Optimization-based Task and Motion Planning under Signal Temporal Logic Specifications using Logic Network Flow

Xuan Lin[1], Jiming Ren[1], Samuel Coogan[2], and Ye Zhao[1]

***Abstract*— This paper proposes an optimization-based task and motion planning framework, named "Logic Network Flow", to integrate signal temporal logic (STL) specifications into efficient mixed-binary linear programmings. In this framework, temporal predicates are encoded as polyhedron constraints on each edge of the network flow, instead of as constraints between the nodes as in the traditional Logic Tree formulation. Synthesized with Dynamic Network Flows, Logic Network Flows render a tighter convex relaxation compared to Logic Trees derived from these STL specifications. Our formulation is evaluated on several multi-robot motion planning case studies. Empirical results demonstrate that our formulation outperforms Logic Tree formulation in terms of computation time for several planning problems. As the problem size scales up, our method still discovers better lower and upper bounds by exploring fewer number of nodes during the branch-and-bound process, although this comes at the cost of increased computational load for each node when exploring branches.**

I. Introduction

Task and motion planning (TAMP) with temporal logic provides formal guarantees for provably correct robot plans and task completion [1]. Particularly, TAMP with Signal Temporal Logic (STL) constraints is often formulated as an optimization problem solved by mixed-integer linear program (MILP) [2], [3]. However, this STL-based planning problem is theoretically intractable due to its NP-hard nature. In practice, although MILP solvers, *e.g.*, via branch and bound (B&B), can solve in a reasonable computation time, they still suffer from the worst-case (*i.e.*, exponential) complexity. To take a step toward circumventing these worst-case scenarios, this study presents a novel MILP formulation by transforming STL specifications into a form of network flow to render a tighter convex relaxation for the MILP. This formulation offers a promise in improving the efficiency of the B&B process, which can further facilitate the solve of STL-based optimization problems more efficiently.

STL offers an expressive task-specification language for specifying a variety of temporal tasks, and provides a powerful framework to integrate discrete and continuous actions. Planning of dynamical systems under STL specifications has been widely explored for robot manipulation [4], [5], locomotion [6], and multi-agent systems [3], [7]. As an evaluation of the task completion, a robustness metric is introduced [8] to facilitate the search for the optimal solution. An approximation to robustness has been designed [9]–[11] to entirely avoid mixed-integer programming through gradient-based optimization. This "smooth" method improves computational speed but sacrifices the completeness of the results that is guaranteed by MILP.

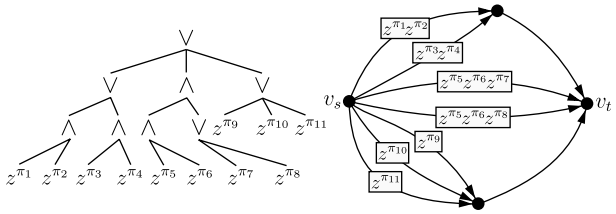

Fig. 1: An example of an Logic Tree (*Left*) and an Logic Network Flow (*Right*) for $\varphi = ((z^{\pi_1} \wedge z^{\pi_2}) \vee (z^{\pi_3} \wedge z^{\pi_4})) \wedge ((z^{\pi_5} \wedge z^{\pi_6}) \wedge (z^{\pi_7} \vee z^{\pi_8})) \wedge (z^{\pi_9} \vee z^{\pi_{10}} \vee z^{\pi_{11}})$.

To improve computational efficiency when solving MILP with STL constraints, existing efforts have focused on either reducing the problem size [12] or tightening the convex formulation [13]. In particular, [14] discussed several approaches for designing MILP formulations with tight convex relaxations. In addition, [13] introduced a compact formulation named graph-of-convex-set to solve hybrid motion planning problems leveraging a structure similar to network flows, which is a classical model developed for urban traffic flow management [15].

In this paper, we reformulate the problem to achieve a tighter convex relaxation inspired by [13]. Our main contributions include converting temporal logic specifications into a Logic Network Flow that encodes STL constraints. As shown in Fig. 1, temporal logic predicates are placed on the edges of a Logic Network Flow (*right*) instead of on the leaf nodes of a Logic Tree (*left*), which is designed in previous literatures [16], [17]. Our formulation is evaluated on multi-robot coordination and searching tasks. Simulation studies show that through integrating Logic Network Flows with dynamics encoded as Dynamic Network Flows [18], our formulation can discover tighter lower and upper bounds via exploring fewer numbers of nodes during the B&B process compared to [16], [17]. The trade-off is that solving a single node tends to be more computationally expensive. In future works, we aim to reduce the computation time on single nodes with techniques such as parallel computing.

II. Background

A. Temporal Logic Preliminaries

We consider a discrete-time nonlinear system in form of

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{1}$$

[1]George W. Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA, 30332 USA (e-mail: {xlin373, jren313, ye.zhao}@gatech.edu).

[2]School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: sam.coogan@gatech.edu)

TABLE I: Validity semantics of Signal Temporal Logic

$(\boldsymbol{x},t) \models \varphi_1 \wedge \varphi_2$	$\Leftrightarrow$	$(\boldsymbol{x},t) \models \varphi_1 \wedge (\boldsymbol{x},t) \models \varphi_2$
$(\boldsymbol{x},t) \models \varphi_1 \vee \varphi_2$	$\Leftrightarrow$	$(\boldsymbol{x},t) \models \varphi_1 \vee (\boldsymbol{x},t) \models \varphi_2$
$(\boldsymbol{x},t) \models \Diamond_{[t_1,t_2]}\varphi$	$\Leftrightarrow$	$\exists t^{'} \in [t+t_1, t+t_2], (\boldsymbol{x},t^{'}) \models \varphi$
$(\boldsymbol{x},t) \models \Box_{[t_1,t_2]}\varphi$	$\Leftrightarrow$	$\forall t^{'} \in [t+t_1, t+t_2], (\boldsymbol{x},t^{'}) \models \varphi$
$(\boldsymbol{x},t) \models \varphi_1 \mathcal{U}_{[t_1,t_2]}\varphi_2$	$\Leftrightarrow$	$\exists t^{'} \in [t+t_1, t+t_2], (\boldsymbol{x},t^{'}) \models \varphi_2$
		$\wedge\, \forall t^{''} \in [t+t_1, t^{'}](\boldsymbol{x},t^{''}) \models \varphi_1$

where $\boldsymbol{x}_t \in \mathcal{X} \subseteq \mathbb{R}^{n_x} \times \mathbb{B}^{n_z}$ represents the state vector, consisting of continuous variables of size n_x and binary variables of size n_z; $\boldsymbol{u}_t \in \mathcal{U} \subseteq \mathbb{R}^{n_u}$ represents the control input of size n_u, with $\mathbb{B} = \{0,1\}$ and $t = 0, 1, \ldots, T$ denoting the time indices. Given an initial state $\boldsymbol{x}_0 \in \mathcal{X}_0$ and the control input at each step, a run of the system is expressed as $\xi = (\boldsymbol{x}_0\boldsymbol{u}_0)(\boldsymbol{x}_1\boldsymbol{u}_1)\cdots$ via rolling out Eqn. (1).

In this paper, we focus on bounded-time signal temporal logic (STL) formulas built upon convex predicates. That said, the maximum trajectory length T to determine logic satisfiability is finite. We recursively define the syntax of STL formulas as follows [19]: $\varphi := \pi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Diamond_{[t_1,t_2]}\ \varphi \mid \Box_{[t_1,t_2]}\ \varphi \mid \varphi_1\ \mathcal{U}_{[t_1,t_2]}\ \varphi_2$, where the semantics consists of not only boolean operations "and" ($\wedge$) and "or" ($\vee$) but also temporal operators "always" ($\Box$), "eventually" ($\Diamond$), and "until" ($\mathcal{U}$). $\varphi, \varphi_1, \varphi_2$ are formulas, and π is an atomic predicate $\mathcal{X} \to \mathbb{B}$ whose truth value is defined by the sign of the convex function $g^\pi : \mathcal{X} \to \mathbb{R}$. In this paper, we assume that the convex function is a combination of linear functions, which can be expressed as $g^\pi(t) = (\boldsymbol{a}^\pi)^\top \boldsymbol{x}_t + b^\pi$. A binary predicate variable $z_t^\pi \in \mathbb{B}$ is assigned to each predicate at timestep t such that:

$$\begin{aligned} (\boldsymbol{a}^\pi)^\top \boldsymbol{x}_t + b^\pi \geq 0 \Leftrightarrow z_t^\pi = 1, \\ (\boldsymbol{a}^\pi)^\top \boldsymbol{x}_t + b^\pi < 0 \Leftrightarrow z_t^\pi = 0 \end{aligned} \tag{2}$$

A run ξ that satisfies an STL formula φ is denoted as $\xi \models \varphi$. The satisfaction of a formula φ having a state signal $\boldsymbol{x}$ beginning from time t is defined inductively as in Table I.

B. Logic Tree

Logic Tree, also referred as STL Tree [12], STL Parse Tree [20], and AND-OR Tree [3], is a hierarchical data structure encapsulating STL formulas to facilitate efficient optimization solve. Here we first provide its definition and an example of translating an STL formula to a Logic Tree:

Definition 1. A Logic Tree (LT) T^φ constructed from an STL specification φ is defined as a tuple $(\circ, \Pi, \mathcal{N}, \tau)$, where:

- $\circ \in \{\wedge, \vee\}$ denotes the combination type;
- $\Pi = \{\pi_1, \ldots, \pi_{|\Pi|}\}$ is the set of $|\Pi|$ predicates associated with each leaf node in the tree T^φ. Each leaf node is assigned a variable z^{π_i} to indicate its validity.
- $\mathcal{N} = \{T^{\varphi_0}, T^{\varphi_1}, \ldots, T^{\varphi_n}\}$ represents the set of $n+1$ internal nodes having at least one child, where the root node is denoted by $T^\varphi = T^{\varphi_0}$. Each node is associated with an STL formula φ_i and a combination type $\circ$. Similarly, each internal node is assigned a variable z^{φ_i} to indicate the formula's validity.

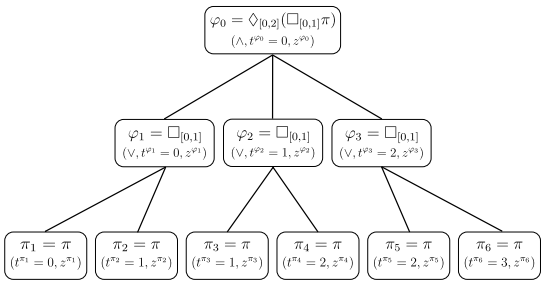

Fig. 2: The Logic Tree for $\Diamond_{[0,2]}(\Box_{[0,1]}\pi)$, given in Example 1.

- $\tau = \{t^{\varphi_0}, t^{\varphi_1}, \ldots, t^{\varphi_n}\} \cup \{t^{\pi_1}, \ldots, t^{\pi_{|\Pi|}}\}$ is a list of starting times corresponding to each of the STL formulas at the internal nodes and predicates at the leaf nodes.

Example 1. Consider a specification $\Diamond_{[0,2]}(\Box_{[0,1]}\pi)$ whose corresponding LT is shown in Fig. 2. This tree has 10 nodes including 6 leaf nodes and 4 internal nodes. The root node has a combination type of disjunction corresponding to the operator $\Diamond$ in formula and three second-level conjunction nodes correspond to the operator $\Box$ in the formula.

To encode temporal logic constraints represented by the LT into an optimization formulation, [16] and [17] propose a MILP, where all variables assigned to the internal nodes $z^{\varphi_i}, \forall i \in \{0, \ldots, n\}$ are continuous variables and all variables $z^{\pi_i}, \forall i \in \{1, \ldots, |\Pi|\}$ on the leaf nodes are binary variables.

For each internal node with a conjunction combination type: $\varphi = \wedge_{i=1}^{p}\varphi_i$ where φ_i is either a formula or a predicate of the child nodes, the following constraints are enforced:

$$z^\varphi \leq z^{\varphi_i}, \quad i = 1, \ldots, p, \quad z^\varphi \geq 1 - p + \sum_{i=1}^{p} z^{\varphi_i} \tag{3}$$

Similarly, for each internal node with a disjunction combination type: $\varphi = \vee_{i=1}^{q}\varphi_i$, the following constraints are applied:

$$z^\varphi \geq z^{\varphi_i}, \quad i = 1, \ldots, q, \quad z^\varphi \leq \sum_{i=1}^{q} z^{\varphi_i} \tag{4}$$

On the root node, $z^{\varphi_0} = 1$ must hold to satisfy the STL specification. Notably, although z^{φ_i} are continuous variables, constraints (3) and (4) guarantee that z^{φ_i} remain binary as long as z^{π_i} take binary values.

In summary, the LT-based optimization formulation is to solve planning problems subject to the dynamics constraint (1) and the temporal logic specification φ, while minimizing the objective function $f(\boldsymbol{x}_t, \boldsymbol{u}_t)$. After constructing the LT T^φ, the optimization can be formulated as:

$$\begin{aligned} \underset{\substack{\boldsymbol{x}_t \in \mathcal{X},\ \boldsymbol{u}_t \in U \\ z^{\pi_i} \in \mathbb{B},\ z^{\varphi_i} \in [0,1]}}{\text{minimize}} \quad & f(\boldsymbol{x}_t, \boldsymbol{u}_t) \\ \text{s.t.} \quad & (1), \quad \boldsymbol{x}_0 \in \mathcal{X}_0 \\ & (2), \quad \forall\, \pi \in \Pi \\ & (3)\ (4), \quad \forall\, T^{\varphi_i} \in \mathcal{N} \\ & z^{\varphi_0} = 1 \end{aligned} \tag{5}$$

C. Branch and Bound

The problem formulation proposed in this study is a mixed-binary linear program (MBLP), which is known to be NP-complete [21], and Branch and Bound (B&B) is a well-received method to solve MBLPs. For a feasible optimization problem, B&B converges to the global optimum; otherwise, it provides a certificate of infeasibility. In this section, we briefly introduce B&B, and refer readers to [22] for a more detailed description.

Consider a MBLP with continuous variables $\boldsymbol{x} \in \mathbb{R}^{n_x}$, binary variables $\boldsymbol{z} \in \mathbb{B}^{n_z}$, and an optimal objective value LP^*. B&B maintains a search tree, where each node corresponds to a linear programming (LP) problem. These LP problems on nodes are created by relaxing some binary variables $\boldsymbol{z}[j]$, $j = \{1, \dots, n_z\}$ to continuous variables, and imposing bounds on them. The root node of the tree is a linear program LP_0 that relaxes all binary variables to continuous variables.

Each LP_i in the search tree is associated with a lower bound, $\underline{LP_i}$, on its optimal objective value LP_i^*. Heuristics are applied to effectively obtain lower bounds. The B&B algorithm also keeps an incumbent solution, $\overline{LP}$, which is the best objective value found so far. This value also serves as an upper bound on LP^*. If no feasible solution has been found up to the current iteration, $\overline{LP}$ is set to $+\infty$. The success of B&B relies on efficiently pruning the search tree using both upper and lower bounds, which occurs when $\underline{LP_i}$ is a tight lower bounds of LP_i^*, and $\overline{LP}$ is a tight upper bound of LP^* [22].

The relaxation gap of B&B is defined as $G_a = |\overline{LP} - \underline{LP}|/|\overline{LP}|$, where $\underline{LP}$ is the best lower bound among all $\underline{LP_i}$. G_a is used to measure the tightness of bounds and the solver will terminate when $G_a = 0$. In particular, the root relaxation gap is defined as $G_r = |\overline{LP} - \underline{LP_0}|/|\overline{LP}|$.

III. Proposed Method

A. Logic Network Flow

In this paper, we propose a new formulation to encode signal temporal logic specifications, named Logic Network Flow (LNF), which comes with a tighter convex relaxation. We first provide the definition for LNFs.

Definition 2. A Logic Network Flow $\mathcal{F}^\varphi$ from the specification φ is defined as a tuple $(\mathcal{G}, \mathcal{P}, \Pi, \tau)$, where:

- $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a directed graph with a vertex $v_s \in \mathcal{V}$ be the source vertex and a vertex $v_t \in \mathcal{V}$ be the target vertex.
- $\Pi = \{\pi_1, \dots, \pi_{|\Pi|}\}$ is the set of $|\Pi|$ predicates associated with each leaf node in the tree T^φ (same to Def. 1). We define $\boldsymbol{\omega}_{\text{flow}} \in \mathbb{R}^{|\Pi|}$ as a vector of values of all elements in Π.
- $\mathcal{P}$ is a collection of sets of n_e ($n_e \leq |\Pi|$) predicates that must hold true to pass through each edge $e \in \mathcal{E}$, defined as $P_e \coloneqq \{\pi_i | \pi_i \in \Pi, z^{\pi_i} = 1\} \in \mathcal{P}$.
- $\tau = \{t^{\pi_1}, \dots, t^{\pi_{|\Pi|}}\}$ is a list of starting times corresponding to each predicate in Π.

For a vertex $v \in \mathcal{V}$ in an LNF, let $\mathcal{E}_v^{\text{in}}$ denote the set of incoming edges to v, and $\mathcal{E}_v^{\text{out}}$ the set of outgoing edges from v. We present Algorithm 1, a recursive algorithm that translates an LNF from an LT. A similar approach can also be used to construct LNFs directly from STL specifications. To initialize the recursion process in Algorithm 1, we input a source vertex v_s, a "dangling" outgoing edge $e \in \mathcal{E}_{v_s}^{\text{out}}$ without the other end, and an empty predicate set $P_e = \varnothing$ associated with the edge e. Subsequently, we run the program: BuildNode$(T^{\varphi_0}, v_s, e, P_e)$, which returns the target vertex v_t. Fig. 1, 3 and 4 show a few examples of converting LNFs to LTs for nodes with conjunction and disjunction combination types.

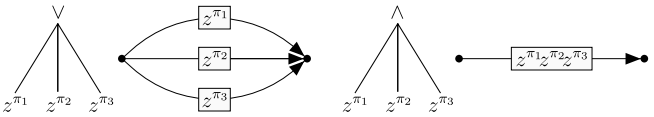

Fig. 3: An illustration of the strategy to translate conjunction and disjunction combination types from an Logic Tree to an Logic Network Flow in Algorithm 1.

Algorithm 1 BuildNode

Input: An LT node T^{φ_i}, a vertex v, an outgoing edge e of v, and a predicate set P_e for edge e.
Output: A vertex v, an outgoing edge e of v, and a predicate set P_e for edge e.

if $T^{\varphi_i} = \pi_\psi$ is a leaf node **then**
  Add π_ψ to P_e.
  return v, e, P_e
if $\circ(T^{\varphi_i}) = \wedge$ **then**
  for each child T^{φ_j} of T^{φ_i} **do**
    v, e, P_e = BuildNode$(T^{\varphi_j}, v, e, P_e)$
  return v, e, P_e
if $\circ(T^{\varphi_i}) = \vee$ **then**
  (*Assume n to be the number of subnodes of T^{φ_i}*)
  Duplicate e, P_e for $n-1$ times, denote as e_j, $P_{e,j}$, where $j = 1, \dots, n$.
  for each child T^{φ_j} of T^{φ_i} **do**
    v, $e_{o,j}$, $P_{o,j}$ = BuildNode$(T^{\varphi_j}, v, e_j, P_{e,j})$
  Initialize a new vertex v_{φ_i}, an outgoing edge e_{φ_i}, and a set $P_{e_{\varphi_i}} = \varnothing$ for the edge e_{φ_i}
  Assign $\mathcal{E}_{v_{\varphi_i}}^{\text{in}} = \{e_{o,j}, j = 1, \dots, n\}$.
  return v_{φ_i}, e_{φ_i}, $P_{e_{\varphi_i}}$.

B. Optimization Formulation

Given a $\mathcal{F}^\varphi$, we propose an optimization formulation with a tighter and more compact convex relaxation, similar to the approach in [13], since an LNF can be considered as a special instance of graph-of-convex-sets. For each edge $e \in \mathcal{E}$ in the LNF shown in Fig. 4, we associate a binary variable $y_e \in \mathbb{B}$ indicating if this edge is traversed by the flow $\boldsymbol{\omega}_{\text{flow}}$, and a multi-dimensional continuous vector $\boldsymbol{\omega}_e \in [0,1]^{|\Pi|}$ defined as follows: if $\boldsymbol{\omega}_{\text{flow}}$ passes through the edge e, we require $\boldsymbol{\omega}_e[i] = 1$ for $\pi_i \in P_e$ where π_i is the i^{th} predicate in Π; otherwise, we set $\boldsymbol{\omega}_e = \mathbf{0}$. The relationship between y_e and $\boldsymbol{\omega}_e$ can be imposed through convex hull constraints [14]. We define $\boldsymbol{H}_e \in \mathbb{R}^{2|\Pi| \times |\Pi|}$, $\boldsymbol{h}_e \in \mathbb{R}^{2|\Pi|}$, such that $\boldsymbol{H}_e \boldsymbol{\omega}_e \leq \boldsymbol{h}_e$ captures a closed predicate polyhedron within

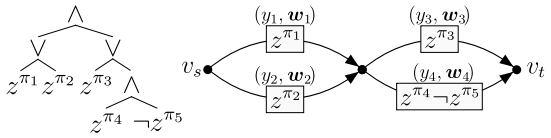

Fig. 4: An example of an LT transformed into an LNF by applying Algorithm 1. Each edge in the LNF possesses a binary variable y_i and a vector variable $\boldsymbol{\omega}_i$, given in Example 4.

which $\boldsymbol{\omega}_e$ remains. The convex hull constraint is expressed as follows so as $y_e = 0 \Rightarrow \boldsymbol{\omega}_e = \mathbf{0}$ can be inferred:

$$\boldsymbol{H}_e \boldsymbol{\omega}_e \leq y_e \boldsymbol{h}_e \tag{6}$$

For each vertex $v \in \mathcal{V}$ with the input edges $\mathcal{E}_v^{\text{in}} \subset \mathcal{E}$ and the output edges $\mathcal{E}_v^{\text{out}} \subset \mathcal{E}$, flow conservation constrains are enforced for both y_e and $\boldsymbol{\omega}_e$:

$$\sum_{e \in \mathcal{E}_v^{\text{in}}} y_e = \sum_{e \in \mathcal{E}_v^{\text{out}}} y_e, \quad \sum_{e \in \mathcal{E}_v^{\text{in}}} \boldsymbol{\omega}_e = \sum_{e \in \mathcal{E}_v^{\text{out}}} \boldsymbol{\omega}_e \tag{7}$$

In addition, in-flow constraints are imposed to ensure that one unit of flow is injected into the source vertex and the quantity of flow to any vertex is less than one unit:

$$\sum_{e \in \mathcal{E}_{v_s}^{\text{out}}} y_e = 1, \quad \sum_{e \in \mathcal{E}_{v_s}^{\text{out}}} \boldsymbol{\omega}_e = \boldsymbol{\omega}_{\text{flow}}, \quad \sum_{e \in \mathcal{E}_v^{\text{in}}} y_e \leq 1 \tag{8}$$

Notably, if $\mathcal{G}$ is not acyclic, the inequality constraints in (8) are necessary to prevent cycles in $\mathcal{G}$ from happening. However, since the directed graph $\mathcal{G}$ in an LNF is acyclic, it is unnecessary to add this constraint to our formulation presented in the Sec. III-D. The proof follows directly from topological sorting, which is omitted here for brevity.

Example 2. The LNF in Fig. 4 consists of 4 edges and 3 vertices including v_s and v_t. The variables in the LNF are $[y_1, y_2, y_3, y_4]$, $y_i \in [0, 1]$, and $[\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \boldsymbol{\omega}_3, \boldsymbol{\omega}_4]$, $\boldsymbol{\omega}_i \in [0, 1]^5$. Edge 1 is associated with the predicate set $P_1 = \{z^{\pi_1}\}$, so we enforce $\boldsymbol{\omega}_1[1] \leq y_1$. Similar constraints are applied to Edge 2 and Edge 3. For Edge 4, with predicate set $P_4 = \{z^{\pi_4} \neg z^{\pi_5}\}$, we enforce $\boldsymbol{\omega}_4[4] \leq y_4$ and $\boldsymbol{\omega}_4[5] = 0$. At the middle vertex, we impose flow conservation constraints $y_1 + y_2 = y_3 + y_4$ and $\boldsymbol{\omega}_1 + \boldsymbol{\omega}_2 = \boldsymbol{\omega}_3 + \boldsymbol{\omega}_4$. For the source vertex v_s, the in-flow constraints are $y_1 + y_2 = 1$ and $\boldsymbol{\omega}_1 + \boldsymbol{\omega}_2 = \boldsymbol{\omega}_{\text{flow}}$.

In summary, the LNF formulation ensures that a continuous signal $\boldsymbol{\omega}_{\text{flow}}$ passes through the graph from the source vertex to the target vertex. If $\boldsymbol{\omega}_{\text{flow}}$ reaches the target vertex, the specification φ is satisfied. So far, we treat $\boldsymbol{\omega}_{\text{flow}}$ as the in-flow to the LNF. In Sec. III-D, we will explain how $\boldsymbol{\omega}_{\text{flow}}$ serves as a connection coupling the LNF with the dynamics through Eqn. (2).

C. Incorporating Dynamics as Dynamic Network Flow

In this study, we abstract the dynamics constraint in (1) using a Dynamic Network Flow (DNF) [18]. This approach is employed such that, when integrating with LNFs, it preserves the tightness of the convex relaxation from LNFs. We define the state space $\mathcal{X}$ as a set of discrete points, and generate the DNF connecting these discrete points through running trajectory optimization offline. This framework also holds the promise to synthesize with continuous state spaces.

To build a DNF, a set of discrete points $\mathcal{S} = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_m\}$, $\boldsymbol{p}_i \in \mathcal{X}$ are selected on a map to represent locations that the robots are required to visit based on STL specifications. For each pair of points $(\boldsymbol{p}_i, \boldsymbol{p}_j)$, a trajectory with horizon K departing $\boldsymbol{p}_i$ and arriving $\boldsymbol{p}_j$ is constructed, where K is the timestep to travel between $\boldsymbol{q}_j$ and $\boldsymbol{q}_{j+1}$.

A DNF $\mathcal{G}_t = (\mathcal{E}_t, \mathcal{V}_t)$ is composed of $N \times |\mathcal{S}|$ vertices for a planning trajectory of a horizon N. Each vertex $v \in \mathcal{V}_t$ has a subscript p representing a point $\boldsymbol{p} \in \mathcal{S}$ and a superscript t accounting for the time steps, i.e., all vertices affiliated to $\boldsymbol{p}$ can be expressed in a sequence of vertices $v_p^1, v_p^2, \ldots, v_p^N$ in the graph. If traversing from $\boldsymbol{p}$ to $\boldsymbol{q}$ takes K time steps, edges are connected from v_p^t to v_q^{t+K}, $\forall t$. Edges are also connected from v_p^t to v_p^{t+1} representing the robot staying still at timestep t. We refer readers to [18] for detailed examples of the DNFs.

For each edge $e \in \mathcal{E}_t$, the variable $r_e \in [0, 1]$ represents the flow carried by the edge. Meanwhile, the flow incurs a cost $c_e r_e$ proportional to the amount of flow r_e with coefficient c_e. Similarly, we impose flow conservation constraints on all vertices and in-flow constraints on the source vertex p_s:

$$\sum_{e \in \mathcal{E}_v^{\text{in}}} r_e = \sum_{e \in \mathcal{E}_v^{\text{out}}} r_e, \forall v \in \mathcal{V}_t, \quad \sum_{e \in \mathcal{E}_{p_s}^{\text{out}}} r_e = 1 \tag{9}$$

D. Complete Formulation

In this subsection, we integrate LNFs with DNFs and establish their connections. Recall that $\boldsymbol{w}_{\text{flow}}$, the flow injecting to the LNF, is a vector of binary predicate variables. Binary variable z^{π_i} with starting time at t holds true if $(\boldsymbol{a}^{\pi_i})^T \boldsymbol{p} + b^{\pi_i} \geq 0$, which is equivalent to a flow traversing v_p^t. Therefore, we arrive to the following constraints connecting r_e and z^{π_i}:

$$\bigvee\nolimits_{e \in \mathcal{E}_{v_p^t}^{\text{in}}} r_e \Leftrightarrow z^{\pi_i} = 1, \ \bigwedge\nolimits_{e \in \mathcal{E}_{v_p^t}^{\text{out}}} \neg r_e \Leftrightarrow z^{\pi_i} = 0 \tag{10}$$

where the binary edge variables r_e in the DNF, or their conjunctions and disjunctions, can be treated as the binary predicate variables in the LNF. Given this connection between the LNF and the DNF, the complete problem formulation of our approach is stated as:

$$\begin{aligned} \underset{\substack{r_e \in [0,1]\ z^{\pi_i} \in \mathbb{B} \\ \boldsymbol{\omega}_e \in [0,1]^{|\Pi|}\ y_e \in \mathbb{B}}}{\text{minimize}} & \sum_{e \in \mathcal{E}_d} c_e r_e \\ \text{s.t.} \quad & (1),\ \boldsymbol{x}_0 \in \mathcal{X}_0;\ (2),\ \forall \pi \in \Pi \\ & (6),\ \forall e \in \mathcal{E};\quad (7),\ \forall v \in \mathcal{V} \\ & (8),\ (9),\ (10) \end{aligned} \tag{11}$$

IV. Experiments

In this section, we present two experiments to evaluate the performance of the proposed optimization: (i) optimizing the motions of a team of robots collaboratively moving on a university campus for multiple delivery tasks; (ii) optimizing the motions of several robots performing a searching task with bipedal locomotion dynamics. Our experiments run on

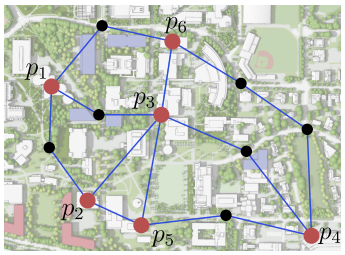

Fig. 5: A discretized Georgia Tech map including 6 sites of interest shown as red dots and 7 interval knots shown as black dots used in Sec. IV-A. It takes each robot dT time to travel each blue line segment divided by black dots in the graph. (Courtesy to Georgia Institute of Technology)

a computer with 12th Gen Intel Core i7-12800H CPU and 16GB memory. All the MIPs are solved using a commercialized solver Gurobi 11.0. For all task specifications, we solve Eqn. (11) by converting them into LNFs, and benchmark Eqn. (5) by converting them into LTs.

A. Multi-agent Coordination Tasks

We deploy 4 mobile robots on a map to visit 6 sites of interest, denoted by p_1 to p_6, and 7 interval nodes, as shown in Fig. 5. The minimum time to execute a task at any site is assumed to be $dT = 1$ min, and the planning horizon is $N = 30$. The robots' motions are modeled as 4 DNFs, each encompassing $(6 + 7) \times 30 = 390$ vertices and 1140 edges. Random costs are assigned to the edges following a uniform distribution between $[0, 1]$, reflecting potential unexpected factors that impact the operating power consumption such as wind gusts. The random costs are also used to give more comprehensive evaluations of computational performance through means and variances.

In this example, tasks assigned to the robots are categorized into three different types. The first type consists of delivery tasks, requesting one robot to pick up an item at a site (*e.g.*, p_1) between time interval $[t_1, t_2]$, and then deliver it to another location (*e.g.*, p_2) after t intervals. The specification is expressed as:

$$\varphi^{p_1 \to p_2}_{\text{deliver}} = \vee^4_{i=1}(\Diamond_{[t_1,t_2]}(\Box_{[0,2]} z^{i,p_1}_t \wedge \Box_{[t,t+2]} z^{i,p_2}_t))$$

The second type of tasks are designed as team tasks, which demand two robots to visit a site (*e.g.*, p_1) simultaneously between time interval $[t_1, t_2]$, and to stay there for a duration t. The specification is shown as:

$$\varphi^{p_1}_{\text{team}} = \Diamond_{[t_1,t_2]}(\vee^4_{i,j=1,i\neq j}(\Box_{[0,t]} z^{i,p_1}_t \wedge \Box_{[0,t]} z^{j,p_1}_t))$$

The third type of tasks are charging duties, asking each robot to visit a charging station (*e.g.*, p_5) every $10 \sim 20$ min of operation. The specification is:

$$\varphi^{p_5}_{\text{charge}} = \wedge^4_{i=1}(\Diamond_{[10,20]}(\Box_{[0,1]} z^{i,p_5}_t))$$

The specifications are converted into LNFs and LTs. We benchmark the performance of our optimization formulation in Eqn. 11 with the one using LTs in Eqn. 5. The robots are initially distributed at 4 random vertices. For each test, 10 sets of random costs are sampled and applied to DNFs. We run the test on three different specifications with increasing complexities: (i) $\varphi_1 = \varphi^{p_3}_{\text{team}} \wedge \varphi^{p_5}_{\text{charge}}$; (ii) $\varphi_2 = \varphi^{p_3}_{\text{team}} \wedge \varphi^{p_4 \to p_6}_{\text{deliver}} \wedge \varphi^{p_5}_{\text{charge}}$; (iii) $\varphi_3 = \varphi^{p_3}_{\text{team}} \wedge \varphi^{p_4 \to p_6}_{\text{deliver}} \wedge \varphi^{p_2 \to p_5}_{\text{deliver}} \wedge \varphi^{p_5}_{\text{charge}}$.

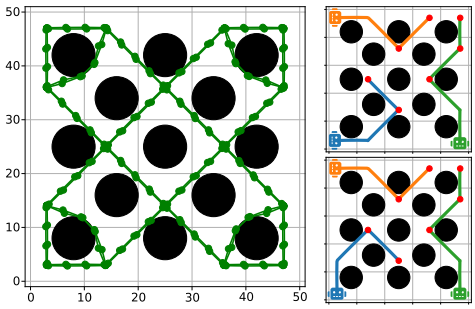

Fig. 6: *Left*: Trajectory library for bipedal robot walking while avoiding obstacles (shown in black circles). Any position on the map can be locally connected to the vertices (shown by green dots) nearby, so it can be reached by following the trajectories (shown by green lines). *Right*: Two examples of globally optimal trajectories under two different random costs, both satisfying φ_{search}. Red dots show the positions required for searching. The orange, blue, and green curves represent the paths taken by each robot.

Table II reports the number of binary and continuous variables, and the number of constraints for each specification. By tracing the Gurobi log file, we list the time when the solver discovers the global optimal solution (labeled "T find opt.") and the number of nodes explored by the B&B before then (labeled "# Node find opt."). The whole optimization process finishes when B&B proves the optimality of the incumbent solution. The row "T prove opt." and "# Node prove opt." respectively lists total time and the total number of nodes explored when the optimality is proven. For the bottom four rows of Table II, the values are shown in the format of *median* $\pm$ *median absolute deviation* for the reason that there is a large variance in solving times, particularly when the problem becomes more complicated (because of the exponential worst solving time). The row "G_r" records the root relaxation gap between the relaxation of the MBLP formulations and the global optimal solution as explained in Sect. II-C. Fig. 7(1)-(3) display the means and variances of the bounds in relation to the number of nodes explored.

Our results indicate that optimization with LNFs are more effective than that with LTs in finding better upper and lower bounds, which is evidenced by the significant reduction in the number of nodes explored to achieve bounds of the same quality. We attribute this to the tighter lower bounds, which allow the B&B to prune the tree more efficiently and avoid unnecessary branching by detecting earlier if certain nodes are not worth further exploration. However, LNFs show an advantage in computational speed only for φ_1 and φ_2. This is because, as the logic specification becomes more complex, LNFs introduce more continuous variables and constraints compared to LTs, which result in larger convex relaxations at each node of the B&B tree. For example, the average solving speed per node in φ_1 is only 4 times slower for LNFs than LTs, but 14 times slower in φ_3. Nonetheless, there is a promise that techniques such as parallel computing could reduce solving times for larger-scale convex programs, potentially improving the scalability of LNFs as future work.

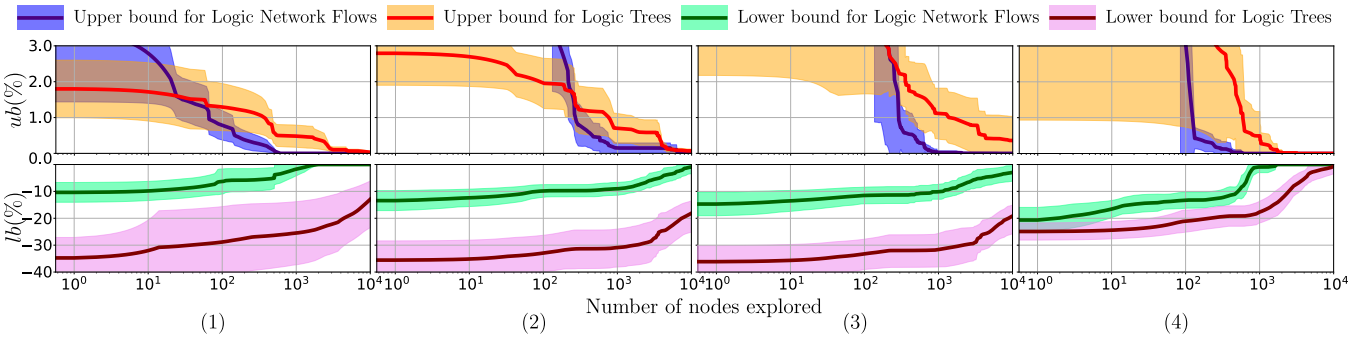

Fig. 7: Solved upper and lower bounds plotted against the number of nodes explored during the B&B process for four tasks, demonstrating the comparison between LNFs and LTs. Shaded regions show the variance. Generally, LNFs discover the same bound by exploring less number of nodes. Four figures on the top shows the upper bounds in percentage values, computed by (upper_bound − global_optimum)/global_optimum × 100%. Four figures on the bottom shows the lower bounds in percentage values, computed by (lower_bound − global_optimum)/global_optimum × 100%. Subfigure (1) shows $\varphi_1 = \varphi^{p_3}_{\text{team}} \wedge \varphi^{p_5}_{\text{charge}}$; Subfigure (2) shows $\varphi_2 = \varphi^{p_3}_{\text{team}} \wedge \varphi^{p_4 \to p_6}_{\text{deliver}} \wedge \varphi^{p_5}_{\text{charge}}$; Subfigure (3) shows $\varphi_3 = \varphi^{p_3}_{\text{team}} \wedge \varphi^{p_4 \to p_6}_{\text{deliver}} \wedge \varphi^{p_2 \to p_5}_{\text{deliver}} \wedge \varphi^{p_5}_{\text{charge}}$; Subfigure (4) shows $\varphi_{\text{search}} = \wedge_{j=1}^{7}(\Diamond_{[t_{j,1},t_{j,2}]}(\vee_{i=1}^{3}(\Box_{[0,1]} z_t^{i,p_j})))$.

TABLE II: Computation results for planning robots' motions in 10 trails

	$\varphi^{p_3}_{\text{team}} \wedge \varphi^{p_5}_{\text{charge}}$		$\varphi^{p_3}_{\text{team}} \wedge \varphi^{p_4 \to p_6}_{\text{deliver}} \wedge \varphi^{p_5}_{\text{charge}}$		$\varphi^{p_3}_{\text{team}} \wedge \varphi^{p_4 \to p_6}_{\text{deliver}} \wedge \varphi^{p_2 \to p_5}_{\text{deliver}} \wedge \varphi^{p_5}_{\text{charge}}$		$\wedge_{j=1}^{7}(\Diamond_{[t_{j,1},t_{j,2}]}(\vee_{i=1}^{3}(\Box_{[0,1]} z_t^{i,p_j})))$	
	Flow	Tree	Flow	Tree	Flow	Tree	Flow	Tree
# of binary vars	124		228		300		633	
# of cont. vars	20682	7048	42094	6989	66354	6962	462177	95678
# of constr.	17417	3385	39902	3739	65187	4093	415807	46037
G_r (%)	**14 ± 5**	51.9 ± 11.3	**15.7 ± 4.3**	54.4 ± 11.4	**16.8 ± 4.4**	55.2 ± 9.5	**25.7 ± 5.2**	56.6 ± 8.1
T find opt. (s)	**5.5 ± 1.5**	11.5 ± 8.5	**44.0 ± 16.5**	46.0 ± 41.0	138.5 ± 59.5	**78.5 ± 41.0**	58.0 ± 22.0	**32.5 ± 16.0**
# N find opt. (10^3)	**0.3 ± 0.3**	3.6 ± 1.7	**0.6 ± 0.3**	3.9 ± 3.6	**0.8 ± 0.2**	15.6 ± 8.0	**0.5 ± 0.2**	1.9 ± 1.1
T prove opt. (s)	**7.8 ± 2.5**	53.4 ± 22.0	**231 ± 104**	379 ± 257	1061 ± 494	**897 ± 540**	**91 ± 34**	153 ± 124
# N prove opt. (10^3)	**0.9 ± 0.6**	33.1 ± 19.4	**7.3 ± 2.7**	79.8 ± 22.9	**10.8 ± 7.4**	140.2 ± 632.9	**1.0 ± 0.3**	10.8 ± 6.1

B. Planning Robot Motions to Search over a Map

In this experiment, three bipedal robots are deployed to search a 50×50 meters map containing 13 circular obstacles. The task involves robots visiting several specified sites of interest within a range of time specified in φ. To expedite the runtime computation, we utilize an offline-generated trajectory library to link these sites on the map. Those trajectories are generated using linear inverted pendulum dynamics [23] tailored for bipedal locomotion. The resulting map is given on the left side of Fig. 6. When a search position is notified to the robot during runtime, a short trajectory with a horizon less than ten seconds is instantaneously planned with less than 100 ms (through IPOPT) to connect the point to the nearest node in the trajectory library.

In this example, 7 positions are chosen from the map and the swarm is required to visit each position for at least one time. The specification is notated as:

$$\varphi_{\text{search}} = \wedge_{j=1}^{7}(\Diamond_{[t_{j,1},t_{j,2}]}(\vee_{i=1}^{3}(\Box_{[0,1]} z_t^{i,p_j})))$$

Here we set $dT = 4$ sec and the planning horizon $N = 45$. Similar to Sec. IV-A, three DNFs are constructed and random costs are assigned to the edges in uniform distributions of $[0, 1]$. The variant edge costs reflect different terrain traversability. Two examples of solved paths with each of the 7 search positions visited at least once by one robot are shown on the right side of Fig. 6. The two examples differ due to the different random traversability costs.

The 4^{th} column of Table II shows the number of binary variables, continuous variables and constraints, as well as the computation results explained in Sec. IV-A. The subfigure (4) in Fig. 7 displays the mean and variance curves of the bounds versus the number of nodes explored. We observe a similar dramatic decrease in the number of nodes required to achieve the same upper and lower bounds. However, computational speeds show greater variability because of the problem size.

V. Conclusion

This paper proposes the LNF, a novel method for encoding STL specifications as an MBLP to enhance the efficiency of the B&B process. While the initial results are promising, several limitations remain in the current work. First, the findings in this study are empirical: they lack a formal proof to justify the improvement in tightening the bounds. Additionally, LNFs tend to involve more continuous variables and constraints, which increases computation time at each node in the B&B search tree. In future work, we plan to employ techniques like parallel computing to reduce the computation time per node. Nevertheless, LNFs serve as a valuable alternative to LTs and offer a promising future direction to improving the computational speed of problems with temporal logic specifications.

REFERENCES

[1] E. Plaku and S. Karaman, "Motion planning with temporal-logic specifications: Progress and challenges," *AI communications*, vol. 29, no. 1, pp. 151–162, 2016.

[2] G. A. Cardona, D. Kamale, and C.-I. Vasile, "Mixed integer linear programming approach for control synthesis with weighted signal temporal logic," in *Proceedings of ACM International Conference on Hybrid Systems: Computation and Control*. Association for Computing Machinery, 2023.

[3] D. Sun, J. Chen, S. Mitra, and C. Fan, "Multi-agent motion planning from signal temporal logic specifications," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3451–3458, 2022.

[4] R. Takano, H. Oyama, and M. Yamakita, "Continuous optimization-based task and motion planning with signal temporal logic specifications for sequential manipulation," in *Proceedings of IEEE international conference on robotics and automation*. IEEE, 2021, pp. 8409–8415.

[5] F. Nawaz, S. Peng, L. Lindemann, N. Figueroa, and N. Matni, "Reactive temporal logic-based planning and control for interactive robotic tasks," *arXiv preprint arXiv:2404.19594*, 2024.

[6] Z. Gu, Y. Zhao, Y. Chen, R. Guo, J. K. Leestma, G. S. Sawicki, and Y. Zhao, "Robust-locomotion-by-logic: Perturbation-resilient bipedal locomotion via signal temporal logic guided model predictive control," *arXiv preprint arXiv:2403.15993*, 2024.

[7] 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.

[8] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Proceedings of International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2010, pp. 92–106.

[9] N. Mehdipour, C.-I. Vasile, and C. Belta, "Average-based robustness for continuous-time signal temporal logic," in *Proceedings of IEEE Conference on Decision and Control*. IEEE, 2019, pp. 5312–5317.

[10] Y. Gilpin, V. Kurtz, and H. Lin, "A smooth robustness measure of signal temporal logic for symbolic control," *IEEE Control Systems Letters*, vol. 5, no. 1, pp. 241–246, 2020.

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

[12] V. Kurtz and H. Lin, "Mixed-integer programming for signal temporal logic with fewer binary variables," *arXiv preprint arXiv:2204.06367*, 2022.

[13] T. Marcucci, "Graphs of convex sets with applications to optimal control and motion planning," Ph.D. dissertation, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2024.

[14] T. Marcucci and R. Tedrake, "Mixed-integer formulations for optimal control of piecewise-affine systems," in *Proceedings of ACM International Conference on Hybrid Systems: Computation and Control*, 2019, pp. 230–239.

[15] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, "Network flows," 1988.

[16] E. M. Wolff, U. Topcu, and R. M. Murray, "Optimization-based trajectory generation with linear temporal logic specifications," in *Proceedings of IEEE International Conference on Robotics and Automation*. IEEE, 2014, pp. 5319–5325.

[17] V. Raman, M. Maasoumy, and A. Donzé, "Model predictive control from signal temporal logic specifications: A case study," in *Proceedings of ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems*, 2014, pp. 52–55.

[18] J. Yu and S. M. LaValle, "Multi-agent path planning and network flow," in *Algorithmic Foundations of Robotics X: Proceedings of Workshop on the Algorithmic Foundations of Robotics*. Springer, 2013, pp. 157–173.

[19] C. Belta and S. Sadraddini, "Formal methods for control synthesis: An optimization perspective," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, no. 1, pp. 115–140, 2019.

[20] K. Leung, N. Aréchiga, and M. Pavone, "Backpropagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods," *The International Journal of Robotics Research*, vol. 42, no. 6, pp. 356–370, 2023.

[21] R. M. Karp, *Reducibility among combinatorial problems*. Springer, 2010.

[22] M. Conforti, G. Cornuéjols, G. Zambelli, M. Conforti, G. Cornuéjols, and G. Zambelli, *Integer programming models*. Springer, 2014.

[23] T. Koolen, T. De Boer, J. Rebula, A. Goswami, and J. Pratt, "Capturability-based analysis and control of legged locomotion, part 1: Theory and application to three simple gait models," *The international journal of robotics research*, vol. 31, no. 9, pp. 1094–1113, 2012.