Robotics and Autonomous Systems 142 (2021) 103801

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Distributed safe planning for satisfying minimal temporal relaxations of TWTL specifications☆

Ryan Peterson [a], Ali Tevfik Buyukkocak [a], Derya Aksaray [a,*], Yasin Yazıcıoğlu [b]

[a] Department of Aerospace Engineering and Mechanics, University of Minnesota, Minneapolis, MN, 55455, USA
[b] Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, 55455, USA

ARTICLE INFO

Article history:
Received 24 July 2020
Received in revised form 3 April 2021
Accepted 3 May 2021
Available online 7 May 2021

Keywords:
Multi-agent systems
Distributed planning
Formal methods
Collision avoidance

ABSTRACT

We investigate a multi-agent planning problem, where each agent aims to achieve an individual task while avoiding collisions with other agents. Each agent's task is expressed as a Time-Window Temporal Logic (TWTL) specification defined over a discretized environment. We propose a distributed receding horizon algorithm for online planning of agent trajectories. We show that under mild assumptions on the environment, the resulting trajectories are always safe (collision-free) and lead to the satisfaction of the TWTL specifications or a finite temporal relaxation. Accordingly, each agent is guaranteed to safely achieve its task, possibly with some minimal finite delay. Performance of the proposed algorithm is demonstrated via numerical simulations and experiments with quadrotors.

1. Introduction

The control and coordination of multi-agent systems for package delivery, disaster relief, warehouse logistics, smart transportation, or persistent surveillance have received significant attention in recent years (e.g., [1–5]). Collision avoidance is critical for the safe operation of multi-agent systems while performing such complex tasks over a shared environment. Since the complexity of joint planning grows exponentially with the number of agents, significant effort has been devoted to the design of distributed algorithms with safety and performance guarantees. In this paper, we propose a distributed algorithm for satisfying complex high-level specifications while ensuring safety.

1.1. Related work

In the literature, potential fields (e.g., [6]), sequential convex programming (e.g. [7]), priority based planning (e.g., [8]), control barrier functions (CBFs) (e.g., [9]) or buffered Voronoi cells (e.g., [10]) have been employed for distributed collision avoidance. More recently, some of these methods have been extended to account for constraints on agent dynamics as well as deadlock detection and avoidance (e.g., [11,12]). In particular, [12] explicitly defines different types of deadlock which may occur, and how those deadlocks are accounted for through their implementation of CBFs. There also exist methods such as optimal reciprocal collision-avoidance (ORCA) [13] and a buffered Voronoi cell collision-avoidance strategy which requires less information than ORCA and can produce smooth collision-free trajectories with reduced number of deadlocks [14]. However, these methods do not guarantee deadlock avoidance. Other approaches such as distributed model predictive control (e.g. [15,16]), differential games (e.g., [17]), and game theoretic methods (e.g., [18]) have also been used for distributed planning. The methods described thus far typically do not accommodate complex spatio-temporal requirements that can be expressed as temporal logics.

Recently, there has been a significant interest in the analysis and control of dynamical systems under temporal logic specifications. For instance, linear temporal logic (LTL) [19] has been extensively used in motion planning and control of robots (e.g., [20–22]). To ensure collision avoidance while satisfying more complex tasks defined as LTLs in a distributed manner, hybrid controllers can be employed, which can essentially combine a high-level mission planner with a local planner which enforces collision avoidance (e.g., [23,24]). While these results are promising, LTL cannot express tasks with explicit time constraints. For example, consider an agent that is required to perform the following task: "visit A for 1 time unit within 5 time units, and after this visit B for 3 time units within 10 time units, and visiting C must be performed after visiting both A and B twice". Such tasks with time constraints can be expressed via bounded temporal logics (e.g., [25–28]).

☆ Some preliminary results of this work were presented at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (Peterson et al., 2020). This work was partially supported at the University of Minnesota, United States by Honeywell Aerospace and MnDRIVE, University of Minnesota, United States.

* Corresponding author.
E-mail addresses: pete9936@umn.edu (R. Peterson), buyuk012@umn.edu (A.T. Buyukkocak), daksaray@umn.edu (D. Aksaray), ayasin@umn.com (Y. Yazıcıoğlu).

https://doi.org/10.1016/j.robot.2021.103801
0921-8890/

Ensuring safety under bounded temporal logics for multi-agent systems has received considerable attention in recent years (e.g., [29–32]). When it is impossible to find safe paths that satisfy the original specifications, it may be desired to satisfy some relaxed versions of the specifications instead. The framework in [33] gives feedback on why the specification is not satisfiable and how to modify it. In [34], the robots slow down to give their iterative optimization algorithm more time to solve when necessary. The framework in [32] looks to minimize the violation by considering both hard and soft constraints, where the hard constraints (such as collision avoidance) must be satisfied. Alternatively, our algorithm allows for the temporal relaxation of time-window temporal logic (TWTL) specifications, and we show that, under mild assumptions on the environment topology, our algorithm ensures the completion of all TWTL specifications with finite temporal relaxation.

This work is closely related to [20,29,35]. Similar to this paper, a multi-agent receding horizon algorithm is proposed in [20] to generate each agent's path independently using only local information. However, each task was defined by LTL (which cannot express explicit time constraints) and collision avoidance was not considered. In [35], collision avoidance was ensured by penalizing transitions in the centralized graph which captures the environment and TWTL specifications for all agents in the system. This centralized algorithm is not scalable as the number of agents increases. Moreover, in [35], if a safe path satisfying the TWTL cannot be found, the algorithm terminates and does not allow for relaxations of the TWTL specifications. Finally, the work in [29] considers a global task that needs to be achieved by the multi-agent system and allows for temporal relaxation with TWTL specifications. However, collision avoidance is not incorporated in path planning.

The collision avoidance procedure in this work is closely related to [36,37]. In [36], a distributed prioritized planning algorithm is introduced, and [37] is essentially an asynchronous extension of [36] called *revised priority planning*. These algorithms assume complete peer-to-peer communication of trajectories while planning, and the entire path must be calculated before execution which does not lend itself well to unexpected interruptions. Our proposed collision avoidance procedure dynamically allocates priorities based on how close the agents are to completing their tasks. Our algorithm differs from previous versions of distributed prioritized planning (e.g., [36,37]) by (1) detecting and resolving deadlocks (under mild assumptions on the environment), (2) considering partial/local information while computing agent paths in a receding horizon manner using dynamic programming, and (3) having the capability of changing goal regions of agents (allowable by the temporal logic) while avoiding collisions and resolving deadlocks (whereas state-of-the-art collision avoidance algorithms assume fixed goal regions for each agent).

1.2. Contributions

This paper introduces a distributed algorithm for safe satisfaction of TWTL specifications, which can efficiently express complex time bounded tasks [28], for multi-agent systems operating in shared environments. In this paper, each agent is assigned with an individual TWTL specification and is assumed to move over a discretized environment. Agents communicate with the other agents in their local neighborhoods to plan collision-free paths in a receding horizon manner. The main contribution of this paper is a distributed algorithm for safe satisfaction of TWTL specifications by resolving conflicts via online re-planning, relaxing time windows if infeasibility occurs, and resolving possible deadlocks when detected. We theoretically show that the proposed algorithm produces individual agent paths that avoid both collisions and deadlock, and complete tasks in finite time.

Some preliminary results of this work has been presented in [38]. This paper is a significant extension of [38], and the main differences are as follows: (1) [38] shows that if the environment has n-connectivity where n is the number of agents, there will not occur any deadlocks. Thus, [38] shows that agent paths with finite temporal relaxations can always be found under this environment assumption. In this paper, we eliminate the n-connectivity assumption and introduce a deadlock resolution algorithm that can resolve conflicts under an assumption that is milder than n-connectivity (i.e., the premise of Lemma 1). We also prove that the proposed algorithm can find paths with finite relaxations by resolving deadlocks (Theorem 1); (2) in this paper, we generalize the formulation and introduce a mapping for the conflicting transitions. Accordingly, one can flexibly define conflicting transitions depending on the problem's assumptions and the capabilities of the low-level controller. (3) Moreover, [38] finds agent paths via exhaustive search. In this paper, we implement dynamic programming, which dramatically reduces the computational time to solve for the agent paths; (4) Finally, we present a benchmark analysis and compare our algorithm with some of the well-known methods in addition to different priority settings.

1.3. Organization

The remainder of the paper is organized as follows: Section 2 introduces TWTL, its temporal relaxation, and some graph theory preliminaries. Section 3 states the problem. The proposed algorithm and theoretical results are presented in Section 4. Experimental results on a team of quadrotors, numerical results, and benchmark analysis are shown in Section 5. The paper is concluded with a summary and possible future work in Section 6.

2. Preliminaries

2.1. Notation

We denote $\mathbb{Z}, \mathbb{Z}^+, \mathbb{R}, \mathbb{R}^+$ as the set of all integers, positive integers, real numbers, and positive real numbers, respectively. Throughout this paper, t denotes discrete time. Sets are often given by capital letters, S, and lower case letters denote a member of a set, $s \in S$. Vectors or sequences are shown in bold, $\mathbf{s}$. Cardinality is indicated by $|\cdot|$ (i.e., the number of elements in a finite vector, sequence, or set), and 2^S indicates the power set (set of all subsets) of S.

2.2. Time Window Temporal Logic (TWTL)

Syntax and Semantics: The syntax of TWTL is defined as:

$$\phi := \alpha \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \phi_1 \cdot \phi_2 \mid \mathrm{H}^d\alpha \mid [\phi]^{[a,b]},$$

where ϕ is a TWTL specification, $\alpha \in AP$ is an atomic proposition that has a binary truth value (e.g., being at region α or not) and AP is the set of atomic propositions; $\wedge$, $\vee$, and $\neg$ are the Boolean operators for conjunction, disjunction, and negation, respectively; $\cdot$ is the concatenation operator such that $\phi_1 \cdot \phi_2$ specifies that first ϕ_1 and then immediately ϕ_2 must be satisfied. The semantics are defined with respect to finite output words $\mathbf{o}$ over AP where $\mathbf{o}(t)$ denotes the discrete-time element on $\mathbf{o}$. The *hold* operator $\mathrm{H}^d\alpha$ specifies that an atomic proposition $\alpha \in AP$ should be true for d time units (i.e., $\mathbf{o} \models \mathrm{H}^d\alpha$ if $\mathbf{o}(t) = \alpha \quad \forall t \in [0, d]$), while the *within* operator $[\phi]^{[a,b]}$ bounds the satisfaction of ϕ within a time window $[a, b]$.

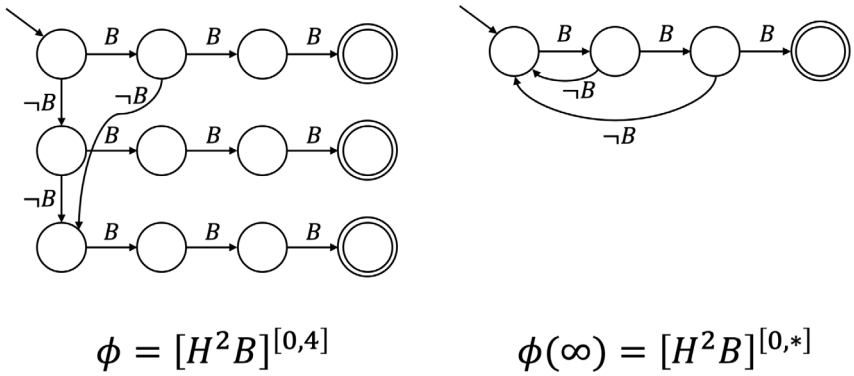

Fig. 1. [Left] Conventional dFSA of a TWTL ϕ, [Right] Annotated dFSA that represents all temporal relaxations of ϕ.

Temporal Relaxation: Given a TWTL specification, its temporal relaxation can be written by adding slack variables to the time windows of each hold operator to represent shorter or longer time windows. For instance, consider the following TWTL formula:

$$\phi = [\mathrm{H}^2 A]^{[0,5]} \cdot \left[\mathrm{H}^1 B \wedge [\mathrm{H}^2 C]^{[0,6]}\right]^{[2,10]}$$

In plain English, ϕ can be interpreted as: "Service A for 2 time units within the time bound [0,5], then immediately after this, service B for 1 time unit and service C for two time units within [0,6], which both must be performed within [2,10]". The relaxed version of ϕ is written as:

$$\phi(\boldsymbol{\tau}) = [\mathrm{H}^2 A]^{[0,5+\tau_1]} \cdot \left[\mathrm{H}^1 B \wedge [\mathrm{H}^2 C]^{[0,6+\tau_2]}\right]^{[2,10+\tau_3]}$$

where $\boldsymbol{\tau} = (\tau_1, \tau_2, \tau_3) \in \mathbb{Z}^3$ is the temporal relaxation vector. Overall, the temporal relaxation of $\phi(\boldsymbol{\tau})$ is quantified by $|\boldsymbol{\tau}|_{TR} = \max_j(\tau_j)$ [28].

For any TWTL specification, ϕ, an annotated deterministic finite state automaton (dFSA) can be constructed that captures all feasible temporal relaxations of ϕ. Specifically, for the example given above, this means the dFSA captures all possible values of $(\tau_1, \tau_2, \tau_3) \in \mathbb{Z}^3$ which modify the time windows of the initial TWTL formula.

Definition 1 (*Annotated Deterministic Finite State Automaton [28]*)**.** Consider a TWTL formula ϕ that is defined over the set of atomic propositions AP. An annotated dFSA of ϕ represents all temporal relaxations of ϕ, and it is a tuple $\mathcal{A}_\infty = (S_{\mathcal{A}_\infty}, s_0, \Sigma, \delta_{\mathcal{A}_\infty}, F_{\mathcal{A}_\infty})$ where

- $S_{\mathcal{A}_\infty}$ *is a finite set of states;*
- $s_0 \in S_{\mathcal{A}_\infty}$ *is the initial state;*
- $\Sigma = 2^{AP}$ *is the input alphabet;*
- $\delta_{\mathcal{A}_\infty} : S_{\mathcal{A}_\infty} \times \Sigma \to S_{\mathcal{A}_\infty}$ *is the transition function;*
- $F_{\mathcal{A}_\infty} \subseteq S_{\mathcal{A}_\infty}$ *is the set of accepting states.*

Consider a TWTL specification $\phi = [H^2B]^{[0,4]}$, which means that "service region B for two consecutive time steps within a time window $[0, 4]$". A conventional dFSA of ϕ represents all possible satisfactory cases within the specified time window. For instance, the left graph in Fig. 1 illustrates the conventional dFSA of ϕ. On the other hand, the right graph in Fig. 1 illustrates the annotated dFSA that captures all satisfactory cases for all temporal relaxations of ϕ. This is mainly achieved by adding backward edges, which also result in a more compact representation for the annotated dFSA. In this paper, we construct the annotated dFSA of each TWTL specification, and the reader is referred to [28] for further details on the construction and properties of the annotated dFSA.

Once we construct the annotated dFSA, we also add a self-transition to each accepting state $s \in F_{\mathcal{A}_\infty}$. This modification enables an agent to continue mission execution after it satisfies its specification.

2.3. Graph theory

A weighted directed graph is a tuple $\mathcal{G} = (X, \Delta, w)$ where X is a set of nodes, $\Delta \subseteq X \times X$ is a set of edges between the nodes, and $w : \Delta \to \mathbb{R}^+$ denotes the weight function. We use $\Delta_x^{in}, \Delta_x^{out} \subseteq \Delta$ to denote the set of incoming and outgoing edges of a node $x \in X$. Accordingly, $\Delta_x = \Delta_x^{in} \cup \Delta_x^{out}$ denotes the set of all edges incident to x. A node $x' \in X$ is said to be an out-neighbor of another node $x \in X$ if $(x, x') \in \Delta$. We use $\mathcal{N}(x)$ to denote the set of out-neighbors of x. For brevity, we will use the term "neighbor" when referring to an "out-neighbor". We use $\mathcal{N}^H(x)$ to denote the set of all nodes that are reachable from x in at most H hops.

A graph is strongly connected if there exists a path from any node x to any other node x'. A path $\mathbf{x}$ on a graph $\mathcal{G}$ is a sequence of nodes such that there exists an edge from any node in $\mathbf{x}$ to the next node in the sequence. We use $|\mathbf{x}|$ to denote the path length, i.e., the total number of edges traversed on $\mathbf{x}$. The weighted graph distance between the nodes, $d(x, x')$, is equal to the cumulative weight of edges traversed along the shortest (minimum cumulative weight) path from x to x'.

Let $X' \subseteq X$ be a subset of nodes. The set X' induces a sub-graph that is the graph with the node set X' and the edge set consisting of the edges between the nodes in X'. A (strongly) connected component in $\mathcal{G}$ is a maximal sub-graph in which every node is reachable from every other node in the sub-graph.

3. Problem statement

3.1. Agent model

Dynamics: We consider a set of identical agents, $N = \{1, \ldots, n\}$, moving in a discretized 3D environment, whose abstraction is initially given as a graph $\mathcal{G} = (X, \Delta, w)$. In general, several methods (e.g., [39,40]) can be used to construct such an abstraction; however, the construction of the abstraction is not in the scope of this paper. Given an environment graph, $\mathcal{G}$, we model the dynamics of each agent as a deterministic weighted transition system. Moreover, the agents move synchronously on $\mathcal{G}$ meaning state transitions for any agent happen at the same time.

Definition 2 (*Deterministic Weighted Transition System*)**.** A deterministic weighted transition system (dWTS) is a tuple $\mathcal{T} = (X, x_0, \Delta, w, AP, l)$ where:

- X *is a finite set of states;*
- $x_0 \in X$ *is the initial state;*
- $\Delta \subseteq X \times X$ *is the set of transitions;*

- $w : \Delta \to \mathbb{R}^+$ is the weight function;
- *AP is a finite set of atomic propositions;*
- $l : X \to 2^{AP}$ *is the labeling function.*

We make a mild assumption that the transition system $\mathcal{T}$ is (strongly) connected, that is, any $x' \in X$ can be reached from any $x \in X$ via a finite number of transitions. If this assumption does not hold, then there exists a state x'' in the environment such that some states are not reachable from x''. This would cause some arbitrary specifications to be infeasible due to the lack of basic connectivity requirements of the environment. To avoid such infeasibilities, we exclude the structural issues of the environment by assuming it strongly connected.

A path (or run) of the system is a sequence of states $\mathbf{x} = x_0 x_1 \ldots$. This path $\mathbf{x}$ generates an output word $\mathbf{o} = o_0 o_1 \ldots$, where $o_t = l(x_t)$, $\forall t \geq 0$. The language corresponding to a transition system $\mathcal{T}$ is the set of all generated output words, denoted by $\mathcal{L}(\mathcal{T})$. The weight of a transition can be defined by the time or input (e.g., fuel) cost required to traverse transitions or a combination of the two. Without loss of generality, we will consider normalized weights, i.e., $w(x, x') \in (0, 1]$ for all $(x, x') \in \Delta$.

Communications: We consider a setting where each agent plans its own trajectory in a receding horizon manner to safely complete its own task. Accordingly, for any horizon length $H \geq 1$, each agent needs to communicate with the other agents within $2H$-hops of its current position on the environment graph to compute a collision-free path over the next H time steps. We assume that the agents have such local ($2H$-hop) communication capability, and we use $N_i^{2H} \subseteq N$ to denote the set of all agents within $2H$-hops of agent i's current position. Note that we do not assume a separate communication graph for agents. Communicating with agents within $2H$-hop neighborhood implies that communicating with agents that are $2H$-hop distance away from the agent's current position. Such a local communication capability defines a connected multi-hop communication network for each agent. We use $\bar{N}_i \subseteq N$ to denote the set of agents that are connected to agent i through such multi-hop communications. Accordingly, $\bar{N}_i$ is the smallest set of agents that satisfy the following two conditions:

$$i \in \bar{N}_i, \quad \text{and } N_j^{2H} \subseteq \bar{N}_i, \ \forall j \in \bar{N}_i. \tag{1}$$

Note that both N_i^{2H} and $\bar{N}_i$ are determined by the current positions of the agents and change over time as the agents move. In our proposed algorithms, such local communications are mainly used to provide each agent i with two types of information: (1) H-hop path of each agent in N_i^{2H}, and (2) current position, priority, and update indicator of each agent in $\bar{N}_i$. These variables will be clearly defined in Section 4.

Specifications: Each agent i aims to satisfy a TWTL formula, ϕ_i, that is defined over the atomic proposition set, AP, of the transition system $\mathcal{T}_i$. It is assumed that agents do not know about the other agents' specifications. In the presence of violations, instead of terminating the mission, each agent i tries to satisfy a minimal temporal relaxation of ϕ_i, i.e., $\phi_i(\boldsymbol{\tau}^i)$, which will be formally defined in the problem statement.

3.2. Problem statement

Suppose that there are n identical agents, each of which has its own respective transition system $\mathcal{T}_i$. We address the problem of planning paths such that each agent satisfies its individual TWTL specification while avoiding collisions with the others. In order to enforce collision avoidance, the agents require some representation of infeasible transitions (i.e., transitions which may result in collision).

Definition 3 (*Conflicting Transition*)**.** A conflicting transition is a transition which is not allowed while a transition (x, x') is being traversed.

For example, if an agent takes a transition (x, x') to move to state x', then another agent located at x' may not be allowed to stay in the same state. Then, given a transition (x, x'), the transition (x', x') is defined as a conflicting transition. A mapping of all conflicting transitions is defined for every transition $(x, x') \in \Delta$ as $C_{\mathcal{G}}$, where $C_{\mathcal{G}} : \Delta \to 2^{\Delta}$. A particular set of conflicting transitions for some transition (x, x') is denoted as $C_{\mathcal{G}}(x, x')$. Hence, when an agent takes a transition (x, x'), the other agents cannot take any of the transitions in $C_{\mathcal{G}}(x, x')$ in the same time step. We make the following mild assumptions on the mapping $C_{\mathcal{G}}$.

Assumption 1. All conflicts are local on $\mathcal{G}$ such that

$$(x, x') \in C_{\mathcal{G}}(y, y') \Rightarrow x \in \mathcal{N}^2(y), \tag{2}$$

and they are symmetric, i.e., for every $x, x', y, y' \in X$

$$(x, x') \in C_{\mathcal{G}}(y, y') \iff (y, y') \in C_{\mathcal{G}}(x, x'). \tag{3}$$

Furthermore, a self-transition can only conflict with the transitions incoming to that state, i.e.,

$$C_{\mathcal{G}}(x, x) \subseteq \Delta_x^{in}, \ \forall x \in X. \tag{4}$$

Accordingly, we say that the paths of agents are safe if the concurrent transitions do not conflict with each other.

Definition 4 (*Safe Path*)**.** A path of agent i, $\mathbf{x}_i = x_t^i x_{t+1}^i \ldots$ over the environment graph $\mathcal{G} = (X, \Delta, w)$ is safe if, for all $t \geq 0$ and for all $j \neq i$, $(x_t^i, x_{t+1}^i) \notin C_{\mathcal{G}}(x_t^j, x_{t+1}^j)$.

We aim to solve a multi-agent path planning problem which results in safe paths over $\mathcal{G}$ that satisfy the individual TWTL specifications with minimal temporal relaxation.

Problem 1. Let a multi-agent system consist of n identical agents, each of which has a transition system $\mathcal{T}_i$ and an individual TWTL specification ϕ_i. Find safe paths $\mathbf{x}_1, \ldots, \mathbf{x}_n$ that satisfy minimally relaxed TWTL specifications $\phi_1(\boldsymbol{\tau}^1), \ldots, \phi_n(\boldsymbol{\tau}^n)$, i.e.,

$$\begin{aligned} &\min_{\mathbf{x}_1, \ldots, \mathbf{x}_n} \sum_{i=1}^{n} |\boldsymbol{\tau}^i|_{TR} \\ &s.t. \ \ (x_t^i, x_{t+1}^i) \notin C_{\mathcal{G}}(x_t^j, x_{t+1}^j), \ \forall i \neq j \in \{1, \ldots, n\}, \\ &\quad \mathbf{o}_i \models \phi_i(\boldsymbol{\tau}^i), \quad \forall i \in \{1, \ldots, n\}. \end{aligned} \tag{5}$$

where $\mathbf{o}_i$ is the output word associated with the state path $\mathbf{x}_i = x_0^i x_1^i \ldots$ over $\mathcal{T}_i$, and $|\boldsymbol{\tau}^i|_{TR} \in \mathbb{Z}$ is the temporal relaxation of ϕ_i.

3.3. Challenges of a centralized solution

The multi-agent problem defined in (5) can be solved via an automata-theoretic approach. To this end, a product automaton $\mathcal{P}_i$ can be constructed for each agent given its transition system $\mathcal{T}_i$ and its dFSA $\mathcal{A}_{\infty,i}$. The purpose of the product automaton is to encode all possible satisfactory cases given the feasible movement on the transition system.

Definition 5 (*Weighted Product Automaton*)**.** Let $\mathcal{T} = (X, x_0, \Delta, w, AP, l)$ be a transition system and $\mathcal{A}_\infty = (S_{\mathcal{A}_\infty}, s_0, \Sigma, \delta_{\mathcal{A}_\infty}, F_{\mathcal{A}_\infty})$ be a finite state automaton capturing all temporal relaxations of a TWTL specification. A (weighted) product automaton $\mathcal{P} = \mathcal{T} \times \mathcal{A}_\infty$ is a tuple $\mathcal{P} = (S_{\mathcal{P}}, p_0, \Delta_{\mathcal{P}}, w_p, F_{\mathcal{P}})$, where

- $S_{\mathcal{P}} = X \times S_{\mathcal{A}_\infty}$ *is the finite set of states;*

- $p_0 := (x_0, s_0) \in S_{\mathcal{P}}$ *is the initial state;*
- $\Delta_{\mathcal{P}} \subseteq S_{\mathcal{P}} \times S_{\mathcal{P}}$ *is the set of transitions;*
- $w_p : \Delta_{\mathcal{P}} \to \mathbb{R}^+$ *is the weight function defined as:* $w_p\big((x,s),(x',s')\big) = w\big((x,x')\big)$;
- $F_{\mathcal{P}} = X \times F_{\mathcal{A}_\infty}$ *is the set of accepting states.*

Let $p = (x, s) \in S_{\mathcal{P}}$ and $p' = (x', s') \in S_{\mathcal{P}}$ be states in product automaton $\mathcal{P}$. A transition from p to p', i.e., $(p, p') \in \Delta_{\mathcal{P}}$, implies a transition $(x, x') \in \Delta$ and $\delta(s, l(x')) = s'$. The notions of path and acceptance are the same as in the dFSA. A satisfying run of $\mathcal{T}$ with respect to ϕ can be obtained by computing a path from the initial state to an accepting state over $\mathcal{P}$ and projecting the path onto $\mathcal{T}$.

The centralized solution of (5) requires construction of an aggregated product automaton $\mathcal{P}_{full} = \mathcal{T}_{full} \times \mathcal{A}_{\infty,full}$ where $\mathcal{T}_{full} = \mathcal{T}_1 \times \mathcal{T}_2 \times \cdots \times \mathcal{T}_n$, and $\mathcal{A}_{\infty,full} = \mathcal{A}_{\infty,1} \times \mathcal{A}_{\infty,2} \times \cdots \times \mathcal{A}_{\infty,n}$. Such an aggregated product automaton captures all possible agent movements and enables to find safe paths satisfying the TWTL specifications (or their temporal relaxations). While such an approach can result in optimal agent paths, the complexity of constructing $\mathcal{P}_{full}$ exponentially grows as the number of agents n increases. Hence, we propose a distributed approximate solution to (5) by constructing the individual product automaton of each agent and solve a planning problem over individual product automata in a distributed fashion.

4. Receding horizon safe path planning with TWTL satisfaction

The proposed distributed algorithm comprises two parts. In the offline portion, a product automaton $\mathcal{P}_i$ is constructed for each agent given its transition system $\mathcal{T}_i$ and dFSA $\mathcal{A}_{\infty,i}$ (representing all temporal relaxations of a TWTL specification ϕ_i). The energy of each state $p \in \mathcal{P}_i$ is computed as will soon be discussed.

In the online portion of the algorithm, agents move according to their updated H-hop paths $\mathbf{p}_i$ at each time step over $\mathcal{P}_i$. Note that an updated path computed over $\mathcal{P}_i$ is $\mathbf{p}_i = (x_t^i, s_t^i)(x_{t+1}^i, s_{t+1}^i) \ldots$, thus the corresponding path $\mathbf{x}_i$ on $\mathcal{T}_i$ can always be extracted from $\mathbf{p}_i$ (using the projection of the path $\mathbf{p}_i$ onto $\mathcal{T}_i$).

Whenever an agent encounters other agents in its local neighborhood, a negotiation protocol, which assigns priorities to the agents, is performed. This protocol is required to decide which agents are "yielded" to by considering their respective paths for collision avoidance. Such a priority assignment is partially achieved by using an energy function defined over the product automaton states.

Definition 6 (*Energy Function*)**.** The energy of a state p over a product automaton $\mathcal{P}$ is defined similar to [41] as,

$$E(p, \mathcal{P}) = \begin{cases} \min\limits_{p' \in F_{\mathcal{P}}} d\big(p, p'\big), & \text{if } p \notin F_{\mathcal{P}}, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

where $d\big(p, p'\big)$ is the weighted graph distance between p and p'. If there is no such reachable accepting state $p' \in F_{\mathcal{P}}$, the energy of p is $E(p, \mathcal{P}) = \infty$. Accordingly, the energy function serves as a measure of "progress" toward satisfying the given TWTL specification. Any states in $\mathcal{P}$ with infinite energy, $E(p, \mathcal{P}) = \infty$, are pruned from $\mathcal{P}$ to ensure the satisfiability of the specification. Note that the energy function, $E(p, \mathcal{P})$, differs for each agent since each agent has a different product automaton, $\mathcal{P}_i$, with different accepting states, $F_{\mathcal{P}_i}$.

Definition 7 (*Agent Priorities*)**.** Each agent $i \in \{1, \ldots, n\}$ has a time-varying priority determined by $\pi_t^i = (i, \eta_t^i)$, where

$$\eta_t^i = \begin{cases} \dfrac{1}{E(p_t^i, \mathcal{P}_i)}, & \text{if } p_t^i \notin F_{\mathcal{P}_i}, \\ 0, & \text{otherwise.} \end{cases}$$

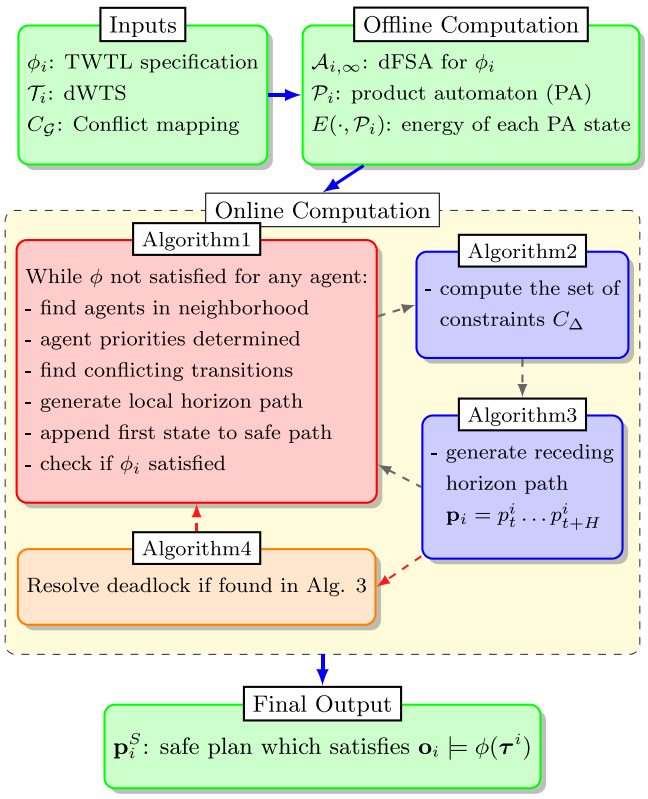

Fig. 2. Outline of the proposed algorithm.

Agent i has higher priority than agent j, i.e., $\pi_t^i > \pi_t^j$, if either of the following is true: (1) $\eta_t^i > \eta_t^j$, (2) $\eta_t^i = \eta_t^j$ and $i < j$. Accordingly, no pair of agents have equal priorities and Π_t denotes the sequence of agent ID numbers sorted based on their priorities from the highest to the lowest.

In the remainder of the paper, we will omit the time index of Π_t whenever it is clear from the context. In accordance with the agent priorities as defined above, the highest priority agent is the one who has not completed its task but is the closest to completion. Agents who have completed their tasks have the lowest priority. The ID numbers are only used to break ties (lower ID number implies higher priority). In our proposed planning approach, each agent i yields to the other agents with higher priority by avoiding transitions in conflict with their transitions, i.e.,

$$(x_t^i, x_{t+1}^i) \notin C_{\mathcal{G}}(x_t^j, x_{t+1}^j), \ \forall j : \pi_t^j > \pi_t^i. \tag{7}$$

Since the conflicts are symmetric as per (4), (7) is equivalent to the safety constraint in (5). We say that an agent i is in deadlock if it has no feasible transitions (x_t^i, x_{t+1}^i) to satisfy (7). Resolving such deadlocks requires re-planning for some higher priority agents as we will explain in the next subsection.

Definition 8 (*Deadlock*)**.** We say that an agent i is in deadlock at time t if, given (x_t^j, x_{t+1}^j) for all j such that $\pi_t^j > \pi_t^i$, there exists no feasible transition $(x_t^i, x_{t+1}^i) \in \Delta$ that satisfies (7).

4.1. Algorithm descriptions

The high-level outline of the proposed algorithm is shown in Fig. 2. Note that the proposed algorithm shown is for a single agent and runs for each agent separately in a distributed manner.

Prior to execution, the dFSA is constructed from the agent's respective TWTL formula ϕ_i, the agent's product automaton $\mathcal{P}_i$ is generated, and the energy of each state in the product automaton $E(\cdot, \mathcal{P}_i)$ is computed.

We use $N_i^{2H} \subseteq N$ to denote the set of all agents within $2H$ hops of agent i's current position and $\bar{N}_i \subseteq N$ to denote the set of agents that may indirectly affect agent i's current plan as per (1). Note that when the agents have a communication range of at least $2H$ hops, each agent i can exchange information with the agents in N_i^{2H} via direct communications and with the agents in $\bar{N}_i$ via multi-hop communications (messages relayed by intermediate agents). In our proposed algorithms, such local communications are used to provide each agent i with two types of information: (1) H-hop path $\mathbf{p}_j = p_t^j, p_{t+1}^j, \ldots, p_{t+H}^j$ of each agent $j \in N_i^{2H}$, and (2) current position x_t^j, priority π_t^j, and update indicator U_{flag}^j of each agent $j \in \bar{N}_i$. The update indicator is required for a proper path update protocol. Furthermore, when agent i is in deadlock, all agents in $\bar{N}_i$ share their current state $p_t^j \in S_{\mathcal{P}_j}$, and all higher priority agents $\overline{HP}_i \subseteq \bar{N}_i$ share their desired next state $p_{t+1}^j \in S_{\mathcal{P}_j}$ with agent i. The need for this information is discussed below in further detail.

We now consider Algorithms 1–3, which run online at every time step, then discuss Alg. 4 which is performed only if deadlock occurs. Note in Alg. 1, whenever agent i "obtains" information from other agents it is assumed that the agent broadcasts a request to the desired agents and that request for information is satisfied (i.e., the information is communicated to agent i at that moment). This is similarly performed in Alg. 4. First, we discuss Alg. 1 which takes in the initial information computed offline.

In Alg. 1, agent i first computes its priority based on Definition 7. Then, it obtains the priorities and current positions from all other agents in $\bar{N}_i$ (line 5). Next, the priority ordering for all $\bar{N}_i$ agents is computed using Definition 7 (line 6). The set of agents in the local neighborhood N_i^{2H} which have a higher priority, $HP_i \subseteq N_i^{2H}$, is computed, and agent i obtains the update indicator from each of these agents (line 7). The loop in lines 8–12 is incorporated to ensure that all agents in HP_i have updated their paths (based on their local neighborhoods) before agent i's path can be updated. If lines 10–12 are executed this means a higher priority agent was found to be in deadlock and the next desired state for agent i, p_{des}^i, which ensures deadlock resolution was found in Alg. 4. Therefore, the updated path can be generated (line 11) and the standard receding horizon update is not required (line 12). The loop in lines 8–12 is required to ensure the path information obtained from all agents in agent i's local neighborhood, N_i^{2H}, is current (line 13), and that proper conflict transitions are generated by Alg. 2 (line 14). These conflicting transitions, $C_\Delta \subseteq \Delta$, are then used in Alg. 3 (line 15) in order to find a finite horizon safe path over $\mathcal{P}_i$ based on Definition 4. Line 16 ensures the agent has an H-hop path since Alg. 3 may result in varied path lengths (discussed later).

If the agent is in deadlock (determined by Alg. 3), then the current state is obtained from all agents in $\bar{N}_i$ and the desired next state is obtained from all higher priority agents in $\overline{HP}_i \subseteq \bar{N}_i$ to be used in Alg. 4 (line 18). The set of desired next states for all corresponding agents required for deadlock resolution (this includes agent i), P_{t+1}, is returned by Alg. 4 (line 19). The desired next state for agent i is extracted from P_{t+1}, the updated H-hop path is generated, and the update flag is set to *true* (lines 20–22). Then the next desired state for each agent corresponding to P_{t+1}, found in Alg. 4, is broadcast to those agents (line 23).

Whether deadlock occurred or not, agent i next obtains the update indicator from all other agents in $\bar{N}_i$ (line 24). The loop in lines 25–28 ensures synchronous movement amongst all agents in $\bar{N}_i$ and updating the local path if a new next state is received from a lower priority agent due to deadlock. As in conventional receding horizon approaches, only the first new state p_{t+1}^i of the generated path is executed and the time is updated (line 29).

Algorithm 1: Online Safe Path Planning for Agent i

Input: $\mathcal{P}_i$, $E(\cdot, \mathcal{P}_i)$, $C_\mathcal{G}$; $\mathcal{G}$ - PA, Energy of PA states, Conflict mapping, Environment graph

1 **Note:** $p_t^i = (x_t^i, s_t^i)$ is an element of $\mathbf{p}_i = p_t^i, \ldots, p_{t+H}^i$;
2 **Initialization:** $x_t^i = p_{x,t}^i$; $t = 0$; $U_{flag}^i = false$
3 **while** *(1)* **do**
4 Compute π_t^i via Definition 7 ;
5 Obtain π^k and $x_t^k \in X$, from all agents $k \in \bar{N}_i$;
6 Compute priority ordering $\overline{\Pi}$ for all agents $k \in \bar{N}_i$;
7 Compute set of higher priority neighbors $HP_i \subseteq N_i^{2H}$, and obtain U_{flag}^j from all $j \in HP_i$;
8 **while** U_{flag}^j *is false for any agent* $j \in HP_i$ **do**
9 Obtain U_{flag}^j from all $j \in HP_i$;
10 **if** p_{des}^i *is received from a higher priority agent* **then**
11 $\mathbf{p}_i = p_t^i, p_{des}^i, \ldots, p_{des}^i$;
12 $U_{flag}^i = true$; jump to line 24;
13 Obtain $\mathbf{p}_j$ from all $j \in HP_i$ agents;
14 $C_\Delta \leftarrow$ **Alg2**$(HP_i, p_t^i, \mathbf{p}_j | \forall j \in HP_i, C_\mathcal{G}, \mathcal{P}_i)$
15 $\mathbf{p}_i, U_{flag}^i, D_{flag} \leftarrow$ **Alg3**$(\mathcal{P}_i, p_t^i, C_\Delta)$
16 Append the last element of $\mathbf{p}_i$ at the end of $\mathbf{p}_i$ for $H - |\mathbf{p}_i|$ times;
17 **if** $D_{flag} == true$ **then**
18 Obtain p_t^k from all $k \in \bar{N}_i$, and p_{t+1}^k from all $k \in \overline{HP}_i$ agents;
19 $P_{t+1} \leftarrow$ **Alg4**$(p_t^k$ and $p_{t+1}^k \forall k \in \bar{N}_i, \overline{\Pi}, C_\mathcal{G}, \mathcal{G})$;
20 $p_{des}^i \in P_{t+1}$;
21 $\mathbf{p}_i = p_t^i, p_{des}^i, \ldots, p_{des}^i$;
22 $U_{flag}^i = true$;
23 Broadcast desired next state $p_{des}^k \in P_{t+1}$ to the corresponding agents;
24 Obtain U_{flag}^k from all agents $k \in \bar{N}_i$;
25 **while** U_{flag}^k *is false for any agent* $k \in \bar{N}_i$ **do**
26 Obtain U_{flag}^k from agents $k \in \bar{N}_i$;
27 **if** p_{des}^i *is received from a lower priority agent* **then**
28 $\mathbf{p}_i = p_t^i, p_{des}^i, \ldots, p_{des}^i$;
29 Move to p_{t+1}^i; $U_{flag}^i = false$; $t = t + 1$;

Algorithm 2: Generate Conflict Transitions for Agent i

Input: $HP_i, p_t^i, C_\mathcal{G}, \mathcal{P}_i$ - Agents with higher priority than agent i, current state of agent i, conflict mapping, product automaton
Input: $\mathbf{p}_j | \ \forall j \in HP_i$ - Recall $\mathbf{p}_j = p_t^j, \ldots, p_{t+H}^j$
Output: C_Δ - Set of conflict transitions

1 **Note:** $x_h^i \neq x_h^j, \forall j \in HP_i$
2 **Initialization:** $C_\Delta = \emptyset$, $C_{\Delta,h} = \emptyset$
3 $C_{\Delta,h} = \{(x, x') \mid (x, x') \in C_\mathcal{G}(x_{h-1}^j, x_h^j), \ \forall j \in HP_i\}$;
4 $C_\Delta = C_\Delta \cup C_{\Delta,h}, \forall h \in \{1, \ldots, H\}$;
5 **if** $HP_i == \emptyset$ *and* $E(p_t^i, \mathcal{P}_i) > 0$ **then**
6 $C_\Delta = \{(x_t^i, x')$ from $(p_t^i, p') \in \Delta_{\mathcal{P}_i} | E(p', \mathcal{P}_i) \geq E(p_t^i, \mathcal{P}_i)\}$;
7 **return** C_Δ

In Alg. 2 (called from line 14 of Alg. 1), the conflict set, denoted by $C_\Delta \subseteq \Delta$, is accounted for in order to guarantee a safe update in Alg. 3. The conflict set, C_Δ, enforces that transitions which may result in conflict, defined by the mapping $C_\mathcal{G}$ (see

Assumption 1), are not traversed (line 3). If any conflicts at hop h, $C_{\Delta,h}$, are found, they are added to the set of conflict transitions, i.e., $C_{\Delta,h} \subseteq C_\Delta$, where their associated hop h is preserved (line 4). Lines 5–6 ensure the energy of the highest priority agent is monotonically decreasing by making transitions to higher energy states infeasible for the first hop only. This is required in order to guarantee progress toward TWTL formulae satisfaction discussed in Section 4.2-Theorem 1

Algorithm 3: Receding Horizon Plan for Agent i

Input: $\mathcal{P}_i$, p_t^i - Product automaton and current state
Input: C_Δ - Set of conflict transitions
Output: $\mathbf{p}_i$, U^i_{flag}, D_{flag} - Conflict-free path; Update flag; Deadlock flag

1 **Note:** $C_{\Delta,h} \subseteq C_\Delta$ is the conflict set of transitions at hop h;
2 **for** $h = 1 : H$ **do**
3 Generate $S_{\mathcal{P}_i,h}$, $\Delta_{\mathcal{P}_i,h}$, reachable states and transitions;
4 $S_{\mathcal{P}_i,h} = S_{\mathcal{P}_i,h} \setminus \{(x', s) \in S_{\mathcal{P}_i,h} | \ (x, x') \in C_{\Delta,h}\}$;
5 $\Delta_{\mathcal{P}_i,h} = \Delta_{\mathcal{P}_i,h} \setminus \{\big((x, s), (x', s')\big) \in \Delta_{\mathcal{P}_i,h} | \ (x, x') \in C_{\Delta,h}\}$;
6 **if** $\Delta_{\mathcal{P}_i,h} = \emptyset$ **then**
7 **if** $h == 1$ **then**
8 $U^i_{flag} = false$ and $D_{flag} = true$;
9 **return** p_t^i, U^i_{flag}, D_{flag}
10 **else**
11 $h = h - 1$;
12 exit the for loop;
13 **for** *each state* $p \in S_{\mathcal{P}_i,h}$ **do**
14 **if** $p \in F_{\mathcal{P}_i}$ **then**
15 exit both for loops;
16 $H_{new} = h$;
17 $target\_state = \underset{p \in S_{\mathcal{P}_i,H_{new}}}{\arg\min} E(p, \mathcal{P}_i)$;
18 **if** $H_{new} == 1$ **then**
19 $\mathbf{p}_i = p_t^i, target\_state$;
20 **else**
21 $\mathbf{p}_i = DP\big(p_t^i, target\_state, \{S_{\mathcal{P}_i,1}, \ldots, S_{\mathcal{P}_i,H_{new}}\}, \{\Delta_{\mathcal{P}_i,1}, \ldots, \Delta_{\mathcal{P}_i,h}\}\big)$;
22 $U^i_{flag} = true$ and $D_{flag} = false$;
23 **return** $\mathbf{p}_i$, U^i_{flag}, D_{flag}

Alg. 3 essentially generates the set of all feasible (conflict-free) states of $\mathcal{P}_i$ that can be reached in H-hops. First, the product automaton state with minimum energy (*target_state*) is chosen, then a minimum cost path from the current node p_t^i to the *target_state* is generated. In further detail, the sets $S_{\mathcal{P}_i,h} \subseteq S_{\mathcal{P}_i}$ and $\Delta_{\mathcal{P}_i,h} \subseteq \Delta_{\mathcal{P}_i}$ with the associated hop h are first generated (line 3). Next, any conflicting states (p with x' where $(x, x') \in C_{\Delta,t}$) and conflicting transitions ((p, p') with $(x, x') \in C_{\Delta,t}$) are removed from the sets $S_{\mathcal{P}_i,h}$ and $\Delta_{\mathcal{P}_i,h}$ (lines 4–5).

Instead of generating a sub-graph of all states that can be reached from p_t^i in H-hops, i.e., $\mathcal{N}^H(p_t^i)$, the sets $S_{\mathcal{P}_i,h} \subseteq S_{\mathcal{P}_i}$ and $\Delta_{\mathcal{P}_i,h} \subseteq \Delta_{\mathcal{P}_i}$ are generated; where $S_{\mathcal{P}_i,h}$ is the set of states that can be reached at hop h, and $\Delta_{\mathcal{P}_i,h}$ is the set of transitions that can be taken at hop h. These are used for two reasons: (1) this allows for proper handling of collision-avoidance at each hop h without excess pruning that could lead to infeasibilities, and (2) this facilitates the Dynamic Programming (DP) algorithm to generate a "min-cost" path ([42], Chapter 11). Alternatively, $S_{\mathcal{P}_i,h}$ and $\Delta_{\mathcal{P}_i,h}$ can be thought of as a stage in the sequence.

Line 6 checks if there are no feasible transitions at hop h. If this occurs at the first hop, $h = 1$, then the agent is in deadlock and Alg. 3 is broken out of early (lines 7–9). If there are no feasible transitions at a later hop, $h > 1$, then this means a full H-hop path will not be obtained, however the agent is not in deadlock (lines 10–12). Lines 13–15 check if any of the states in $S_{\mathcal{P}_i,h} \subseteq S_{\mathcal{P}}$ are accepting states. If that is the case, then the loops are broken out of early since any accepting state $p \in F_{\mathcal{P}_i}$ has $E(p, \mathcal{P}_i) = 0$; this is the best state that can be reached in the least number of hops. Next, H_{new} is assigned (line 16) which accounts for paths shorter than H hops due to exiting the previous loop (lines 2–15) early for the reasons previously stated. The minimum energy state that can be reached after H_{new} hops is chosen as the *target_state* (line 17). Accordingly, the goal becomes to find the minimum cost path over $\mathcal{P}_i$ reaching the lowest energy state in H_{new} hops.

Note that if the path is only 1 hop, it is simply from the current state to the neighboring *target_state* (lines 18–19). Otherwise, in line 21, the dynamic programming (DP) algorithm back traverses from the *target_state* to the current state p_t^i using states of $S_{\mathcal{P}_i,h} \subseteq S_{\mathcal{P}_i}$ and transitions of $\Delta_{\mathcal{P}_i,h} \subseteq \Delta_{\mathcal{P}_i}$ in order to only generate feasible paths over $\mathcal{T}_i$. The minimum cost path is generated using a standard DP algorithm (see [42], Chapter 11). After the minimum cost path over $\mathcal{P}_i$ from p_t^i to *target_state* is obtained, the update flag is set to *true*, deadlock flag to *false* (line 22), then both flags and the conflict-free path are returned to Alg. 1 (see Fig. 3).

Algorithm 4: Deadlock Resolution

Input: $\overline{\Pi}$; $C_\mathcal{G}$; $\mathcal{G}$ - Priority ordering, conflict mapping, environment graph
Input: $p_t^k \ \forall k \in \overline{N}_i$ and $p_{t+1}^k \ \forall k \in \overline{HP}_i$ - Current and next states
Output: P_{t+1} - Next states within $\overline{N}_i$ neighborhood

1 **Note:** X_t, X_{t+1}, and P_{t+1} are in order of highest to lowest priority. Recall $p = (x, s)$.
2 **Initialization:** $X_t = \{x_t^{\overline{\Pi}(1)}, \ldots, x_t^{\overline{\Pi}(|\overline{N}_i|)}\}$;
$X_{t+1} = \{x_{t+1}^{\overline{\Pi}(1)}, \ldots, x_{t+1}^{\overline{\Pi}(|\overline{HP}_i|)}\}$;
3 **Initialization:** $P_{t+1} = \emptyset$; $P_{flag} = false$; $x_D = x_t^i$;
4 **while** $\{x_D\} \neq \emptyset$ **do**
5 **for** $k = 1 : |X_{t+1}|$ **do**
6 **if** $X_{t+1}(k) == x_D$ **then**
7 **if** *ID of* $X_{t+1}(k)$ *belongs to* $\overline{\Pi}(1)$ **then**
8 $P_{flag} = true$;
9 exit while loop;
10 **else**
11 Get $\overline{\Pi}(\cdot)$ corresponding to k;
12 $P_{t+1} = P_{t+1} \cup p_t^{\overline{\Pi}(\cdot)}$;
13 $X_{t+1} = X_{t+1} \setminus \{x_D\}$;
14 $x_D = X_t(\overline{\Pi}(\cdot))$;
15 jump to line 4;
16 $\{x_D\} = \emptyset$
17 **if** $P_{flag} == true$ **then**
18 $P_{t+1} = \{p_{t+1}^{\overline{\Pi}(1)}, p_t^{\overline{\Pi}(2)}, \ldots, p_t^{\overline{\Pi}(|\overline{N}_i|)}\}$;
19 $\mathcal{G}' = \Big(X, \Delta \setminus C_\mathcal{G}(x_t^{\overline{\Pi}(1)}, x_D), w\Big)$;
20 $x_T = \underset{x' \in X \setminus X_t}{\arg\min} d(x_D, x')$;
21 $\mathbf{x} = Dijkstra\big(x_D, x_T, \mathcal{G}'\big)$;
22 **for** $j = 1 : |\mathbf{x}|-1$ **do**
23 **if** $\mathbf{x}(j) \in X_t$ **then**
24 Denote m as the agent ID occupying $\mathbf{x}(j)$;
25 Broadcast $\mathbf{x}(j+1)$ to agent m, and obtain state $p_{t+1}^m = (\mathbf{x}(j+1), s) \in S_{\mathcal{P}_m}$ reached from p_t^m in 1-hop;
26 Find index l in $\overline{\Pi}$ corresponding to agent m;
27 $P_{t+1}(l) = p_{t+1}^m$;
28 **return** P_{t+1}

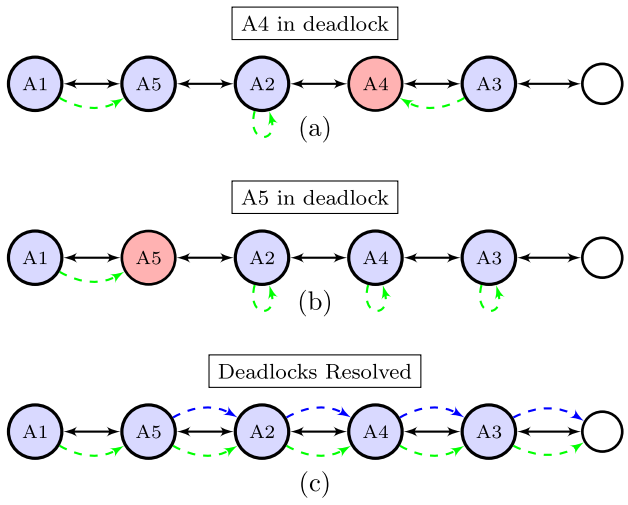

Fig. 3. Illustration of deadlock resolution using Alg. 4 for priority ordering $\Pi = \{A1, A2, A3, A4, A5\}$. In this example, $C_{\mathcal{G}}(x, x') = \Delta^{in}_{x'} \cup \{(x', x)\}$ for every $(x, x') \in \Delta$. Note, Alg. 4 is called twice due to the given priority ordering. Black arrows indicate the environment transitions, Δ, an agent's desired next transition is shown by a green dashed arrow, and in (c) the path found in Alg. 4 line 21 is shown by the blue dashed arrows. This is only a portion of an environment for illustration purposes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Lastly, Alg. 4 (called from line 19 of Alg. 1) is required to resolve any deadlocks that might occur. Alg. 4 generates the set of next states, denoted P_{t+1}, for the set of agents in $\bar{N}_i$ required for deadlock resolution. Agent i stays stationary (i.e., $p^i_{t+1} = p^i_t$), if a higher priority agent (let us call it agent j) desires to occupy agent i's current state in $\mathcal{G}$ (i.e., $x^j_{t+1} = x^i_t$) then agent j will also be stationary. Now, if another agent in $\bar{N}_i \subseteq \bar{N}_i$ (let us call it agent k) desires to occupy agent j's current state in $\mathcal{G}$ (i.e., $x^k_{t+1} = x^j_t$) then agent k will also remain stationary. This "cascading" goes on until either (1) an agent is not required to be displaced, meaning the remaining agents in $\overline{HP}_i$ will remain with their initially desired next states p_{t+1}, or (2) the highest priority agent desires to displace a deadlock state, denoted x_D. If the latter occurs, then all agents in $\bar{N}_i$, except the highest priority agent, are initially set to remain stationary. Then, the agent being displaced by the highest priority agent finds a shortest path over $\mathcal{G}'$ (where $\mathcal{G}'$ excludes transitions made infeasible by the highest priority agent's desired next transition) from its current state to the closest unoccupied state in $\mathcal{G}'$ using *Dijkstra's* shortest path algorithm. Note the highest priority agent is always yielded to in order to guarantee progress toward TWTL formulae satisfaction discussed in Section 4.2-Theorem 1.

In further detail, Alg. 4 first generates sets X_t, X_{t+1} in order of highest to lowest priority from the given inputs (line 2). The set of desired next states P_{t+1} is initially empty, the path flag, P_{flag}, is initially set to *false*, and the deadlock state, x_D, is the current state of agent i (line 3). Line 4 indicates that the loop executes until there is no longer a deadlock state (or is exited early). The inner loop (lines 5–15) checks if a higher priority agent desires to occupy the assigned deadlock state x_D. If it is found that the agent in $\overline{HP}_i$ which desires to occupy x_D at its next state x_{t+1} is the highest priority agent (line 7), then P_{flag} is set to *true* and the while loop is exited (lines 8–9). If the agent which desires to occupy x_D is not the highest priority agent, then this agent will be stationary (i.e., $p_{t+1} = p_t$) and added to the set of desired next states P_{t+1} (lines 11–12). The current deadlock state is then removed from the set of next states X_{t+1} which are searched through (line 13); the current state x_t (of the agent which desires to occupy x_D) is assigned as the updated deadlock state (line 14), and this process is then repeated (line 15). If none of the desired next states $x_{t+1} \in X_{t+1}$ correspond to the deadlock state, x_D, then there is no other deadlock state and the while loop is exited (line 16).

Now, if the P_{flag} is set to *true*, then all agents in $\bar{N}_i$ (except the highest priority agent) are assigned to stay at their current state (line 18). The updated environment graph $\mathcal{G}'$ excludes transitions made infeasible due to the highest priority agent's desired next transition (line 19) (note that here, $x_D \in X$ is the desired next state for the highest priority agent). The closest unoccupied state is found and assigned to be the target state x_T (line 20). A shortest path $\mathbf{x}$, from x_D to x_T, is then computed over $\mathcal{G}'$ using *Dijkstra's* algorithm (line 21). Then, a message passing protocol (e.g., [43]) is followed for safe next state reassignment. Specifically, if an agent is found to be currently occupying a state on the path $\mathbf{x}$, including the agent occupying x_D, (line 23), this agent, denoted m, is assigned the next state along the path. This next state is then broadcast to agent m, where agent m obtains the state $p \in S_{\mathcal{P}_m}$ that can be reached in 1-hop from its current state p^m_t (i.e., $p^m_{t+1} = (\mathbf{x}(j+1), s) \in S_{\mathcal{P}_m} |\ \mathbf{x}(j+1) \in \mathcal{N}^1_m$) (line 25). Then the desired next state for agent m is updated in the set P_{t+1} (lines 26–27), and this procedure is repeated (lines 22–27) until the entire path (of possible occupied states) has been checked. Note that the last state $\mathbf{x}(|\mathbf{x}|)$ is not checked since this state is not initially occupied.

4.2. Performance guarantees

In this section, we will show that an agent i always finds a safe path over the environment graph $\mathcal{G}$ by executing the proposed algorithm. Moreover, we show that the resulting path satisfies either the original TWTL formula or a finite relaxation of it, $\phi_i(\boldsymbol{\tau}^i)$.

Lemma 1 (*Deadlock Resolution*). *Let Assumption 1 hold and let n agents follow Alg. 1 in an environment $\mathcal{G} = (X, \Delta, w)$ such that for every $(x, x') \in \Delta$, each (strongly) connected component in*

$$\mathcal{G}' = (X, \Delta \setminus C_{\mathcal{G}}(x, x'), w)$$

contains at least n states and any shortest path $\mathbf{x} = x_0 x_1 \ldots$ on $\mathcal{G}'$ is conflict-free, i.e.,

$$(x_i x_{i+1}) \notin C_{\mathcal{G}}(x_j, x_{j+1}),\ \forall x_i, x_j \in \mathbf{x}. \tag{8}$$

Then, at any time $t \geq 0$ each agent i moves to its next state $p^i_{t+1} = (x^i_{t+1}, s^i_{t+1})$ such that

$$(x^i_t, x^i_{t+1}) \notin C_{\mathcal{G}}(x^j_t, x^j_{t+1}),\ \forall i \neq j \in \{1, \ldots, n\}. \tag{9}$$

Proof. See Appendix A. □

Theorem 1 (*Finite Relaxation*). *Let Assumption 1 hold and let n agents follow Alg. 1 in an environment $\mathcal{G} = (X, \Delta, w)$ such that for every $(x, x') \in \Delta$, each (strongly) connected component in $\mathcal{G}' = (X, \Delta \setminus C_{\mathcal{G}}(x, x'), w)$ contains at least n states and any shortest path $\mathbf{x} = x_0 x_1 \ldots$ on $\mathcal{G}'$ is conflict-free, i.e., $(x_i x_{i+1}) \notin C_{\mathcal{G}}(x_j, x_{j+1}),\ \forall x_i, x_j \in \mathbf{x}$. Then, each agent i satisfies its TWTL specification $\phi_i(\boldsymbol{\tau}^i)$ such that $|\boldsymbol{\tau}^i| < \infty$.*

Proof. See Appendix A. □

5. Experimental results and evaluation

The code base for trajectory generation is derived from the PyTWTL[1] package which handles the construction of $\mathcal{A}_\infty$ corresponding to a given TWTL specification, ϕ, and the creation of the

[1] hyness.bu.edu/twtl.

product automaton $\mathcal{P}$ given a particular transition system $\mathcal{T}$. The algorithms presented in this paper which are integrated into the PyTWTL framework as well as a video of the experiment can be found at https://github.com/pete9936/pyTWTL_ObsAvoid.

Our proposed algorithm is verified experimentally on a team of five Crazyflies 2.0. This is conducted in a 3 m × 3 m × 1.5 m motion-capture space, corresponding to the environment shown in Fig. 4, using a VICON camera system with 8 cameras. We use the Crazyswarm[2] package to perform the low-level controls algorithms, communication, and interface of the VICON system with the Crazyflies. Details of which can be found in [44]. The experiments were carried out on a desktop with 4 cores running Ubuntu 16.04, 4.0 GHz CPU, and 32 GB of RAM.

5.1. Scenario 1

This scenario considers five quadrotors in the environment shown in Fig. 4 with a narrow corridor. The scenario shows both the collision avoidance and deadlock resolution protocols. Initial agent priority ordering corresponds to $\Pi = \{A1, A3, A4, A5, A2\}$, where Agent 1 has the highest priority (and maintains highest priority until its task completion) and therefore completes its task, ϕ_1, without yielding to other agents. Deadlock occurs twice in this scenario and is resolved using Alg. 4.

This scenario illustrates how our algorithm is useful in dealing with complex specifications. For a pick-up and delivery task, it may be the case that multiple agents desire to deliver to the same location, however, if the desired location is currently occupied then an alternative near-by location can be used instead. Many path planning algorithms which consider collision avoidance (e.g., [45]) cannot account for such a scenario. While this is just one example, the ability to account for alternative drop-off locations using local information (and enforcing safety) is a notable property of our algorithm. We depict this scenario on the environment shown in Fig. 4, where the following temporally relaxed TWTL formulae are considered:

$$\phi_1 = [\mathrm{H}^2 A]^{[0,3+\tau_1^1]} \cdot [\mathrm{H}^1 D]^{[0,5+\tau_2^1]} \cdot [\mathrm{H}^0 B1]^{[0,5+\tau_3^1]}$$
$$\phi_2 = [\mathrm{H}^3 (B \vee C)]^{[0,6+\tau_1^2]} \cdot [\mathrm{H}^2 E]^{[0,7+\tau_2^2]} \cdot [\mathrm{H}^0 B2]^{[0,3+\tau_3^2]}$$
$$\phi_3 = [\mathrm{H}^2 (B \vee C)]^{[0,6+\tau_1^3]} \cdot [\mathrm{H}^2 E]^{[0,7+\tau_2^3]} \cdot [\mathrm{H}^0 B3]^{[0,3+\tau_3^3]}$$
$$\phi_4 = [\mathrm{H}^2 (B \vee C)]^{[0,6+\tau_1^4]} \cdot [\mathrm{H}^2 E]^{[0,7+\tau_2^4]} \cdot [\mathrm{H}^0 B4]^{[0,3+\tau_3^4]}$$
$$\phi_5 = [\mathrm{H}^2 A]^{[0,5+\tau_1^5]} \cdot [\mathrm{H}^1 F]^{[0,5+\tau_2^5]} \cdot [\mathrm{H}^0 B5]^{[0,5+\tau_3^5]}$$

All five TWTL specifications are examples of servicing in sequence which can be thought of as pick-up and delivery tasks while starting and ending at the agent's base. In practice, these task specifications can be made far more complex due to the richness of the TWTL language. It should be noted that not all agents will remain in their designated bases since agents which have reached an accepting state (i.e., completing the task) will yield to other agents in the environment which have not yet reached an accepting state (higher priority agents). This is made possible due to the self-loop of the accepting states of the dFSA, $F_{\mathcal{A}_\infty}$.

For the 4 × 7 × 1 environment shown in Fig. 4, each respective agent has a transition system $\mathcal{T}_i$ of (19; 105) states[3] and transitions, and similarly sized product automaton $\mathcal{P}_i$ of ≈ (80; 440) states and transitions. The algorithm's offline initialization (generating $\mathcal{A}_{\infty,1\ldots5}$, $\mathcal{P}_{1\ldots5}$, $E_{1\ldots5}$) takes 0.31 s. Generating the collision-free paths (the online portion of our algorithm) with a horizon length $H = 2$ takes 0.07 s. However, the metric of greater concern is the average time for an individual agent's receding horizon update which is ≈ 0.7 ms, which is fast enough for real-time execution.

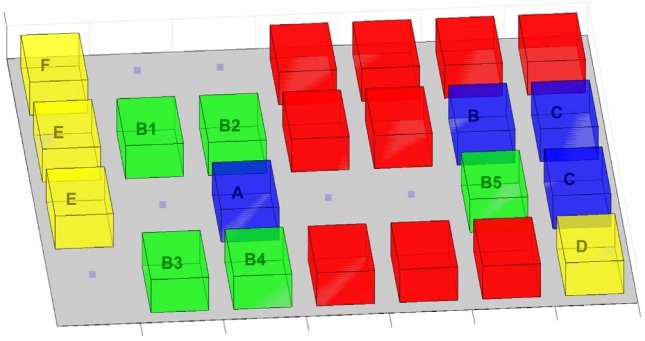

Fig. 4. *Scenario-1*: The 3D discretized environment shared by 5 agents. Initial positions given by the green nodes, obstacles are shown in red, three pick-up regions (A, B, C) are shown in blue, and three drop-off regions (D, E, F) are shown in yellow. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
Temporal relaxation (τ) of paths.

	Nominal paths					Safe paths				
	x_1	x_2	x_3	x_4	x_5	$x_1{}^S$	$x_2{}^S$	$x_3{}^S$	$x_4{}^S$	$x_5{}^S$
τ_1	0	**0**	**0**	**−1**	**0**	0	**+1**	**+4**	**+4**	**+7**
τ_2	0	**0**	0	**0**	**−1**	0	**+6**	0	**+1**	**0**
τ_3	0	−1	−2	−1	0	0	−1	−2	−1	0

Each agent computes its nominal path over its product automaton $\mathcal{P}_i$ (from the agent's initial state p_0^i to an accepting state in $F_{\mathcal{P}_i}$ via a Dijkstra's weighted shortest path algorithm) irrespective of other agents' paths. These nominal paths give a baseline metric for temporal relaxation (Table 1). The temporal relaxation for both the nominal and collision-free policies for each agent are given in Table 1. Recall that negative relaxation of the TWTL formulae implies that the formulae are satisfied within a stricter time window than was originally allotted.

For the given scenario, some snapshots of the experiment at notable instances are shown in Fig. 5, and priority ordering at a given time step is evident by looking at Fig. 6. Notice that while agents are yielding to those of higher priority, their energies do not decrease (seen in Fig. 6).

5.2. Scalability analysis

In this section, we run some simulations to demonstrate the scalability of the proposed algorithm. First, we explore the impact of horizon length (H) on the average time for an individual agent's path update. We consider Scenario 1 in this study, and the results are reported in Fig. 7. In this figure, the average iteration time is an average of all agents and all iterations for the respective run. Note that by using our dynamic programming based receding horizon algorithm, the execution time growth is approximately linear with respect to H.

The proposed algorithm runs over a product automaton which captures both the transition system and the automaton of an agent. The size of the product automaton has a direct effect on the computation time of the algorithm. So, we present some results to illustrate the influence of the product automaton size. Fig. 8 depicts various points corresponding to different sizes of product automata that are obtained by different environments and varying lengths of TWTL specifications. For example, the first

[2] https://github.com/USC-ACTLab/crazyswarm.

[3] The transition system contains only the feasible states, not the states with obstacles.

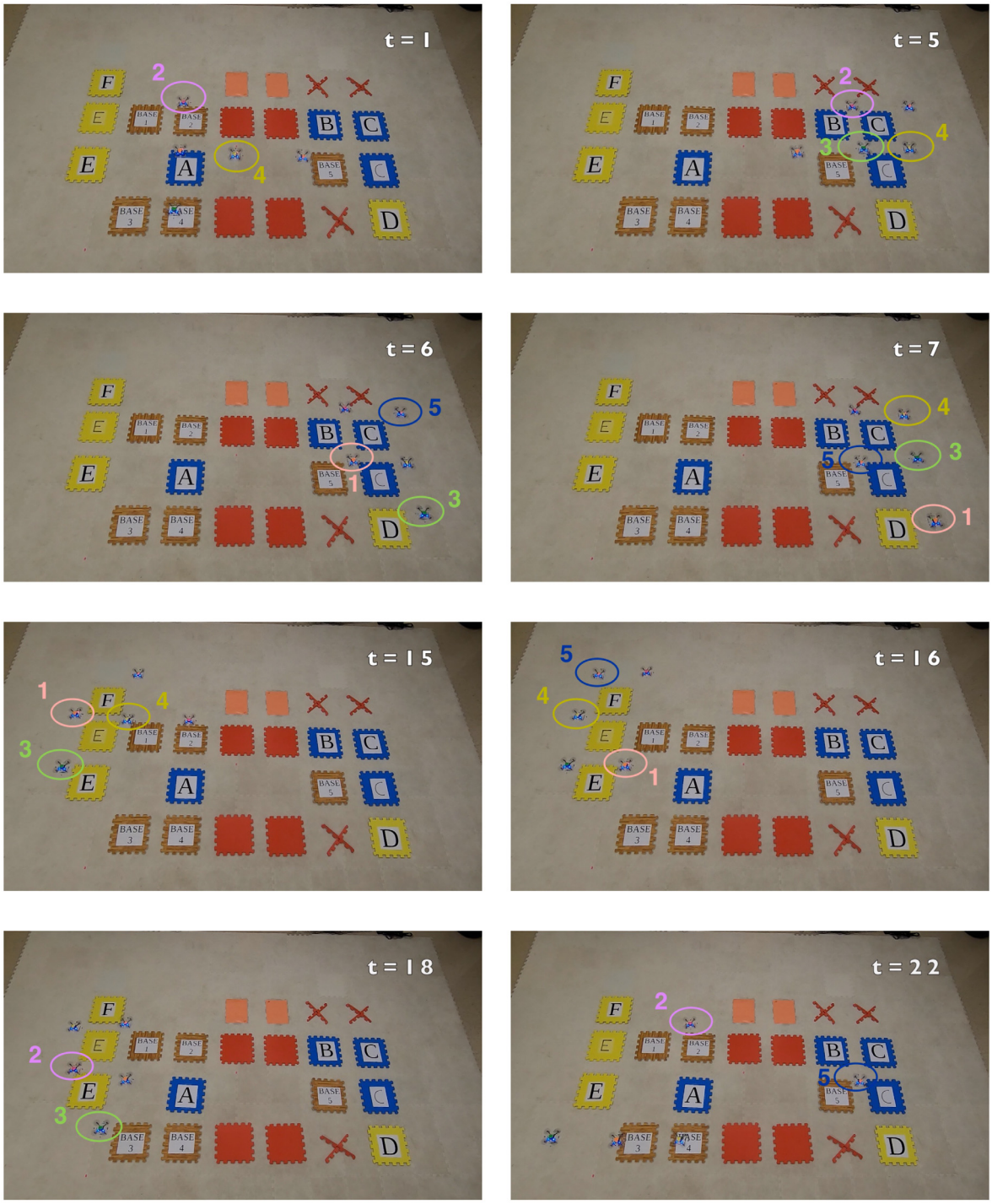

Fig. 5. Scenario 1: At $t = 1$ agent 2 yields to agent 4 due to priority ordering (see Fig. 4); At $t = 6$ agent 5 triggers deadlock resolution, Alg. 4; At $t = 7$ agent 4 triggers deadlock resolution; At $t = 16$ agent 1 has already finished its task and has since yielded to other agents still in progress (this is why agent 1 is not still at its base); At $t = 18$ agent 2 finally begins its delivery at E and agent 3 returns to base; At $t = 22$ agents 2 and 5 return to their bases and all agents have completed their tasks.

three points in Fig. 8 represent missions executed in Scenario 1 (Fig. 4) under varying lengths of TWTL specifications. The first point in Fig. 8 corresponds to the case when the agents try to achieve the original TWTL specifications of Scenario 1. In that case, the product automaton has approximately $\mathcal{P}_i \approx (80; 440)$ states and transitions for each agent. The second and third points correspond to the TWTL formulae in Scenario 1 concatenated by itself once and twice, respectively. By doing that, the lengths of the specifications increase and the formulae enforce the agent to repetitively achieve its pick-up and delivery task (e.g., complete the pick-up and delivery task, and immediately after that complete the same task). For points 2 and 3, the corresponding sizes of the product automata are $\mathcal{P}_i \approx (140; 760)$ and $\mathcal{P}_i \approx (200; 1100)$, respectively. The fourth point corresponds to a scenario over a $6 \times 6 \times 3$ environment with TWTL specifications similar to Scenario 1 ($\mathcal{P}_i \approx (395; 6060)$). The fifth point runs

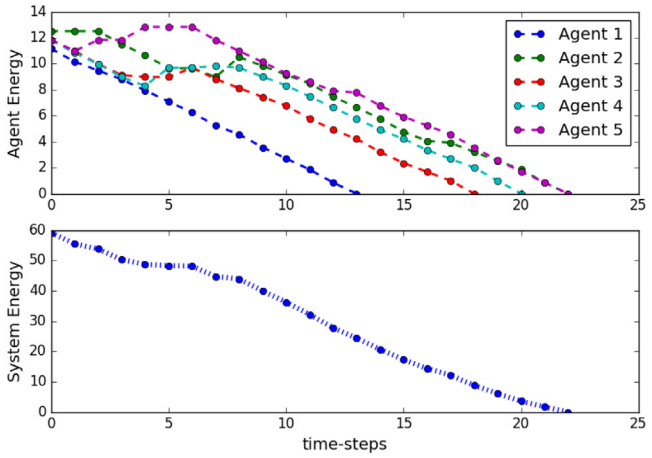

Fig. 6. *Scenario-1*: The agent energy values are shown as well as the collective energy of the system. These energy values correspond to each state along the path which leads to satisfaction of each agents' respective TWTL formula.

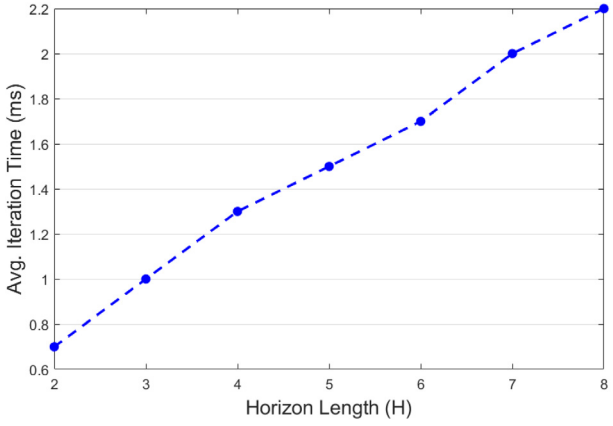

Fig. 7. Impact of horizon length on execution time.

the scenario without returning to the base[4] on a $6 \times 12 \times 4$ environment ($\mathcal{P}_i \approx (750; 12700)$), while the sixth point presents the full scenario on the $6 \times 12 \times 4$ environment where each agent has $\mathcal{P}_i \approx (1000; 16900)$ states and transitions. Overall, Fig. 8 shows that an agent's path update scales approximately linearly with the size of $\mathcal{P}$ using our algorithm.

We also investigate how the number of agents affect the online execution time of the proposed algorithm. To this end, we gradually increase the number of agents in the simulations (up to 10 agents) and assign them with TWTL specifications of similar structure to those in Scenario 1. Fig. 9 shows that the online execution time (i.e., the average iteration time) increases approximately linearly with the number of agents. Note that the average iteration time is an average of all agents and all iterations for the respective run. Also, it is observed that the average time for an individual agent's path update grows less than linearly with respect to the number of agents in the environment since this depends only on the number of agents in neighborhood $\bar{N}_i$ defined as in (1).

[4] This implies a shorter TWTL formula than the case in point 6 where the task structure is pick-up, delivery, go to base.

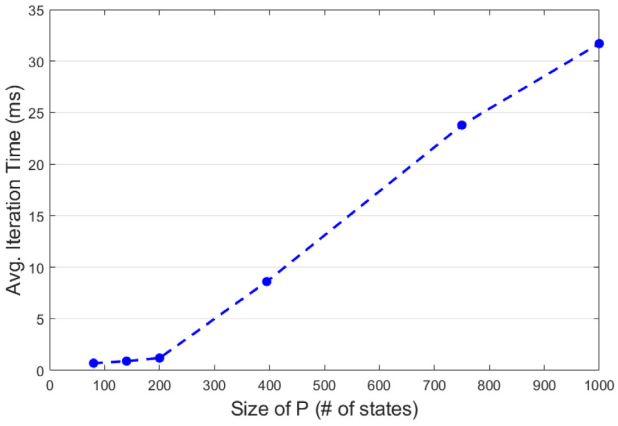

Fig. 8. Impact of product automaton, $\mathcal{P}_i$, size on execution time.

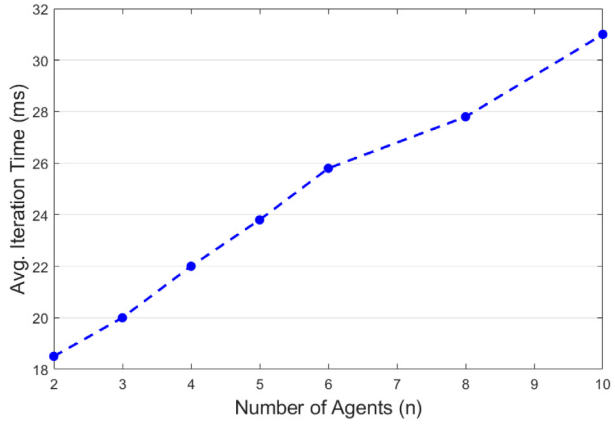

Fig. 9. Impact of the number of agents on execution time.

5.3. Benchmark analysis

In the benchmark analysis, the proposed distributed algorithm is compared with some of the widely used approaches in the literature, i.e., ORCA [13], a centralized solution based on the mixed-integer linear program (MILP) encoding of temporal logic specifications [46,47], and an automata-theoretic centralized solution whose challenges are described in Section 3.3. We also compare our priority ordering method with that of a fixed-random-priority ordering as well as a "longest first" priority ordering which gives agents with higher energy as defined by Eq. (6) a higher priority.

The benchmark analysis is implemented for $n = 3$ agents, and a horizon length of $H = 2$. Local specifications for three agents are given in different $3 \times 6 \times 1$ environments (shown in Fig. 10) which include randomly placed bases, goal regions, and static obstacles. The TWTL specifications run over these environments are given as:

$$\phi_1 = [\mathrm{H}^1 P_1]^{[0,5+\tau_1^1]} \cdot \left([\mathrm{H}^3 D_1] \vee [\mathrm{H}^3 D_2] \vee [\mathrm{H}^3 D_3]\right)^{[0,7+\tau_2^1]}$$

$$\phi_2 = [\mathrm{H}^1 P_1]^{[0,5+\tau_1^2]} \cdot \left([\mathrm{H}^3 D_1] \vee [\mathrm{H}^3 D_2] \vee [\mathrm{H}^3 D_3]\right)^{[0,7+\tau_2^2]}$$

$$\phi_3 = [\mathrm{H}^1 P_2]^{[0,5+\tau_1^3]} \cdot \left([\mathrm{H}^3 D_1] \vee [\mathrm{H}^3 D_2] \vee [\mathrm{H}^3 D_3]\right)^{[0,7+\tau_2^3]}$$

Since TWTL does not have a MILP encoding and the authors of [28] state that TWTL and MTL have similar expressivity, we formulate a MILP with MTL specifications. To this end, we write the

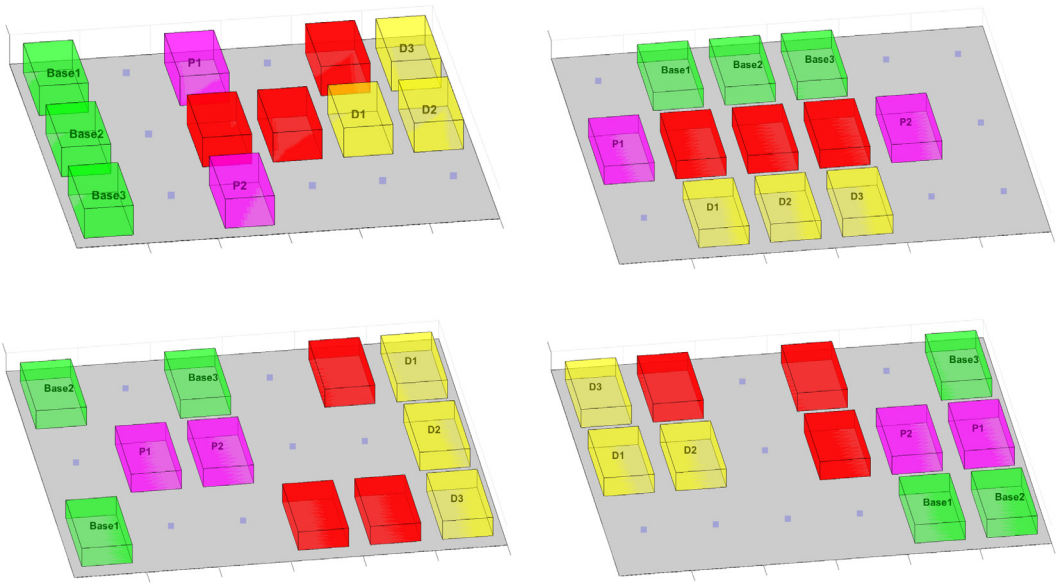

Fig. 10. Environments 1–4 for benchmark analysis: Env. 1 (top left), Env. 2 (top right), Env. 3 (bottom left), Env. 4 (bottom right).

MTL specifications equivalent to the above TWTL specifications.

$$\phi_1 = \Diamond_{[0,4]}\Big(\Box_{[0,1]}P_1 \wedge \Diamond_{[1,5]}\big(\Box_{[0,3]}D_1 \vee \Box_{[0,3]}D_2 \vee \Box_{[0,3]}D_3\big)\Big)$$

$$\phi_2 = \Diamond_{[0,4]}\Big(\Box_{[0,1]}P_2 \wedge \Diamond_{[1,5]}\big(\Box_{[0,3]}D_1 \vee \Box_{[0,3]}D_2 \vee \Box_{[0,3]}D_3\big)\Big)$$

$$\phi_3 = \Diamond_{[0,4]}\Big(\Box_{[0,1]}P_2 \wedge \Diamond_{[1,5]}\big(\Box_{[0,3]}D_1 \vee \Box_{[0,3]}D_2 \vee \Box_{[0,3]}D_3\big)\Big)$$

where $\Box_{[a,b]}$ and $\Diamond_{[a,b]}$ denotes *always* and *eventually* operators of MTL, respectively, with the time window $[a, b]$. Note that the equivalent MTL specifications include many nested operators that result in more complex representations than TWTL specifications [28]. The optimization problem with MTL specifications is solved to find satisfying trajectories in a centralized manner. Safety constraints are applied such that $x_t^i - x_t^j > r_{min},\ \forall t,\ \forall i, j$ with $i \neq j$. The agents are represented with single integrator dynamics which are allowed to move along either cardinal or diagonal directions by constraining the velocities as $v_{x,t}^i, v_{y,t}^i \in \{-0.5, 0, 0.5\}/\Delta t$ m/s, $\forall i, t$ where 0.5 m is the cardinal transition length in the environment graph. The time step Δt is selected in accordance with the TWTL application. We refer the reader to [46] for further details relating to the symbols and syntax of MTL and its encoding into MILP .

Since logical expressions such as *disjunction* cannot be defined in ORCA, and there are three delivery regions which agents are free to choose between, a total of $3^3 = 27$ runs for each of the four environments in Fig. 10 are implemented to examine all possible delivery options. In each run, an order of goal regions to be visited is defined for each agent with the required number of multiplicity of the regions that agents need to hold in. Since the holding operation of TWTL is not preemptive, whenever an agent's holding is interrupted by another one (due to collision avoidance effort), we reset the holding requirement for that agent. We observe that ORCA may fail when multiple agents are required to hold in the same delivery region which makes ORCA prone to oscillations and deadlocks, especially when agents have common regions of interest. Note that in the only $3! = 6$ of the runs per environment, the agents are provided unique delivery options. In the implementation of ORCA, again $0.5/\Delta t$ m/s velocity limit is applied where agents can see other agents and obstacles within a horizon of 2 hops as in the TWTL application.

The automata-theoretic centralized solution to the safe multi-agent path planning problem with TWTL specifications requires the construction of a global product automaton that includes the transition system and automaton states of all agents. In the scenarios above with three agents, the transition system of each agent has 15 states,[5] and the resultant product automaton for a single agent with the TWTL specification presented above has 39 states for the Environments 1 and 2, and 51 states for the Environments 3 and 4. Accordingly, the centralized solution requires the generation of a product automaton that has $\sim 39^3$ or $\sim 51^3$ states. Such a centralized approach finds the optimal trajectories, but it is not scalable, which is one of the main reasons why we propose a distributed solution in this paper.

5.3.1. Time complexity

The mission completion times as well as total run-times (including the offline computation for TWTL) are shown for different methods in Table 2. Note that the average completion time for ORCA is the average of the trials which were able to finish the servicing and holding requirements of all agents regardless of their finishing time. The number of successful ORCA trials (out of 27 runs per environment) were 12 (44.44%), 20 (74.07%), 14 (51.85%), and 6 (22.22%) for Environments 1, 2, 3, and 4, respectively. The main reason for such behavior is the lack of decision-making ability of the agents in terms of high-level planning such as waiting for others to finish their ongoing service tasks (holding) instead of interrupting them. Moreover, we observe that only a few of the trials over Environments 1 and 2 resulted in trajectories satisfying the TWTL specification within the desired time windows. None of the trials in Environments 3 and 4 satisfy the specification before the TWTL deadline. This alone however does not lead to infeasible trials since ORCA has no explicit time limit consideration. The agents simply try to reach their goals as soon as possible while avoiding collisions so we treat these results as the best ORCA can perform.

The results of the ORCA trials in terms of mission completion time together with TWTL (centralized and distributed) and MILP-MTL solutions are illustrated in Fig. 11. As it can be seen, the

[5] Each environment in Fig. 10 has 15 feasible states without any obstacles.

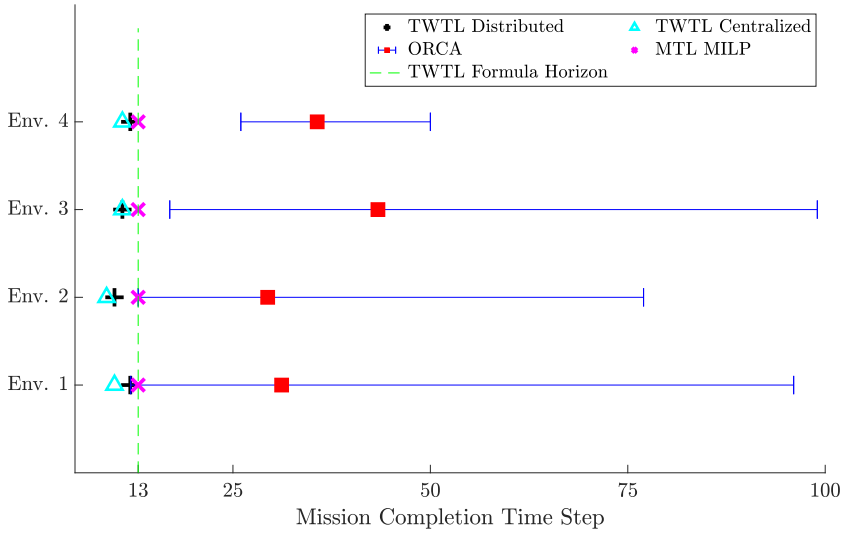

Fig. 11. The mission completion time for TWTL (both the proposed distributed approach and the centralized version), MILP with MTL constraints, and ORCA including min (fastest), max (slowest), and mean of the completion times. Note that the time horizons of the initial TWTL specifications are 13 time steps, and MILP-MTL does not optimize temporal relaxation.

ORCA method leads to a distribution of mission completion times and does not result in completing the mission in desired time windows in most of the cases although there exists a feasible solution. MILP-MTL solves for optimal trajectories satisfying equivalent temporal logic (MTL) specifications in a centralized manner by formulating an optimization problem with approximately 2800 variables, where the temporal logic is encoded as a constraint. The MILP approach does not accommodate temporal relaxation. Accordingly, when the time bounds of the mission are tight and the agents have to violate the specifications, the MILP approach cannot provide trajectories servicing the desired locations, whereas our TWTL method allows for temporal relaxations in case of infeasible mission specifications. On the other hand, the automata-theoretic centralized approach with TWTL specifications requires constructing a global product automaton for each environment. Even after pruning the infeasible states (multiple agents occupying the same location) and conflicting transitions (Definition 3), the sizes of the global product automata are $\mathcal{P}_{full}^{Env.1} \approx (45870; 2819728)$, $\mathcal{P}_{full}^{Env.2} \approx (45870; 2065584)$, $\mathcal{P}_{full}^{Env.3} \approx (106050; 9202735)$, and $\mathcal{P}_{full}^{Env.4} \approx (106050; 8210137)$, respectively. Such an approach provides the fastest mission completions (the optimal solution) but requires an enormous amount of computation time as shown in Table 2 and Fig. 11. Overall, these empirical results indicate that our proposed distributed approach can produce near-optimal solutions with significantly reduced computation times.

5.3.2. Priority orderings

Our priority ordering method which gives priority to agents with lower energy defined by (6) was run on the four environments shown in Fig. 10 as well as an additional 6 environments, for a total of 10 different $3 \times 6 \times 1$ environments. All but one environment produced successful trials. The unsuccessful trial occurred due to a violation of the environment property defined in Lemma 1, i.e., in an environment $\mathcal{G} = (X, \Delta, w)$ such that for every $(x, x') \in \Delta$, each (strongly) connected component in $\mathcal{G}' = (X, \Delta \setminus C_{\mathcal{G}}(x, x'), w)$ contains at least n states (where $n = 3$ here).

A fixed-random-priority ordering for the three agents was considered on environments 1–4. All combinations of priority orderings were considered in these cases yielding 6 different orderings for each environment giving a total of 24 trials. Once an agent completed its task, that agent is assigned lowest overall priority (same as our method) so the agent may avoid blocking the environment for other active agents. The change in completion time for the team of agents using fixed-random-priority ordering was not statistically significant. Furthermore, a fixed-random-priority ordering does not guarantee a Lyapunov-like convergence to ensure a finite temporal relaxation as our priority ordering method does (see Theorem 1).

A priority ordering based on giving higher priority to agents with higher energy as in Def. (6) was investigated but is unsuitable for TWTL specifications. This is due to the hold operators since agent priorities may oscillate when two or more agents desire to go to a common goal region. Performing this priority ordering method for the given specifications produced infeasible trials for all 10 environments. This priority ordering method may possibly work better when considering physical distance as the "Energy Function" as in [48], but in the case of temporal logic specifications the satisfaction of the specification should be

Table 2
The comparison of mission completion time step and total run-time. Columns represent the results of the proposed distributed strategy with TWTL specifications, the centralized automata-theoretic solution with TWTL specifications, ORCA method, and MILP solution under equivalent MTL constraints, respectively.

Scenario	Parameters	TWTL distributed	TWTL centralized	ORCA (average, fastest)	MTL MILP
Env. 1	Mission completion step	12	10	31.17, 12	13
	Total run-time (s)	0.16	5900	0.013, 0.012	43.53
Env. 2	Mission completion step	10	9	29.40, 13	13
	Total run-time (s)	0.15	5800	0.013, 0.012	14.64
Env. 3	Mission completion step	11	11	43.36, 17	13
	Total run-time (s)	0.18	62 000	0.014, 0.013	47.16
Env. 4	Mission completion step	12	11	35.67, 26	13
	Total run-time (s)	0.20	58 000	0.015, 0.013	18.07

taken into account which is why we have defined priority as in Definition 7.

6. Conclusions and future work

We presented an algorithm for generating collision-free paths for a multi-agent system which satisfy individual tasks encoded as TWTL specifications in finite time. The proposed algorithm takes advantage of the offline computation of each agent's product automaton and use them to generate safe paths in an online fashion. The proposed algorithm guarantees both collision avoidance among agents and the satisfaction of the given TWTL formula (with a finite relaxation) given some mild assumptions on the environment. Simulation and experimental results show that the proposed algorithm can be used in real-time applications and scales well with increasing environment size and number of agents. Future work may include extending the ideas in this paper to account for heterogeneous teams and asynchronous movement. Moreover, further improvements can be done to the current algorithm by incorporating sampling-based ideas to reduce computational complexity and still provide optimality guarantees (e.g., [49,50]).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A

A.1. Proof of Lemma 1

Since the conflicts are symmetric as per (4), (9) is equivalent to (7). Accordingly, first we will show that at any time $t \geq 0$, each agent i who reaches the last line of Alg. 1 and moves to the computed p^i_{t+1} satisfies (7). For the sake of contradiction, suppose that this is not true and there exists $j \neq i$ such that $\pi^i_t < \pi^j_t$ and $(x^i_t, x^i_{t+1}) \in C_{\mathcal{G}}(x^j_t, x^j_{t+1})$. Note that there are two possibilities for p^i_{t+1}: (1) p^i_{t+1} was determined by the agent itself using Alg. 4 (line 12), or (2) p^i_{t+1} was determined by a lower priority agent using Alg. 4 due to deadlock (line 23). We will show that both of these two cases lead to contradiction.

Case 1: Let p^i_{t+1} be determined by agent i. Due to (2), agents with conflicting transitions must be within 2 hops from each other. Hence, $x^j_t \in \mathcal{N}^2(x^i_t)$. Accordingly, (x^i_t, x^i_{t+1}) would be included in C_Δ of agent i (Alg. 2, line 3). Since Alg. 3 generates a path $\mathbf{p}_i$ in sub-automaton $\mathcal{P}'_i \subseteq \mathcal{P}_i$ that is pruned of C_Δ (lines 4–5), we end up with a contradiction: $(x^i_t, x^i_{t+1}) \notin C_{\mathcal{G}}(x^j_t, x^j_{t+1})$.

Case 2: If p^i_{t+1} was determined by a lower priority agent using Alg. 4, then there are two possibilities:

- *Case 2a* ($p^i_{t+1} = p^i_t$): If agent i was asked to stay stationary (in lines 12 or 18 in Alg. 4), due to (3), a conflict could occur only if another agent attempted to move to agent i's state. However, if $P_{flag} = false$, lines 11–15 in Alg. 4 would stop any agent attempting to move to i's state. If $P_{flag} = true$, then agent i was made stationary in line 18 and its desired next state was not modified in lines 23–27 of Alg. 4. This implies that agent i was not on the shortest path computed in line 21, which is from the next state (x_D) of the highest priority agent in $\bar{N}_i$, i.e., $\overline{\Pi}(1)$, to the nearest unoccupied state in $\mathcal{G}' = (X, \Delta \setminus C_{\mathcal{G}}(x^{\overline{\Pi}(1)}_t, x_D), w)$. Note that such a shortest path always exist due to the premise of Lemma 1, which states that every connected component on $\mathcal{G}'$ must have at least n nodes (hence an unoccupied state). Since all agents in $\bar{N}_i$ other than the highest priority agent and the agents on the shortest path are asked to remain stationary, due to (3), i is again guaranteed to not have any conflicts, leading to a contradiction.
- *Case 2b* ($p^i_{t+1} \neq p^i_t$): if agent i was asked to move to specific state p^i_{t+1} by a lower priority agent, then it must be on the shortest path computed in line 21. Since this path is computed on $\mathcal{G}'$, which is the original graph minus all the edges in conflict with the planned transition of the highest priority agent in $\bar{N}_i$, we know that none of the agents moving along the path will have a conflict with the highest priority agent. Furthermore, since any shortest path on $\mathcal{G}'$ is conflict-free as per the premise of Lemma 1, these agents will not have any conflicts with each other either. Finally, every other agent who is not on the shortest path will remain stationary (hence they cannot have any conflicts with i either). Consequently, once again i is guaranteed to not have any conflicts, leading to a contradiction.

Accordingly, each agent i who reaches the last line of Alg. 1 and moves to the computed p^i_{t+1} is guaranteed to satisfy (7).

Next, we will show that every agent i is guaranteed to reach the last line (line 29) of Alg. 1. Note that it can be shown that no infinite loop is possible in Algs. 2, 3, 4, which can be called during the execution of Alg. 1. Furthermore, for each agent i, U^i_{flag} starts as false in each t and if it ever becomes true it stays true until line 29. Moreover, each agent i reaches line 29 if and only if U^k_{flag} is true for all $k \in \bar{N}_i$. For the sake of contradiction, suppose that there exist agents whose U_{flag} never becomes true during the execution in time t, and let i be the highest priority agent among such agents. Accordingly, since U^j_{flag} is true for all $\pi^i_t < \pi^j_t$, agent i must have passed the while loop in lines 8–12 and reached line 15 (or jumped to line 24 with an updated path from Alg. 4) where Alg. 3 returns either of the following : (1) U^i_{flag} is true and D_{flag} is false, or (2) U^i_{flag} is false and D_{flag} is true. Even if the second case occurs, U^i_{flag} will eventually become true in line 18 after agent i runs Alg. 4. Hence, we reach a contradiction. Since U^i_{flag} eventually becomes true for every agent i, all agents are guaranteed to reach line 29 in their runs of Alg. 1. As per the first part of the proof, their transitions will satisfy (8).

A.2. Proof of Theorem 1

In light of Lemma 1, we know that when the premise of the theorem holds, agents make conflict-free transitions at each time $t \geq 0$. Here, we will show that under such transitions realized by Alg. 1, the energy of the highest priority agent in the system strictly decreases in each time step until all agents have zero energy, i.e., they all have satisfied their specifications.

Suppose that agent i has the highest priority, i.e., $\pi^i_t > \pi^j_t, \forall j \in N \setminus \{i\}$ at time t. Then Alg. 2 (line 6) always updates C_Δ with the transitions that drive agent i to higher energy states in the next time step, i.e., $C_\Delta = \{\big((x^i_t, s^i_t)(x^i_{t+1}, s^i_{t+1})\big) \in \Delta_{\mathcal{P}_i,1} | E(p^i_{t+1}, \mathcal{P}_i) \geq E(p^i_t, \mathcal{P}_i)\}$. Accordingly, line 5 in Alg. 3 guarantees to prune the higher energy states in the next hop hence $E(p^i_{t+1}, \mathcal{P}_i) < E(p^i_t, \mathcal{P}_i)$ is always true.

Given that agent i was the minimum energy agent at time t, one can state $E(p^{min}_t, \mathcal{P}_{min}) = E(p^i_t, \mathcal{P}_i)$. Moreover, by definition, $E(p^{min}_{t+1}, \mathcal{P}_{min}) \leq E(p^i_{t+1}, \mathcal{P}_i)$. Overall, the previous three energy relations imply $E(p^{min}_{t+1}, \mathcal{P}_{min}) < E(p^{min}_t, \mathcal{P}_{min})$, i.e., the smallest positive energy strictly decreases even when the highest priority agent has changed. Therefore, in a system with a finite number of agents moving in a finite graph, all agents reach an accepting state $p \in F_{\mathcal{P}}$ with $E(p, \mathcal{P}) = 0$ in a finite number of transitions since $E(p^i_t, \mathcal{P}_i) \neq \infty$ by the pruning of infinite energy states.

Appendix B. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.robot.2021.103801.

References

[1] X.C. Ding, A.R. Rahmani, M. Egerstedt, Multi-UAV convoy protection: An optimal approach to path planning and coordination, IEEE Trans. Robot. 26 (2) (2010) 256–268.

[2] W. Burgard, M. Moors, D. Fox, R. Simmons, S. Thrun, Collaborative multi-robot exploration, in: Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065), 1, IEEE, 2000, pp. 476–481.

[3] S. Seyedi, Y. Yazıcıoğlu, D. Aksaray, Persistent surveillance with energy-constrained UAVs and mobile charging stations, IFAC-PapersOnLine 52 (20) (2019) 193–198.

[4] R. Bhat, Y. Yazıcıoğlu, D. Aksaray, Distributed path planning for executing cooperative tasks with time windows, IFAC-PapersOnLine 52 (20) (2019) 187–192.

[5] A.Y. Yazıcıoğlu, M. Egerstedt, J.S. Shamma, Communication-free distributed coverage for networked systems, IEEE Trans. Control Netw. Syst. 4 (3) (2016) 499–510.

[6] A. Marchidan, E. Bakolas, Collision avoidance for an unmanned aerial vehicle in the presence of static and moving obstacles, J. Guid. Control Dyn. 43 (1) (2020) 96–110.

[7] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, P. Abbeel, Motion planning with sequential convex optimization and convex collision checking, Int. J. Robot. Res. 33 (9) (2014) 1251–1270.

[8] M. Erdmann, T. Lozano-Perez, On multiple moving objects, Algorithmica 2 (1–4) (1987) 477.

[9] A.D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, P. Tabuada, Control barrier functions: Theory and applications, in: 2019 18th European Control Conference (ECC), IEEE, 2019, pp. 3420–3431.

[10] D. Zhou, Z. Wang, S. Bandyopadhyay, M. Schwager, Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells, IEEE Robot. Autom. Lett. 2 (2) (2017) 1047–1054.

[11] Y. Zhou, H. Hu, Y. Liu, S.-W. Lin, Z. Ding, A distributed approach to robust control of multi-robot systems, Automatica 98 (2018) 1–13.

[12] L. Wang, A.D. Ames, M. Egerstedt, Safety barrier certificates for collisions-free multirobot systems, IEEE Trans. Robot. 33 (3) (2017) 661–674.

[13] J. Van Den Berg, S.J. Guy, M. Lin, D. Manocha, Reciprocal n-body collision avoidance, in: Robotics Research, Springer, 2011, pp. 3–19.

[14] B. Şenbaşlar, W. Hönig, N. Ayanian, Robust trajectory execution for multi-robot teams using distributed real-time replanning, in: Distributed Autonomous Robotic Systems, Springer, 2019, pp. 167–181.

[15] C.E. Luis, A.P. Schoellig, Trajectory generation for multiagent point-to-point transitions via distributed model predictive control, IEEE Robot. Aut. Lett. 4 (2) (2019) 375–382.

[16] L. Dai, Q. Cao, Y. Xia, Y. Gao, Distributed MPC for formation of multi-agent systems with collision avoidance and obstacle avoidance, J. Franklin Inst. B 354 (4) (2017) 2068–2085.

[17] T. Mylvaganam, M. Sassano, A. Astolfi, A differential game approach to multi-agent collision avoidance, IEEE Trans. Automat. Control 62 (8) (2017) 4229–4235.

[18] Z. Wang, R. Spica, M. Schwager, Game theoretic motion planning for multi-robot racing, in: Distributed Autonomous Robotic Systems, Springer, 2019, pp. 225–238.

[19] C. Baier, J.-P. Katoen, et al., Principles of Model Checking, Vol. 26202649, MIT press Cambridge, 2008.

[20] D. Aksaray, K. Leahy, C. Belta, Distributed multi-agent persistent surveillance under temporal logic constraints, IFAC-PapersOnLine 48 (22) (2015) 174–179.

[21] A. Ulusoy, S.L. Smith, X.C. Ding, C. Belta, D. Rus, Optimality and robustness in multi-robot path planning with temporal logic constraints, Int. J. Robot. Res. 32 (8) (2013) 889–911.

[22] E.M. Wolff, U. Topcu, R.M. Murray, Optimization-based trajectory generation with linear temporal logic specifications, in: IEEE Int. Conf. on Robotics and Automation, 2014, pp. 5319–5325.

[23] J. Alonso-Mora, J.A. DeCastro, V. Raman, D. Rus, H. Kress-Gazit, Reactive mission and motion planning with deadlock resolution avoiding dynamic obstacles, Auton. Robots 42 (4) (2018) 801–824.

[24] M. Guo, D.V. Dimarogonas, Multi-agent plan reconfiguration under local LTL specifications, Int. J. Robot. Res. 34 (2) (2015) 218–235.

[25] R. Koymans, Specifying real-time properties with metric temporal logic, Real-Time Syst. 2 (4) (1990) 255–299, http://dx.doi.org/10.1007/BF01995674.

[26] O. Maler, D. Nickovic, Monitoring temporal properties of continuous signals, in: Y. Lakhnech, S. Yovine (Eds.), Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Springer, 2004, pp. 152–166, http://dx.doi.org/10.1007/978-3-540-30206-3_12.

[27] I. Tkachev, A. Abate, Formula-free finite abstractions for linear temporal verification of stochastic hybrid systems, in: Int. Conf. on Hybrid Systems: Computation and Control, 2013.

[28] C.-I. Vasile, D. Aksaray, C. Belta, Time window temporal logic, Theoret. Comput. Sci. 691 (2017) 27–54.

[29] D. Aksaray, C.-I. Vasile, C. Belta, Dynamic routing of energy-aware vehicles with temporal logic constraints, in: Int. Conference on Robotics and Automation (ICRA), IEEE, 2016, pp. 3141–3146.

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

[31] A. Nikou, J. Tumova, D.V. Dimarogonas, Cooperative task planning of multi-agent systems under timed temporal specifications, in: 2016 American Control Conference (ACC), IEEE, 2016, pp. 7104–7109.

[32] S. Ahlberg, D.V. Dimarogonas, Human-in-the-loop control synthesis for multi-agent systems under hard and soft metric interval temporal logic specifications, in: 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE), IEEE, 2019, pp. 788–793.

[33] G.E. Fainekos, Revising temporal logic specifications for motion planning, in: 2011 IEEE International Conference on Robotics and Automation, IEEE, 2011, pp. 40–45.

[34] J. Alonso-Mora, P. Beardsley, R. Siegwart, Cooperative collision avoidance for nonholonomic robots, IEEE Trans. Robot. 34 (2) (2018) 404–420.

[35] C.-I. Vasile, C. Belta, An Automata-Theoretic Approach to the Vehicle Routing Problem, in: Proceedings of the Robotics: Science and Systems (RSS), Berkeley, California, USA, 2014, http://dx.doi.org/10.15607/RSS.2014.X.045.

[36] P. Velagapudi, K. Sycara, P. Scerri, Decentralized prioritized planning in large multirobot teams, in: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2010, pp. 4603–4609.

[37] M. Čáp, P. Novák, A. Kleiner, M. Selecký, Prioritized planning algorithms for trajectory coordination of multiple mobile robots, IEEE Trans. Autom. Sci. Eng. 12 (3) (2015) 835–849.

[38] R. Peterson, A.T. Buyukkocak, D. Aksaray, Y. Yazıcıoğlu, Decentralized safe reactive planning under TWTL specifications, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020.

[39] G.J. Pappas, Bisimilar linear systems, Automatica 39 (12) (2003) 2035–2047.

[40] A. Nikou, D. Boskos, J. Tumova, D.V. Dimarogonas, On the timed temporal logic planning of coupled multi-agent systems, Automatica 97 (2018) 339–345.

[41] X. Ding, M. Lazar, C. Belta, LTL receding horizon control for finite deterministic systems, Automatica 50 (2) (2014).

[42] S.P. Bradley, A.C. Hax, T.L. Magnanti, Applied mathematical programming, 1977.

[43] D. Aksaray, A.Y. Yazıcıoğlu, E. Feron, D.N. Mavris, Message-passing strategy for decentralized connectivity maintenance in multiagent surveillance, J. Guid. Control Dyn. 39 (3) (2016) 542–555.

[44] J.A. Preiss, W. Honig, G.S. Sukhatme, N. Ayanian, Crazyswarm: A large nano-quadcopter swarm, in: Int. Conference on Robotics and Automation (ICRA), IEEE, 2017, pp. 3299–3304.

[45] H. Zhu, J. Alonso-Mora, Chance-constrained collision avoidance for mavs in dynamic environments, IEEE Robot. Autom. Lett. 4 (2) (2019) 776–783.

[46] S. Karaman, E. Frazzoli, Vehicle routing problem with metric temporal logic specifications, in: 2008 47th IEEE Conference on Decision and Control, IEEE, 2008, pp. 3953–3958.

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

[48] W. Wu, S. Bhattacharya, A. Prorok, Multi-robot path deconfliction through prioritization by path prospects, in: 2020 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2020, pp. 9809–9815.

[49] Y. Kantaros, M.M. Zavlanos, STyLuS*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems, Int. J. Robot. Res. 39 (7) (2020) 812–836.

[50] C.I. Vasile, C. Belta, Sampling-based temporal logic path planning, in: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2013, pp. 4817–4822.

Ryan Peterson received his B.S. and M.S. degrees in Aerospace Engineering and Mechanics from the University of Minnesota-Twin Cities. His research interests include multi-agent planning and control of autonomous vehicles with a current focus on provably correct synthesis and deadlock resolution algorithms. He has since gone on to industry as a Systems Engineer with Raytheon Technologies.

Ali Tevfik Buyukkocak is a Ph.D. student with the University of Minnesota, Aerospace Engineering and Mechanics Department. He received his B.S. and M.S. degrees in Aerospace Engineering from Middle East Technical University in Turkey. His research interests include provably-correct multi-agent planning and control algorithms, optimal control theory, and formal methods.

Derya Aksaray is currently an Assistant Professor in the Aerospace Engineering and Mechanics Department at the University of Minnesota. She received her Ph.D. degree in Aerospace Engineering from the Georgia Institute of Technology in 2014. After her Ph.D., she held post-doctoral researcher positions at Boston University from 2014 to 2016 and at the Massachusetts Institute of Technology from 2016 to 2017. Her research interests lie primarily in the areas of control theory, formal methods, and machine learning with applications to autonomous systems and aerial robotics.

Yasin Yazıcıoğlu is a Research Assistant Professor in the Department of Electrical and Computer Engineering at the University of Minnesota. He received the Ph.D. degree in Electrical and Computer Engineering from the Georgia Institute of Technology in 2014. He was a Postdoctoral Research Associate at the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology from 2014 to 2017. His research is mainly focused on distributed decision making, control, and learning with applications to cyber–physical systems, networks, and robotics.