Multi-Agent Temporal Logic Planning via Penalty Functions and Block-Coordinate Optimization

Eleftherios E. Vlahakis, *Member, IEEE*, Arash Bahari Kordabad, Lars Lindemann, *Member, IEEE*, Pantelis Sopasakis, Sadegh Soudjani, *Member, IEEE* and Dimos V. Dimarogonas, *Fellow, IEEE*

***Abstract*— Multi-agent planning under Signal Temporal Logic (STL) is often hindered by collaborative tasks that lead to computational challenges due to the inherent high dimensionality of the problem, preventing scalable synthesis with satisfaction guarantees. To address this, we formulate STL planning as an optimization program under multi-agent STL constraints and introduce a penalty-based unconstrained relaxation that can be efficiently solved via a Block-Coordinate Gradient Descent (BCGD) method, where each block corresponds to a single agent's decision variables, thereby mitigating complexity. By utilizing a quadratic penalty function defined via smooth STL semantics, we show that BCGD iterations converge to a stationary point of the penalized problem under standard regularity assumptions. To enforce feasibility, the BCGD solver is embedded within a two-layer optimization scheme: inner BCGD updates are performed for a fixed penalty parameter, which is then increased in an outer loop to progressively improve multi-agent STL robustness. The proposed framework enables scalable computations and is validated through various complex multi-robot planning scenarios.**

***Index Terms*— Optimization algorithms, multi-agent systems, signal temporal logic.**

I. Introduction

MULTI-AGENT systems (MAS) research deals with the task of coordinating collections of autonomous systems, e.g., in logistics, exploration, and smart infrastructure. These applications require agents to satisfy complex spatio-temporal and logical constraints governing both individual and interactive behaviors. Signal Temporal Logic (STL) [1] has emerged as a powerful framework suitable for these requirements, offering an expressive language to encode time-bounded properties over continuous-time signals.

This work was supported by the Swedish Research Council, the Knut & Alice Wallenberg Foundation, the Horizon Europe Grant SymAware, the ERC Consolidator Grant LEAFHOUND, and KK-stiftelsen under grant no. 202400088.

Eleftherios E. Vlahakis is with the Division of Computer Engineering and Computer Science, University West, 46132, Trollhättan, Sweden; eleftherios.vlahakis@hv.se

Arash Bahari Kordabad and Sadegh Soudjani are with the Max Planck Institute for Software Systems, Kaiserslautern, Germany; {arashbk, sadegh}@mpi-sws.org

Lars Lindemann is with the Automatic Control Laboratory, ETH Zürich, 8092, Zürich, Switzerland; llindemann@ethz.ch

Pantelis Sopasakis is with the School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, Northern Ireland, BT9 5BN, UK; p.sopasakis@qub.ac.uk

Dimos V. Dimarogonas is with the Division of Decision and Control Systems, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, 10044, Stockholm, Sweden; dimos@kth.se

Unlike automata-based LTL synthesis [2], STL's quantitative semantics [3], [4] enable the direct optimization of satisfaction margins over system trajectories. While exact solutions can be obtained via mixed-integer MPC formulations [5], [6], their poor scalability has motivated the development of smooth robustness relaxations [7]–[10] for efficient gradient-based optimization.

However, in MAS settings, collaborative tasks further amplify computational complexity, rendering scalable planning formidable. Important existing multi-agent approaches, including distributed MPC [11], [12], decentralized feedback control [13], and sequential planning [14], address these challenges but are often limited to restricted STL fragments, rely on the existence of feasible solutions, or employ heuristic coordination schemes. Developing scalable multi-agent planning methods that can handle complex collaborative tasks while maintaining computational tractability under a broad class of STL formulas thus remains an important open problem.

To address this challenge, this paper introduces a scalable, optimization-based framework for multi-agent STL planning under arbitrary collaborative tasks, leveraging smooth STL semantics [9] and the computational efficiency of the Block-Coordinate Gradient Descent (BCGD) method [15]. Specifically, we demonstrate that the original planning problem, featuring an objective function that is separable across agents, yet subject to generally coupled multi-agent STL constraints, can be relaxed into an unconstrained problem via a quadratic penalty defined over smooth robustness metrics. This relaxation is solved efficiently via BCGD, where computations are performed at the block level, with each block corresponding to the decision variables of a single agent. Under standard regularity assumptions, we show that the BCGD iterations converge to a stationary point of the penalized problem, providing a computational architecture that remains tractable despite the complexity of the multi-agent specification. To enforce feasible solutions for the original planning problem, the BCGD solver is embedded in a two-layer optimization scheme, forming a penalty method (PM) [16, Chap. 17], where the inner loop optimizes for a fixed penalty parameter, which is then updated in the outer loop to progressively improve multi-agent STL robustness.

This framework is the first to systematically integrate smooth STL semantics, block-coordinate optimization, and penalty functions for efficient multi-agent STL synthesis. We validate BCGD-PM across complex multi-robot scenarios, benchmarking it against an LBFGS-based implementation [16,

Chap. 9], [15, Sec. 7] within the same modular penalty framework. This comparison highlights BCGD's consistency in handling complex multi-agent STL planning. For readability, the main technical proofs are provided in the Appendix.

II. Problem Setup

Notation: The sets of real numbers and nonnegative integers are $\mathbb{R}$ and $\mathbb{N}$, respectively. Let $N \in \mathbb{N}$ so that $\mathbb{N}_{[0,N]} = \{0, 1, \dots, N\}$. Let $x_1, \dots, x_n$ be vectors so that $x = (x_1, \dots, x_n) = [x_1^\intercal \ \cdots \ x_n^\intercal]^\intercal$. Let $a, b \in \mathbb{N}$ with $a \leq b$. We denote by $\boldsymbol{x}(a{:}b) = (x(a), \dots, x(b))$ an aggregate vector consisting of $x(t)$, $t \in \mathbb{N}_{[a,b]}$, representing a trajectory, or an aggregate trajectory when $x(t) = (x_1(t), \dots, x_M(t))$ with $M \in \mathbb{N}$. The cardinality of a set $\mathcal{V}$ is $|\mathcal{V}|$. We call $x^\star$ a *stationary point* of $f : \mathbb{R}^n \to \mathbb{R}$ if its gradient at $x^\star$ is $\nabla f(x^\star) = 0$. Given a sequence $\{x^k\}$, $\bar{x}$ is called an *accumulation point* if there exists a subsequence $\{x^{k_j}\}$ such that $x^{k_j} \to \bar{x}$.

A. Signal temporal logic

We consider STL formulas in positive normal form (PNF):

$$\varphi := \top \mid \pi \mid \neg\pi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \Box_{\mathcal{I}} \phi_1 \mid \phi_1 \, \mathcal{U}_{\mathcal{I}} \phi_2, \quad (1)$$

where $\pi := (\mu(x) \geq 0)$ is a predicate, with predicate function $\mu : \mathbb{R}^{n_x} \to \mathbb{R}$, ϕ_1 and ϕ_2 are STL formulas built recursively using the grammar in (1), $\neg$, $\wedge$, and $\vee$ are the logical operators denoting *negation*, *conjunction*, and *disjunction*, respectively, and $\Box_{\mathcal{I}}$ and $\mathcal{U}_{\mathcal{I}}$ are the *always* and *until* temporal operators, respectively, defined over the discrete interval $\mathcal{I} \subset \mathbb{N}$. We omit the *eventually* operator ($\Diamond_{\mathcal{I}}$) from (1) since $\Diamond_{\mathcal{I}} \phi_1 = \top \mathcal{U}_{\mathcal{I}} \phi_1$.

The above definition has negation appearing only beside atomic predicates. This form of STL specifications in PNF is equivalent to the full class of STL specifications [1], and any STL formula can be transformed into PNF using usual logical identities [17, Prop. 2].

We denote by $\boldsymbol{x}(t) \models \phi$, $t \in \mathbb{N}$, the satisfaction of ϕ, verified over $\boldsymbol{x}(t) = (x(t), x(t+1), \dots)$. The validity of ϕ can be determined recursively using the Boolean semantics of STL; for details, we refer to [1] due to space limitations.

STL is endowed with quantitative semantics [4]: The *robustness function* $\rho^\phi : \bigtimes_{t=0}^{\mathcal{H}^\phi} \mathbb{R}^n \to \mathbb{R}$ of a signal $\boldsymbol{x}(t)$, where $\mathcal{H}^\phi$ is the *horizon* of ϕ [1], indicates how robustly a signal $\boldsymbol{x}(t)$ satisfies a formula ϕ, and is defined recursively as

$$\begin{aligned}
\rho^{\pi}(\boldsymbol{x}(t)) &= \mu(x(t)), \\
\rho^{\neg\pi}(\boldsymbol{x}(t)) &= -\rho^{\pi}(\boldsymbol{x}(t)), \\
\rho^{\phi_1 \wedge \phi_2}(\boldsymbol{x}(t)) &= \min\big(\rho^{\phi_1}(\boldsymbol{x}(t)), \, \rho^{\phi_2}(\boldsymbol{x}(t))\big), \\
\rho^{\phi_1 \vee \phi_2}(\boldsymbol{x}(t)) &= \max\big(\rho^{\phi_1}(\boldsymbol{x}(t)), \, \rho^{\phi_2}(\boldsymbol{x}(t))\big), \\
\rho^{\Box_{\mathcal{I}} \phi_1}(\boldsymbol{x}(t)) &= \min_{\tau \in t \oplus \mathcal{I}} \rho^{\phi_1}(\boldsymbol{x}(\tau)), \\
\rho^{\phi_1 \mathcal{U}_{\mathcal{I}} \phi_2}(\boldsymbol{x}(t)) &= \max_{\tau \in t \oplus \mathcal{I}} \Big(\min\big(\rho^{\phi_2}(\boldsymbol{x}(\tau)), \min_{\tau' \in \mathbb{N}_{[t,\tau]}} \rho^{\phi_1}(\boldsymbol{x}(\tau'))\big)\Big), \quad (2)
\end{aligned}$$

where $\oplus$ is the Minkowski sum. The satisfaction of ϕ by $\boldsymbol{x}(t)$ is indicated by its robustness function in the sense that $\rho^\phi(\boldsymbol{x}(t)) > 0 \Rightarrow \boldsymbol{x}(t) \models \phi$, $\rho^\phi(\boldsymbol{x}(t)) < 0 \Rightarrow \boldsymbol{x}(t) \not\models \phi$, while $\rho^\phi(\boldsymbol{x}(t)) = 0$ does not in general determine satisfiability.

B. Multi-agent system

1) Dynamics: We consider a MAS with M agents, with the i^{th} agent following the dynamics

$$x_i(t+1) = f_i(x_i(t), u_i(t)), \quad (3)$$

where $x_i(t) \in \mathbb{R}^{n_i}$ and $u_i(t) \in \mathbb{R}^{m_i}$ are the state and input vectors, respectively, $f_i : \mathbb{R}^{n_i} \times \mathbb{R}^{m_i} \to \mathbb{R}^{n_i}$ is continuously differentiable and locally Lipschitz in (x_i, u_i), $t \in \mathbb{N}$, and the initial condition $x_i(0)$ is known, with $i \in \mathcal{V} = \{1, \dots, M\}$, where $\mathcal{V}$ is the set of all agents. To formally group agents participating in collaborative tasks, we adopt the notion of *cliques* from graph theory, defined next [18].

Definition 1 *Consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ potentially containing self-loops and multiple edges with node set $\mathcal{V}$ and edge set $\mathcal{E}$. Let $\nu \subseteq \mathcal{V}$ and $\mathcal{E}_\nu \subseteq \mathcal{E}$ be the set of edges connecting the nodes in ν. Then, $(\nu, \mathcal{E}_\nu)$ is called a clique if $\forall \nu_i, \nu_j \in \nu$, $(\nu_i, \nu_j) \in \mathcal{E}_\nu$, i.e., it is a complete subgraph of $\mathcal{G}$.*

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with clique set $\mathcal{K}$. With a slight abuse of notation, we denote a clique $(\nu, \mathcal{E}_\nu) \in \mathcal{K}$ simply by ν, and write $|\nu|$ for the number of nodes in ν. Let $\nu \in \mathcal{K}$ contain the agents $i_1, \dots, i_{|\nu|}$, i.e., $\nu = \{i_1, \dots, i_{|\nu|}\}$. By collecting individual state and input vectors, as $x_\nu(t) = (x_{i_1}(t), \dots, x_{i_{|\nu|}}(t)) \in \mathbb{R}^{n_\nu}$ and $u_\nu(t) = (u_{i_1}(t), \dots, u_{i_{|\nu|}}(t)) \in \mathbb{R}^{m_\nu}$, respectively, we write the aggregate dynamics of $|\nu|$ agents as

$$x_\nu(t+1) = f_\nu(x_\nu(t), u_\nu(t)), \quad (4)$$

with $f_\nu(x_\nu(t), u_\nu(t)) = (f_j(x_j(t), u_j(t)))_{j \in \nu}$. When $\nu = \mathcal{V}$, the aggregate dynamics of the entire MAS are written as

$$x(t+1) = f(x(t), u(t)), \quad (5)$$

with $x(t) = (x_1(t), \dots, x_M(t))$, $u(t) = (u_1(t), \dots, u_M(t))$, and $f = (f_1, \dots, f_M)$.

2) STL specification: The MAS is subject to

$$\phi = \bigwedge_{\nu \in \mathcal{K}_\phi} \phi_\nu, \quad (6)$$

which is a conjunctive STL formula, where each conjunct ϕ_ν is defined over $\boldsymbol{x}_\nu(t)$ and follows the syntax in (1), with the aggregate trajectory $\boldsymbol{x}_\nu(t)$ collecting the individual trajectories of the agents in the clique ν. The set $\mathcal{K}_\phi$ collects all these cliques induced by ϕ, and may include individual agents ($|\nu| = 1$) or groups of agents ($1 < |\nu| \leq |\mathcal{V}|$). Note that different cliques may overlap in their agent sets, indicating that some agents participate in multiple collaborative tasks.

Let $\pi := (\mu(y) \geq 0)$ be a predicate in ϕ, where $\mu : \mathbb{R}^{n_y} \to \mathbb{R}$. We assume that all predicate functions μ appearing in ϕ are continuously differentiable. The vector $y \in \mathbb{R}^{n_y}$ may represent an individual state $x_i \in \mathbb{R}^{n_i}$, for $i \in \mathcal{V}$, an aggregate state $x_\nu \in \mathbb{R}^{n_\nu}$ for a clique $\nu \in \mathcal{K}_\phi$, or an aggregate state $x_\kappa \in \mathbb{R}^{n_\kappa}$ collecting the states of a subset of agents in a clique ν. For example, in formula $\phi_\nu = \pi_\nu^{\kappa_1} \mathcal{U}_{\mathcal{I}} \pi_\nu^{\kappa_2}$, where $\nu = \{1, 2, 3\}$, the predicate $\pi_\nu^{\kappa_1} := (x_1 \geq 0)$ involves one agent $\kappa_1 = \{1\}$, while $\pi_\nu^{\kappa_2} := (x_2 - x_3 \geq 0)$ involves two agents $\kappa_2 = \{2, 3\}$.

C. Problem statement

The multi-agent STL planning synthesis problem is formulated as an optimization problem over the multi-agent control sequence $\boldsymbol{u}=(u(0),\dots,u(N-1))$, where $u(t)=(u_1(t),\dots,u_M(t))$, and $N=\mathcal{H}^\phi$. Given the initial condition $x(0)=x_0$ and the dynamics $x(t+1)=f(x(t),u(t))$, the multi-agent trajectory $\boldsymbol{x}=(x(0),\dots,x(N))$, where $x(t)=(x_1(t),\dots,x_M(t))$, is explicitly determined by $\boldsymbol{u}$. We thus denote the multi-agent STL robustness $\rho^\phi(\boldsymbol{x})$ as $\rho^\phi(\boldsymbol{u})$, and the total cost as $\mathcal{L}(\boldsymbol{u})=\sum_{i\in\mathcal{V}}\mathcal{L}_i(\boldsymbol{u}_i)$, where $\mathcal{L}_i(\boldsymbol{u}_i)=\sum_{t=0}^{N-1}\ell_i(x_i(t),u_i(t),t)+V_{f,i}(x_i(N))$, with $\boldsymbol{u}_i=(u_i(0),\dots,u_i(N-1))$ denoting the i^{th} agent's inputs. Function $\ell_i:\mathbb{R}^{n_i}\times\mathbb{R}^{m_i}\times\mathbb{N}\to\mathbb{R}$ is the running cost for the i^{th} agent, penalizing, e.g., state energy and control effort quantities that cannot be directly encoded through the specification ϕ. The terminal cost $V_{f,i}:\mathbb{R}^{n_i}\to\mathbb{R}$ penalizes deviations from a prescribed terminal condition, which can be chosen, e.g., to enforce horizon-end requirements in ϕ or to induce cyclic trajectory planning by penalizing the distance $x_i(N)-x_i(0)$ for all $i\in\mathcal{V}$. The multi-agent STL planning problem is formulated as

$$\underset{\boldsymbol{u}}{\text{Minimize}}\ \ \mathcal{L}(\boldsymbol{u})=\sum_{i\in\mathcal{V}}\mathcal{L}_i(\boldsymbol{u}_i) \tag{7a}$$

$$\text{subject to } \rho^\phi(\boldsymbol{u})=\min_{\nu\in\mathcal{K}_\phi}\rho^{\phi_\nu}(\boldsymbol{u}_\nu)>0. \tag{7b}$$

Solving (7) is computationally challenging due to the non-smoothness and coupling in the joint STL constraint in (7b), which implies that $\boldsymbol{x}_\nu\models\phi_\nu$, $\forall\nu\in\mathcal{K}_\phi$, i.e., $\boldsymbol{x}\models\phi$. We address this challenge by employing a smooth STL approximation and a penalty-based block-coordinate gradient descent method, which enables agent-level computations, mitigating the complexity of the centralized problem. This approach relies on the following assumption.

Assumption 1 *The function $\mathcal{L}_i:\mathbb{R}^{n_{\boldsymbol{u}_i}}\to\mathbb{R}$, $i\in\mathcal{V}$, is proper, convex, continuous, and level-bounded. Furthermore, the problem in (7) is feasible.*

In the remainder of the paper, Assumption 1 holds. We note that the first part holds for linear dynamics, or, more generally, when the running cost ℓ_i depends only on the input variables and when the terminal cost $V_{f,i}$ is omitted. To streamline the presentation of the proposed optimization framework, we omit explicit input constraints from (7).

III. Multi-agent STL optimization

A. Smooth STL approximations

We recall that the STL constraint in (7b) is non-smooth, involving min and max operations over predicate functions. We underapproximate min and max operators [9] as

$$\min(\mu_1,\dots,\mu_q)\overset{\geq}{\approx}-\frac{1}{\Gamma}\log\left(\sum_{j=1}^{q}e^{-\Gamma\mu_j}\right), \tag{8a}$$

$$\max(\mu_1,\dots,\mu_q)\overset{\geq}{\approx}\frac{\sum_{j=1}^{q}\mu_j e^{\Gamma\mu_j}}{\sum_{j=1}^{q}e^{\Gamma\mu_j}}, \tag{8b}$$

where the approximation becomes tighter for larger values of $\Gamma>0$. For a fixed Γ, we denote by $\varrho^\phi_\Gamma(\boldsymbol{u})$ the smooth approximation of the robustness function $\rho^\phi(\boldsymbol{u})$, which satisfies $\varrho^\phi_\Gamma(\boldsymbol{u})\leq\rho^\phi(\boldsymbol{u})$ for all $\Gamma>0$, since ϕ is in PNF [19]. This implies that the STL constraint in (7b) can be replaced by $\varrho^\phi_\Gamma(\boldsymbol{u})>0$, introducing additional conservatism in satisfying $\boldsymbol{x}\models\phi$ controlled by Γ. In fact, due to the multi-agent structure of ϕ in (6), the strict requirement $\varrho^\phi_\Gamma(\boldsymbol{u})>0$ can be relaxed to $\varrho^\phi_\Gamma(\boldsymbol{u})\geq 0$, as formally stated next.

Proposition 1 *Consider the formula ϕ in (6), where $\mathcal{K}_\phi$ contains at least two cliques, i.e., $|\mathcal{K}_\phi|\geq 2$. Then, for all multi-agent sequences $\boldsymbol{u}$ and $\Gamma>0$, the smooth robustness satisfies $\varrho^\phi_\Gamma(\boldsymbol{u})<\rho^\phi(\boldsymbol{u})$. Consequently, $\varrho^\phi_\Gamma(\boldsymbol{u})\geq 0\Rightarrow\rho^\phi(\boldsymbol{u})>0$.*

B. Unconstrained STL optimization

Due to Proposition 1, a stricter version of the problem in (7) can now be written using smooth STL semantics as

$$\underset{\boldsymbol{u}\in\mathcal{M}}{\text{Minimize}}\ \mathcal{L}(\boldsymbol{u}),\ \text{where}\ \mathcal{M}:=\{\boldsymbol{u}\mid\varrho^\phi_\Gamma(\boldsymbol{u})\geq 0\}. \tag{9}$$

By Assumption 1, this problem has a minimizer [16]. We address this constrained program using a suitable penalty function $R(\boldsymbol{u})$ leading to an unconstrained problem:

$$\underset{\boldsymbol{u}}{\text{Minimize}}\ F_\lambda(\boldsymbol{u}):=\mathcal{L}(\boldsymbol{u})+\lambda R(\boldsymbol{u}), \tag{10}$$

for some $\lambda>0$. To specify a suitable penalty $\lambda R(\boldsymbol{u})$ that relaxes the original constrained formulation into the unconstrained problem in (10) we introduce the quadratic penalty

$$R(\boldsymbol{u}):=\max\{0,-\varrho^\phi_\Gamma(\boldsymbol{u})\}^2, \tag{11}$$

which satisfies $\mathcal{M}=\{\boldsymbol{u}\mid R(\boldsymbol{u})=0\}$, and $R(\boldsymbol{u})>0$ for all $\boldsymbol{u}\notin\mathcal{M}$. This is a differentiable penalty function with gradient

$$\nabla R(\boldsymbol{u})=-2\max\{0,-\varrho^\phi_\Gamma(\boldsymbol{u})\}\nabla\varrho^\phi_\Gamma(\boldsymbol{u}), \tag{12}$$

which exists everywhere and is continuous. We shall first discuss how the unconstrained problem in (10) can be solved for a fixed value of $\lambda>0$ using a block coordinate gradient descent method. Subsequently, we will show how a penalty method can be used to solve the constrained problem in (9).

C. Block coordinate gradient descent

Problem (10) possesses a certain structure that can be exploited to solve it efficiently. First, note that the cost function $\mathcal{L}(\boldsymbol{u})$ is separable into agent-level objective functions $\mathcal{L}_i(\boldsymbol{u}_i)$, which by Assumption 1 are proper, convex, and continuous in $\boldsymbol{u}_i$ for all $i\in\mathcal{V}$. The penalty term $\lambda R(\boldsymbol{u})$, which penalizes violations of the smooth robustness condition $\varrho^\phi_\Gamma(\boldsymbol{u})\geq 0$, is differentiable. These properties allow us to use the block coordinate gradient descent (BCGD) method [15]. In each BCGD iteration, (i) $\lambda R(\boldsymbol{u})$ is approximated by a strictly convex quadratic function, enabling the application of block coordinate descent to generate a descent direction, and (ii) a sufficient descent direction is computed for an agent-level block of $\boldsymbol{u}$, as detailed in Algorithm 1.

Quadratic approximation: We model the variation $R(\boldsymbol{u}^k+\boldsymbol{d})-R(\boldsymbol{u}^k)$ at $\boldsymbol{u}^k$ by $Q^H(\boldsymbol{u}^k,\boldsymbol{d}):=\nabla R(\boldsymbol{u}^k)^\intercal\boldsymbol{d}+$

Algorithm 1 Block Coordinate Gradient Descent (BCGD)

Input: $\boldsymbol{u}^0$ (initial iterate), λ (penalty), K (maximum iterations), $\epsilon > 0$ (tolerance)
for $k = 0, 1, \ldots, K-1$ **do**
  Select agent blocks $J^k \subseteq \mathcal{V}$ and an $H^k \succ 0$
  Solve (13) with $H{=}H^k$ and $J = J^k$ to obtain $\boldsymbol{d}^k$
  If $\|H^k \boldsymbol{d}^k\|_\infty \leq \epsilon$ **break**
  Choose step size α^k and set $\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \alpha^k \boldsymbol{d}^k$
return $\boldsymbol{u}^{\star,k}$

$\frac{1}{2}\boldsymbol{d}^\intercal H^k \boldsymbol{d}$, where $\boldsymbol{d}$ is the update direction to $\boldsymbol{u}^k$ derived from (13), and $H^k{\succ}0$ approximates the Hessian $\nabla^2 R(\boldsymbol{u}^k)$.

Block selection: In iteration k we choose a nonempty subset of agent-level blocks $J^k \subseteq \mathcal{V}$, so that only the elements $(\boldsymbol{u}_i)_{i\in J^k}$ are updated. Over iterations, all blocks should be updated at least once (generalized Gauss–Seidel), or one may select the most "active" blocks using a Gauss–Southwell type rule for efficiency—see [15], [20] for details.

Block update direction: We compute the descent direction

$$\boldsymbol{d}^k{=}\underset{\boldsymbol{d}}{\operatorname{argmin}}\Big\{\lambda Q^H(\boldsymbol{u}^k,\boldsymbol{d}){+}\mathcal{L}(\boldsymbol{u}^k{+}\boldsymbol{d})\Big|\boldsymbol{d}_j{=}0, \forall j{\notin}J\Big\}. \quad (13)$$

Due to the separability of $\mathcal{L}(\boldsymbol{u})$, this step decomposes into independent subproblems for each selected agent block when H^k is chosen block-diagonal, i.e., it is equivalent to solving

$$\underset{(\boldsymbol{d}_j)_{j\in J^k}}{\text{Min.}} \sum_{j\in J^k} \frac{\lambda}{2}\boldsymbol{d}_j^\intercal H_j^k \boldsymbol{d}_j{+}\lambda\nabla R(\boldsymbol{u}^k)_j^\intercal \boldsymbol{d}_j{+}\mathcal{L}_j(\boldsymbol{u}_j^k{+}\boldsymbol{d}_j). \quad (14)$$

Thus, for $j \in J^k$, we have the optimization problems

$$\underset{\boldsymbol{d}_j}{\text{Min.}} \ \frac{\lambda}{2}\boldsymbol{d}_j^\intercal H_j^k \boldsymbol{d}_j + \lambda\nabla R(\boldsymbol{u}^k)_j^\intercal \boldsymbol{d}_j + \mathcal{L}_j(\boldsymbol{u}_j^k + \boldsymbol{d}_j), \quad (15)$$

which are convex and can be solved efficiently, where $H_j^k{\succ}0$ are the diagonal blocks of H^k, enabling a parallel computation across all agents in J^k, when $|J^k|{\geq}1$. Matrix H^k approximates the Hessian $\nabla^2 R(\boldsymbol{u}^k)$, which is costly to compute. In fact, any $H_j^k \succ 0$ may be used, with sufficient decrease along $\boldsymbol{d}_j^k$ ensured via the inexact line search below.

Armijo-type rule for step-size selection: Let $\sigma, \gamma \in (0,1)$, $\Delta F_\lambda^k = F_\lambda(\boldsymbol{u}^k{+}\alpha^k\boldsymbol{d}^k) - F_\lambda(\boldsymbol{u}^k)$, and $\Delta\mathcal{L}^k = \mathcal{L}(\boldsymbol{u}^k{+}\boldsymbol{d}^k) - \mathcal{L}(\boldsymbol{u}^k)$. We select the largest $\alpha^k{\in}\{1/2^j\}_{j\in\mathbb{N}}$ to ensure sufficient decrease of $F_\lambda(\boldsymbol{u})$ along $\boldsymbol{d}^k$, i.e.,

$$\Delta F_\lambda^k \leq \sigma\alpha^k\left(\lambda\nabla R(\boldsymbol{u}^k)^\intercal\boldsymbol{d}^k{+}\gamma(\boldsymbol{d}^k)^\intercal H^k\boldsymbol{d}^k{+}\Delta\mathcal{L}^k\right). \quad (16)$$

Initialization rule: Let $\mathcal{T}_i = \{\nu \in \mathcal{K}_\phi \mid i \in \nu\}$ be the set of cliques containing i. Assuming $\{i\}{\in}\mathcal{T}_i$ for all $i{\in}\mathcal{V}$, implying that for all $i{\in}\mathcal{V}$ there is an individual task ϕ_i, one can obtain initial guesses for $\boldsymbol{u}_i^0$ by minimizing $\mathcal{L}_i(\boldsymbol{u}_i){+}\lambda\ \max\{0, -\varrho_\Gamma^{\phi_i}(\boldsymbol{u}_i)\}^2$ over $\boldsymbol{u}_i$, for $i{\in}\mathcal{V}$, using, e.g., the toolbox in [21]. Otherwise, one may start with $\boldsymbol{u}^0{=}0$.

Termination: Following [15], Alg. 1 stops when $\|H^k\boldsymbol{d}^k\|_\infty{\leq}\epsilon$.

Proposition 2 *For a fixed penalty parameter $\lambda{>}0$, let $\{\boldsymbol{u}^k\}$ be the sequence generated by Alg. 1 applied to (10). Then every accumulation point of $\{\boldsymbol{u}^k\}$ is a stationary point of (10).*

Algorithm 2 Penalty method (PM)

Input: $\boldsymbol{u}^0$ (initial iterate), λ^0 (initial penalty), K_{PM} (maximum iterations), $\eta_\lambda > 1$ (penalty update factor), $\epsilon_{\text{infeas}} > 0$ (max. infeasibility), ϵ^0 (initial tolerance), $\eta_\epsilon \in (0,1]$ (tolerance update parameter)
$\bar{\boldsymbol{u}}^0 = \boldsymbol{u}^0$
for $k = 0, 1, \ldots, K_{\text{PM}} - 1$ **do**
  if $R(\bar{\boldsymbol{u}}^k) < \epsilon_{\text{infeas}}$ **break**
  Use Alg. 1 to solve (10) with $\lambda = \lambda^k$, initial guess $\bar{\boldsymbol{u}}^k$, and tolerance ϵ^k; obtain ϵ^k-approx. solution $\boldsymbol{u}^{\star,k}$
  $\lambda^{k+1} = \eta_\lambda\lambda^k$, $\bar{\boldsymbol{u}}^{k+1} = \boldsymbol{u}^{\star,k}$, $\epsilon^{k+1} = \eta_\epsilon\epsilon^k$
return $\bar{\boldsymbol{u}}^{\star,k}$

The proof of Proposition 2 follows directly from [15, Theorem 1] tailored to the multi-agent STL setting considered here. For fixed λ, the objective F_λ consists of a proper, convex, continuous block-separable cost and a smooth quadratic penalty function. Consequently, the assumptions of [15] are satisfied, and the proposed BCGD scheme converges to a stationary point of F_λ. The most computationally demanding part of Alg. 1 is computing the block elements $\nabla R(\boldsymbol{u}^k)_j$, followed by the computation of the block update direction $\boldsymbol{d}^k$. Next, we embed the BCGD algorithm in a penalty framework.

D. Penalty method for the constrained problem

Here, we propose a penalty method [16, Chap. 17] for the constrained problem (9). Specifically, we solve a sequence of unconstrained problems of the form (10), each corresponding to a fixed penalty parameter $\lambda > 0$, and iteratively increase λ to progressively enforce feasibility. Following [21, Alg. 1], the quadratic penalty method is given in Alg. 2. Due to the nonconvex nature of the problem, no theoretical guarantees of convergence to a global optimum can be provided. However, as we will demonstrate, in practice Alg. 2 can yield an approximate solution $\bar{\boldsymbol{u}}^\star$ that is ϵ_{infeas}-infeasible, satisfying $\varrho_\Gamma^\phi(\bar{\boldsymbol{u}}^\star){>}{-}\sqrt{\epsilon_{\text{infeas}}}$. Due to the true–smooth STL semantics gap (Prop. 1), such near-feasible solutions may satisfy the original (non-smooth) STL constraints (7b). This behavior is formalized as a conditional guarantee in Theorem 1.

Theorem 1 *Suppose that $|\mathcal{K}_\phi|{\geq}2$ and $\boldsymbol{u}^0$ is such that the sequence $\{\bar{\boldsymbol{u}}^k\}$ of Alg. 2 has a limit point $\bar{\boldsymbol{u}}^\star$ with $R(\bar{\boldsymbol{u}}^\star){=}0$. For a given problem of the form (9), there is $\epsilon^\star_{\text{infeas}}$ such that Alg. 2 with $\epsilon_{\text{infeas}}{\leq}\epsilon^\star_{\text{infeas}}$ returns a feasible solution for (7).*

Although the existence of a limit point, $\bar{\boldsymbol{u}}^\star$, in Theorem 1 is not guaranteed to be met under nonlinear dynamics and general multi-agent STL constraints, Section IV demonstrates the effectiveness of the proposed approach in multi-robot STL planning. Moreover, the "inner" optimization problems (10) are solved efficiently with Alg. 1, while Alg. 2 requires only a small number of "outer" iterations to converge.

IV. Numerical validation via ten-robot example

We evaluate the proposed framework on the ten-robot planning problem in [14], [22], considering the multi-agent specification $\phi{=}\bigwedge_{\nu\in\mathcal{K}_\phi}\phi_\nu$ with horizon $\mathcal{H}^\phi{=}100$ over

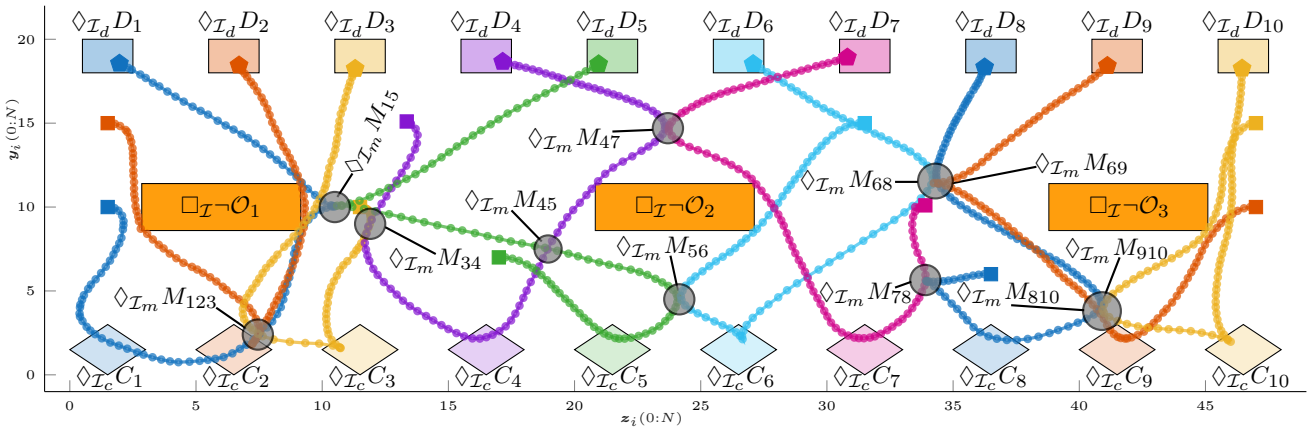

Fig. 1: **RURAMCA** motion planning for ten robots with discrete-time unicycle dynamics: $z_i(t+1) = z_i(t) + v_i(t)\cos\theta_i(t)$, $y_i(t+1) = y_i(t) + v_i(t)\sin\theta_i(t)$, and $\theta_i(t+1) = \theta_i(t) + \omega_i(t)$, where $(z_i(t), y_i(t)) \in \mathbb{R}^2$ denotes the Cartesian position of the i^{th} robot, $\theta_i(t) \in \mathbb{R}$ its heading, and $v_i(t) \in \mathbb{R}$, $\omega_i(t) \in \mathbb{R}$ the linear and angular control inputs, respectively, with a sampling period of 1 s. The trajectories are generated from initial conditions (colored squares) using input sequences obtained via Alg. 2 (using Alg. 1 as inner solver). Simulation setup: $\Gamma{=}2$ for inner smooth operators, $\Gamma{=}1$ for the outer softmin in (7b), $\sigma{=}0.5$ and $\gamma{=}0.995$ (Armijo rule), $\boldsymbol{u}^0{=}0$, $\lambda^0{=}1$, $\eta_\lambda{=}5$, $\epsilon_{\text{infeas}}{=}5.0{\times}10^{-4}$, $\epsilon{=}10^{-6}$, $H^k{=}10^3 I$, $\ell_i{=}v_i(t)^2{+}\omega_i(t)^2$ for all $t{\in}\mathcal{I}$, and $V_{f,i}{=}0$. Block direction update: $\boldsymbol{d}_i^k{=}{-}(10^3\lambda^k{+}2)^{-1}\left(2\boldsymbol{u}_i^k{+}\lambda^k\nabla R(\boldsymbol{u}^k)_i\right)$. $x_i(0)$ (□), $x_i(100)$ (⬠), $i{\in}\mathbb{N}_{[1,10]}$ (filled colored).

$\mathcal{I}{=}\mathbb{N}_{[0,100]}$, where $\mathcal{K}_\phi$ contains 21 cliques. Each task ϕ_ν is either individual or collaborative. For $\nu = \{i\}$, the i^{th} robot must avoid obstacles ($\Box_{\mathcal{I}}\neg\mathcal{O}_l$), visit C_i within $\mathcal{I}_c = \mathbb{N}_{[10,50]}$ ($\Diamond_{\mathcal{I}_c} C_i$), and reach D_i within $\mathcal{I}_d = \mathbb{N}_{[70,100]}$ ($\Diamond_{\mathcal{I}_d} D_i$). The workspace with regions and obstacles is shown in Fig. 1. For $|\nu| > 1$, robots in clique ν must meet ($\Diamond_{\mathcal{I}_m} M_\nu$) within $\mathcal{I}_m{=}\mathbb{N}_{[0,70]}$. The formulas C_i, D_i, $i{\in}\mathcal{V}$, and $\mathcal{O}_l$, $l{\in}\mathbb{N}_{[1,3]}$, are conjunctions of predicates, each defined by a linear predicate function, while $M_\nu{:=}(\min_{\kappa\subseteq\nu}\mu_\kappa(x_\kappa){\geq}0)$, where $x_\kappa{=}(z_q, y_q)_{q\in\kappa}$, with $\kappa{=}(i,j){\subseteq}\nu$ being any subset of two robots in ν, and $\mu_\kappa(x_\kappa) = 0.25 - \left\|[I\ {-}I]\,x_\kappa\right\|$. This scenario is termed R2AM.

We consider two additional planning scenarios building on the R2AM baseline: R2AMCA, which augments R2AM with global inter-agent collision-avoidance specified as $\Box_{\mathcal{I}}\bigwedge_{i\neq j}(\|x_i(t){-}x_j(t)\|{\geq}0.01)$, and RURAMCA, which connects the two reach specifications (R2) in R2AMCA (or R2AM) with the *until* operator specified as $(\Diamond_{\mathcal{I}_c} C_i)\,\mathcal{U}_{[0,50]}(\Diamond_{\mathcal{I}_c} D_i)$. Moreover, robots must fulfill their specific collaborative meeting tasks, $\Diamond_{\mathcal{I}_m} M_\nu$, while ensuring collision-free motion as in the R2AMCA plan.

Motivated by [15], we compare BCGD with LBFGS [16, Chap. 9] implemented in a block-coordinate manner (`jaxopt.LBFGS`), highlighting the modularity of the proposed penalty framework. Both solvers use identical randomized shuffling, updating each block once every 10 iterations. Table I reports runtime, robustness, and feasibility statistics over 100 runs with permuted initial conditions from Fig. 1.

Our numerical results show that BCGD consistently achieves feasible solutions across all scenarios under both linear and unicycle dynamics. Although LBFGS is faster on average (when considering only those problems that it can solve), BCGD is both significantly more robust (it solves all problems) and has a lower 95%-quantile solve time—cf. Table I. These observations are consistent with the benchmarking in [15], albeit outside an STL context.

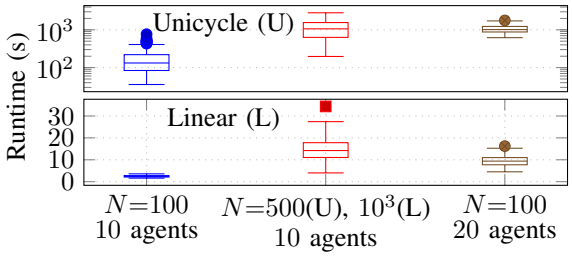

Fig. 2: Runtime boxplots for scalability of BCGD-PM over 100 randomized initial-condition permutations.

To assess scalability, we evaluate BCGD under increasing horizons and problem sizes (see Fig. 2 for runtime statistics). Runtime scales gracefully with the horizon (e.g., $10\times$ increase yields $\sim 6\times$ runtime under linear dynamics) while maintaining feasibility and similar robustness. Doubling agents and cliques leads to subquadratic-to-moderate runtime growth with no loss in feasibility and minor robustness variation. These results indicate favorable scalability of the block-coordinate structure in the reported experiments, particularly under linear dynamics, while remaining reliable for nonlinear models. Additionally, BCGD-PM significantly outperforms our earlier methods in [14], [22] in runtime for R2AM, whereas in R2AMCA and RURAMCA these methods fail to find feasible solutions within a 10^4 s timeout. The proposed algorithm is implemented in Python using JAX. Source code and animations are available at github.com/lefterisvl83/MAS-STL-planning.

V. Conclusions

We presented a scalable framework for multi-agent planning under collaborative STL tasks. We employed a penalty method to tackle the underlying optimization problem under smooth STL semantics, using unconstrained optimization with

Solver	R2AM	R2AMCA	RURAMCA
BCGD (L)	**2.43±0.43** (13.88±11.72) [100]	**2.99±1.29** (10.18±10.36) [100]	**14.07±4.02** (6.55±6.49) [100]
LBFGS (L)	1.62±4.96 (23.13±86.51) [92]	4.19±21.69 (0.66±17.55) [68]	10.23±16.31 (1.62±8.91) [75]
BCGD (U)	**182.86±138.07** (9.36±14.48) [100]	**223.44±130.38** (5.58±6.23) [100]	**337.31±83.81** (3.23±3.44) [100]
LBFGS (U)	83.77±295.29 (4.46±18.00) [88]	97.72±311.45 (3.46±8.24) [83]	180.05±303.65 (2.42±16.88) [80]

TABLE I: Runtime (s), robustness ($\rho \times 10^3$), and feasibility statistics over 100 randomized permutations of the agents' initial conditions in Fig. 1. Entries report mean ± std (robustness in parentheses), with feasibility rates in brackets. L and U denote linear and unicycle dynamics. For each scenario, the solver achieving the lower 95%-quantile runtime is highlighted in bold.

quadratic penalty functions. We showed that the structure of the resulting unconstrained subproblems admits an efficient block-coordinate gradient descent solution, enabling agent-level computations with convergence guarantees while mitigating computational complexity. The proposed framework motivates future research into reactive STL planning for time-varying MAS structures and tasks.

APPENDIX: PROOFS

Proposition 1: Let $a_\nu = \rho^{\phi_\nu}(\boldsymbol{u}_\nu)$ for all $\nu \in \mathcal{K}_\phi$, and let $a_{\min} = \min_{\nu\in\mathcal{K}_\phi} a_\nu = \rho^\phi(\boldsymbol{u})$ denote the true robustness. By definition, the smooth robustness is $\varrho^\phi_\Gamma(\boldsymbol{u}) = -\frac{1}{\Gamma}\log\sum_{\nu\in\mathcal{K}_\phi} e^{-\Gamma a_\nu}$. Writing each a_ν as $a_\nu = a_{\min} + (a_\nu - a_{\min})$ and factoring out $e^{-\Gamma a_{\min}}$ from the summation yields $\sum_{\nu\in\mathcal{K}_\phi} e^{-\Gamma a_\nu} = e^{-\Gamma a_{\min}}\sum_{\nu\in\mathcal{K}_\phi} e^{-\Gamma(a_\nu - a_{\min})}$. Substituting this expression back into the definition of $\varrho^\phi_\Gamma(\boldsymbol{u})$ gives $\varrho^\phi_\Gamma(\boldsymbol{u}) = -\frac{1}{\Gamma}\log\left(e^{-\Gamma a_{\min}}\sum_{\nu\in\mathcal{K}_\phi} e^{-\Gamma(a_\nu - a_{\min})}\right) = a_{\min} - \frac{1}{\Gamma}\log\sum_{\nu\in\mathcal{K}_\phi} e^{-\Gamma(a_\nu - a_{\min})}$. Defining $\delta_\Gamma(\boldsymbol{u}) := \frac{1}{\Gamma}\log\sum_{\nu\in\mathcal{K}_\phi} e^{-\Gamma(a_\nu - a_{\min})}$, we obtain $\varrho^\phi_\Gamma(\boldsymbol{u}) = a_{\min} - \delta_\Gamma(\boldsymbol{u})$. Since $a_\nu \geq a_{\min}$ for all $\nu \in \mathcal{K}_\phi$, it follows that $e^{-\Gamma(a_\nu - a_{\min})} \leq 1$. Because at least one term in $\sum_{\nu\in\mathcal{K}_\phi} e^{-\Gamma(a_\nu - a_{\min})}$ is 1, each a_ν is finite by continuity of the system dynamics and the predicate functions, and $|\mathcal{K}_\phi| \geq 2$ by assumption, the summation inside the logarithm satisfies $1 < \sum_{\nu\in\mathcal{K}_\phi} e^{-\Gamma(a_\nu - a_{\min})} \leq |\mathcal{K}_\phi|$, which implies $0 < \delta_\Gamma(\boldsymbol{u}) \leq \frac{\log|\mathcal{K}_\phi|}{\Gamma}$ for any $\Gamma > 0$, hence, $\varrho^\phi_\Gamma(\boldsymbol{u}) = a_{\min} - \delta_\Gamma(\boldsymbol{u}) < \rho^\phi(\boldsymbol{u})$. Therefore, $\varrho^\phi_\Gamma(\boldsymbol{u}) \geq 0 \Rightarrow \rho^\phi(\boldsymbol{u}) > 0$.

Theorem 1: Let us define the continuous function $\delta_\Gamma(\boldsymbol{u}) := \rho^\phi(\boldsymbol{u}) - \varrho^\phi_\Gamma(\boldsymbol{u}) > 0$ (from Prop. 1), so for every compact set $\mathbb{U} \subset \mathbb{R}^{n_{\boldsymbol{u}}}$, $\alpha_{\mathbb{U}} := \min_{\boldsymbol{u}\in\mathbb{U}} \delta_\Gamma(\boldsymbol{u}) > 0$. Note that for $\mathbb{U} \supseteq \mathbb{U}'$ it holds that $\alpha_{\mathbb{U}} \leq \alpha_{\mathbb{U}'}$. Let $\mathcal{N}(\bar{\boldsymbol{u}}^\star)$ be a compact neighborhood of $\bar{\boldsymbol{u}}^\star$. We have $\delta_\Gamma(\boldsymbol{u}) \geq \alpha_{\mathcal{N}(\bar{\boldsymbol{u}}^\star)}$ for all $\boldsymbol{u} \in \mathcal{N}(\bar{\boldsymbol{u}}^\star)$. By the assumption on the existence of a limit point $\bar{\boldsymbol{u}}^\star$ such that $R(\bar{\boldsymbol{u}}^\star) = 0$, there is a subsequence $\{\bar{\boldsymbol{u}}^{\star,\iota_s}\}_s$ which converges to $\bar{\boldsymbol{u}}^\star$, so there is $s \in \mathbb{N}$ so that $\bar{\boldsymbol{u}}^{\star,\iota_s} \in \mathcal{N}(\bar{\boldsymbol{u}}^\star)$ and, by continuity of R, $R(\bar{\boldsymbol{u}}^{\star,\iota_s}) < \alpha^2_{\mathcal{N}(\bar{\boldsymbol{u}}^\star)}$. Therefore, $\varrho^\phi_\Gamma(\bar{\boldsymbol{u}}^{\star,\iota_s}) > -\alpha_{\mathcal{N}(\bar{\boldsymbol{u}}^\star)}$, hence $\rho^\phi(\bar{\boldsymbol{u}}^{\star,\iota_s}) > 0$.

REFERENCES

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

[2] C. Belta, B. Yordanov, and E. A. Gol, *Formal Methods for Discrete-Time Dynamical Systems*, ser. Studies in Systems, Decision and Control. Springer, 2017, vol. 89.

[3] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.

[4] A. Donzé and O. Maler, "Robust Satisfaction of Temporal Logic over Real-Valued Signals," in *8th International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS, 2010*, Klosterneuburg, Austria, 2010, pp. 92–106.

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

[6] V. Kurtz and H. Lin, "Mixed-Integer Programming for Signal Temporal Logic With Fewer Binary Variables," *IEEE Control Systems Letters*, vol. 6, pp. 2635–2640, 2022.

[7] Y. V. Pant, H. Abbas, and R. Mangharam, "Smooth operator: Control using the smooth robustness of temporal logic," in *2017 IEEE Conference on Control Technol. and Applic. (CCTA)*, 2017, pp. 1235–1240.

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

[9] 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.

[10] S. Han, J. Verhagen, and J. Tumova, "Exact smooth reformulations for trajectory optimization under signal temporal logic specifications," 2025. [Online]. Available: https://arxiv.org/abs/2511.07375

[11] M. Charitidou and D. V. Dimarogonas, "Distributed MPC with continuous-time STL constraint satisfaction guarantees," *IEEE Control Systems Letters*, vol. 8, pp. 211–216, 2024.

[12] X. Zhou, Y. Zou, S. Li, X. Li, and H. Fang, "Distributed model predictive control for multi-robot systems with conflicting signal temporal logic tasks," *IET Control Theory & Appl.*, vol. 16, no. 5, pp. 554–572, 2022.

[13] 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.

[14] E. E. Vlahakis, L. Lindemann, P. Sopasakis, and D. V. Dimarogonas, "Probabilistic tube-based control synthesis of stochastic multi-agent systems under signal temporal logic," in *2024 IEEE 63rd Conference on Decision and Control (CDC)*, 2024, pp. 1586–1592.

[15] P. Tseng and S. Yun, "A coordinate gradient descent method for nonsmooth separable minimization," *Mathematical Programming*, vol. 117, no. 1, pp. 387–423, 2009.

[16] J. Nocedal and S. Wright, *Numerical Optimization*. Springer, 2006.

[17] S. Sadraddini and C. Belta, "Robust temporal logic model predictive control," in *53rd Annual Allerton Conference on Communication, Control, and Computing*. IEEE, 2015, pp. 772–779.

[18] J. Orlin, "Contentment in graph theory: Covering graphs with cliques," *Indagationes Mathematicae*, vol. 80, no. 5, pp. 406–424, 1977.

[19] M. Kazemi and S. Soudjani, "Formal policy synthesis for continuous-state systems via reinforcement learning," in *Integrated Formal Methods: 16th International Conference, IFM 2020, Lugano, Switzerland, November 16–20, 2020, Proceedings 16*. Springer, 2020, pp. 3–21.

[20] P. Tseng, "Convergence of a block coordinate descent method for nondifferentiable minimization," *Journal of Optimization Theory and Applications*, vol. 109, no. 3, pp. 475–94, 2001.

[21] P. Sopasakis, E. Fresk, and P. Patrinos, "OpEn: Code generation for embedded nonconvex optimization," in *IFAC WC*, Berlin, 2020.

[22] E. E. Vlahakis, L. Lindemann, and D. V. Dimarogonas, "Conformal data-driven control of stochastic multi-agent systems under collaborative signal temporal logic specifications," in *2025 IEEE 64th Conference on Decision and Control (CDC)*, 2025, pp. 624–629.