# Safe Multi-Agent Planning under Time-Window Temporal Logic Specifications

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Ryan James Peterson

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Master of Science

Under the supervision of Dr. Derya Aksaray

May, 2020

**Safe Multi-Agent Planning under Time-Window Temporal Logic Specifications**

**by Ryan James Peterson**

## Abstract

This thesis addresses a multi-agent planning problem, where each agent aims to achieve an individual task while avoiding collisions with other agents in the environment. It is assumed that each agent's task is expressed as a Time-Window Temporal Logic (TWTL) specification defined over a 3D environment. A distributed receding horizon algorithm is introduced for online planning of trajectories. Under mild assumptions on the environment topology, the resulting agent trajectories are always safe (collision-free) and lead to the satisfaction of the TWTL specifications or their finite temporal relaxations. Accordingly, each agent is guaranteed to safely achieve its task while avoiding deadlocks. Performance of the proposed algorithm is demonstrated via numerical simulations and through experiments with quadrotors.

# Acknowledgements

First of all, I want to thank my advisor, Dr. Derya Aksaray. Without your continued insight and guidance this work would not have been possible. I am grateful for your support and steady feedback over these last couple of years.

I would like to thank my collaborators, Dr. Yasin Yazıcıoğlu and Ali Tevfik Buyukkocak, for their insight and continued feedback on this work. I am grateful for your time and our many discussions on formal methods and graph theory.

Additionally, I want to thank my colleagues and faculty who advanced my education during my time at the University of Minnesota. Last but not least, I would like to thank my family and friends for providing balance in my life and for their continued support of the unknown.

# Dedication

To my family, and to the late Dave Decoursin who peaked my interest in the wonderful complexities of engineering at an early age.

# Contents

# List of Tables

# List of Figures

# Symbols and Notation

Note: throughout this thesis, finite sets are given by capital letters, $S$, lower case letters denote a member of a set, $s \in S$, and vectors or sequences are shown in bold, $\mathbf{s}$.

$\neg, \vee, \wedge, \rightarrow$ Boolean operators for negation, conjunction, disjunction, and implication

$S_1 \times S_2$ Cartesian product of sets $S_1$ and $S_2$

$S \setminus \{s\}$ Removing an element $s$ from the set $S$

$2^S$ Power set (set of all subsets) of a set $S$

$|\cdot|$ Cardinality operator

$\emptyset$ Empty set

$\mathbb{R}$ Set of all real numbers

$\mathbb{R}^+$ Set of all positive real numbers

$\mathbb{R}^N$ Euclidean space of dimension $N$

$\mathbb{Z}$ Set of all integers

$\mathbb{Z}^+$ Set of all positive integers

$n$ Integer number of agents

| | |
|---|---|
| $N$ | Set of all agents in the environment ($\{1, \ldots, n\}$) |
| $p$ | A state of the product automaton |
| $\mathbf{p}$ | A path over the product automaton |
| $r$ | Radius of an agent in $\mathbb{R}^3$ |
| $t$ | Discrete time |
| $x$ | A state of the transition system or environment graph |
| $\mathbf{x}$ | A path over the transition system or environment graph |
| $\pi$ | Agent priority |
| $\phi$ | TWTL formula |
| $\mathcal{L}$ | Finite language |
| $\mathcal{O}(\cdot)$ | Complexity class |

# Chapter 1

# Introduction

## 1.1 Motivation

The control of multi-agent systems, such as a team of package delivery drones, disaster relief robots, warehouse vehicles or autonomous cars, has gained significant attention in recent years. While performing these complex tasks over a shared environment, collision avoidance is critical for the safe operation of these multi-agent systems. To this end, significant effort has gone into developing collision avoidance algorithms for multi-agents systems, however, many of these algorithms only account for simple point-to-point tasks which may not be suitable to many real-world robotics problems. Moreover, many of these algorithms do not have guarantees on avoiding deadlocks which results in infeasible solutions or no solution at all.

Consider a package delivery task where a robot is tasked with delivering a package from "Room A" to "Room B" while avoiding other agents in the environment. Now, let's consider the scenario where the door to "Room B" is closed or some other agent is blocking the door for a long period of time. It may be desired that the robot drops off the package near the door or possibly in a nearby room

if the room is not accessible. Alternatively, if the robot determines the agent currently blocking the door will soon move out of the doorway, it will wait to access the room. To capture these possible alternatives, spatial and possibly temporal relaxations must be considered. In order to account for task relaxations in a real-world environment, local information needs to be incorporated during execution to update the environment with the newly appeared and disappeared obstacles.

In the literature, multi-agent systems have often been considered in coverage (e.g.,[1, 2, 3]), persistent surveillance (e.g.,[4, 5, 6]), or formation (e.g.,[7, 8, 9]) problems. However, multi-agent systems can achieve more complex tasks (e.g.,[10, 11]). For example, consider the package delivery task given above for a single agent (e.g., robot). Now, let's consider the scenario where many packages need to be delivered from different locations in the building to "Room B", and a team of robots is available for the task. Room B can only be occupied by one robot at any given time and that robot may only stay in the room for at most one minute, and if a robot must wait more than two minutes to access Room B then it chooses to drop off the package just outside of Room B. The robots must visit a recharging station at least once every 30 minutes during operation, and immediately after delivering the last package all robots must go to "Room C" where they are stored when not in use. While performing these deliveries, the robots must also avoid colliding with obstacles in the environment and each other.

The example above is an example to a high-level complex task specification which can be rigorously described using temporal logic. Using temporal logic allows for high-level plans to be encoded in order to design control policies which satisfy these high-level plans and verify the feasibility of the designed control policies. Addressing the high-level complex specifications for a multi-agent system can take the form of a three-step hierarchical approach as follows (e.g.,[12, 13, 14]): First, the agent dynamics are abstracted into a finite discrete transition system using some form of partitioning. Second, a discrete plan that satisfies the high-level

tasks are synthesized. Third, the discrete plan is translated into a sequence of controllers for the original system.

Since the complexity of centralized path planning grows exponentially with the number of agents and local unknown information can not be incorporated during execution, significant effort has been devoted to the design of distributed or decentralized algorithms (e.g., [15, 16, 17, 18, 19]).

This thesis introduces a distributed algorithm which enables each agent to satisfy its individual complex high-level specifications encoded in a time bounded logic. The proposed algorithm allows for relaxation of the specifications which may be required while considering collision situations between other agents. The algorithm guarantees safety and deadlock avoidance.

### 1.1.1 Motivating Example

As a simple example, consider a 2 agent scenario where each agent must first visit a pick-up region, then visit the same drop-off region while avoiding collision with one another. This is shown by the environment in Fig. 1.1. In this scenario "D" is the designated drop-off region for both agents, where only one agent may occupy "D" at any given time.

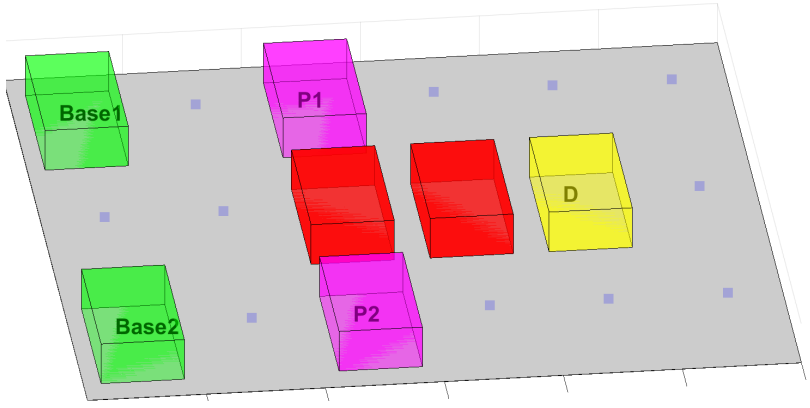


Figure 1.1: Initial positions given in green, obstacles are red, pick-up regions (P1, P2) are magenta, and the drop-off region D is shown in yellow.

Using the distributed algorithm presented in this thesis, each agent is unaware of the other agent's intended task. Upon seeing that "D" is occupied, the other agent must wait for "D" to be unoccupied. This waiting may cause the initial mission criteria to be violated. In state-of-the-art collision avoidance algorithms (discussed in the following section), this is a conflict that can not be avoided without any mission modifications. Alternatively, the algorithm presented in this thesis can account for local information (such as a desired location being occupied) in order to modify the initial mission specifications, and thus avoid infeasibility in scenarios such as this.

## 1.2 Background and Related Works

To address collision avoidance for multi-agent systems in a distributed manner, potential fields [20], sequential convex programming (e.g. [21, 22]), and priority based planning [23] have been employed. However, these methods do not account for navigating among other decision-making agents, where other agent tasks are considered during interactions, and these methods can lead to deadlocks, i.e., local-minima where the agent can not make progress towards its goal. Regarding potential field methods, [24] accounts for local-minima (i.e., deadlocks) using streamlines of the collision-avoidance vector fields, and [25] relies on human assistance to avoid or resolve deadlocks.

Alternatively, methods such as control barrier functions (CBFs) (e.g., [26, 27]) or buffered Voronoi cells (e.g., [28, 29]) have been employed which can effectively account for decision-making agent interaction. In recent years, these methods have been improved upon to account for constraints on agent dynamics as well as deadlock detection and avoidance. In particular, [27] explicitly defines different types of deadlock which may occur, and how those deadlocks are accounted for through their implementation of CBFs. The approach in [30], which employs a buffered Voronoi cell collision-avoidance strategy, does not explicitly guarantee deadlock avoidance but shows marked improvement compared with other methods such as

optimal reciprocal collision-avoidance (ORCA) [31]. A recent approach known as $\epsilon$CCA [32] is derived from reciprocal velocity obstacle (RVO) [33] methods with additional repulsive forces and formulates the problem as a quadratic optimization similar to CBF implementation. This method is expanded upon in [34] to account for deadlocks using a mission planner. Other approaches such as distributed model predictive control (e.g. [35, 36]), differential games (e.g., [37, 38]), and game theoretic methods (e.g., [39, 40]) have been used for distributed planning. The methods described thus far typically have not accommodated 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 [41]. For instance, linear temporal logic (LTL) [42] has been extensively used in motion planning and control of robots (e.g., [15, 43, 44, 45]). To ensure collision avoidance while satisfying more complex tasks defined as LTLs in a distributed manner, researchers have employed what are sometimes referred to as *hybrid* controllers. These *hybrid* controllers essentially combine a high-level mission planner with a local planner which enforces collision avoidance (e.g., [34, 46, 47]). These local planners which enforce collision avoidance are derived from the many methods previously outlined. While these results are promising, LTL can not express tasks with 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., [48, 49, 50, 51]).

Ensuring safety under bounded temporal logics for multi-agent systems has thus received considerable attention in recent years. Regarding signal temporal logic (STL) [49], a common approach has been to formulate the problem as a mixed integer linear program (MILP) in a receding horizon manner as in [52, 53]. However, this implementation can lead to infeasible solutions and is restricted to small

systems due to its poor computational scalability. Alternatively, for a fragmented version of metric temporal logic (MTL) [48] known as metric interval temporal logic (MITL) [54], graph automata-based approaches have been implemented [55, 56]. Yet, collision avoidance is not considered in [55], while [56] accounts for collision avoidance and incorporates human input using an inverse reinforcement learning (IRL) [57] algorithm. However, the collision avoidance approach is overly conservative by considering all other agents as fixed obstacles while an updated path is found using a modified Dijkstra's algorithm [58].

Since it may be sometimes impossible to find safe paths which satisfy the initial specifications (i.e., infeasible), some form of specification relaxation can be desirable. These infeasibilities may arise from conflicting objectives among agents, poorly constructed temporal logic formulae, or path deviations required to enforce collision avoidance. Different approaches have been presented to address infeasible initial specifications. One approach is to add slack variables as in [35], but this could lead to endangering safety. The framework in [59] gives feedback on why the specification is not satisfiable and how to modify it. In [32], the robots slow down to give their iterative optimization algorithm more time to solve when necessary. The framework in [56] looks to minimize the violation by considering both hard and soft constraints, where the hard constraints (such as collision avoidance) must be satisfied. Alternatively, the algorithm presented in this thesis allows for the temporal relaxation of time-window temporal logic (TWTL) specifications and it is shown that, under mild assumptions on the environment topology, the algorithm ensures the completion of all TWTL specifications with finite relaxation.

This work is closely related to [15, 60, 16]. In [15], a multi-agent receding horizon approach is proposed to generate each agent's path independently using only local information. Their algorithm ensures LTL satisfaction using the idea of energy function defined over a product automaton. However, tasks under explicit time constraints are not considered, and collision avoidance is not considered. In [60], 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 [60], 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 [16] 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 and practically achieved by the assumption of quadrotors flying at different altitudes.

The collision avoidance procedure in this work is closely related to [61, 62]. In [61], a decentralized prioritized planning method is introduced, and [62] is essentially an asynchronous extension of [61] called *revised priority planning*. Both of these variants of prioritized planning use an $A^*$ planner with a heuristic function to facilitate completion. Though the idea of using a heuristic function for dynamic priority has been around for a while (e.g.,[63]). Before execution in both methods, each robot computes their entire trajectory (from start to goal state) using the $A^*$ planner, and broadcasts this trajectory to other robots which may conflict with the computed trajectory. This is repeated in an iterative manner until collisions among all robots are avoided. In [61], infeasibility is not explicitly accounted for, while in [62], infeasibilities are avoided through assumptions on the environment (i.e., *well-formed infrastructures*) and the initial robot states (which could be potentially prohibitive). These algorithms assume complete peer-to-peer communication while planning, and the entire path must be calculated before execution which does not lend itself well to unexpected interruptions.

The proposed collision avoidance procedure makes use of an energy function (i.e., a heuristic function) as an estimate of the cost from its current state to the goal state to assign priorities in a dynamic manner which facilitates completion. However, the proposed algorithm in this thesis differs from previous versions of distributed prioritized planning (e.g., [61, 62]) by 1) detecting and resolving deadlocks (under mild assumptions on the environment), and 2) considers limited communication

of information while computing agent paths in a receding horizon manner using dynamic programming.

## 1.3 Thesis Overview and Contributions

**"The objective of this thesis is to develop an efficient and safe path planning algorithm for multi-agent systems where each agent aims to achieve a complex task with spatial, temporal, and logical requirements."**

In this thesis, a distributed algorithm is introduced for safe satisfaction of TWTL specifications, which are particularly fitted to express complex time bounded robotics tasks [51], for multi-agent systems operating in shared environments. In this setting, agents which move over the environment on linear paths between adjacent states are considered, where each agent is assigned an individual TWTL specification. Agents do not know the other agents' tasks and only communicate their path information with other agents in their local neighborhoods to plan collision-free paths in a receding horizon manner.

Many of the path planning algorithms described in the previous section which consider collision avoidance (e.g.,[30, 35, 62, 64]) can not update plans based on local information in order to improve upon task satisfaction. The proposed algorithm can account for local information during path planning in order to improve upon task satisfaction as will be made clear by example in Sec. 6.4.

The contributions of this thesis are as follows:

- A distributed algorithm for safe satisfaction of TWTL specifications.
- Detection and resolution of possible deadlocks (under mild assumptions on the environment).
- Theoretical analysis for collision avoidance, deadlock resolution, and finite temporal relaxation.

- Demonstration of the algorithm through both numerical simulations and physical experiments on quadrotors.

## 1.4 Organization

Accordingly, the remainder of this thesis is organized as follows:

- Chapter 2 provides some graph theory preliminaries, an overview of the TWTL syntax and semantics, and its temporal relaxation.
- Chapter 3 provides the information to formulate the problem statement. This chapter also details the centralized solution to the problem.
- Chapter 4 outlines the proposed algorithm, and goes through the details of the overall algorithm.
- Chapter 5 presents the theoretical results and a discussion of computational complexity.
- Chapter 6 demonstrates the proposed method via numerical simulations and presents the results of these simulations. These results are used to discuss the scalability properties of the proposed method with respect to the number of agents, receding horizon length, size of the environment graph and task specifications.
- Chapter 7 demonstrates the algorithm on a team of quadrotors and describes the experimental setup.
- Chapter 8 presents a final discussion of the presented algorithm as well as possible extensions and future work.

In addition, the appendices provide supplementary material. Appendix A provides a glossary of terms and a table of acronyms used. Appendix B provides further details on the collision avoidance procedure.

# Chapter 2

# Preliminaries

The purpose of this chapter is to give the necessary background information for the graph theoretic approach used for planning under TWTL specifications.

## 2.1 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. The terms $\Delta_x^{in}, \Delta_x^{out} \subseteq \Delta$ are used 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 $(x, x') \in \Delta$. $\mathcal{N}(x)$ denotes the set of out-neighbors of $x$. For brevity, the term "neighbor" will be used when referring to an "out-neighbor" throughout this thesis. The set of all nodes that are reachable from $x$ in at most $H$ hops is denoted as $\mathcal{N}^H(x)$.

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. The path length

is denoted by $|\mathbf{x}|$, 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. In this thesis, all connected components are assumed to be strongly connected.

## 2.2 Time Window Temporal Logic (TWTL)

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

$$\phi := s \,|\, \phi_i \wedge \phi_j \,|\, \phi_i \vee \phi_j \,|\, \neg\phi_i \,|\, \phi_i \cdot \phi_j \,|\, H^d s \,|\, [\phi_i]^{[a,b]},$$

where $s \in \mathcal{S}$ is a site label and $\mathcal{S}$ is the set of site labels; $\wedge$, $\vee$, and $\neg$ are the Boolean operators for conjunction, disjunction, and negation, respectively; $\cdot$ is the concatenation operator such that $\phi_i \cdot \phi_j$ specifies that first $\phi_i$ and then immediately $\phi_j$ must be satisfied. The semantics are defined with respect to finite output words $\mathbf{o}$ over $\mathcal{S}$ where $\mathbf{o}(k)$ denotes the $k^{th}$ element on $\mathbf{o}$. The *hold* operator $H^d s$ specifies that a region $s \in \mathcal{S}$ should be visited for $d$ time units (i.e., $\mathbf{o} \models H^d s$ if $\mathbf{o}(t) = s \;\; \forall t \in [0, d]$), while the *within* operator $[\phi]^{[a,b]}$ bounds the satisfaction of $\phi$ within a time window $[a, b]$. The language of all finite output words $\mathbf{o}$ satisfying $\phi$ is denoted by $\mathcal{L}(\phi)$.

The rich expressivity of TWTL is best shown through examples. The following set of examples consider an agent (e.g., robot) servicing different regions in an environment:

*servicing without time constraints* – "service $A$ for 2 time units and do not visit

C"

$$\phi_1 = H^2 A \wedge \neg C$$

*servicing within a time bound* – "service $A$ for 2 time units within [0,4]"

$$\phi_1 = [H^2 A]^{[0,4]}$$

*servicing in sequence:* "service $A$ for 4 time units within [0,10], then immediately after this service B for 1 time unit within [5,12]"

$$\phi_1 = [H^4 A]^{[0,10]} \cdot [H^1 B]^{[5,12]}$$

*servicing within time windows* – "service $A$ for 2 time units and service B for 2 time units within [0,6], which both must be performed within [0,10]"

$$\phi_1 = [H^2 A \wedge [H^2 B]^{[0,6]}]^{[0,10]}$$

*enabling conditions* – "within 10 time units, if $A$ is serviced for 2 time units, this implies that either $B$ or $C$ was serviced for 1 time unit"

$$\phi_1 = [H^2 A \Rightarrow H^1 (B \vee C)]^{[0,10]}$$

These examples show that TWTL can be used for a variety of complex task specifications, however other bounded temporal logics such as STL and MTL are actually more expressive than TWTL since they can express continuous time properties. A main benefit of TWTL over other bounded temporal logics is the existence of concatenation, within, and hold operators which can lead to more compact representations of specifications. Another important property of TWTL is that the automaton for the satisfying language of a TWTL formula ($\mathcal{L}(\phi)$) can be directly constructed (i.e., does not need to be translated to another logic). Moreover, a TWTL formula with all possible temporal relaxations can be encoded in a compact manner which is discussed in the next section. The interested reader is referred to [51] for further details on TWTL.

## 2.3 Temporal Relaxation and State Automaton

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:

$$\phi = [H^2 A]^{[0,5]} \cdot \left[H^1 B \wedge [H^2 C]^{[0,6]}\right]^{[2,10]} \tag{2.1}$$

In plain English, $\phi$ 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 $\phi$ is written as:

$$\phi(\boldsymbol{\tau}) = [H^2 A]^{[0,5+\tau_1]} \cdot \left[H^1 B \wedge [H^2 C]^{[0,6+\tau_2]}\right]^{[2,10+\tau_3]} \tag{2.2}$$

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 $|\boldsymbol{\tau}|_{TR} = \max_j(\tau_j)$ [51]. Note that $\boldsymbol{\tau}$ may actually be negative, meaning $\phi(\boldsymbol{\tau})$ represents a stronger specification than the initial specification $\phi$.

For any TWTL specification, $\phi$, a deterministic finite state automaton (dFSA) can be constructed that captures all feasible temporal relaxations of $\phi$, i.e., $\phi(\boldsymbol{\tau})$. 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. Throughout this thesis $\phi$ is often used to denote the relaxed TWTL specification $\phi(\boldsymbol{\tau})$.

**Definition 1.** *(Deterministic Finite State Automaton) [51] A dFSA that represents all temporal relaxations is represented as 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;*
- $\Sigma$ *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.*

A transition is denoted as $(s, s') = \delta_{\mathcal{A}_\infty}$, where $\sigma \in \Sigma$. Alternatively, a transition on the dFSA can be interpreted as a control policy which generates a symbol $\sigma \in \Sigma$ that results in the transition from $s$ to $s'$. A finite path of the dFSA $\mathbf{s} = s_0 \dots s_{n+1}$ is generated by a sequence of symbols $\sigma = \sigma_0 \dots \sigma_n$ where $\sigma$ is a finite input *word* over $\Sigma$. An input *word* is said to be *accepting* if the generated path $\mathbf{s}$ ends with some accepting state, i.e., $s_{n+1} \in F_{\mathcal{A}_\infty}$ for $|\sigma| = n$. The (*accepting*) language is the set of all *accepting* input words, and is denoted by $\mathcal{L}_{\mathcal{A}_\infty}$ (or $\mathcal{L}_{\mathcal{A}}$). To illustrate the dFSA which represents all temporal relaxations of a TWTL specification, a finite state automaton which corresponds to the TWTL specification $\phi = [H^2 A]^{[0,4]}$ is first shown in Fig. 2.1. The initial state is defined by an arrow pointing to it ($s_0$), and accepting states are defined by double circles.

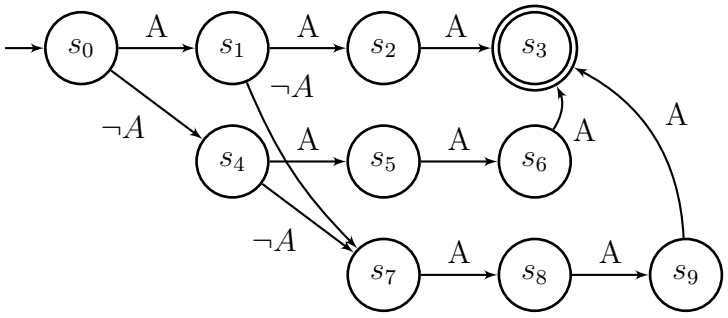


Figure 2.1: Finite state automaton corresponding to $\phi = [H^2 A]^{[0,4]}$.

The finite state automaton depicted in Fig. 2.1 captures all policies which satisfy the TWTL specification $\phi = [H^2 A]^{[0,4]}$, where the (*accepting*) language is $\mathcal{L}_{\mathcal{A}} = \{AAA, A\neg AAAA, \neg AAAA, \neg A\neg AAAA\}$. It is evident that the standard finite state automaton implementation depicted in Fig. 2.1 grows without bound with respect to the time window. A solution to this state explosion was first introduced

in [51] where the authors presented the dFSA (annotated automaton) which is used in this work, $\mathcal{A}_\infty$. By incorporating backwards facing edges, all of the time bounds (and all temporal relaxations) are captured in a compact form shown in Fig. 2.2 for the specification $\phi = [H^2 A]^{[0,*]}$. Note that any path captured in the graph shown in Fig. 2.1 is also captured in the graph shown in Fig. 2.2.

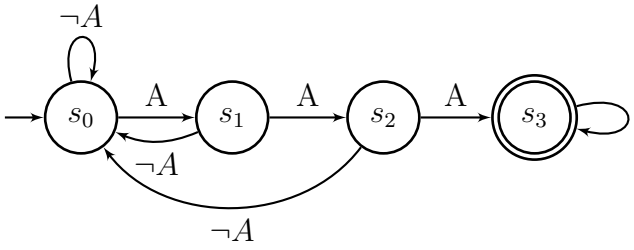


Figure 2.2: Annotated automaton, $\mathcal{A}_\infty$, corresponding to $\phi = [H^2 A]^{[0,*]}$.

# Chapter 3

# Problem Formulation

## 3.1 Agent Model

### 3.1.1 Dynamics and Transition System

An agent moving in a discretized 3D environment, whose abstraction is initially given as a graph $\mathcal{G} = (X, \Delta, w)$ is considered. In general, several methods (e.g., [65, 66, 67]) can be used to construct such an abstraction; however, the construction of the abstraction is not the scope of this thesis. Given an environment graph, $\mathcal{G}$, the dynamics of each agent are modeled as a deterministic weighted transition system. Moreover, the agents move synchronously on $\mathcal{G}$ meaning any state transition happens 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.*

A transition is denoted by $(x, x') \in \Delta$. It is assumed the transition system $\mathcal{T}$ is (strongly) connected, that is, for each $x \in X$, $\exists x' \in X$ that can be reached in a finite number of transitions. 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 fuel cost required to traverse transitions or a combination of the two. Without loss of generality, normalized weights are considered, i.e., $w(x, x') \in (0, 1]$ for all $(x, x') \in \Delta$.

The transition system $\mathcal{T} = (X, x_0, \Delta, w, AP, l)$ shown in Fig. 3.1 is defined by:

- $X = \{x_0, x_1, x_2, x_3, x_4, x_5\}$
- $x_0 = x_0$ (this can be any state in $X$)
- $\Delta$ shown by the arrows between states
- $w$ values are shown on the transitions
- $AP = (A, \neg A, B, \neg B, C, \neg C, D, \neg D)$
- $l = \{x_0 \to A, x_1 \to B, x_2 \to B, x_3 \to C, x_3 \to D\}$

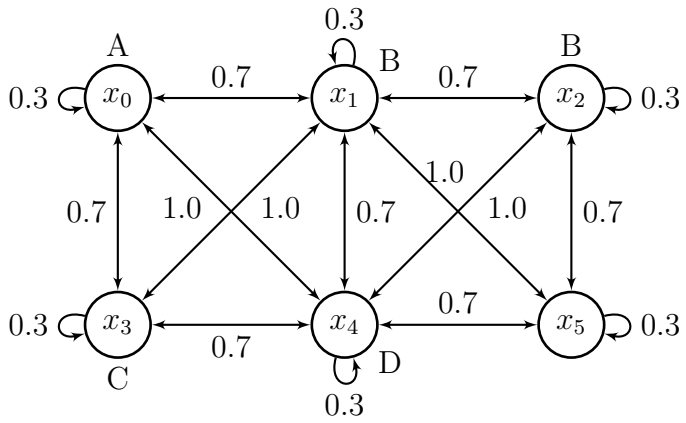


Figure 3.1: Example of a simple transition system, $\mathcal{T}$

### 3.1.2 Communication

In this thesis 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 $2H$-hop of its current position on the environment graph to compute a collision-free path over the next $H$ time steps. Therefore, it is assumed that the agents have such local ($2H$-hop) communication capability and the term $N_i^{2H} \subseteq N$ is used to denote the set of all agents within $2H$ hops of agent $i$'s current position. Such a local communication capability defines a connected multi-hop communication network for each agent. The term $\overline{N}_i \subseteq N$ is used to denote the set of agents that are connected to agent $i$ through such multi-hop communications, i.e., the set of agents that satisfy the conditions

$$\overline{N}_i = \bigcup_{j \in \overline{N}_i} N_j^{2H},\ i \in \overline{N}_i. \tag{3.1}$$

Note that both $N_i^{2H}$ and $\overline{N}_i$ are determined by the current positions of the agents and change over time as the agents move. In the 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 $\overline{N}_i$. These notions will be discussed further in Chapter 4.

### 3.1.3 Specification

Each agent $i$ aims to satisfy a TWTL formula, $\phi_i$, 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 presence of violations, instead of terminating the mission, it will be allowed to satisfy a temporally relaxed version of $\phi_i$, i.e., $\phi_i(\boldsymbol{\tau}^i)$.

## 3.2 Problem Statement

Suppose that there are $n$ identical agents, each with its own respective transition system $\mathcal{T}_i$. This work addresses the problem of planning paths for $n$ agents, each of which is required to satisfy an individual TWTL specification while avoiding collisions.

### 3.2.1 Collision Avoidance and Safe Path

In order to enforce collision avoidance, the agents require some representation of infeasible transitions (i.e., transitions which may result in collision).

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 $C_{\mathcal{G}}(x, x')$. Informally, this represents the transitions which might result in a collision while the transition $(x, x')$ is being traversed. 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. The following mild assumptions are made on the mapping $C_{\mathcal{G}}$.

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

$$(x, x') \in C_{\mathcal{G}}(y, y') \Rightarrow x \in \mathcal{N}^2(y), \tag{3.2}$$

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

$$(x, x') \in C_{\mathcal{G}}(y, y') \iff (y, y') \in C_{\mathcal{G}}(x, x'). \tag{3.3}$$

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

$$C_{\mathcal{G}}(x, x) \subseteq \Delta_x^{in}, \ \forall x \in X. \tag{3.4}$$

Accordingly, it is said that the paths of agents are safe if the concurrent transitions do not conflict with each other. These notions are shown in Fig. 3.2.

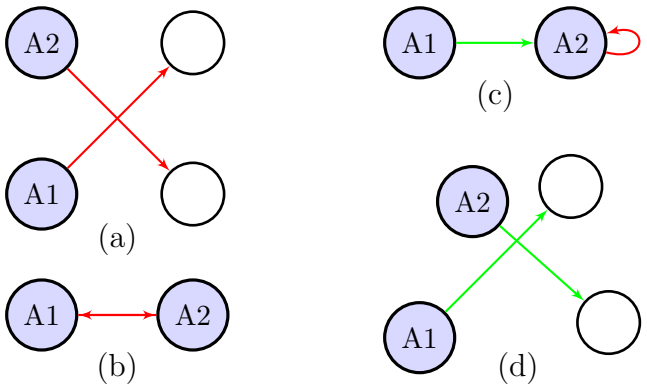


Figure 3.2: Conflicts captured by $C_{\mathcal{G}}$; (a) intersecting diagonal movements, (b) traversing the same transition, and (c) transition to a new state makes self-transition infeasible. (d) Case where the intersecting diagonal movements may not result in conflict.

**Remark 1.** *In the algorithm presented, accounting for conflicting transitions using a mapping such as $C_{\mathcal{G}}$ may be overly conservative. It might be more appropriate to incorporate local low-level controllers (such as [26, 30, 32]) which allow for deviations from the straight-line transition and variations in velocity (see Appendix B) in order to resolve these conflicts. A low-level controller could also account*

*for imperfect motion/noise in a more robust manner (e.g., [64]). These low-level resolution considerations are not in the scope of this thesis, but may be topics of future research.*

Finally, a safe path can be formally defined as:

**Definition 3.** *(Safe path) A path of agent $i$, $\mathbf{x_i} = x_t^i x_{t+1}^i \dots$ over the environment graph $\mathcal{G} = (X, \Delta, w)$ is safe if, for all $t \geq 0$ and for all $j \neq i$, $(x_t^i, x_{t+1}^i) \cap C_{\mathcal{G}}(x_t^j, x_{t+1}^j) = \emptyset$.*

### 3.2.2 Problem Definition

This thesis aims to solve a multi-agent path planning problem which results in agent paths over $\mathcal{T}$ (or $\mathcal{G}$) that always ensures collision avoidance and satisfies the TWTL specifications (or minimally relaxed versions of the original TWTL specifications).

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

$$\begin{aligned} &\min_{\mathbf{x}_1,\dots,\mathbf{x}_n} \sum_{i=1}^{n} |\boldsymbol{\tau}^i|_{TR} \\ &\quad s.t. \;\; (x_t^i, x_{t+1}^i) \cap C_{\mathcal{G}}(x_t^j, x_{t+1}^j) = \emptyset, \quad \forall i \in \{1, ..., n\}, \; \forall j \neq i \\ &\qquad \mathbf{o}_i \models \phi_i(\boldsymbol{\tau}^i), \quad \forall i \in \{1, ..., n\}. \end{aligned} \tag{3.5}$$

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

## 3.3 Product Automaton

The multi-agent problem defined in (3.5) 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 4.** *(Weighted Product Automaton) Let $\mathcal{T} = (X, x_0, \Delta, w, AP, l)$ be a transition system and $\mathcal{A}_\infty = (S_{\mathcal{A}_\infty}, s_0, \Sigma, \delta, F_{\mathcal{A}_\infty})$ 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 $\phi$ 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 projection of a path $\mathbf{p} = (x, s)(x', s') \ldots$ onto $\mathcal{T}$ is given by $\gamma_\mathcal{T}(\mathbf{p}) = \mathbf{x} = x\, x' \ldots$.

The construction of a product automaton is shown by example in Fig. 3.3, where it is clear that both the feasible movement on the transition system and TWTL specification are encoded in $\mathcal{P}$. Note that in Fig. 3.3 (c) the additional accepting states due to the self-transition on the dFSA are not shown for clarity.

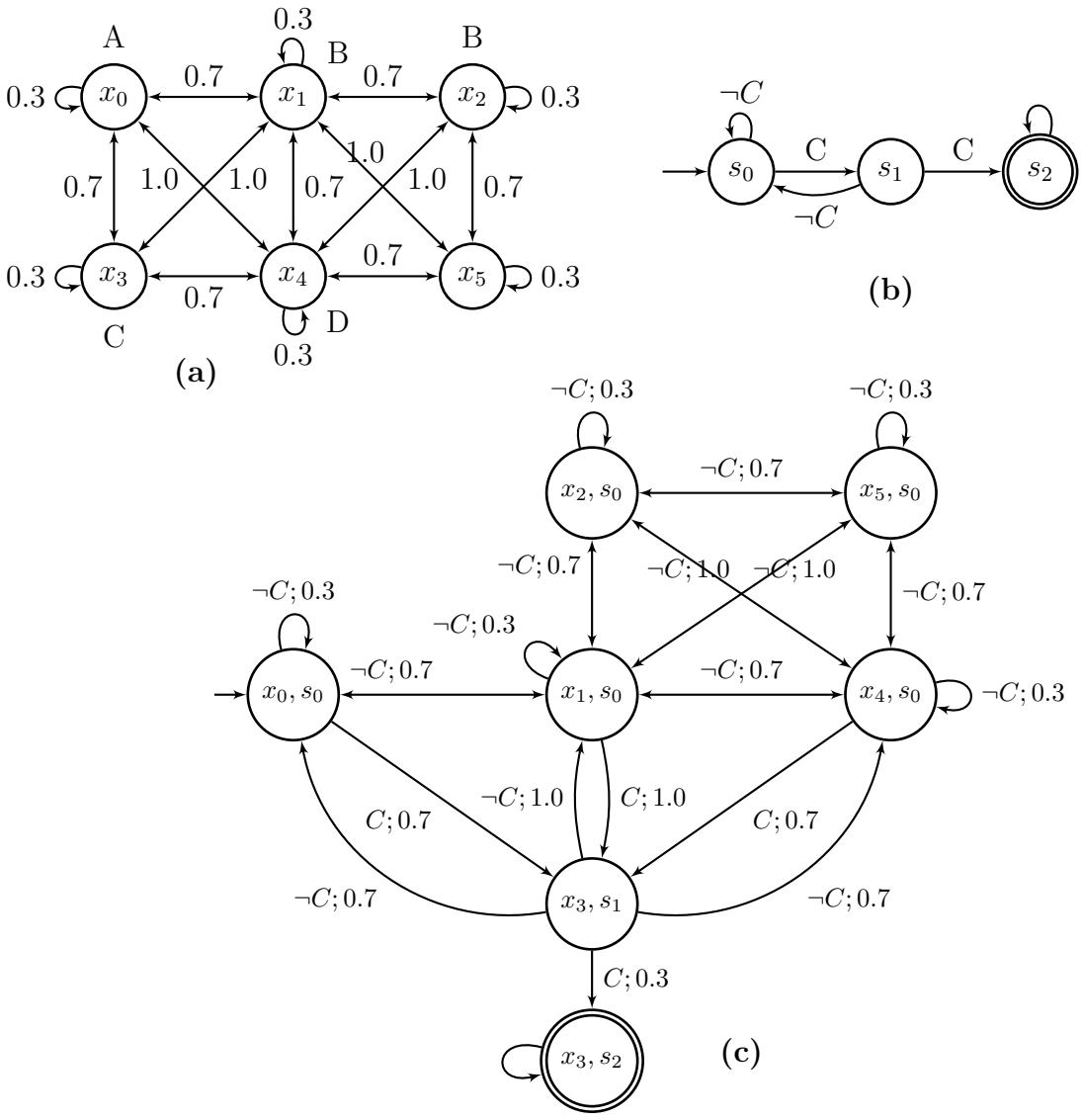


Figure 3.3: (a) The dWTS, $\mathcal{T}$, shown previously in Fig. 3.1. (b) The annotated automaton, $\mathcal{A}_\infty$, corresponding to $\phi = [H^1 C]^{[0,*]}$. (c) Shows the corresponding (weighted) product automaton, $\mathcal{P} = \mathcal{T} \times \mathcal{A}_\infty$.

## 3.4 Centralized Solution

The centralized solution of (3.5) requires construction of an aggregated product automaton $\mathcal{P}_{full} = \mathcal{T}_{full} \times \mathcal{A}_{\infty,full}$ where $\mathcal{T}_{full} = \mathcal{T}_1 \times \mathcal{T}_2 \times \ldots \times \mathcal{T}_n$, and $\mathcal{A}_{\infty,full} =$

$\mathcal{A}_{\infty,1} \times \mathcal{A}_{\infty,2} \times \ldots \times \mathcal{A}_{\infty,n}$. Such an aggregated product automaton captures all possible agent movements and enables safe paths to be found that satisfy the TWTL specifications (or their temporal relaxations). While such an approach can result in optimal agent trajectories, the complexity of constructing $\mathcal{P}_{full}$ grows exponentially as the number of agents $n$ increases. Hence, the proposed algorithm is a distributed approximate solution to Problem 1 which constructs the individual product automaton of each agent and solves a planning problem over individual product automata by using local neighborhood path information.

# Chapter 4

# Receding Horizon Safe Path Planning with TWTL Satisfaction

## 4.1 Solution Approach

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 $\phi_i$), and 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 projection function $\gamma_{\mathcal{T}}(\mathbf{p})$). 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 5.** *(Energy Function) The energy function on the states of a product automaton $\mathcal{P}$ is defined similar to [68] as,*

$$E(p,\mathcal{P}) = \begin{cases} \min\limits_{p'\in F_{\mathcal{P}}} d\big(p,p'\big), & \text{if} \quad p \notin F_{\mathcal{P}} \\ 0, & \text{if} \quad p \in F_{\mathcal{P}} \end{cases} \tag{4.1}$$

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" towards 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 a (strongly) connected graph topology for $\mathcal{P}$. Note that the energy function, $E$, differs for each agent since each agent has a different product automaton, $\mathcal{P}_i$, with different accepting states, $F_{\mathcal{P}_i}$. Fig. 4.1 shows the energy function defined on states of $\mathcal{P}$.

### 4.1.1 Priority Ordering and Deadlock

**Definition 6.** *(Agent Priorities) Each agent $i \in \{1,\dots,n\}$ has a time-varying priority determined by $\pi_t^i = (i,\eta_t^i)$, where*

$$\eta_t^i = \begin{cases} \dfrac{1}{E(p_t^i,\mathcal{P}_i)}, & \text{if } p_t^i \notin F_{\mathcal{P}_i}, \\ 0, & \text{otherwise.} \end{cases} \tag{4.2}$$

*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 $\Pi_t$ denotes the sequence of agent ID numbers sorted based on their priorities from the highest to the lowest.*

Going forward, the time index of $\Pi_t$ is omitted 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

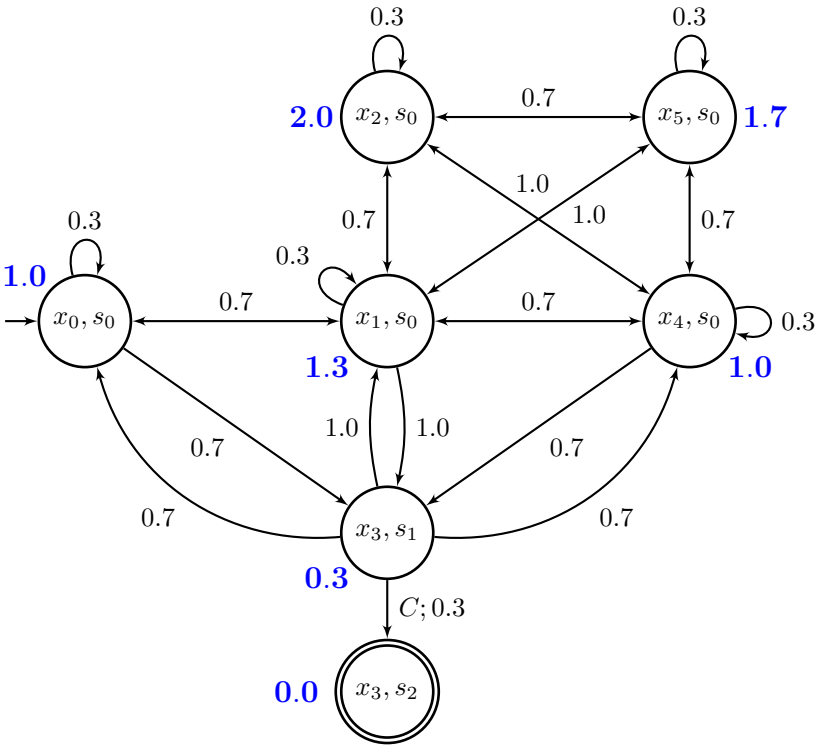


Figure 4.1: Energy function over the states of $\mathcal{P}$ (from Fig. 3.3). The energy of each state computed using (4.1) shown in **blue**, and the weight of each transition is shown.

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 the proposed planning approach, each agent $i$ yields to the other agents with higher priority by avoiding transitions in conflict with their transitions, i.e.,

$$(x_t^i, x_{t+1}^i) \notin C_{\mathcal{G}}(x_t^j, x_{t+1}^j),\ \forall j : \pi_t^j > \pi_t^i. \tag{4.3}$$

Since the conflicts are symmetric as per (3.4), (4.3) is equivalent to the safety constraint in (3.5). An agent $i$ is said to be in deadlock if it has no feasible transitions $(x_t^i, x_{t+1}^i)$ to satisfy (4.3). Resolving such deadlocks requires re-planning for some higher priority agents as will be explained in the next section.

**Definition 7.** *(Deadlock) Consider an agent $i$, where $\Delta_{\mathcal{P}_i,t} \subseteq \Delta_{\mathcal{P}_i}$ denotes the set of feasible transitions over $\mathcal{P}_i$ at time $t$. If $\Delta_{\mathcal{P}_i,t} = \emptyset$ (i.e., there are no feasible transitions), then agent $i$ is said to be in deadlock.*

### 4.1.2 Local Neighborhood and Information Sharing

The term $N_i^{2H} \subseteq N$ is used to denote the set of all agents within $2H$ hops of agent $i$'s current position and $\overline{N}_i \subseteq N$ denotes the set of agents that may indirectly affect agent $i$'s current plan as per (3.1). 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 $\overline{N}_i$ via multi-hop communications (messages relayed by intermediate agents). In the 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 $\pi_t^j$, and update indicator $U_{flag}^j$ of each agent $j \in \overline{N}_i$. The update indicator is required for a proper path update protocol. Furthermore, when agent $i$ is in deadlock, all agents in $\overline{N}_i$ share their current state $p_t^j \in S_{\mathcal{P}_j}$, and all higher priority agents $\overline{HP}_i \subseteq \overline{N}_i$ share their desired next state $p_{t+1}^j \in S_{\mathcal{P}_j}$ with agent $i$. The need for this information is discussed in further detail in the following section.

## 4.2 Algorithm Descriptions

### 4.2.1 Outline

The high-level outline of the proposed method is shown in Fig. 4.2. Note that the proposed algorithm shown is for a single agent and runs independently (for each agent) in a distributed manner.

Prior to execution, the dFSA is constructed from the agent's respective TWTL formula $\phi_i$, the agent's product automaton $\mathcal{P}_i$ is generated, and the energy of each

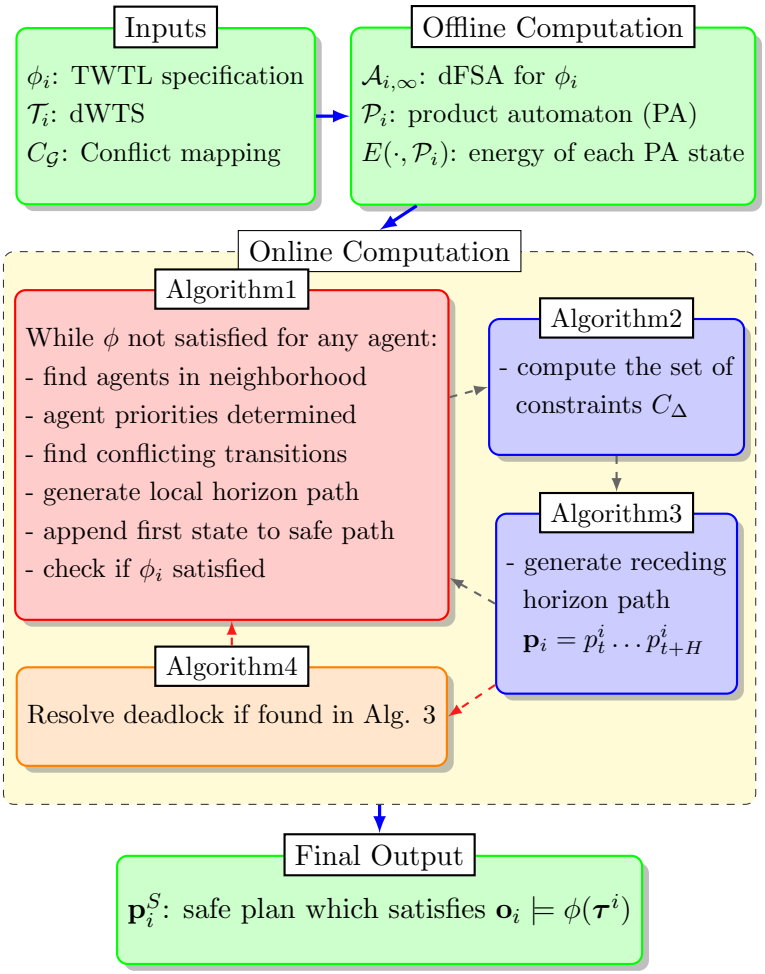


Figure 4.2: Outline of the proposed algorithm.

state in the product automaton $E(\cdot, \mathcal{P}_i)$ is computed.

A detailed analysis of Algorithms 1-3, which run online at every time step, will first be presented. Then Algorithm 4, which is only performed if deadlock occurs, will be discussed.

### 4.2.2 Algorithm 1

Algorithm 1 takes in the product automaton, $\mathcal{P}_i$, energy of the product automaton states, $E(\cdot, \mathcal{P}_i)$, and conflict mapping $C_\mathcal{G}$ to produce a safe path that satisfies the

agent's respective TWTL specification. 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).

In Algorithm 1, agent $i$ first computes its priority based on Def. 6. Then, it obtains the priorities and current positions from all other agents in $\overline{N}_i$ (line 5). Next, the priority ordering for all $\overline{N}_i$ agents is computed using Def. 6 (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 Def. 3. Line 16 ensures the agent has an $H$-hop path since Alg. 3 may result in varied path lengths (discussed later).

**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:** $p_t^i = \left(x_t^i, s_t^i\right)$ 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 $\pi_t^i$ via Def. 6 ;

5 Obtain $\pi^k$ and $x_t^k \in X$, from all agents $k \in \overline{N}_i$ ;

6 Compute priority ordering $\overline{\Pi}$ for all agents $k \in \overline{N}_i$ ;

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**$\left(HP_i, \mathbf{p}_j | \forall j \in HP_i, C_{\mathcal{G}}, \mathcal{P}_i\right)$

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 $k \in \overline{N}_i$, and $p_{t+1}^k$ from all $k \in \overline{HP}_i$ agents;

19 $P_{t+1} \leftarrow$ **Alg4**$(p_t^k$ and $p_{t+1}^k \forall k \in \overline{N}_i, \overline{\Pi}, C_{\mathcal{G}}, \mathcal{G})$;

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 $k \in \overline{N}_i$;

25 **while** $U_{flag}^k$ *is false for any agent* $k \in \overline{N}_i$ **do**

26 Obtain $U_{flag}^k$ from agents $k \in \overline{N}_i$;

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 $p_{t+1}^i$; $U_{flag}^i = false$; $t = t + 1$;

If the agent is in deadlock (determined by Alg. 3), then the current state is obtained from all agents in $\overline{N}_i$ and the desired next state is obtained from all higher priority agents in $\overline{HP}_i \subseteq \overline{N}_i$ 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 $\overline{N}_i$ (line 24). The loop in lines 25-28 ensures synchronous movement amongst all agents in $\overline{N}_i$ 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 $p^i_{t+1}$ of the generated path is executed and the time is updated (line 29).

### 4.2.3 Algorithm 2

In Algorithm 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, $C_\Delta$, enforces that transitions which may result in conflict, defined by the mapping $C_\mathcal{G}$ (see Assumption 1), are not traversed (lines 5-6). 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 7). Lines 8-9 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 Sec. 5.1-Thm. 1.

**Algorithm 2:** Generate Conflicting Transitions for Agent $i$

**Input:** $HP_i, C_{\mathcal{G}}$, $\mathcal{P}_i$ - Agents with higher priority than 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:** $C_\Delta$ - 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 **for** $h = 1 : H$ **do**

4  **for** *each agent* $j$ *in* $HP_i$ **do**

5  **for** *each transition* $(x, x') \in C_{\mathcal{G}}(x_{h-1}^j, x_h^j)$ **do**

6  $C_{\Delta,h} = C_{\Delta,h} \cup \{(x, x')\}$

7  $C_\Delta = C_\Delta \cup C_{\Delta,h}$

8 **if** $HP_i == \emptyset$ **then**

9  $C_\Delta = \{(p, p') \in \Delta_{\mathcal{P}_i} | E(p', \mathcal{P}_i) > E(p, \mathcal{P}_i)\}$

10 **return** $C_\Delta$

### 4.2.4 Algorithm 3

Algorithm 3 essentially generates the set of all feasible (conflict-free) states of $\mathcal{P}_i$ that can be reached in $H$-hops. An example of the procedure for Alg. 3 is shown in Fig. 4.3. 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).

**Algorithm 3:** Receding Horizon Plan for Agent $i$

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

**Note:** $C_{\Delta,h} \subseteq C_\Delta$ is the conflict set of transitions at hop $h$; $V_t(p_h^i)$ is the state cost at hop $h$ used in DP;
**for** $h = 1 : H$ **do**
  Generate $S_{\mathcal{P}_i,h}, \Delta_{\mathcal{P}_i,h}$, reachable states and transitions;
  $S_{\mathcal{P}_i,h} = S_{\mathcal{P}_i,h} \setminus \{(x', s) \in S_{\mathcal{P}_i,h} |\ (x, x') \in C_{\Delta,h}\}$;
  $\Delta_{\mathcal{P}_i,h} = \Delta_{\mathcal{P}_i,h} \setminus \{\big((x, s), (x', s')\big) \in \Delta_{\mathcal{P}_i,h} |\ (x, x') \in C_{\Delta,h}\}$;
  **if** $\Delta_{\mathcal{P}_i,h} = \emptyset$ **then**
    **if** $h == 1$ **then**
      $U_{flag}^i = false$ and $D_{flag} = true$
      **return** $p_t^i$, $U_{flag}^i$, $D_{flag}$
    **else**
      $h = h - 1$;
      exit the for loop;
  **for** *each state* $p \in S_{\mathcal{P}_i,h}$ **do**
    **if** $p \in F_{\mathcal{P}_i}$ **then**
      exit both for loops;
$H_{new} = h$;
$target\_state = \underset{p \in S_{\mathcal{P}_i,H_{new}}}{} E(p, \mathcal{P}_i)$ ;
**if** $H_{new} == 1$ **then**
  $\mathbf{p}_i = p_t^i, target\_state$
**else**
  Initialize: $p_{H_{new}}^i = target\_state$; $V(p_{H_{new}}^i) = 0$ ;
  **for** $h = H_{new} - 1 : 0$ **do**
    **for** *each state* $p_h^i \in S_{\mathcal{P}_i,h}$ **do**
      $V(p_h^i) = \infty$;
      **for** *each transition* $(p_h^i, p_{h+1}^i) \in \Delta_{\mathcal{P}_i,h}$ **do**
        $V(p_h^i) = \min\big(V(p_h^i), V(p_{h+1}^i) + w_p(p_h^i, p_{h+1}^i)\big)$
  Initialize: $\mathbf{p}_i = p_t^i$;
  **for** $h = 1 : H_{new} - 1$ **do**
    $p_{t+h}^i = \underset{p_h^i \ s.t.\ (p_{h-1}^i, p_h^i) \in \Delta_{\mathcal{P}_i,h}}{} V(p) + w_p(p_{h-1}^i, p_h^i)$;
    $\mathbf{p}_i = \mathbf{p}_i, p_{t+h}^i$
  $\mathbf{p}_i = \mathbf{p}_i, target\_state$;
$U_{flag}^i = true$ and $D_{flag} = false$
**return** $\mathbf{p}_i, U_{flag}^i, D_{flag}$

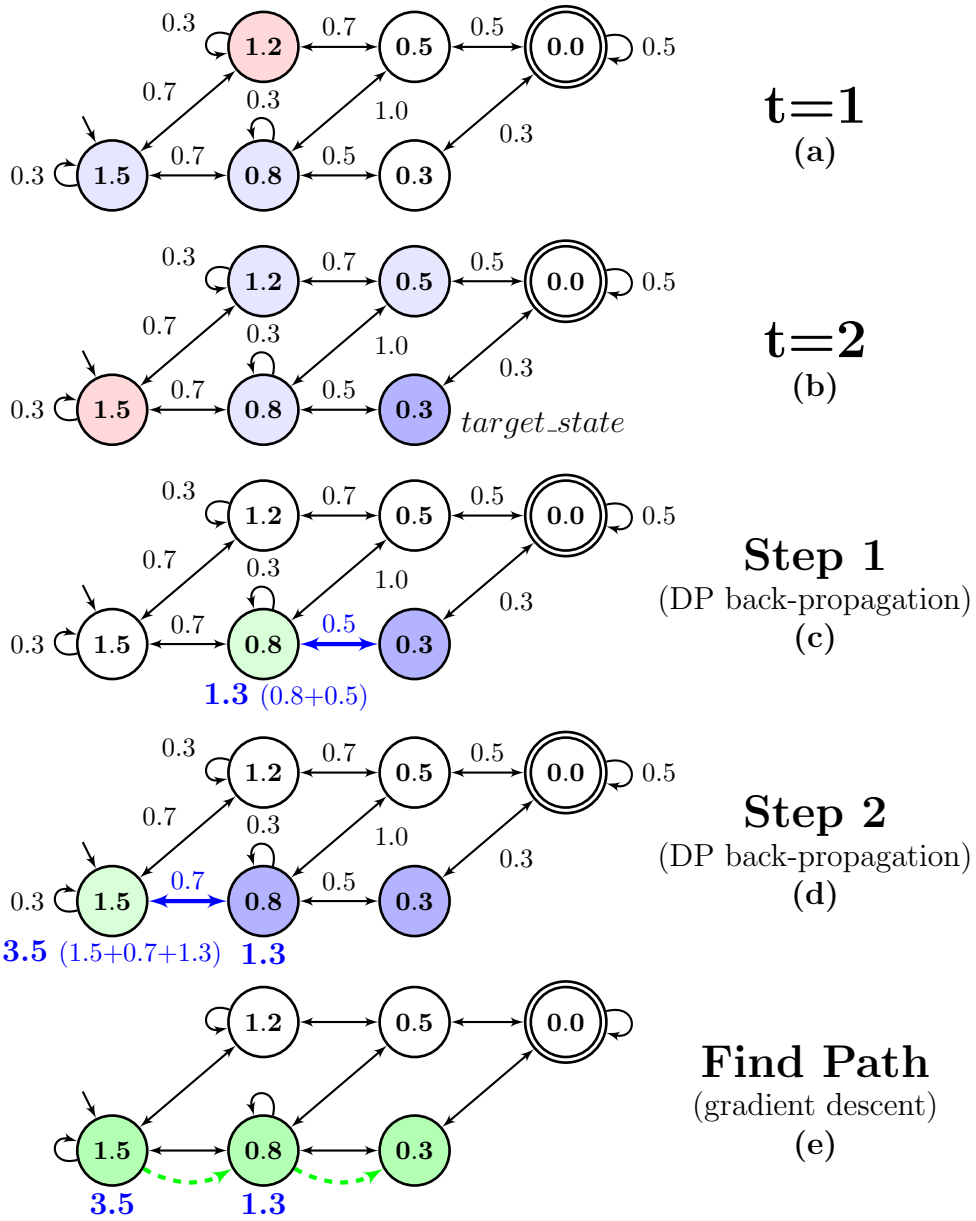


Figure 4.3: Illustration of Alg. 3 over some $\mathcal{P}$, where $H = 2$ hops. The energy of each state $p \in S_{\mathcal{P}}$ is shown in the node, and transition weights shown on edges. Light gray nodes in (a) and (b) represent states that can be reached at that time-step (Alg. 3 lines 2-15), and red nodes represent conflict states at that time-step. The target state is found in (b) (Alg. 3 line 17), then back-propagation is performed in (c) and (d) (Alg. 3 lines 22-26). Finally, the lowest cost path is generated in (e) (Alg. 3 lines 27-31).

Instead of generating a sub-graph of all states that can be reached from $p_t^i$ in $H$ hops (i.e., $\mathcal{N}_i^H$), 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 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 DP algorithm to generate a "min-cost" path since a DP path generation algorithm is not suited for cyclic graphs ([69], 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 lowest 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). In lines 22-26, the 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 $\mathcal{T}_i$. While back traversing, the minimum path cost to reach that state is saved in $V(p_h^i)$ as is typical in any DP algorithm (see [69], Chapter 11). In further detail, the cost to reach a state (line 26) is given as the minimum summation of the prior cost to reach that state, $V_h(p_{h+1}^i)$, and the transition cost, $w_p(p_h^i, p_{h+1}^i)$.

Lines 28-31 generate the minimum cost path over $\mathcal{P}_i$ from $p_t^i$ to $target\_state$ based on the previous computations of $V(p)$. The state associated with the minimum

cost is chosen while ensuring a transition exists from the previous state (line 29). This state is then appended to the safe path (line 30). Lastly, the $target\_state$ is appended to the path (line 30) since a transition is known to exist between it and the previous state in the path. Finally, the update flag is set to *true*, deadlock flag to *false* (line 32), then both flags and the conflict-free path are returned to Alg. 1.

### 4.2.5 Algorithm 4

Algorithm 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 $\overline{N}_i$ required for deadlock resolution. Agent $i$ stays stationary (i.e., $p^i_{t+1} = p^i_t$), if a higher priority agent (let's 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 $\overline{N}_i \subseteq \overline{N}_i$ (let's 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", , which is similar to a message passing strategy (e.g., **??**), 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 $\overline{N}_i$, 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 Sec. 5.1.2.

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 $\overline{N}_i$ (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). 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 $p_t^m$ (i.e., $p_{t+1}^m = (\mathbf{x}(j+1), s) \in S_{\mathcal{P}_m} |\ \mathbf{x}(j+1) \in \mathcal{N}_m^1$) (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.

Multiple calls to Alg. 4 (depending on priority ordering) may be required to resolve deadlock for the system. An illustration of deadlock resolution using Alg. 4 is shown in Fig. 4.4. Assumptions and requirements to ensure deadlocks can always be resolved are discussed in Sec. 5.1.1.

**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

**Note:** $X_t$, $X_{t+1}$, and $P_{t+1}$ are in order of highest to lowest priority. Recall $p = (x, s)$.

**Initialization:** $X_t = \{x_t^{\overline{\Pi}(1)}, \ldots, x_t^{\overline{\Pi}(|\overline{N}_i|)}\}$; $X_{t+1} = \{x_{t+1}^{\overline{\Pi}(1)}, \ldots, x_{t+1}^{\overline{\Pi}(|\overline{HP}_i|)}\}$;

**Initialization:** $P_{t+1} = \emptyset$; $P_{flag} = false$; $x_D = x_t^i$;

**while** $\{x_D\} \neq \emptyset$ **do**

  **for** $k = 1 : |X_{t+1}|$ **do**

    **if** $X_{t+1}(k) == x_D$ **then**

      **if** *ID of* $X_{t+1}(k)$ *belongs to* $\overline{\Pi}(1)$ **then**

        $P_{flag} = true$;

        exit while loop;

      **else**

        Get $\overline{\Pi}(\cdot)$ corresponding to $k$;

        $P_{t+1} = P_{t+1} \cup p_t^{\overline{\Pi}(\cdot)}$;

        $X_{t+1} = X_{t+1} \setminus \{x_D\}$;

        $x_D = X_t(\overline{\Pi}(\cdot))$;

        jump to line 4;

  $\{x_D\} = \emptyset$

**if** $P_{flag} == true$ **then**

  $P_{t+1} = \{p_{t+1}^{\overline{\Pi}(1)}, p_t^{\overline{\Pi}(2)}, \ldots, p_t^{\overline{\Pi}(|\overline{N}_i|)}\}$;

  $\mathcal{G}' = \Big(X, \Delta \setminus C_{\mathcal{G}}(x_t^{\overline{\Pi}(1)}, x_D), w)\Big)$;

  $x_T = \underset{x' \in X \setminus X_t}{} d(x_D, x')$;

  $\mathbf{x} = Dijkstra\big(x_D, x_T, \mathcal{G}'\big)$;

  **for** $j = 1 : |\mathbf{x}| - 1$ **do**

    **if** $\mathbf{x}(j) \in X_t$ **then**

      Denote $m$ as the agent ID occupying $\mathbf{x}(j)$;

      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;

      Find index $l$ in $\overline{\Pi}$ corresponding to agent $m$;

      $P_{t+1}(l) = p_{t+1}^m$;

**return** $P_{t+1}$

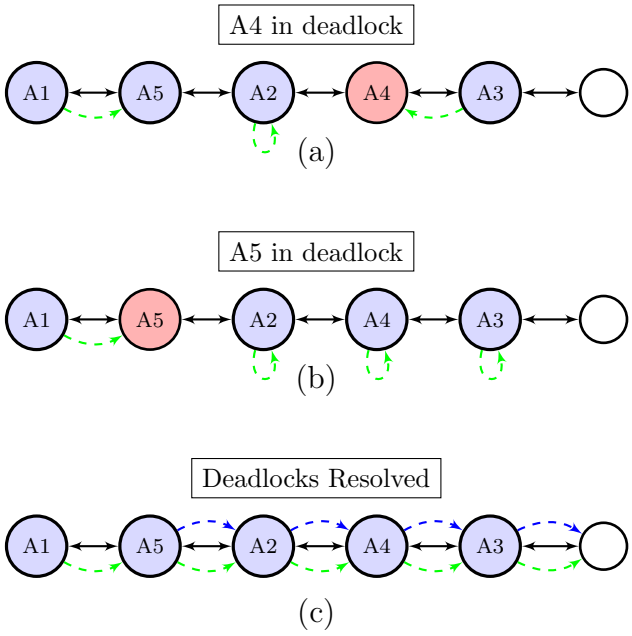


Figure 4.4: Illustration of deadlock resolution using Alg. 4 for priority ordering $\Pi = \{A1, A2, A3, A4, A5\}$. Note, Alg. 4 is called twice due to the given priority ordering. Black arrows indicate the environment transitions, $X$, an agent's next desired transition is shown by a green dashed arrow, 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.

# Chapter 5

# Theory and Complexity

## 5.1 Theoretical Results

This chapter presents theoretical guarantees that an agent $i$ always finds a safe path over the environment graph $\mathcal{G}$ by executing the proposed algorithm. Moreover, it will be shown that the resulting path satisfies either the original TWTL formula or a finite relaxation of it, $\phi_i(\boldsymbol{\tau}^i)$.

### 5.1.1 Safety

**Lemma 1.** *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*

$$\mathcal{G}' = (X, \Delta \setminus C_{\mathcal{G}}(x, x'), w) \tag{5.1}$$

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

$$(x_i x_{i+1}) \notin C_{\mathcal{G}}(x_j, x_{j+1}),\ \forall x_i, x_j \in \mathbf{x}. \tag{5.2}$$

*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*

$$(x^i_t, x^i_{t+1}) \notin C_{\mathcal{G}}(x^j_t, x^j_{t+1}),\ \forall i \neq j \in \{1, ..., n\}. \tag{5.3}$$

.

*Proof.* Since the conflicts are symmetric as per (3.4), (5.3) is equivalent to (4.3). Accordingly, it is first shown 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 (4.3). 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). It will be shown that both of these two cases lead to contradiction.

*Case 1*: Let $p^i_{t+1}$ be determined by agent $i$. Due to (3.2), 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 $C_\Delta$ of agent $i$ (Alg. 2, lines 5-6). Since Alg. 3 generates a path $\mathbf{p}_i$ in sub-automaton $\mathcal{P}'_i \subseteq \mathcal{P}_i$ that is pruned of $C_\Delta$ (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 (3.3), 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 $\overline{N}_i$, 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 (3.3), $i$ 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 $\bar{N}_i$, it is known 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 (4.3).

Next, it will be shown 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 $k \in \overline{N}_i$. 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, a contradiction is reached. 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 (5.2).

□

### 5.1.2 Finite Relaxation

**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* $\mathbf{x} = x_0 x_1 \dots$ *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.* 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 9) always updates $C_\Delta$ with the transitions that drive agent $i$ to higher energy states in the next time step, i.e., $C_\Delta = \{\big((x^i_t, s^i_t)(x^i_{t+1}, s^i_{t+1})\big) \in \Delta_{\mathcal{P}_i,1} | E(p^i_{t+1}, \mathcal{P}_i) < E(p^i_t, \mathcal{P}_i)\}$. 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 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 $E(p_t^i, \mathcal{P}_i) \neq \infty$ by the pruning of infinite energy states. □

## 5.2 Computational Complexity

This section discusses the computational complexity for a single agent since the presented algorithm is distributed. The bulk of the computation lies in generating the product automaton and computing the energy function, which are both performed offline before execution. The online computation is dependent on the length of the receding horizon updates $H$.

### 5.2.1 Offline Computation

The complexity of generating the weighted product automaton $\mathcal{P}$ is highly dependent on the size of the transition system $\mathcal{T}$ and the size of the TWTL formula used to construct $\mathcal{A}_\infty$, i.e., $|\phi|$, since the size of the product automaton is bounded by $\mathcal{O}\big(|X| \times |\phi| \times 2^{|\phi|+|AP|}\big)$. Again, $|X|$ is the number of states in $\mathcal{T}$, and $|\phi| \times 2^{|\phi|+|AP|}$ is the maximum number of states in $\mathcal{A}_\infty$ created from $\phi$. Note that the size of $\mathcal{A}_\infty$ is independent of the deadlines of the *within* operators $\phi$ [51]. This is computed offline prior to execution.

The reader is referred to Section 4.3 of [68] for further details on the energy function computational complexity but the main result is shown here. The complexity of computing the energy function is $\mathcal{O}(|F_{\mathcal{P}}|^3 + |F_{\mathcal{P}}|^2 + |S_{\mathcal{P}}|^2 \times |F_{\mathcal{P}}|)$ for a single agent. This is quite computationally expensive, but is only computed once offline.

### 5.2.2 Online Computation

The complexity of generating a local horizon path in Alg. 3 using the DP algorithm outlined is highly dependant on the horizon length $H$. The maximum number of transitions at each state of $\mathcal{P}$ is denoted as $|\Delta_{\mathcal{P}}(p)|$. To generate all $H$-transition reachable states and transitions is $\mathcal{O}\big(|\Delta_{\mathcal{P}}(p)|^H\big)$ and to search the generated states is $\mathcal{O}\big(|\Delta_{\mathcal{P}}(p)|^H\big)$. Both of these computations are upper bounded by the size of the product automaton $|S_{\mathcal{P}}|$. The DP path generation complexity is similar to that of a depth-first search (conventionally given as $\mathcal{O}(V + E)$). The complexity is roughly $\mathcal{O}\big(|\Delta_{\mathcal{P}}(p)|^{H-1}\big)$ for the recursive back propagation (with upper bound $|\Delta_{\mathcal{P}}|$), and to choose the optimal path is $\mathcal{O}\big(H \cdot |\Delta_{\mathcal{P}}(p)|\big)$ (with upper bound $|S_{\mathcal{P}}|$). For the overall algorithm this gives worst case time complexity of $\mathcal{O}\big(2 \cdot |\Delta_{\mathcal{P}}(p)|^H + |\Delta_{\mathcal{P}}(p)|^{H-1} + H \cdot |\Delta_{\mathcal{P}}(p)|\big)$.

# Chapter 6

# Simulations

The code base for the simulations is derived from the PyTWTL[1] package which handles the construction of $\mathcal{A}_\infty$ corresponding to a given TWTL specification, $\phi$, and the creation of the product automaton $\mathcal{P}$ given a particular transition system $\mathcal{T}$. The algorithms presented in this thesis which are integrated into the PyTWTL framework as well as the videos corresponding to the simulations can be found at `https://github.com/pete9936/pyTWTL_ObsAvoid`.

Four different scenarios are considered. In *scenario-1*, 5 agents are considered in the environment shown in Fig. 6.1. In *scenario-2*, up to 10 agents are considered in a much larger environment given by Fig. 6.3. *Scenario-3* considers 3 agents in a narrow corridor (Fig. 6.4), and *scenario-4* considers 2 agents with more complex specifications (Fig. 6.7). In all cases each agent is given a different TWTL specification defined over the environment, and an agent can stay put or move to any neighboring cell (non-obstacle) in one time-step. The radius ($r' = r + \epsilon$) for each agent is chosen as 0.1 m. (for the conflict mapping $C_\mathcal{G}$, see Appendix B), and each discretized region of the environment is considered as $0.4 \times 0.4 \times 0.4$ m.

[1] `hyness.bu.edu/twtl`

## 6.1 Scenario 1

The following temporally relaxed TWTL formulae are considered for *scenario-1*:

$$\phi_1 = [H^2 B]^{[0,5+\tau_1]} \cdot [H^1 F]^{[0,4+\tau_2]} \cdot [H^1 Base1]^{[0,4+\tau_3]}$$
$$\phi_2 = [H^2 A]^{[0,4+\tau_1]} \cdot [H^1 E]^{[0,3+\tau_2]} \cdot [H^1 Base2]^{[0,3+\tau_3]}$$
$$\phi_3 = [H^1 A]^{[0,4+\tau_1]} \cdot [H^2 E]^{[0,3+\tau_2]} \cdot [H^2 Base3]^{[0,3+\tau_3]}$$
$$\phi_4 = [H^1 C]^{[0,4+\tau_1]} \cdot [H^1 D]^{[0,4+\tau_2]} \cdot [H^1 Base4]^{[0,3+\tau_3]}$$
$$\phi_5 = [H^1 C]^{[0,4+\tau_1]} \cdot [H^1 D]^{[0,4+\tau_2]} \cdot [H^1 Base5]^{[0,3+\tau_3]}$$

In plain English, $\phi_1$ for example, can be interpreted as: "Service B for 2 time units within [0,6], and after this, service A for 2 time units within [0,5]." Where $\boldsymbol{\tau} = (\tau_1, \tau_2, \tau_3) \in \mathbb{Z}^3$ is the temporal relaxation vector which captures all possible relaxations of the time windows in $\phi_1$.

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. These were kept relatively simple and of similar form to illustrate how the number of agents and horizon length impacts performance and computation time (shown in Tables 6.2 and 6.3). In practice these task specifications can be made far more complex due to the richness of the TWTL language.

An agents nominal trajectory based on $\mathcal{P}_1, \ldots, \mathcal{P}_5$ is computed, where each agent computes its nominal plan over $\mathcal{P}_i$ (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 6.1).

The temporal relaxation for both the nominal and collision-free policies for each agent are given in Table 6.1. Recall that negative relaxation of the TWTL formulae implies that the formulae are satisfied within a stricter time window than was

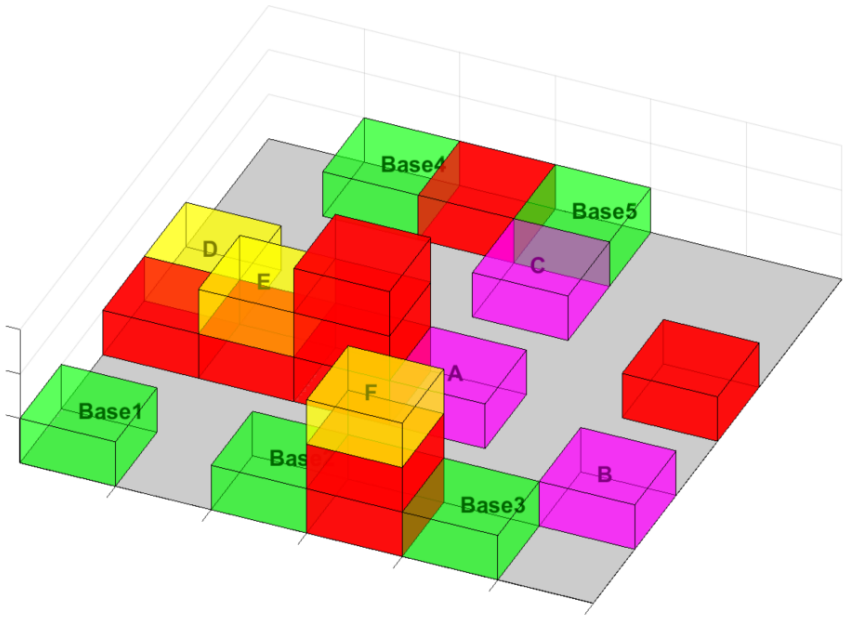


Figure 6.1: *Scenario-1*: The 3D discretized environment shared by 5 agents. Initial positions given by the green nodes, obstacles are red, pick-up regions (A, B, C) are magenta, and drop-off regions (D, E, F) are shown in yellow.

originally allotted. For the given scenario, both Agent's 3 and 5 require additional temporal relaxation as is shown in Table 6.1. This is due to Agent 3 waiting while Agent 2 satisfies the "service A for 2 time units" portion of its formula since Agent 2 is given higher priority. Similarly, Agent 5 yields to Agent 4 while it satisfies the "service C for 1 time unit" portion of its formula. This priority ordering is evident in Fig. 6.2. Notice that while Agent's 3 and 5 are waiting, their energy is not decreasing.

| | Nominal paths | | | | | Safe paths | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $\mathbf{x_1}$ | $\mathbf{x_2}$ | $\mathbf{x_3}$ | $\mathbf{x_4}$ | $\mathbf{x_5}$ | $\mathbf{x_1}^S$ | $\mathbf{x_2}^S$ | $\mathbf{x_3}^S$ | $\mathbf{x_4}^S$ | $\mathbf{x_5}^S$ |
| $\tau_1$ | +1 | 0 | **0** | -1 | **−2** | +1 | 0 | **+3** | -1 | **+1** |
| $\tau_2$ | -1 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 |
| $\tau_3$ | -1 | -1 | 0 | -1 | 0 | -1 | -1 | 0 | -1 | 0 |

Table 6.1: Temporal Relaxation ($\tau$) of paths.

For the $6 \times 6 \times 3$ environment shown in Fig. 6.1, each respective agent has a transition system $\mathcal{T}_i$ of $(102; 1594)$ states and transitions, and similarly sized product automaton $\mathcal{P}_i$ of about $(300; 4500)$ states and transitions. The algorithm's offline initialization (generating $\mathcal{A}_{\infty,1...5}$, $\mathcal{P}_{1...5}$, $E_{1...5}$) takes 6.51 seconds. Generating the collision-free paths (the online portion of our algorithm) with a horizon length $H = 2$ takes 0.37 seconds (s). However, the metric of greater concern is the average time for an individual agent's receding horizon update which is $\approx 7$ milliseconds (ms). Fast enough for real-time execution.

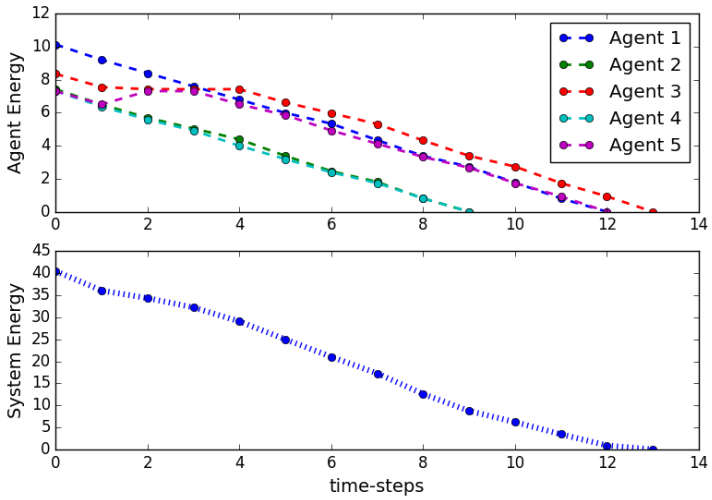


Figure 6.2: *Scenario-1*: The agent energy values at each state along the paths are shown as well as the collective energy of the system. These energy values correspond to each state along the path which leads to satisfaction of each agents' respective TWTL formula.

Referring to Table 6.2, it is seen that the average time for an individual agent's path update is dependent on the number of transitions $H$ as was discussed in Sec. 5.2. Note that by using the DP receding horizon algorithm, execution time growth is approximately linear with respect to $H$.

| | $H = 2$ | $H = 4$ | $H = 6$ |
|---|---|---|---|
| Online Run-time (s) | 0.37 | 0.67 | 1.09 |
| Avg. Iteration time (ms) | 7 | 12 | 20 |

Table 6.2: How Receding Horizon Length Impacts Execution Time

## 6.2 Scenario 2

In order to explore how the number of agents and size of the environment (and therefore size of the product automaton) affects the computation time, *scenario-2* which uses the environment shown in Fig. 6.3 is considered. This scenario considers up to 10 agents assigned TWTL specifications of the same structure as those in the *scenario-1*. A comparison of the computation time with 5 agents as in *scenario-1* is presented, as well as how the number of agents affects execution time.

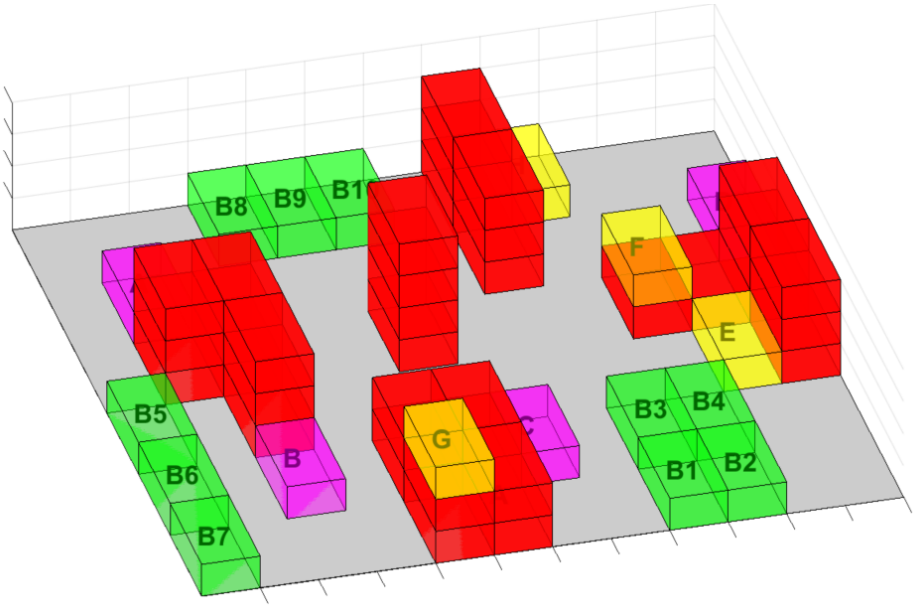


Figure 6.3: *Scenario-2*: 6 x 12 x 4 environment shared by 10 agents. Initial positions given by the green nodes, obstacles are red, pick-up regions (A, B, C, D) are magenta, and drop-off regions (D, E, F, G) are shown in yellow.

For the $6 \times 12 \times 4$ environment shown in Fig. 6.3, each respective agent has

the transition system $\mathcal{T}$ of $(253; 4229)$ states and transitions, and similarly sized product automaton $\mathcal{P}_i$ of about $(750; 12700)$ states and transitions. The offline initialization takes 45.4 seconds. Comparing this scenario to scenario 1, shown in Table 6.3, it can be seen that an agent's path update scales linearly with the size of $\mathcal{P}$ using the presented algorithm.

| | scenario 1 | scenario 2 |
|---|---|---|
| Number of states | 300 | 750 |
| Number of transitions | 4500 | 12700 |
| Online Run-time (s) | 0.37 | 1.80 |
| Avg. Iteration time (ms) | 7 | 24 |

Table 6.3: How Size of $\mathcal{P}$ Impacts Execution Time (*n=5, H=2*)

| | $n = 2$ | $n = 5$ | $n = 10$ |
|---|---|---|---|
| Online Run-time (s) | 0.43 | 1.80 | 5.83 |
| Avg. Iteration time (ms) | 18 | 24 | 35 |

Table 6.4: How Number of Agents Impacts Execution Time (*H=2*)

Table 6.4 shows that the online execution time increases approximately linearly with the number of agents. This is due to the fact that all $n$ agents must update their respective paths at each step before moving to the next step/iteration since the simulations are run on a centralized machine. However, in general the presented algorithm can be run in a distributed manner where this computation would only increase based on the number of agents in the local neighborhood at each step, $|\mathcal{N}_i^{2H}|$. 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 its local neighborhood.

## 6.3 Scenario 3

In this scenario, the following temporally relaxed TWTL formulae are considered:

$$\phi_1 = [H^1 B]^{[0,5+\tau_1]}$$
$$\phi_2 = [H^1 A]^{[0,5+\tau_1]}$$
$$\phi_3 = [H^3 C]^{[0,7+\tau_1]}$$

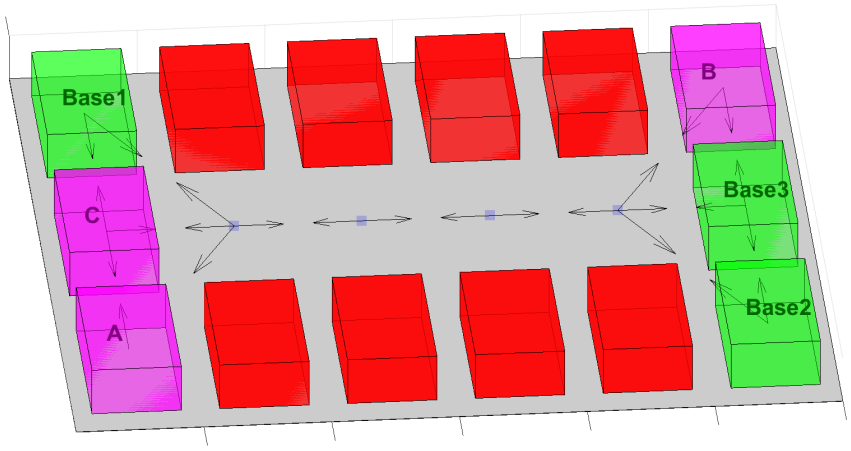


Figure 6.4: *Scenario-3*: A narrow corridor environment shared by 4 agents. Initial positions given by the green nodes and obstacles are shown in red. Three regions of interest (A, B, C) are shown in magenta. Black arrows indicate transitions (note that self-loop transitions are not explicitly shown).

This scenario clearly shows both the collision avoidance and deadlock resolution protocols. The initial agent priority ordering is $\Pi = \{A1, A2, A3\}$, where Agent 1 has the highest priority (and maintains highest priority throughout) and therefore completes its task, $\phi_1$, without yielding to other agents. However, this scenario does encounter deadlock and requires deadlock resolution using Alg. 4. This deadlock resolution protocol (using Alg. 4) is illustrated by Fig. 6.5. Note that the total system energy actually increases between time-steps 2-4, Fig. 6.6, though as

discussed earlier in Sec. 5.1.2, satisfaction of all TWTL specifications in finite-time is guaranteed.

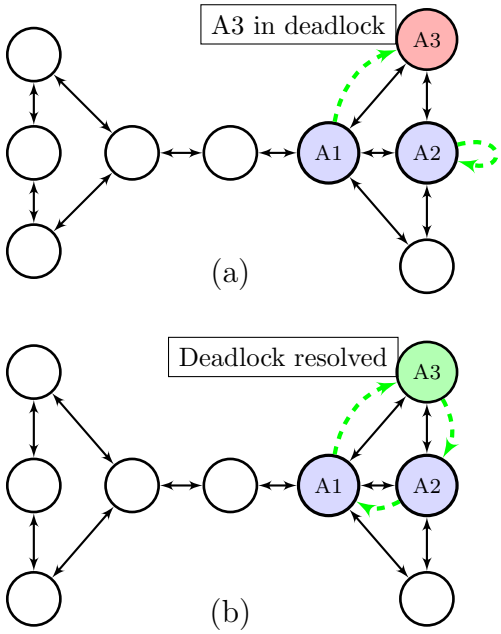


Figure 6.5: Resolving deadlock in *scenario-3* using Alg. 4. Initial priority ordering is $\Pi = \{A1, A2, A3\}$. Black arrows indicate the environment transitions (self-transitions not shown), and an agent's next desired transition is shown by a green dashed arrow. (a) $A4$ has no feasible transitions (after $A1, A2$ compute their next desired transition), and results in deadlock. (b) Using Alg. 4 results in each agent having a feasible next transition, therefore resolving deadlock.

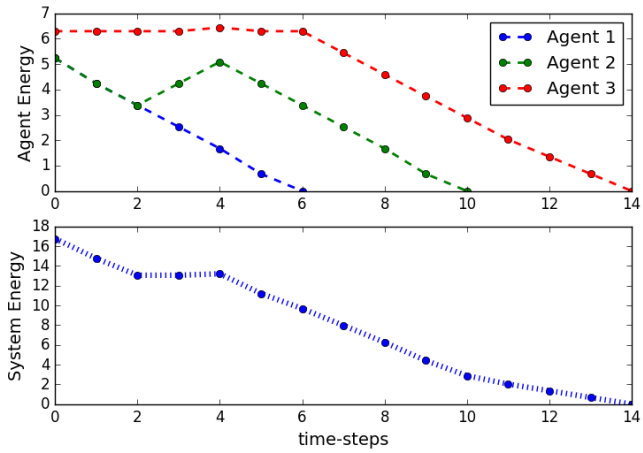


Figure 6.6: *Scenario-3*: The agent energy values and the collective energy of the system at each time-step.

## 6.4 Scenario 4

This scenario illustrates how the algorithm is useful in dealing with more complex specifications. Consider, a pick-up and delivery task for multiple agents. It may be the case that both 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.,[30, 35, 62, 64]) can not account for such a scenario. This scenario is depicted on the environment shown in Fig. 6.7, where the following temporally relaxed TWTL formulae are considered:

$$\begin{aligned} \phi_1 &= [H^1 P1]^{[0,4+\tau_1]} \cdot [H^1(D1 \vee D2 \vee D3)]^{[0,4+\tau_2]} \\ \phi_2 &= [H^1 P2]^{[0,4+\tau_1]} \cdot [H^1(D1 \vee D2 \vee D3)]^{[0,4+\tau_2]} \end{aligned} \tag{6.1}$$

In plain English, $\phi_1$ can be interpreted as: "Service P1 for 1 time units within the relaxed time window $[0, 4+\tau_1]$, then immediately after this, service either D1, D2, or D3 for 1 time unit within the relaxed time window $[0, 4+\tau_2]$."

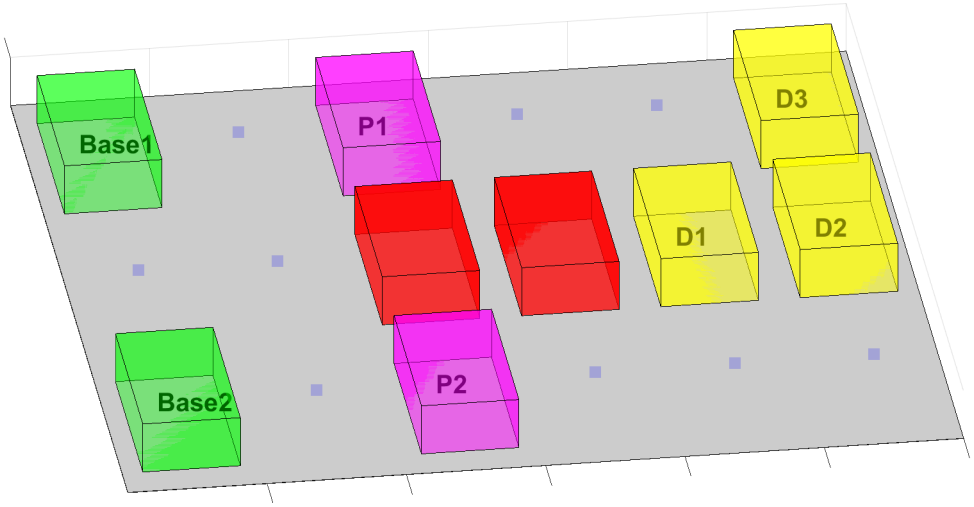


Figure 6.7: *Scenario-4*: 3 x 6 x 1 environment shared by 2 agents. Initial positions given by the green nodes, obstacles are red, pick-up regions (P1, P2) are magenta, and the possible drop-off regions (D1, D2, D3) are shown in yellow.

The TWTL specifications given in (6.1) are compared to the more conventional pick-up and delivery tasks which would not be able to account for alternative delivery locations based on local information obtained during execution. These temporally relaxed TWTL formulae are as follows:

$$\begin{aligned} \phi_1 &= [H^1 P1]^{[0,3+\tau_1]} \cdot [H^1 D1]^{[0,3+\tau_2]} \\ \phi_2 &= [H^1 P2]^{[0,3+\tau_1]} \cdot [H^1 D1]^{[0,3+\tau_2]} \end{aligned} \tag{6.2}$$

The temporal relaxations of the collision-free policies are compared for both (6.1) and (6.2) as well as their nominal policies (the nominal policies for (6.1) and (6.2) are identical) in Table 6.5.

In this scenario, the temporal relaxation of $\tau_2 = +1$ is required for Agent 2 in order to drop off in region $D2$ instead of the nominal $D1$ in order to satisfy $\phi_2$ in (6.1), while a temporal relaxation of $\tau_2 = +2$ is required to satisfy $\phi_2$ in (6.2) since Agent 2 must wait for Agent 1 to finish dropping off in region $D1$

| | Nominal paths | | Safe paths (6.1) | | Safe paths (6.2) | |
|---|---|---|---|---|---|---|
| | $\mathbf{x_1}$ | $\mathbf{x_2}$ | $\mathbf{x_1}^S$ | $\mathbf{x_2}^S$ | $\mathbf{x_1}^S$ | $\mathbf{x_2}^S$ |
| $\tau_\mathbf{1}$ | 0 | 0 | 0 | 0 | 0 | 0 |
| $\tau_\mathbf{2}$ | 0 | **0** | 0 | **+1** | 0 | **+2** |

Table 6.5: Temporal Relaxation ($\tau$) of paths.

to avoid collision. 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 the presented algorithm.

# Chapter 7

# Experiments

The proposed algorithm is verified experimentally on a team of Crazyflies 2.0. This platform is used due to its high level of maneuverability [70] which allows for the safe operation in small densely populated environments. Also, the control architecture is open-source allowing for modifications if necessary. The modified Crazyflie 2.0 platform used in experiments is shown in Fig. 7.1. Note that the only modifications to the physical platform are the additional white frame and `VICON` reflective markers (gray spheres) attached. The additional white frame simply allows for further space to adhere the reflective markers since each Crazyflie must have a unique marker configuration in order to correctly identify it. The interested reader is referred to [71] for an in depth review of the Crazyflie 2.0 and its dynamics.

The experimental scenarios considered are identical to *scenario-1*, *scenario-3*, and *scenario-4* detailed in Chapter 6. Videos of the experiments can be found at `https://github.com/pete9936/pyTWTL_ObsAvoid`. The experiments are conducted in a $3m \times 3m \times 1.5m$ motion-capture space, which emulate the simulation environments of *scenario-1* and *scenario-3* (shown in Fig. 6.1 and Fig. 6.4). The lab environment makes use of a `VICON` camera system with 8 cameras in order to

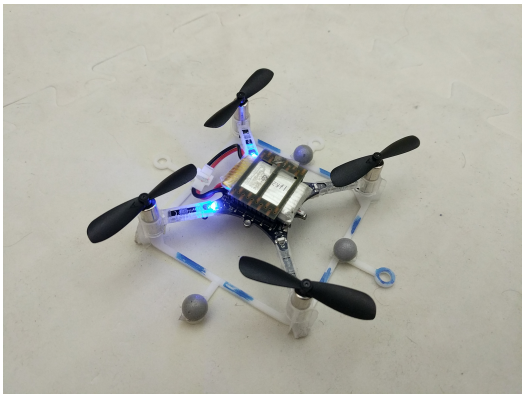

Figure 7.1: The physical platform (Crazyflie 2.0) experiments are performed with.

obtain high quality position and attitude information for the Crazyflies.

## 7.1 Experimental Procedure

### 7.1.1 Architecture

All experiments were carried out on a workstation with 4 cores running Ubuntu 16.04, 4.0GHz CPU, and 32GB of RAM. The Crazyswarm[1] package is used to perform the low-level controls algorithms, communication, and interface of the `Vicon` system with the Crazyflies.

The Crazyswarm package employs a cascaded control architecture where the position control is the outer loop and the attitude control is the inner loop. The outer loop (position) control is performed on the workstation at 120 Hz. This corresponds to the frequency with which the `VICON` system publishes position and attitude information to the workstation. In these experiments the standard `Vicon Tracker` rigid-body motion-capture was used. This object tracker requires a unique marker arrangement for each object tracked, hence the additional white frame previously discussed. Alternatively, a raw point cloud can be obtained from the `VICON` system in order to run an *Iterative Closest Point* (ICP) algorithm [72]

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

which handles identical marker arrangements. However, the ICP method was not employed since unique marker arrangements can easily be created with the relatively few agents considered in these experiments (maximum of 5). The inner loop (attitude) control is performed on-board the Crazyflie 2.0 hardware at 500 Hz. Communication between these layers is handled via the Crazyflie standard radio communication protocol which sends the desired attitude and thrust from the workstation (*Crazyflie Client*) to the quadrotor (at a rate of 120 Hz). The reader is referred to [73] for further details of the Crazyswarm architecture.

### 7.1.2 Trajectory Generation

Quadrotor dynamics are differentially flat in the outputs $\mathbf{y} = (x, y, z, \psi)$, where $\psi$ is the yaw angle in world coordinates [74]. This property is used to generate trajectories which are at least four times differentiable. The trajectories are represented by $7^{th}$-order piecewise polynomials from the current state to some desired state. These piecewise polynomials satisfy given waypoints and continuity constraints described in ([73], Sec. VII-A).

The algorithms presented in Chapter 4 are executed as a set of Python scripts, where the waypoints are published to the Crazyswarm module online from Alg. 1 for each respective agent in a receding horizon manner in order to generate and execute the local trajectories. The waypoints published from Alg. 1 are the Euclidean positions in $\mathbb{R}^3$ of the respective states $x \in X$ of the graph $\mathcal{G} = (X, \Delta, w)$.

### 7.1.3 Platform Considerations

In these experiments a "downwash zone" below each quadrotor is considered, since entering these regions would lead to loss of flight. This is accounted for by considering the agents as cylinders of radius, $r'$, and height, $h > r$, in $\mathbb{R}^3$ instead of spheres of radius $r \in \mathbb{R}^3$ in the conflict mapping $C_{\mathcal{G}}$ described in Appendix B. In (B.1), $\epsilon \geq 0$ is some dilation to the radius to account for imperfect motion/noise, therefore, $r' = r + \epsilon$. A value of $r' = 0.1$ m. was found to be sufficient for

experiments. It was experimentally found that a "downwash zone" of $\approx 0.6$ m. is sufficient for the Crazyflie 2.0 platform, i.e., $h = 0.6$ m. This restriction is not formally considered in the method since this is both an environment and agent-dependent constraint.

# Chapter 8

# Conclusions

An algorithm for generating collision-free paths for a multi-agent system which satisfy individual tasks encoded as TWTL specifications in finite time was introduced. The outlined algorithm takes advantage of the offline computation of each agents product automaton and energy of the respective product automaton states in order to generate safe paths online in an efficient manner. The algorithm presented 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 algorithm can be used in real-time applications and scales well with increasing environment size (or specification length) and number of agents.

While this work has shown a practical algorithm to ensure the satisfaction of complex high-level specifications in a distributed manner that guarantees safety, further work remains to improve upon this algorithm before it may reliably account for real-world applications. Care was taken to design the algorithm in a general manner that allows for further extension of ideas such as uncertainty, asynchronous movement, and accounting for non-broadcasting dynamic obstacles (e.g., humans).

To this end, the following areas of research and extensions are recommended:

- Incorporate a low-level controller to aid in resolving collisions in the case of intersecting transitions.
- Consider heterogeneous teams of agents, and account for asynchronous movement of the agents over the environment.
- Account for uncertainty in the environment in a more rigorous manner. In this work, uncertainty is accounted for in the conflicting transitions mapping $C_{\mathcal{T}}$ by adding some dilation to the agent radius while moving over the environment, i.e., $r + \epsilon$ (see (B.1)). Uncertainty in the agent dynamics, and therefore transitions on the environment $\mathcal{G}$, are sometimes accounted for by using a Markov Decision Process (MDP) of some form. Uncertainty could also be accounted for through the low-level controller implementation as in [64].
- Explore methods to further limit communication between agents in the environment to further decrease the required communication bandwidth.
- Develop a method to account for dynamic obstacles in the environment which do not broadcast/share path information (e.g., a human). This would require the agent to actively monitor the environment, most likely using an on-board camera or similar sensor. When a new obstacle (dynamic or static) is observed, some estimation of its future trajectory/path would be required. This obstacle could then be considered similar to how a higher priority agent in this work is dealt with, where the agent must yield to this obstacle's estimated path.

# References


[1] Jorge Cortes, Sonia Martinez, Timur Karatas, and Francesco Bullo. Coverage control for mobile sensing networks. *IEEE Transactions on robotics and Automation*, 20(2):243–255, 2004.

[2] Mac Schwager, Daniela Rus, and Jean-Jacques Slotine. Decentralized, adaptive coverage control for networked robots. *The International Journal of Robotics Research*, 28(3):357–375, 2009.

[3] A Yasin Yazıcıoğlu, Magnus Egerstedt, and Jeff S Shamma. Communication-free distributed coverage for networked systems. *IEEE Transactions on Control of Network Systems*, 4(3):499–510, 2016.

[4] Nikhil Nigam, Stefan Bieniawski, Ilan Kroo, and John Vian. Control of multiple uavs for persistent surveillance: algorithm and flight test results. *IEEE Transactions on Control Systems Technology*, 20(5):1236–1251, 2011.

[5] Derya Aksaray, A Yasin Yazıcıoğlu, Eric Feron, and Dimitri N Mavris. Message-passing strategy for decentralized connectivity maintenance in multiagent surveillance. *Journal of Guidance, Control, and Dynamics*, 38(11):542–555, 2016.

[6] Sepehr Seyedi, Yasin Yazicioğlu, and Derya Aksaray. Persistent surveillance with energy-constrained uavs and mobile charging stations. *IFAC-PapersOnLine*, 52(20):193–198, 2019.

[7] Soon-Jo Chung, Aditya Avinash Paranjape, Philip Dames, Shaojie Shen, and Vijay Kumar. A survey on aerial swarm robotics. *IEEE Transactions on Robotics*, 34(4):837–855, 2018.

[8] Kwang-Kyo Oh, Myoung-Chul Park, and Hyo-Sung Ahn. A survey of multi-agent formation control. *Automatica*, 53:424–440, 2015.

[9] Xiwang Dong, Bocheng Yu, Zongying Shi, and Yisheng Zhong. Time-varying formation control for unmanned aerial vehicles: Theories and applications. *IEEE Transactions on Control Systems Technology*, 23(1):340–348, 2014.

[10] Raghavendra Bhat, Yasin Yazıcıoğlu, and Derya Aksaray. Distributed path planning for executing cooperative tasks with time windows. *IFAC-PapersOnLine*, 52(20):187–192, 2019.

[11] Yunus Emre Sahin, Petter Nilsson, and Necmiye Ozay. Provably-correct coordination of large collections of agents with counting temporal logic constraints. In *2017 ACM/IEEE 8th International Conference on Cyber-Physical Systems (ICCPS)*, pages 249–258. IEEE, 2017.

[12] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE transactions on robotics*, 25(6):1370–1381, 2009.

[13] Paulo Tabuada and George J Pappas. Linear time logic control of discrete-time linear systems. *IEEE Transactions on Automatic Control*, 51(12):1862–1877, 2006.

[14] Marius Kloetzer and Calin Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, 53(1):287–297, 2008.

[15] Derya Aksaray, Kevin Leahy, and Calin Belta. Distributed multi-agent persistent surveillance under temporal logic constraints. *IFAC-PapersOnLine*, 48(22):174–179, 2015.

[16] Derya Aksaray, Cristian-Ioan Vasile, and Calin Belta. Dynamic routing of energy-aware vehicles with temporal logic constraints. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3141–3146. IEEE, 2016.

[17] Yushan Chen, Xu Chu Ding, Alin Stefanescu, and Calin Belta. Formal approach to the deployment of distributed robotic teams. *IEEE Transactions on Robotics*, 28(1):158–171, 2011.

[18] Ioannis Filippidis, Dimos V Dimarogonas, and Kostas J Kyriakopoulos. Decentralized multi-agent control from local ltl specifications. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 6235–6240. IEEE, 2012.

[19] Eric S Kim, Sadra Sadraddini, Calin Belta, Murat Arcak, and Sanjit A Seshia. Dynamic contracts for distributed temporal logic control of traffic networks. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 3640–3645. IEEE, 2017.

[20] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[21] John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: science and systems*, volume 9, pages 1–10. Citeseer, 2013.

[22] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[23] Michael Erdmann and Tomas Lozano-Perez. On multiple moving objects. *Algorithmica*, 2(1-4):477, 1987.

[24] Andrei Marchidan and Efstathios Bakolas. Collision avoidance for an unmanned aerial vehicle in the presence of static and moving obstacles. *Journal of Guidance, Control, and Dynamics*, 43(1):96–110, 2020.

[25] Dingjiang Zhou and Mac Schwager. Assistive collision avoidance for quadrotor swarm teleoperation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1249–1254. IEEE, 2016.

[26] Aaron D Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *2019 18th European Control Conference (ECC)*, pages 3420–3431. IEEE, 2019.

[27] Li Wang, Aaron D Ames, and Magnus Egerstedt. Safety barrier certificates for collisions-free multirobot systems. *IEEE Transactions on Robotics*, 33(3):661–674, 2017.

[28] Dingjiang Zhou, Zijian Wang, Saptarshi Bandyopadhyay, and Mac Schwager. Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells. *IEEE Robotics and Automation Letters*, 2(2):1047–1054, 2017.

[29] Saptarshi Bandyopadhyay, Soon-Jo Chung, and Fred Y Hadaegh. Probabilistic swarm guidance using optimal transport. In *2014 IEEE Conference on Control Applications (CCA)*, pages 498–505. IEEE, 2014.

[30] Baskın Şenbaşlar, Wolfgang Hönig, and Nora Ayanian. Robust trajectory execution for multi-robot teams using distributed real-time replanning. In *Distributed Autonomous Robotic Systems*, pages 167–181. Springer, 2019.

[31] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

[32] Javier Alonso-Mora, Paul Beardsley, and Roland Siegwart. Cooperative collision avoidance for nonholonomic robots. *IEEE Transactions on Robotics*,

34(2):404–420, 2018.

[33] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 1928–1935. IEEE, 2008.

[34] Javier Alonso-Mora, Jonathan A DeCastro, Vasumathi Raman, Daniela Rus, and Hadas Kress-Gazit. Reactive mission and motion planning with deadlock resolution avoiding dynamic obstacles. *Autonomous Robots*, 42(4):801–824, 2018.

[35] Carlos E Luis and Angela P Schoellig. Trajectory generation for multiagent point-to-point transitions via distributed model predictive control. *IEEE Robotics and Automation Letters*, 4(2):375–382, 2019.

[36] Li Dai, Qun Cao, Yuanqing Xia, and Yulong Gao. Distributed mpc for formation of multi-agent systems with collision avoidance and obstacle avoidance. *Journal of the Franklin Institute*, 354(4):2068–2085, 2017.

[37] Thulasi Mylvaganam, Mario Sassano, and Alessandro Astolfi. A differential game approach to multi-agent collision avoidance. *IEEE Transactions on Automatic Control*, 62(8):4229–4235, 2017.

[38] Thulasi Mylvaganam, Mario Sassano, and Alessandro Astolfi. Constructive $\epsilon$-nash equilibria for nonzero-sum differential games. *IEEE Transactions on Automatic Control*, 60(4):950–965, 2014.

[39] Na Li and Jason R Marden. Designing games for distributed optimization. *IEEE Journal of Selected Topics in Signal Processing*, 7(2):230–242, 2013.

[40] Zijian Wang, Riccardo Spica, and Mac Schwager. Game theoretic motion planning for multi-robot racing. In *Distributed Autonomous Robotic Systems*, pages 225–238. Springer, 2019.

[41] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. *Formal methods for discrete-time dynamical systems*, volume 89. Springer, 2017.

[42] Christel Baier, Joost-Pieter Katoen, et al. *Principles of Model Checking*, volume 26202649. MIT press Cambridge, 2008.

[43] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus. Optimality and Robustness in Multi-Robot Path Planning with Temporal Logic Constraints. *Int. Journal of Robotics Research*, 32(8):889–911, 2013.

[44] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding Horizon Temporal Logic Planning for Dynamical Systems. In *IEEE Conference on Decision and Control (CDC)*, pages 5997 –6004, 2009.

[45] Eric M Wolff, Ufuk Topcu, and Richard M Murray. Optimization-based trajectory generation with linear temporal logic specifications. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5319–5325. IEEE, 2014.

[46] Meng Guo and Dimos V Dimarogonas. Multi-agent plan reconfiguration under local ltl specifications. *The International Journal of Robotics Research*, 34(2):218–235, 2015.

[47] Jana Tmová and Dimos V Dimarogonas. A receding horizon approach to multi-agent planning from local ltl specifications. In *2014 American Control Conference*, pages 1775–1780. IEEE, 2014.

[48] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[49] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In Yassine Lakhnech and Sergio Yovine, editors, *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[50] I. Tkachev and A. Abate. Formula-free Finite Abstractions for Linear Temporal Verification of Stochastic Hybrid Systems. In *Proc. of the 16th Int. Conference on Hybrid Systems: Computation and Control*, Philadelphia, PA, April 2013.

[51] Cristian-Ioan Vasile, Derya Aksaray, and Calin Belta. Time window temporal logic. *Theoretical Computer Science*, 691:27–54, 2017.

[52] Vasumathi Raman, Alexandre Donzé, Mehdi Maasoumy, Richard M Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A Seshia. Model predictive control with signal temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, pages 81–87. IEEE, 2014.

[53] Sadra Sadraddini and Calin Belta. Robust temporal logic model predictive control. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 772–779. IEEE, 2015.

[54] Rajeev Alur, Tomás Feder, and Thomas A Henzinger. The benefits of relaxing punctuality. *Journal of the ACM (JACM)*, 43(1):116–146, 1996.

[55] Alexandros Nikou, Jana Tumova, and Dimos V Dimarogonas. Cooperative task planning of multi-agent systems under timed temporal specifications. In *2016 American Control Conference (ACC)*, pages 7104–7109. IEEE, 2016.

[56] Sofie Ahlberg and Dimos 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)*, pages 788–793. IEEE, 2019.

[57] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, pages 663–670, 2000.

[58] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[59] Georgios E Fainekos. Revising temporal logic specifications for motion planning. In *2011 IEEE International Conference on Robotics and Automation*, pages 40–45. IEEE, 2011.

[60] Cristian-Ioan Vasile and Calin Belta. An Automata-Theoretic Approach to the Vehicle Routing Problem. In *Proceedings of the Robotics: Science and Systems (RSS)*, Berkeley, California, USA, July 2014.

[61] Prasanna Velagapudi, Katia Sycara, and Paul Scerri. Decentralized prioritized planning in large multirobot teams. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4603–4609. IEEE, 2010.

[62] Michal Čáp, Peter Novák, Alexander Kleiner, and Martin Selecký. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. *IEEE transactions on automation science and engineering*, 12(3):835–849, 2015.

[63] Li Chun, Zhiqiang Zheng, and Wensen Chang. A decentralized approach to the conflict-free motion planning for multiple mobile robots. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 2, pages 1544–1549. IEEE, 1999.

[64] Hai Zhu and Javier Alonso-Mora. Chance-constrained collision avoidance for mavs in dynamic environments. *IEEE Robotics and Automation Letters*, 4(2):776–783, 2019.

[65] Alessandro Abate, Alessandro D'Innocenzo, and Maria D Di Benedetto. Approximate abstractions of stochastic hybrid systems. *IEEE Transactions on Automatic Control*, 56(11):2688–2694, 2011.

[66] George J Pappas. Bisimilar linear systems. *Automatica*, 39(12):2035–2047, 2003.

[67] Alexandros Nikou, Dimitris Boskos, Jana Tumova, and Dimos V Dimarogonas. On the timed temporal logic planning of coupled multi-agent systems.

*Automatica*, 97:339–345, 2018.

[68] Xuchu Ding, Mircea Lazar, and Calin Belta. Ltl receding horizon control for finite deterministic systems. *Automatica*, 50(2):399–408, 2014.

[69] Stephen P Bradley, Arnoldo C Hax, and Thomas L Magnanti. Applied mathematical programming. 1977.

[70] Samir Bouabdallah and Roland Siegwart. Design and control of a miniature quadrotor. In *Advances in unmanned aerial vehicles*, pages 171–210. Springer, 2007.

[71] Benoit Landry et al. *Planning and control for quadrotor flight through cluttered environments*. PhD thesis, Massachusetts Institute of Technology, 2015.

[72] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. International Society for Optics and Photonics, 1992.

[73] James A Preiss, Wolfgang Honig, Gaurav S Sukhatme, and Nora Ayanian. Crazyswarm: A large nano-quadcopter swarm. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3299–3304. IEEE, 2017.

[74] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*, pages 2520–2525. IEEE, 2011.

# Appendix A

# Glossary and Acronyms

Throughout this thesis, care has been taken to explain any subject matter specific terminology and acronyms when they are first introduced in the text. As a further point of reference, this appendix defines the specific terminology used in a glossary, and contains a table of acronyms and their meaning.

## A.1 Glossary

- Alphabet: A set of symbols, where a sequence of elements from the *alphabet* generates a *word*.
- Atomic Proposition: A statement or assertion that must be true or false.
- Cardinality: The number of elements in a vector, sequence, or set.
- Deadlock: Situation where the agent can not make progress towards its goal (in some form; criteria defined by the designer)
- Graph Automaton(a): A graph that incorporates information of some form (e.g., edge weights, mappings, labels, etc...)
- Language: A set of *words* over an *alphabet*.

- Logic Constraint: Statements (definite clauses) which are enforced in logic such as the Boolean $\neg, \vee, \wedge$ operators.
- Formal Methods: Mathematically rigorous techniques to synthesize and verify specifications
- Temporal Logic: A formal set of symbols and rules which form a language used to define complex specifications in terms of time which resembles that of natural language.

## A.2 Acronyms

Table A.1: Acronyms

| Acronym | Meaning |
|---|---|
| AP | Atomic Propositions |
| DP | Dynamic Programming |
| dFSA | Discrete Finite State Automaton |
| dWTS | Discrete Weighted Transition System |
| LTL | Linear Temporal Logic |
| MTL | Metric Temporal Logic |
| MILP | Mixed Integer Linear Program |
| MITL | Metric Interval Temporal Logic |
| PA | Product Automaton |
| SCP | Sequential Convex Programming |
| STL | Signal Temporal Logic |
| SITL | Signal Interval Temporal Logic |
| TS | Transition System |
| TWTL | Time Window Temporal Logic |

# Appendix B

# Collision Avoidance Implementation

This appendix contains the details of implementing collision avoidance for conflicting transitions, and the assumptions on agents and dynamics to generate the conflict mapping $C_{\mathcal{G}}$.

## B.1 Intersecting Transitions

In order to enforce collision avoidance, the agents require some physical representation in the discretized environment. The presented algorithm considers agents as spheres defined by their smallest enclosing radius $r \in \mathbb{R}^3$ which move about the environment on linear paths between adjacent states at a constant velocity (i.e., single integrator dynamics, $\dot{x} = u$).

Let agents $i$ and $j$ move over the environment graph $\mathcal{G} = (X, \Delta, w)$ via transitions denoted as $(x_t^i, x_{t+1}^i) \in \Delta$ and $(x_t^j, x_{t+1}^j) \in \Delta$, respectively. Note that states have a physical position in the environment, i.e., $v(x_t) \in \mathbb{R}^3$, which denotes the

centroid of a discrete state in the environment. The positions along the straight-line transitions in $\mathbb{R}^3$ are defined as a function $f(\cdot)$. If the distance between any pair of points along these line segments is less than some defined distance then these are considered intersecting transitions in $\mathcal{G}$ that are expressed by the following set of equations:

$$\begin{aligned} f(a) =& a \cdot v(x_t^i) + (1-a) \cdot v(x_{t+1}^i) \quad \text{where} \;\; a \in [0,1] \\ f(b) =& b \cdot v(x_t^j) + (1-b) \cdot v(x_{t+1}^j) \quad \text{where} \;\; b \in [0,1] \\ & \exists \; a, b \in [0,1] \quad \text{s.t.} \quad dist(f(a), f(b)) < r_{i,j} + \epsilon \\ \implies & \text{Intersecting Transitions, i.e.,} \\ & (x_t^i, x_{t+1}^i) \;\; \cap_{\mathcal{G}} \;\; (x_t^j, x_{t+1}^j) \neq \emptyset \end{aligned} \tag{B.1}$$

where $dist(\cdot)$ is the Euclidean distance between two points; $r_{i,j}$ is the sum of radii for a pair of agents, i.e., $r_i + r_j$; and $\epsilon \geq 0$ is some dilation defined by the user to account for imperfect motion. Note that the intersecting transition case captures both collisions due to traversing the same edge and due to intersecting diagonal movements.

Any conflicting transitions found using (B.1) for a transition $(x, x')$ are added to the set of conflicts $c_{\mathcal{G}}(x, x')$ where $c_{\mathcal{G}}(x, x') \in C_{\mathcal{G}}$.