Semester project

Solving Signal Temporal Logic Planning Problems Online via a Learning-based Framework

Student: QI YI

Supervisor: Prof. Maryam Kamgarpour
Dr. Tony Alan Wood

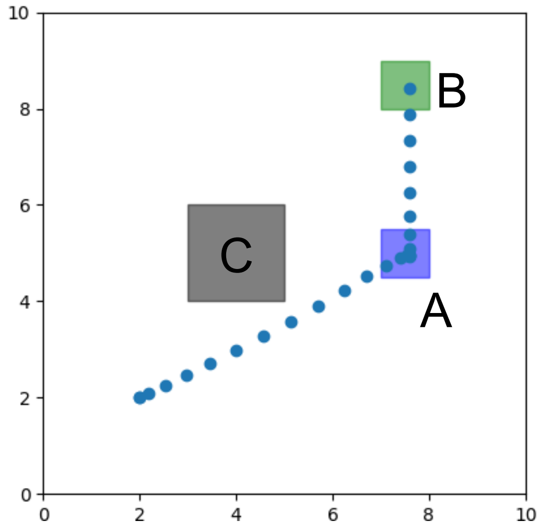
sycamore lab
SYSTEMS CONTROL AND MULTIAGENT OPTIMIZATION RESEARCH

Example

Why we need STL?

Imagining a cleaning robot:

- needs to go to area A to charge itself first
- then go to B to implement the task
- always avoid obstacle C



However, this task can not be formulated with standard optimal control constraints.

Reason: associated with temporal relations and logical behaviors

Need to introduce signal temporal logic (STL) to impose a high-level logical behavior into the robot.

Motivation

- WHAT: Complex robot motion planning problems combined with STL.
- WHY: 1. Need to encode high-level logical behaviors to capture complex task
  2. Slow computation when solving MIPs. (real-time computations)
- HOW: 1. Design an efficient STL encoding, using as few as possible binary variables
  2. Develop an online optimization framework via learning-based method to solve MIPs combined with STL

CONTENTS

Technical background

STL

STL: a formal language to describe the timed behaviors of systems may or may not satisfy

The syntax of STL is defined as : $\varphi := \pi | \neg\varphi | \bigvee_i \varphi_i | \bigwedge_i \varphi_i | \Box_{[t_1,t_2]}\varphi | \diamond_{[t_1,t_2]}\, \varphi \mid \varphi_1 \mathcal{U}_{[t_1,t_2]}\varphi_2$

Read as: predicate |not | or | and |always |eventually| until

x ⊨ φ: x satisfy φ

Predicates π*:* a symbol which represents a property or a relation. (e.g $x \geq 2$))

Technical background

Big-M constraints

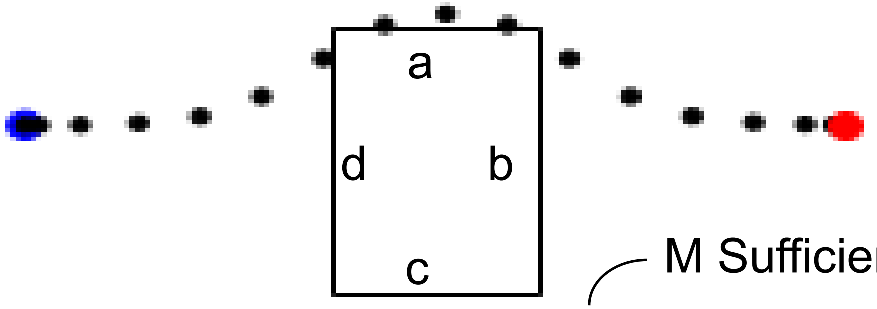
Example: Obstacle avoiding $\varphi \coloneqq \neg O$ for a 2-D position $(\mathrm{p}_1, \mathrm{p}_2)$

Need 4 predicates combined with 'OR' : $(\mathrm{p}_1 \geq \mathrm{a}) \vee (\mathrm{p}_1 \leq c) \vee (\mathrm{p}_2 \geq b) \vee (\mathrm{p}_2 \leq d)$

Introduce 4 binary variables δ using Big-M constraints:

$$\mathrm{p}_1 - a \geq M(1-\delta_1) \qquad \mathrm{p}_1 - c \leq M(1-\delta_3) \qquad \mathrm{p}_2 - b \geq M(1-\delta_2) \qquad \mathrm{p}_2 - d \leq M(1-\delta_4)$$

$\delta_1 = 1 \rightarrow \mathrm{p}_1 \geq a$ Predicate works!



M Sufficient large

$\delta_3 = 0 \rightarrow \mathrm{p}_1 \leq c + M$ Trivial constraint !

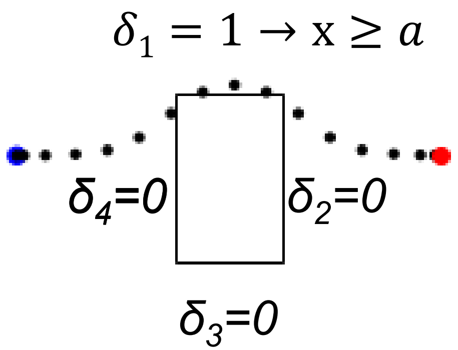
Need at least one predicate works: $\delta_1 + \delta_2 + \delta_3 + \delta_4 \geq 1$

Technical background

Mixed-integer programming

Parameterized Mixed-integer Convex Programs (MICP). Given the STL specification φ:

$$
\begin{aligned}
\min \quad & f_0(x, \delta; \theta^{\varphi}) \\
\text{s.t.} \quad & f_i(x; \theta^{\varphi}) \leq 0 \quad i = 1, \ldots, m_c \\
& g_i(x, \delta_i; \theta^{\varphi}) \leq 0 \quad i = 1, \ldots, m_I \\
& \delta \in (0, 1)^{n_\delta}
\end{aligned}
$$



$x \in \mathbf{R}^{n_c}$: continuous decision variables
$\delta \in (0, 1)^{n_I}$: binary variables,
$\theta^{\varphi} \in \mathbf{R}^{n_p}$: parameters depending on STL φ. (e.g. initial conditions, obstacles, goal)
f_i : constraints only contain x
g_i: mixed-integer constraints contain binary variables δ_i

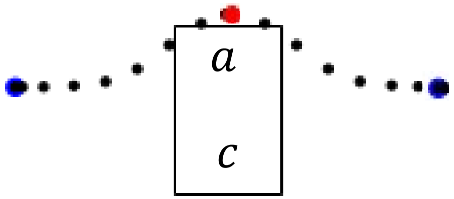
Tight constraint

Tight constraints:

$$\mathcal{T}^{\pi}(\theta^{\varphi}) = \{i | g_i^{\pi}(x^*;\theta^{\varphi}) \leq M(1-\delta_i^{\pi}) \iff \delta_i^{\pi} = \delta_i^{\pi *}\},$$

Insight behind: constraints that really work when solving MIP, corresbonding to $\delta_1 = 1$

$\delta_1 = 1 \Rightarrow p_1 \geq a$ tight



$\delta_3 = 0 \Rightarrow p_1 \leq c + M$ trivial

At some time, only the constraint with δ_1 matters,

discard other constraints still get a optimal solution.

STL integer strategy

STL integer strategy is defined as a tuple:

$$\mathcal{S} = (\mathcal{T}^{\pi}, \delta)$$

T^{π}: the index of the tight constraint; δ: the binary variables.

MICP be simplified as a convex optimization problem with fewer constraints:

$$\begin{aligned} \min \quad & f_0(x; \delta^*, \theta^{\varphi}) \\ \text{s.t.} \quad & f_i(x; \theta^{\varphi}) \leq 0 \quad i = 1, ..., m_c \\ & \hat{g}_i(x, \delta_i; \theta^{\varphi}) \leq 0 \quad i \in \mathcal{T}^{\pi}(\theta^{\varphi}) \\ & \delta = \delta^*(\theta^{\varphi}) \end{aligned}$$

Easier to solve because it is continuous, convex and has fewer constraints (in milliseconds)

2.Problem formulation

STL motion planning problem

Example 1

MPC with obstacles avoidance

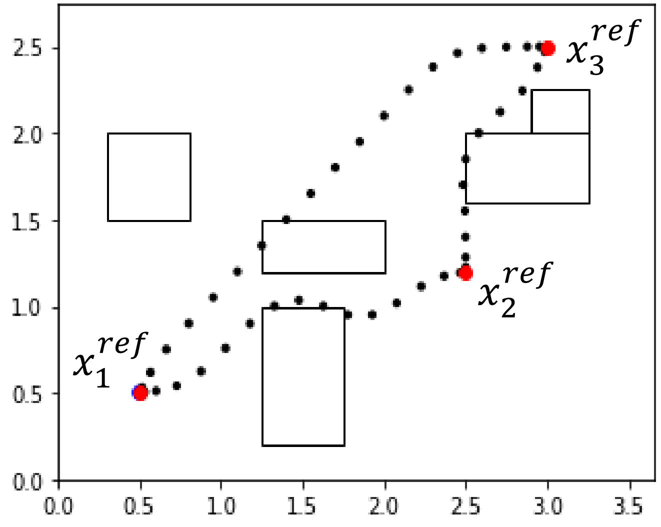
Given φ for obstacles avoidance: $\varphi = \Box_{[0,T]}(\wedge_{i=1}^{n} \neg O_i)$

Given: $Q \succeq 0$, $R \succeq 0$: symmetric cost matrices; T: prediction horizon;
(p_0, v_0), (p^{ref}, v^{ref}):initial state and reference state

Solve T→ move 1 step →new state

MPC solve it online recursively in a receding horizon:

$$
\begin{aligned}
\min \quad & \|p_{t+T} - p_t^{ref}\|_2^2 + Q\sum_{k=0}^{T-1} \|p_{t+k} - p_t^{ref}\|_2^2 + R\|u_t\|_2^2 \\
\text{s.t.} \quad & [p_{t+1}, v_{t+1}]^T = A[p_t, v_t]^T + Bu_t \quad t = 0, ..., T-1 \\
& \|p_t\|_2^2 \le p_{max} \qquad \|u_t\|_2^2 \le u_{max} \qquad t = 0, ..., T-1 \\
& p_0, v_0 \quad fixed \\
& p_t \vDash \varphi \qquad t = 0, ..., T
\end{aligned}
$$



MPC with 3 reference points

Key challenge: define $p_t \vDash \varphi$ with MIP, using as few binary variables as possible.

Contributions: formulate the static LQR as real-time MPC with STL constraints,
computing online in a receding horizon (time-varying destinations and initial conditions).

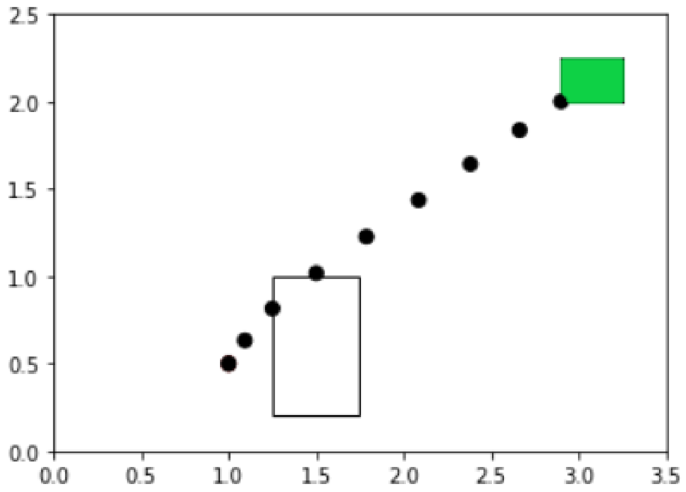
STL motion planning problem

Example 2
STL Motion planner with robust satisfaction

$Q \succeq 0$, $R \succeq 0$: cost matrices; (p_0, v_0): initial state; y: output signal [p, v, u]

STL planning problem considering robustness:

$$
\begin{aligned}
\min \quad & -\rho^{\varphi}(y) + Q\sum_{t=0}^{T} \|p_t\|_2^2 + R\|u_t\|_2^2 \\
\text{s.t.} \quad & [p_{t+1}, v_{t+1}]^T = A[p_t, v_t]^T + Bu_t \quad t = 0, \dots, T-1 \\
& y_{t+1} = C[p_t, v_t]^T + Du_t \quad t = 0, \dots, T-1 \\
& p_0, v_0 \quad fixed \\
& \rho^{\varphi}(y) \geq 0
\end{aligned}
$$



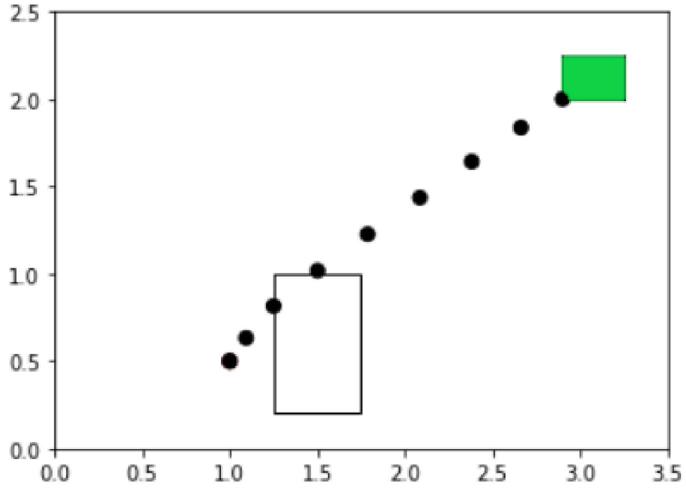
$$\varphi_1 = \Box_{[1,10]} \neg O \wedge \Diamond_{[1,10]} G$$

"Robustness value" $\rho^{\varphi}(y)$: measure how strongly a formula is satisfied by a signal y, which is positive only if $y \vDash \varphi$

Apart from the robustness measure $\rho^{\varphi}(y)$, this is a convex optimization problem.
Goal: define the $\rho^{\varphi}(y)$ in a efficient way

STL motion planning problem

Example 2
STL Motion planner with robust satisfaction



$$\varphi_1 = \square_{[1,10]} \neg O \wedge \Diamond_{[1,10]} G$$

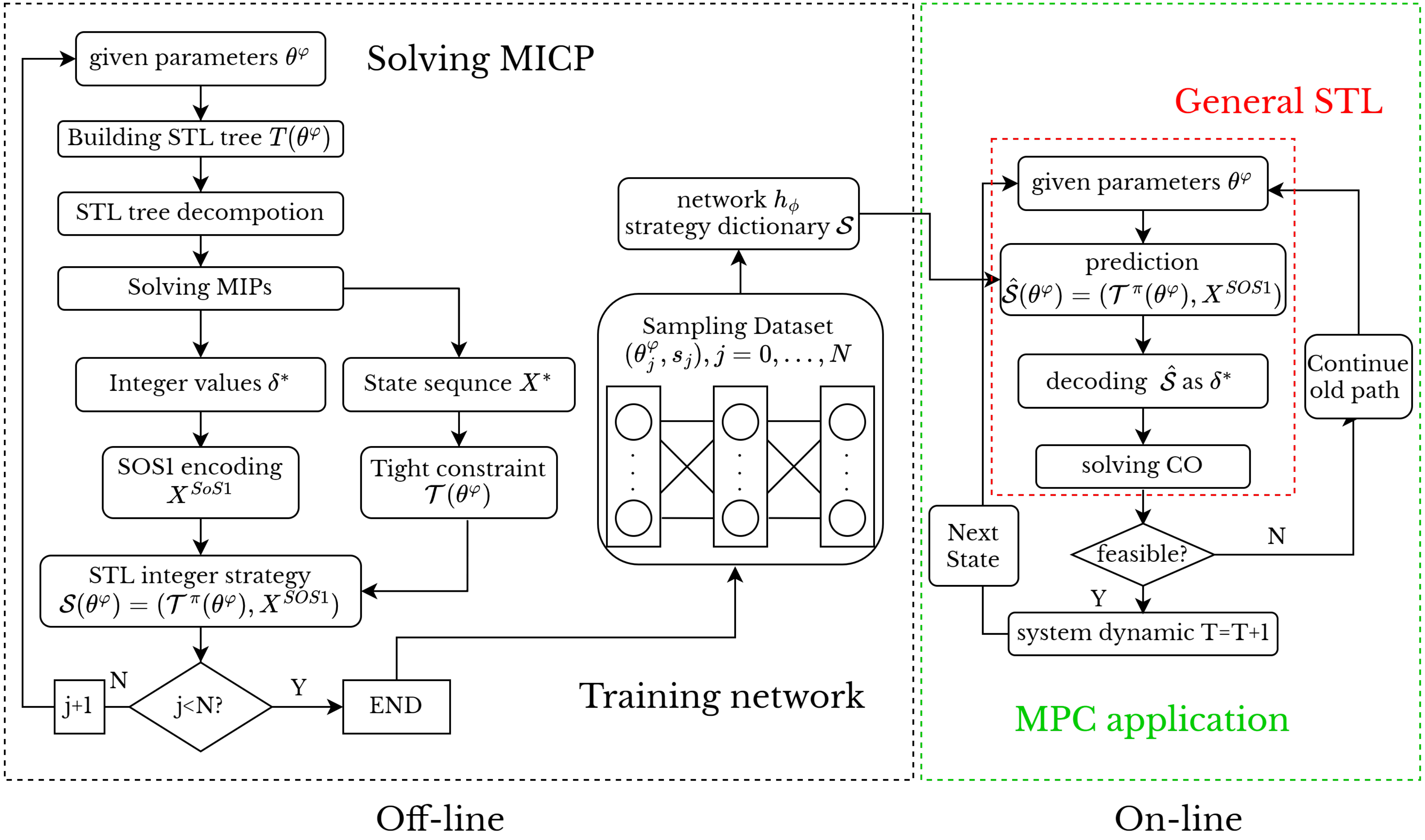
For □ always formula: set a constraint for each timestep to satisfy obstacle avioding all the time.

For ◊ eventually formula: cannot use standard optimal control constraints
Reason: associated with the temporal property, which means the constraints can be satisfied at different timesteps

3. Solution approach

Solution approach

Framework of OMISTL



Online Mixed-Integer Optimization Based on Signal Temporal Logic (OMISTL)

Solution approach

Framework of OMISTL

Our goal:

1. Training a neural network offline. Learn mapping from parameters θ^{φ} to the optimal STL integer $S(\theta^{\varphi})$
2. Predict a strategy with given parameter
3. Solve the convex optimization problem online.

For Ex1 θ^{φ} is: $(p_0, v_0, p^{ref}, v^{ref}, obs)$
For Ex2 θ^{φ} is: $(p_0, v_0, obs, goal)$

For the offline part:

- Solve MICPs to obtain sampling points $(\theta^{\varphi}_j, s_j)$
  $\theta^{\varphi}_j \in R^p$: parameters (p_0, v_0, x_{obs}) $s_j \in S$: label of optimal strategies
  S: dictionary to store the encountered strategies.
- Use a Feedforward Neural Network to implement the classification task.
  Each inner layers feature a ReLU function, cross-entropy loss, softmax function output

For the online part:

- Given parameter θ^{φ}_j, predict $\hat{s}_j$ so that it is as close as possible to the true s_j
- Solve the reduced CO problem with the STL integer strategy

Solution approach

STL tree structure

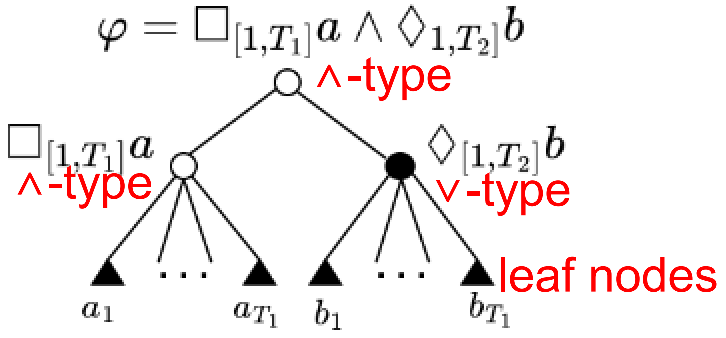
T: time horizon N^{π} : number of predicates $N^{\vee}$:number of disjunctive subformulas

Definition of disjunctive (sub)formulas:

Conjuntive formula: ∧and, □ always (parent node=1 implies all subnodes=1)
Disjunctive formula: ∨or, ◊ eventually, U until (parent node=1 implies at least one subnode =1)

Decomposed as: ∧-type nodes (conjunctive formulas) and ∨-type nodes (disjunctive formulas) and leaf nodes (predicates).

Consider the specification $\varphi = \square_{[1,T]} a \wedge \lozenge_{[1,T]} b$
Has two∧-type nodes (associated with φ and □a), one ∨-type node (◊b) denoted, and 2T leaf nodes



Solution approach

STL encoding

For leaf node: we use the big-M constraint, which is associated with a linear predicate π:

$$\varphi \leq -g^{\pi}(y_t) + M(1 - \delta^{\pi})$$

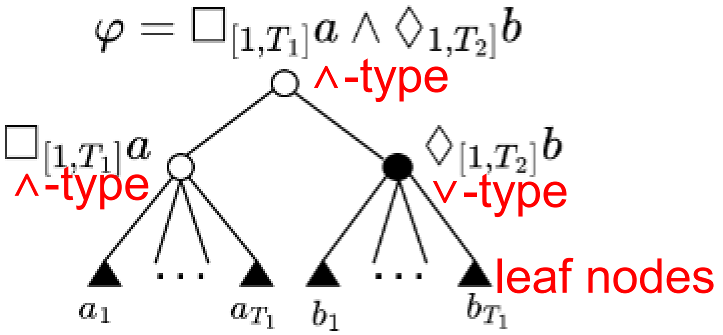
For ∧-type nodes: no more binaries needed, $\delta^\varphi = 1$ implies all the subformulas $\varphi_1, \ldots, \varphi_N$ must also hold:

$$\delta^{\varphi} \leq \delta_i^{\varphi} (\forall i = 1, \ldots, N)$$

For ∨-type nodes : need to introduce $\log_2(N_i + 1)$ binary variables, using SOS1 constraints:

$$\left[1 - \delta^{\varphi}, \delta^{\varphi_1}, \delta^{\phi_2}, \ldots, \delta^{\varphi_N}\right] \in SOS1.$$

A vector λ is in Special Ordered Set of Type 1(SOS1), if contains exactly one 1 element, others are zeros. E.g. SOS1 encoding X^{SOS1}=[1,1,0] (Gray Code 5) implies an SOS1 vector λ = [0, 0, 0, 0, 1, 0, 0, 0].



sycamore lab
SYSTEMS CONTROL AND MULTIAGENT OPTIMIZATION RESEARCH

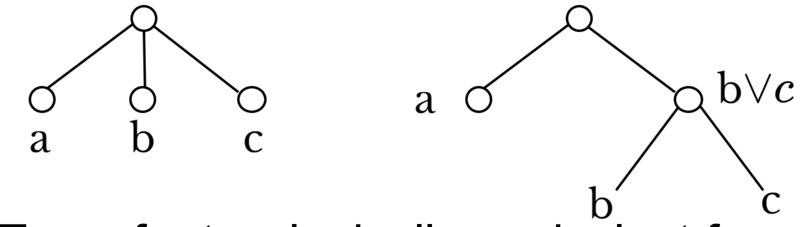
Solution approach

Simplify STL tree

φ_1 requires 2 binary variables   the logically equivalent φ_2 requires 4.

Prefer "flatter" STL Tree

$\varphi_1 = a \vee b \vee c$ $\varphi_2 = a \vee (b \vee c)$



STL Trees for two logically equivalent formulas

Formula flattening: search for nodes which have the same combination type as their parent, then the children node can be moved up a level and that node removed.

STL tree decomposition: further exploit the STL tree structure and decompose the tree into a more succinct form.

Contribution: Compared with standard STL encoding

- Adopt STL tree structure, formula flattening method, STL tree decomposition
- Reduce binaries. MPC with $\neg O$: TN_{f} to $T\log_2 N_{\mathrm{f}}$ General STL: TN^{π} to $\Sigma_1^{N^{\vee}}(\log_2(N_i+1))$

4. Case studies

Parameter setting

System: a double-integrator dynamics:

$$x_{t+1} = \begin{bmatrix} I & I \\ 0 & I \end{bmatrix} x_t + \begin{bmatrix} 0 \\ I \end{bmatrix} u_t$$

MPC example: T = 10, number of obstacles n=6

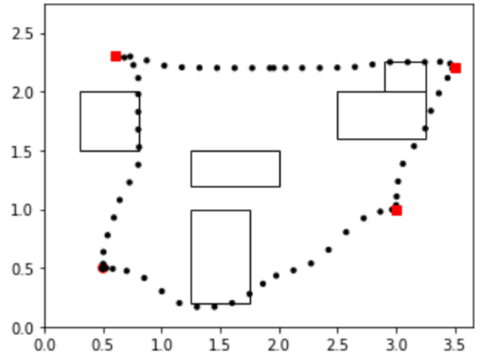
STL example: T=10, number of obstacles n=1

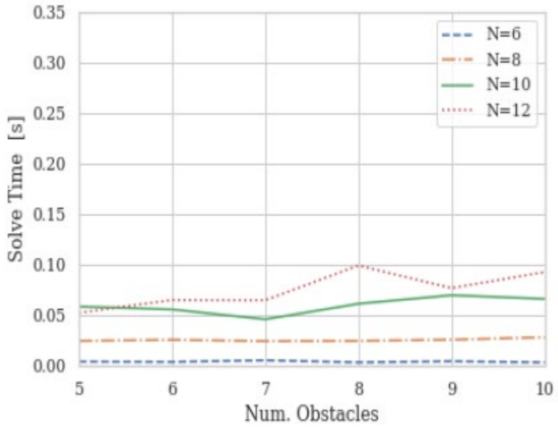
Network setting: sample 10000 parameters θ^{φ} to collect strategies, 90% for training and 10% for testing. Layer number L =3 and each contains 128 neurons, learning rate r=10e-4, batch size B=64, number of epochs E=500

Case studies

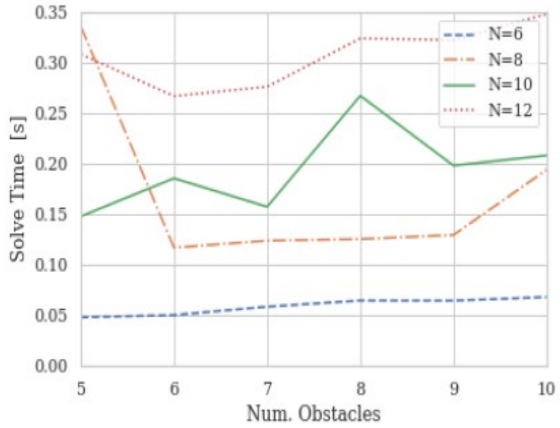
Obstacle avoiding example

Num. of obstacles 5-10
horizons N=6,8,10,12

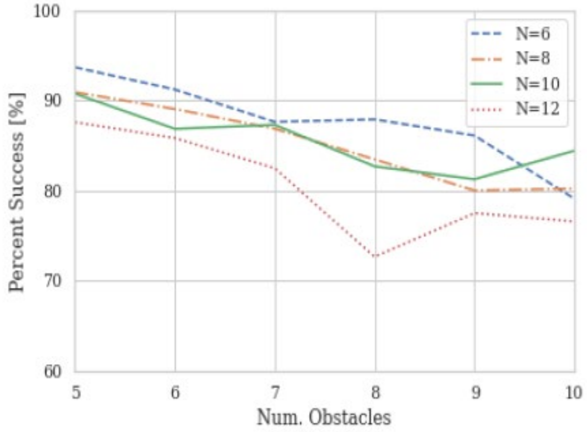




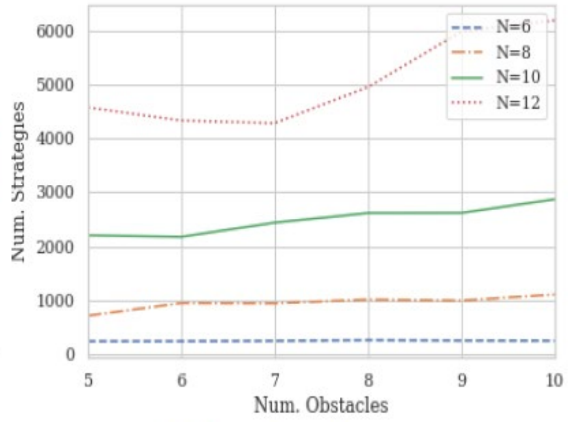
(a) OMISTL solve time



(b) Gurobi solve time



(c) Percentage of feasible solutions



(d) Number of strategies

(a) and (b) demonstrate OMISTL consistently outperforms Gurobi in all problem settings, where the solving time is always within 0.1s.

(c) shows number of feasible solutions decreases continuously with the increasing problems size, while the number of integer strategies is increasing in (d).

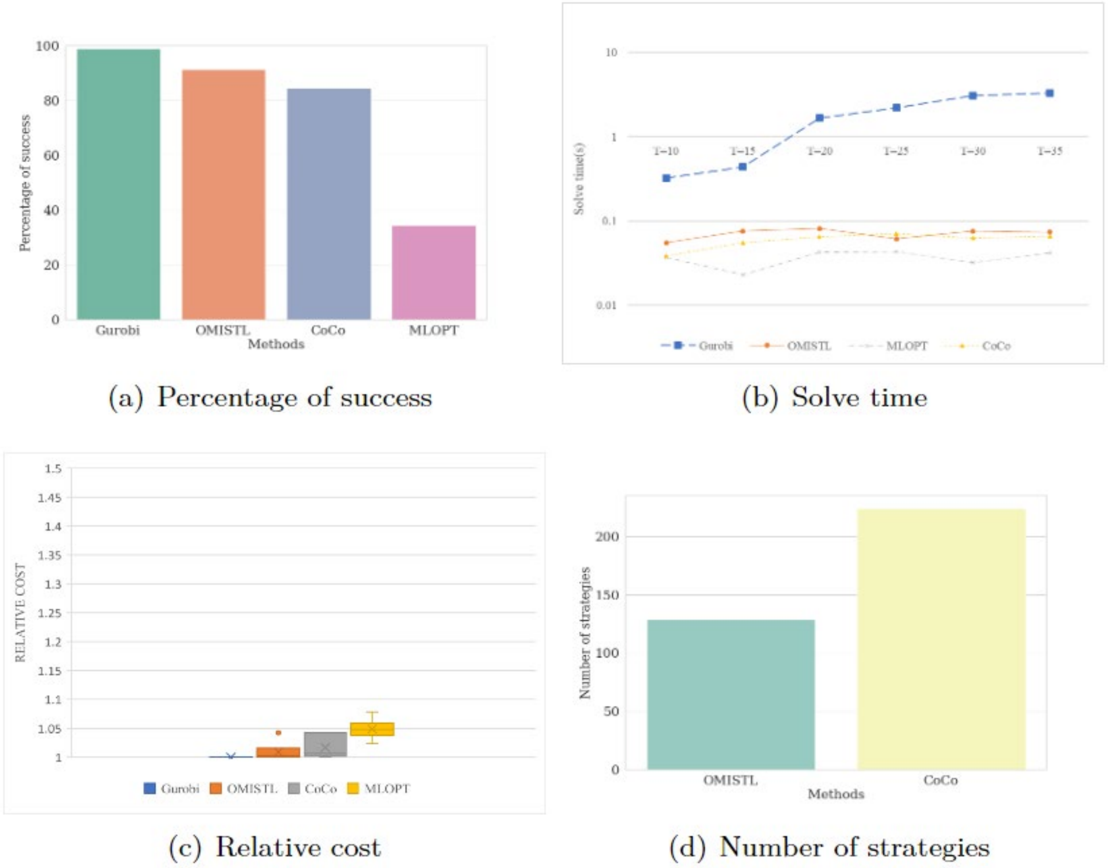
Obstacle avoiding

Encoding comparison in different horizons

N		5	10	15	20	25	30
Integer variables	OMISTL	50	100	150	200	250	300
	MLOPT	100	200	300	400	500	600
Total variables	OMISTL	184	364	544	724	904	1084
	MLOPT	134	264	394	524	654	784
Constraints	OMISTL	384	759	1134	1509	1884	2259
	MLOPT	184	359	534	709	884	1059
Strategy number	OMISTL	241	940	4436	9758	12291	19021
	MLOPT	3536	8329	13990	17892	24421	29542

OMISTL has fewer integer variables and integer strategies compared with OMISTL, but has more continuous variables and constraints.

numerical experiments



Comparison with Gurobi and other learning-based approaches

Compared with other learning-based approaches:

(a):OMISTL provides more feasible solutions

(b):less time solving MICP compared with Gurobi, similar to other learning-based methods.

(c):better solutions, the cost is closed to the optimal solution.

(d): fewer strategies

Performance comparison

Benchmarks: three different specifications   Implementing: python and CVXPY   Solving: Gurobi 9.1.2
Compute STL problem by 1000 times and take the average

Contributions:

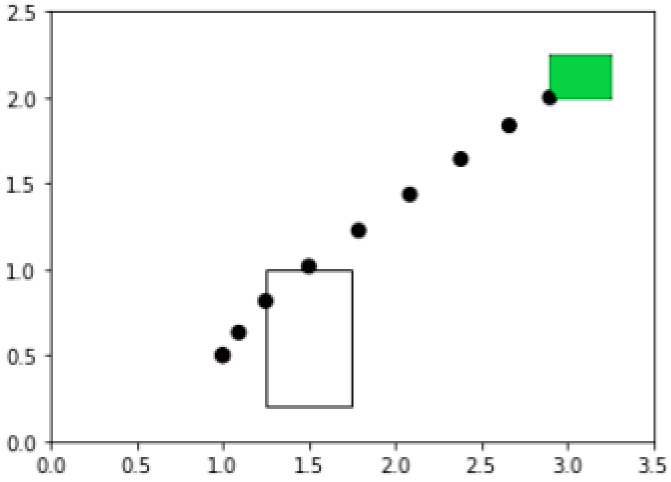
- Compared with previous learning-based method: fewer binary variables (nearly a half), higher network accuracy(80%-90%).
- Compared with state-of-the-art STL encoding: less solving time. (a speed-up of 1-2 orders of magnitude)

Specifications	Horizon(T)	Parameters	Binary variables		Average Solve time(s)	
			CoCo	OMISTL	Gurobi	OMISTL
MPC obstacle avoiding	10	29	200	124	0.4133	0.0032
	15	29	300	178	0.6324	0.0574
	20	29	400	239	0.7421	0.0951
Always eventually	10	12	80	43	0.4133	0.0513
	15	12	120	63	0.6424	0.0342
	20	12	160	107	1.1421	0.0621
Two-target	10	16	498	47	0.4133	0.0032
	15	16	856	78	0.5232	0.0224
	20	16	1023	123	0.7214	0.0312

Table 4.1: Comparison for three specifications

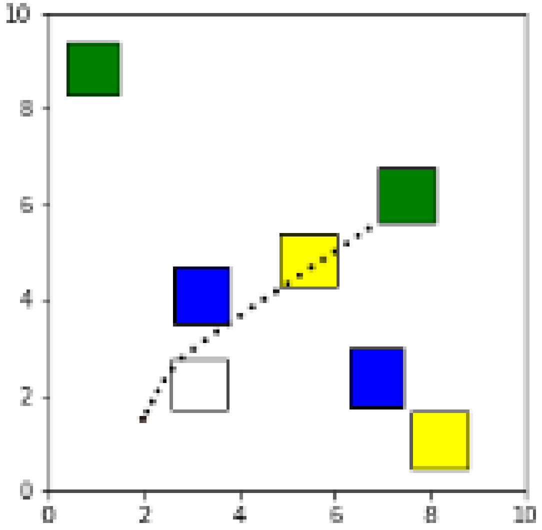
numerical experiments

General STL example results



Alway eventually example

$$\varphi_2 = \square_{[1,10]} \neg O \wedge \lozenge_{[1,10]} G$$



Multi-target example

$$\varphi_3 = \bigwedge_{i=1}^{3} \left(\bigvee_{j=1}^{2} \lozenge_{[0,T]} T_i^j \right) \wedge \square_{[0,T]} (\neg O)$$

numerical experiments

General STL example

Comparison for three specifications

Specifications	Horizon(T)	Disjunctions (maximum)	Binary variables Typical	OMISTL	Average Solve time(s) Gurobi	OMISTL
Obstacle avoiding	10	4	200	100	0.1433	0.0214
	15	4	300	150	0.2132	0.0226
	20	4	400	200	0.2921	0.0245
Always eventually	10	10	50	34	0.0463	0.0156
	15	15	75	49	0.0728	0.0158
	20	20	100	65	0.1093	0.0162
Multi-target	15	30	150	60	1.0312	0.0336
	20	40	200	78	4.9011	0.0334
	25	50	250	93	11.0621	0.0345

OMISTL significantly reduce the number of binary variables

OMISTL gains speed-ups of 1-2 orders of magnitude when solving online, which is depending on the number of binary variables and disjunctions

Conclusions

- A learning-based framework is proposed to solve MPC and STL motion planning.
- The proposed efficient STL encoding approach leads to fewer integer variables (logarithmic).
- Our framework gains speed-ups of 1-2 orders of magnitude in online computation time.
- The solution is better than existing learning-based method with 92% globally optimal.