiple independent objects; pspace- hardness of the "warehouseman's problem"," *The International Journal of Robotics Research*, vol. 3, no. 4, pp. 76–88, 1984.

[25] C.-I. Vasile, V. Raman, and S. Karaman, "Sampling-based synthesis of maximally-satisfying controllers for temporal logic specifications," pp. 3840–3847, 2017.

[26] O. Maler and D. Nickovic, "Monitoring Temporal Properties of Continuous Signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, ser. Lecture Notes in Computer Science, Y. Lakhnech and S. Yovine, Eds. Berlin, Heidelberg: Springer, 2004, pp. 152–166.

[27] P. Yu and D. V. Dimarogonas, "Hierarchical control for uncertain discrete-time nonlinear systems under signal temporal logic specifications," in *2021 60th IEEE Conference on Decision and Control (CDC)*, 2021, pp. 1450–1455.

[28] S. M. LaValle *et al.*, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[29] Z. Zhang, D. Wu, J. Gu, and F. Li, "A path-planning strategy for unmanned surface vehicles based on an adaptive hybrid dynamic stepsize and target attractive force-rrt algorithm," *Journal of Marine Science and Engineering*, vol. 7, no. 5, 2019.