KTH Electrical Engineering

Human-in-the-Loop Control Synthesis for Multi-Agent Systems under Metric Interval Temporal Logic Specifications

SOFIE AHLBERG

Licentiate Thesis
Stockholm, Sweden, 2019

KTH Royal Institute of Technology
School of Electrical Engineering and Computer Science
Division of Decision and Control
SE-100 44 Stockholm
SWEDEN

TRITA-EECS-AVL-2019:52
ISBN: 978-91-7873-215-9

Akademisk avhandling som med tillstånd av Kungliga Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie licenciatexamen i reglerteknik fredagen den 14 juni 2019 klockan 10.00 i sal Q31 Kungliga Tekniska högskolan, Malvinas Väg 6B, KTH, Stockholm.

© Sofie Ahlberg, June 2019. All rights reserved.

Tryck: Universitetsservice US AB

Abstract

With the increase of robotic presence in our homes and work environment, it has become imperative to consider human-in-the-loop systems when designing robotic controllers. This includes both a physical presence of humans as well as interaction on a decision and control level. One important aspect of this is to design controllers which are guaranteed to satisfy specified safety constraints. At the same time we must minimize the risk of not finding solutions, which would force the system to stop. This require some room for relaxation to be put on the specifications. Another aspect is to design the system to be adaptive to the human and its environment.

In this thesis we approach the problem by considering control synthesis for multi-agent systems under hard and soft constraints, where the human has direct impact on how the soft constraint is violated. To handle the multi-agent structure we consider both a classical centralized automata based framework and a decentralized approach with collision avoidance. To handle soft constraints we introduce a novel metric; hybrid distance, which quantify the violation. The hybrid distance consists of two types of violation; continuous distance or missing deadlines, and discrete distance or spacial violation. These distances are weighed against each other with a weight constant we will denote as the human preference constant. For the human impact we consider two types of feedback; direct feedback on the violation in the form of determining the human preference constant, and direct control input through mixed-initiative control where the human preference constant is determined through an inverse reinforcement learning algorithm based on the suggested and followed paths. The methods are validated through simulations.

Sammanfattning

I takt med att robotar blir allt vanligare i våra hem och i våra arbetsmiljöer, har det blivit allt viktigare att ta hänsyn till människan plats i systemen när regulatorerna för robotorna designas. Detta innefattar både människans fysiska närvaro och interaktion på besluts- och reglernivå. En viktig aspekt i detta är att designa regulatorer som garanterat uppfyller givna villkor. Samtidigt måste vi minimera risken att ingen lösning hittas, eftersom det skulle tvinga systemet till ett stopp. För att uppnå detta krävs det att det finns rum för att mjuka upp villkoren. En annan aspekt är att designa systemet så att det är anpassningsbart till människan och miljön.

I den här uppsatsen närmar vi oss problemet genom att använda regulator syntes för multi-agent system under hårda och mjuka villkor där människan har direkt påverkan på hur det svaga villkoret överträds. För att hantera multi-agent strukturen undersöker vi både det klassiska centraliserade automata-baserade ramverket och ett icke-centraliserat tillvägagångsätt med krockundvikning. För att hantera mjuka villkor introducerar vi en metrik; hybrida avståndet, som kvantifierar överträdelsen. Det hybrida avståndet består av två typer av överträdelse (kontinuerligt avstånd eller missandet av deadlines, och diskret avstånd eller rumsliga överträdelser) som vägs mot varandra med en vikt konstant som vi kommer att kalla den mänskliga preferens kontanten. Som mänsklig påverkan överväger vi direkt feedback på överträdelsen genom att hon bestämmer värdet på den mänskliga preferens kontanten, och direkt påverkan på regulatorn där den mänskliga preferens konstanten bestäms genom en inverserad förstärkt inlärnings algoritm baserad på de föreslagna och följda vägarna. Metoderna valideras genom simuleringar.

To my father, who I wish could be here,
and I know is watching proudly on the other side.

Acknowledgements

First, I want to direct my deepest gratitude to my supervisor Prof. Dimos Dimarogonas, for introducing me to the topic at hand and giving me the opportunity to continue to study it. I am grateful for all our discussions, and your feedback and support. I would also like to thank Jean-Pierre Meyer, Meng Guo and Alexandros Nikou for interesting discussions and helpful suggestions on my work. I want to direct a special thanks to Meng for our joint work and to Alexandros for guiding me onto this path as my master thesis supervisor.

Next, I would like to thank Pian Yu, Alexandros Nikou, Xiao Tan and Dionysios Theodosis Palimeris for proof-reading this thesis, it is much appreciated.

As for our department, thank you Emma Tegling for being a great roommate! Our small talks and your encouraging words have lighten up my days. Linnea Persson, thank you for our fun times outside of the office and for all our joint course work. I also want to thank all the members in the COIN project and the former members of ”the old reading group” for creating such a nice work environment.

Finally, I want to thank my family, and my husband in particular, for being supportive of me at times working late nights and weekends, helping me to put reasonable demands on myself and always cheering me on.

Sofie Ahlberg
May 2019

Contents

Abstract **iii**

Sammanfattning **v**

Acknowledgements **vii**

Abbreviations **xii**

List of Figures **xv**

1 Introduction **1**
1.1 Motivation 1
1.2 Literature Overview 4
1.3 Thesis Outline and Contribution 5

2 Notation and Preliminaries **9**
2.1 Notation 9
2.2 Model Preliminaries 9
2.3 Control Design Preliminaries 12

3 Abstraction with Time Constraints **15**
3.1 Introduction 15
3.2 Problem Statement 16
3.3 Constructing a Weighted Transition System 16
3.4 Simulations 20
3.5 Conclusion 23

4 Control Synthesis for Multi-Agent Systems under Hard MITL Tasks **25**
4.1 Introduction 25
4.2 Problem Statement 26
4.3 Control Strategy 27
4.4 Simulations 30

4.5 Conclusion 33

5 Control Synthesis for Single-Agent Systems under Soft MITL Tasks 35
5.1 Introduction 35
5.2 Preliminaries and Notation 36
5.3 Problem Statement 39
5.4 Control Strategy 39
5.5 Simulations 51
5.6 Conclusion 53

6 Control Synthesis for Single-Agent Systems under Hard and Soft MITL Tasks 55
6.1 Introduction 55
6.2 Problem Statement 56
6.3 Control Strategy 56
6.4 Conclusion 60

7 Human-in-the-Loop Control with Preference Learning and Collision Avoidance for Multi-Agent Systems 61
7.1 Introduction 61
7.2 Problem Statement 62
7.3 Control Strategy 63
7.4 Simulations 68
7.5 Conclusion 71

8 Summary and Future Work 73
8.1 Summary 73
8.2 Future Work 75

Bibliography 77

Abbreviations

Table 1: Symbols and Notations

Abbreviations and Notation	**Meaning**
LTL	Linear Temporal Logic
MITL	Metric Interval Temporal Logic
STL	Signal Temporal Logic
MTL	Metric Temporal Logic
TBA	Timed Büchi Automaton
WTS	Weighted Transition System
BWTS	Büchi Weighted Transition System
IRL	Inverse Reinforcement Learning
TAhd	Timed Automaton with Hybrid Distance
HRI	Human Robot Interaction
MILP	Mixed-Integer Linear Programming
N	Number of agents
$m_{Set} = \|Set\|$	Number of elements in Set

List of Figures

1.1 Example of applications of human-in-the-loop control systems. . . . 3

2.1 Example of a Timed Büchi Automaton 11
2.2 Example of a Automata Product 14

3.1 Variables for calculating transition times 18
3.2 Partitioning of an environment for the WTS 22
3.3 Example of a WTS 22

4.1 Satisfying paths for a 2-agent system 32

5.1 Example of the construction of a TAhd from soft constraints 48
5.2 Comparison between a TBA and a TAhd 49
5.3 Initial path found using control synthesis with hybrid distance . . . 53
5.4 Suggested paths for control synthesis with hybrid distance given human feedback 53

6.1 Example of a TAhd for hard and soft constraints 59

7.1 Initial and final trajectories for a 2-agent system using the online control synthesis 71

Chapter 1

Introduction

1.1 Motivation

With the progress in the robotics and autonomous control fields we see an increase in robotic presence in environments populated by humans. This has increased the importance of human robot interaction (HRI) and Human-in-the-Loop planning and control. These include both physical interaction and communication, where it is important to create systems that are safe and receptive to human preference. Simultaneously, the size of the robotic systems increase in the sense that a larger number of agents can be found in the same place. This creates the need for combining the above with multi-agent systems.

To achieve safety, we need system designs with strict guarantees, eliminating the risk of incorrect behaviour. To achieve this we turn to formal verification methods to either perform model checking [1] on an already designed system, or to use control synthesis to design correct-by-construction controllers [2–9]. In both approaches we use temporal logic [10–23] to mathematically formalize the constraints we set on the system. Temporal logic allows us to express temporal properties using temporal operators, logic connectives and atomic propositions. By using temporal logic to express tasks such as reachability and avoiding regions, all under deadlines and/or lower time limits, we can apply formal methods to find controllers that are guaranteed to satisfy our constraints. The control synthesis framework typically follows three steps. First, the dynamics and workspace of each considered agent is abstracted into a transition system that expresses a discretized version of the behaviour [24, 25]. In order to guarantee that a solution found for the abstraction is valid for the true system, the behaviour of the transition system must be similar to and the language a subset of the behaviour and language of the original system. Second, the temporal logic specification is translated into an automaton which has a language equivalent to the specification [26–31]. Third, an attempt to find controllers to the original system which satisfy the constraints are made by applying graph search algorithms to the product automaton constructed from the specification and the transition system. If multi-agent systems are considered the number of products normally

increases to combine the information from each agent. While this method provides the guarantees for safety, we can not conclude that a lack of solution is equivalent to an infeasible task. Furthermore, the approach is sensitive to the number of states of the transition system and specification in the sense that they are multiplied every time a product is constructed, leading to sometimes huge automata being computed and searched through. Also, the synthesis is performed offline, and hence the system is not adaptable to any new information obtained during the execution of the plan.

To maximize our chances of finding a solution, we must study the cases where the typical method fails and determine why or how it can be extended to be successful. In this spirit, three different approaches have been applied. First, minimizing over-approximations which may cause the lack of solution by refining the abstraction [32]. Second, using feedback from the synthesis to determine how the specification can be relaxed to achieve satisfaction [33, 34]. Third, by introducing metrics that quantify or predict the probability of the violation we can consider solutions that almost satisfy the constraint or have the highest probability of satisfying it [7, 35, 36].

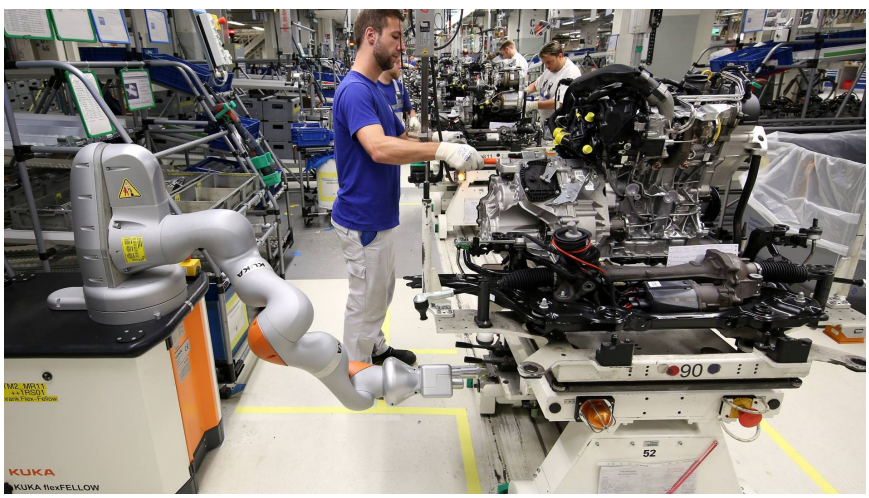
A system that is adaptive to a human user must be able to attain knowledge of the desires of the human and know how to behave to make her happier [37–40]. The knowledge should be as clear as possible for the system to interpret, and easy for the user to give. At the same time, it is crucial that the knowledge is attained in a manner such that we don't lose the guarantees which were gained by using the formal methods tools. That is, the system must allow the human enough control to be able to achieve her goals, without risking that her mistakes to cause a failure in safety.

If we can achieve this, we can apply it to any safety-critical robotic system where humans are present either through some level of co-piloting or by physically sharing the workspace. This includes factories where humans and robots cooperate or share a workspace, commercial robots working in our homes such as vacuum cleaners or lawn mowers, and search and rescue missions where a human co-pilots robots in environments which are unfriendly to humans, see Fig. 1.1.

In this thesis, we start from the standard control synthesis framework for a multi-agent system, extend it to include timed constraints, introduce a novel metric to quantify violation and apply it to find least violating solutions when no perfect solution can be found. Furthermore, we adapt the framework to include hard and soft constraints, allowing us to keep satisfaction guarantees for the hard constraints while relaxing the soft constraints. We investigate two different ways of attaining knowledge from the human; direct feedback and control input through a mixed-initiative controller [41–43] which is converted to the desired feedback by an inverse reinforcement learning (IRL) algorithm [44, 45]. Finally, to avoid the big-sized automata caused by constructing multiple products, we consider a decentralized approach where each agent plans their own path combined with a collision avoidance algorithm.

(a) iRobots's vacuum cleaner Roomba, commonly used in many households [46]

(b) Coyote III, a search and rescue robot which was first developed for space.[47]



(c) Volkswagen committed to using the KUKA robot together with their human employees in their factory in Wolfsburg in 2016[48]

Figure 1.1: Example of applications of human-in-the-loop control systems.

1.2 Literature Overview

This chapter provides a literature overview of temporal logic, control synthesis for single and multi-agent systems under temporal logic constraints, methods for handling what appears to be infeasible tasks, and human-in-the-loop systems.

One of the advantages with temporal logics, is that it allows a user to formulate tasks in a language which has several similarities to structural English. This was further investigated in [10] where one possible translation process was specified to convert structural English into Linear Temporal Logic (LTL). Temporal logic includes several sub-languages such as LTL (mentioned above) [1–4, 10–15], Metric Interval Temporal Logic (MITL)[16–18] and Signal Temporal Logic (STL)[19–23]. The differences between the mentioned sub-languages are whether time limits are considered and if the evaluation of the atomic proposition is boolean or continuous. In the LTL case we consider boolean evaluations without time constraints, while MITL considers boolean evaluations with time intervals related to the operators, and STL consider continuous evaluations with time constraints.

Control synthesis for multi-agent systems under LTL specifications has been addressed in [49–51]. Due to the fact that we are interested in imposing timed constraints to the system, the aforementioned works cannot be directly utilized. Timed constraints have been introduced for the single agent case in [7, 20, 52, 53] and for the multi-agent case in [6, 54]. Authors in [54] addressed the vehicle routing problem, under Metric Temporal Logic (MTL) specifications. The corresponding approach does not rely on automata-based verification, as it is based on a construction of linear inequalities and the solution of a resulting Mixed-Integer Linear Programming (MILP) problem. In [6], an automata based approach was used instead and both individual and cooperative tasks were considered.

When considering infeasible tasks three different approaches have been used. In [32] a method for abstraction refinement to find control policies which could not be found in a sparser partitioning was suggested. The idea is to systematically decrease the size of regions until a path can be found. There is however no guarantee that a solution can be found after refinement, and in theory the refinement can continue indefinitely. In [33] a framework which gives feedback on why the specification is not satisfiable and how to modify it was presented. It was suggested to use this as a basis for formula relaxation to change the original formula as little as possible while achieving a feasible specification. A third approach is to consider how well a formula is satisfied or how likely it is to be satisfied. This is done in [35] and [36], where metrics are introduced to find an approximate or robust solution to the control synthesis. It allows the user to find a solution that is within an error margin (defined by the metric) of the specification. [7] instead treats the environment as stochastic and designs the controller such that the probability of satisfaction is maximized.

Human-in-the-loop systems are any system where a human has impact for instance through co-piloting, a physical presence in the workspace or by giving commands. Regardless of the manner of the human impact, the system has to take it into consideration and adapt to avoid accidents and/or undesired behaviour. For

this purpose, the system must first be aware of the human behaviour and what consequences it has. One way of obtaining this knowledge is to study the human. This was done in [37], where a control policy was created based on data of human decisions. When the knowledge is obtained the system must determine how to respond to it. In [38], a model of human workload information was used to optimize the systems behaviour to balance risk of stress due to full backlogs against risk of low productivity due to empty backlogs. Another approach is to give the human direct impact by making certain decisions for the system in a semi-automated process. This is done in [39], where the human takes the role as a supervisor assigning what type of tasks each robot in a multi-robot system should perform. This allows the human to have direct impact on what type of tasks should be considered as most important or requires more attention, at each time step. [40] considers cooperative tasks, where both human and robot directly impacts the control input, and suggests an adaptive control scheme to attain a system which combines the inputs while avoiding oscillatory behaviour. A question that needs to be handled in the case of human interaction in control synthesis is how the humans modifications of the plan impact the guarantees of task satisfaction. This was investigated in [42], where a control scheme was suggested that only lets the human modify the plan in such a way that the guarantees remain. The control scheme is built on navigation functions which drives the human input to zero if a safety constraint is about to be violated, i.e. mixed-initiative control.

1.3 Thesis Outline and Contribution

In Chapter 2, we introduce notation and preliminaries that are used throughout the thesis. This includes weighted transition systems (WTS), Metric Interval Temporal Logic (MITL), timed Büchi automata, clock constraints, automata products, and the Dijkstra algorithm. The main work of the thesis is divided into five parts in Chapters 3 to 7 which presents methods for abstraction, multi-agent control synthesis for cooperative and individual MITL tasks, control synthesis for least violating solutions, control synthesis for hard and soft constraints, and an online decentralized synthesis framework for multi-agent systems using mixed-initiative control and inverse reinforcement learning (IRL) combined with collision avoidance. The details of the main chapters are given below.

Chapter 3

The first part of this thesis is presented in Chapter 3, and presents a method of abstracting affine dynamics in a square workspace under some constraints on the control input into a WTS. We use an optimization problem to determine if there exist a controller, within the allowed limits, which guarantee that a transition occurs. Next, we use simple algebra to determine the maximum time required to guarantee a transition as a function of the dynamics and control input. The result is a WTS where the weights correspond to overestimations of the transition times.

- C1: [55] Sofie Andersson, Alexandros Nikou and Dimos V. Dimarogonas, Control Synthesis for Multi-Agent Systems under Metric Interval Temporal Logic Specifications. *In the 20th World Congress of the International Federation of Automatic Control (IFAC WC 2017),* 2017.

Chapter 4

Chapter 4 presents the second part of the thesis, a control synthesis framework for a multi-agent system under MITL specifications, considering both individual and cooperative tasks. The framework is automata based and initially follows the standard three steps, adapted to timed constraints by using a timed Büchi automata (TBA) and a WTS abstracted following our approach in Chapter 3. To adapt to the multi-agent system, and allow for cooperative tasks, two additional product automata are constructed to combine the data from i) the individual agents and tasks with each other, and ii) the agents and individual tasks with the cooperative task.

- C1: [55] Sofie Andersson, Alexandros Nikou and Dimos V. Dimarogonas, Control Synthesis for Multi-Agent Systems under Metric Interval Temporal Logic Specifications. *In the 20th World Congress of the International Federation of Automatic Control (IFAC WC 2017),* 2017.

Chapter 5

In Chapter 5, we introduce a novel metric denoted as hybrid distance. The metric quantifies the violation of a given MITL specification by a specific run on a product automaton. It is then used in the synthesis framework to find a least violating run, rendering the specification to be treated as a soft constraint. The hybrid distance uses a human preference constant to weigh different types of violation (missing deadlines and spacial violation) against each other. The result is a simple human-in-the-loop control system where the human gives direct feedback by increasing or decreasing the value of the constant.

- C2: [56] Sofie Andersson and Dimos V. Dimarogonas, Human in the Loop Least Violating Robot Control Synthesis under Metric Interval Temporal Logic Specifications. *In the European Control Conference (ECC),* 2018.

Chapter 6

In Chapter 6 we extend the work in Chapter 5 to include specifications of both hard and soft constraints. The resulting system combines the advantages of Chapters 4 and 5, by keeping the guarantee that the hard constraint is satisfied while allowing the relaxation of the satisfaction of the soft constraint.

- C3: [57] 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 the IEEE International Conference on Automation Science and Engineering (CASE),* 2019.

Chapter 7

In Chapter 7 we put the content of Chapter 6 into a decentralized multi-agent framework where the human is allowed direct control input through a mixed-initiative controller, the human preference constant is determined through an IRL algorithm based on the human input, and a collision avoidance algorithm is applied to ensure safety among the agents without the need to use further product automata.

- C3: [57] 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 the IEEE International Conference on Automation Science and Engineering (CASE),* 2019.
- C4: [58] Meng Guo, Sofie Andersson and Dimos V. Dimarogonas, Human-in-the-Loop Mixed-Initiative Control under Temporal Tasks. *In the IEEE International Conference on Robotics and Automation (ICRA),* 2018.

Chapter 8 concludes the thesis, by summarizing the content and presenting future work possibilities.

Chapter 2

Notation and Preliminaries

In this chapter, we introduce notation and preliminaries which will be used throughout the thesis. The preliminaries stated here are commonly known results within their respective field.

2.1 Notation

Here, we introduce some notation that is used throughout the thesis.

We will use $m_{Set} = |Set|$ to denote the cardinality of Set, i.e. the number of elements in Set. True and false are denoted by $\top$ and $\bot$. $\mathbb{R}$ are the real numbers, $\mathbb{R}_+ \subset \mathbb{R}$ are the positive real numbers and $\mathbb{R}^n$ is the n-dimensional real vector space. Note that we use the symbol x to both denote the state of the agents (i.e. position) and to represent the clocks in the timed automata. These applications are separated and it should be clear in each case which is referred to. In the main chapters we will use the symbol t to denote time, and relate the clock constraints to time (using $t \propto c$) instead of clocks (using $x \propto c$), to simplify for the reader.

2.2 Model Preliminaries

In this thesis we will model the dynamics and workspace of each agent as a weighted transition system (WTS).

Definition 2.1. A *Weighted Transition System* (WTS) is a tuple $T = (\Pi, \Pi^{init}, \Sigma, AP, L, \rightarrow, w)$ where $\Pi = \{\pi_0, ..., \pi_{m_\Pi}\}$ is a finite set of states; $\Pi^{init} \subset \Pi$ is a set of initial states; $\Sigma = \{\sigma_1, ..., \sigma_{m_\Sigma}\}$ is a finite set of actions; AP is a finite set of atomic propositions; $L : \Pi \rightarrow 2^{AP}$ is an labelling function; $\rightarrow \subseteq \Pi \times \Pi$ is a transition relation; the expression $\pi_i \rightarrow \pi_k$ is used to express transition from π_i to π_k, and $w : \rightarrow \rightarrow \mathbb{R}_+$ is a positive weight assignment map; the expression $w(\pi_i, \pi_k)$ is used to express the weight assigned to the transition $\pi_i \rightarrow \pi_k$.

Definition 2.2. A *timed run* $r^t = (\pi_0, \tau_0)(\pi_1, \tau_1)...$ of a WTS T is an infinite sequence where $\pi_0 \in \Pi^{init}$, $\pi_j \in \Pi$, and $\pi_j \rightarrow \pi_{j+1}$ $\forall j \geq 1$ s.t.

- $\tau_0 = 0$,
- $\tau_{j+1} = \tau_j + w(\pi_j, \pi_{j+1})$, $\forall j \geq 1$.

The tasks assigned to the agents are expressed in Metric Interval Temporal Logic (MITL).

Definition 2.3. The *syntax of MITL* over a set of atomic propositions AP is defined by the grammar

$$\phi := \top \mid ap \mid \neg\, \phi \mid \phi \wedge \psi \mid \phi\, \mathcal{U}_{[a,b]}\, \psi \tag{2.1}$$

where $ap \in AP$, $a, b \in [0, \infty]$ and ϕ, ψ are formulas over AP. The operators are *Negation* ($\neg$), *Conjunction* ($\wedge$) and *Until* ($\mathcal{U}$) respectively. Given a timed run $r^t = (\pi_0, \tau_0)(\pi_1, \tau_1), ...$ of a WTS, the semantics of the satisfaction relation is then defined as [16], [18]:

$$(r^t, i) \models ap \Leftrightarrow L(\pi_i) \models ap \text{ (or } ap \in L(\pi_i)), \tag{2.2a}$$

$$(r^t, i) \models \neg\phi \Leftrightarrow (r^t, i) \not\models \phi, \tag{2.2b}$$

$$(r^t, i) \models \phi \wedge \psi \Leftrightarrow (r^t, i) \models \phi \text{ and } (r^t, i) \models \psi, \tag{2.2c}$$

$$(r^t, i) \models \phi\, \mathcal{U}_{[a,b]}\, \psi \Leftrightarrow \exists j \in [a, b],\ s.t.\ (r^t, j) \models \psi \text{ and } \forall i \leq j, (r^t, i) \models \phi. \tag{2.2d}$$

The extended operators *Eventually* ($\Diamond$) and *Always* ($\Box$) are defined as:

$$\Diamond_{[a,b]}\phi := \top \mathcal{U}_{[a,b]}\phi, \tag{2.3a}$$

$$\Box_{[a,b]}\phi := \neg\Diamond_{[a,b]}\neg\phi. \tag{2.3b}$$

To apply formal methods on MITL, the specification is translated into a timed automaton. In the standard control synthesis framework it is normally translated into a Timed Büchi Automaton (TBA). The definition of the TBA is given in Definition 2.4 and a small example is given in Example 2.1.

Definition 2.4. A *Timed Büchi Automaton* (TBA) is a tuple $A = (S, S^{init}, F, AP, X, I_X, E)$ where $S = \{s_0, ..., s_{m_S}\}$ is a finite set of locations, $S^{init} \in S$ is the initial location, $F \subset S$ is a set of accepting locations, AP is a finite set of atomic propositions, X is a finite set of clocks, $I_X : S \to \Phi_X$ is a map labelling each state s_i with some clock constraints Φ_X, and $E \subseteq S \times \Phi_X \times 2^{AP} \times S$ is a set of transitions.

We use $(s, g, a, s') \in E$ to denote that there exists an edge from s to s' under the guard $g \in \Phi_X$ and action $a \in 2^{AP}$, and the clock constraints are defined as:

Definition 2.5. [26] A *clock constraint* Φ_x is a conjunctive formula of the form $x \bowtie a$, where $\bowtie \in \{<, >, \leq, \geq\}$, x is a clock and a is some non-negative constant. Let Φ_X denote the *set of clock constraints* over the set of clocks X.

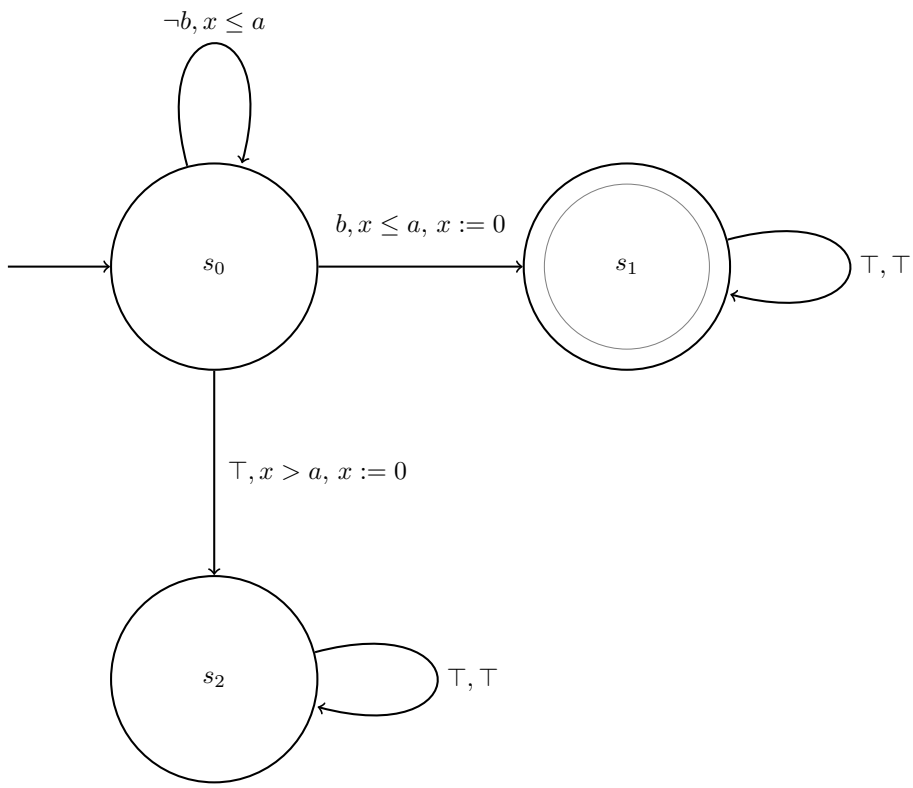


Figure 2.1: A TBA expressing the MITL task $\Diamond_{x \leq a} b$.

Example 2.1. Consider the MITL specification $\Diamond_{x \leq a} b$. A TBA expressing the specification can be constructed by systematically considering what can occur and what consequences it has on the satisfaction of the formula. For instance, initially (in the initial state) there are 4 possible events; i) the action b occurs when $x \leq a$, ii) the action $\neg b$ occurs when $x > a$, iii) the action $\neg b$ occurs when $x \leq a$, and iv) b occurs when $x > a$. In the first case the formula is satisfied, hence there should be an edge between the initial state and an accepting state (which we now create) with the guard $x \leq a$ and the action b. In the second and fourth case the formula is violated and hence we want an edge from the initial state to some violating state (which we now create) with the guard $x > a$ and the action-set $\{b, \neg b\} = \top$. Finally, in the third case the formula is neither satisfied or violated and no progress occur. Hence we add a self-loop to the initial state where the edge has the guard $x \leq a$ and action $\neg b$. This method is then continued on the newly created states until no new states are needed. The resulting TBA is shown in Fig. 2.1.

2.3 Control Design Preliminaries

The control design part of the standard control synthesis framework consists of two steps; i) constructing a product automaton (Definition 2.6), and ii) using a graph search algorithm to find an accepting path (e.g. the Dijkstra Algorithm - Algorithm 1). The path can then be projected onto the WTS to obtain a discrete path in the initial environment, or a sequence of actions can be determined by evaluating the edges along the accepting path.

Definition 2.6. Given a weighted transition system $T = (\Pi, \Pi^{init}, \Sigma, AP, L, \rightarrow, w)$ and a timed Büchi automaton $A = (S, S^{init}, F, AP, X, I_X, E)$ their product is defined as $P = T \otimes A = (Q, Q^{init}, \mathcal{F}, \hat{\Sigma}, \mathcal{AP}, \mathcal{L}, \mathcal{X}, I_{\mathcal{X}}, \rightsquigarrow, \hat{w})$ where: $Q \subseteq \{(\pi, s) \in \Pi \times S\}$ is a set of states, $Q^{init} = \Pi_{init} \times S_{init}$ is a set of initial states, $\mathcal{F} = \{(\pi, s) \in Q : s \in F\}$ is a set of accepting states, $\hat{\Sigma} = \Sigma = 2^{AP}$ is a set of actions, $\mathcal{AP} = AP$ is a set of atomic propositions, $\mathcal{L}(\pi, s) = L(\pi)$ is a labelling function from states to actions, $\mathcal{X} = X$ is a set of clocks, $I_{\mathcal{X}}(s, \pi) = I_X(s)$ is a mapping of clock constraints onto states, $\rightsquigarrow$ is a set of transitions where $q \rightsquigarrow q'$ iff

- $q = (\pi, s)$, $q' = (\pi', s') \in Q$
- $(\pi, \pi') \in \rightarrow$ and
- $\exists\, g, a$, s.t. $(s, g, a, s') \in E$ where $a = L(\pi')$,

and $\hat{w}(q, q') = w(\pi, \pi')$ if $(q, q') \in \rightsquigarrow$ is a weight function assigning a weight constant to each transition,

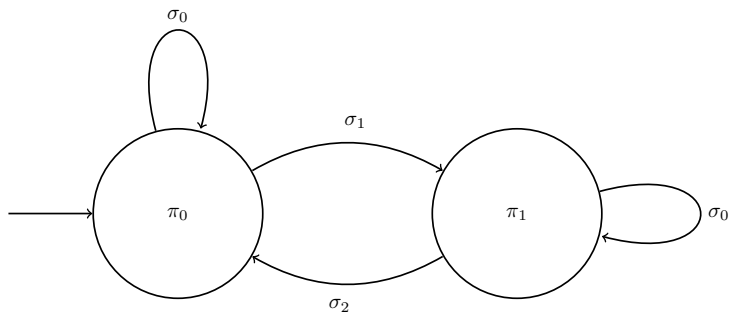
An example of a product automaton is given below.

Example 2.2. In this example we consider the TBA we constructed in Example 2.1, and the WTS illustrated in Fig. 2.2a, where the labelling function L is such that $L(\pi_0) = \emptyset$ and $L(\pi_1) = b$.

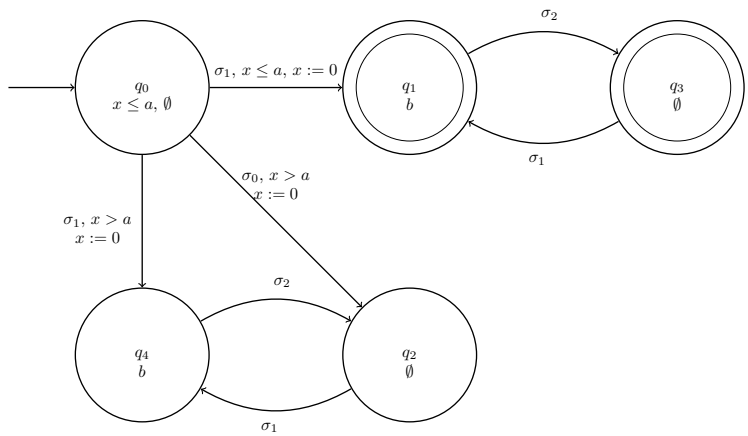
Following Definition 2.6, we create 6 states (all combinations of the states in the WTS and the locations in the TBA), mark the state $q_0 = (\pi_0, s_0)$ as initial and the states $q_1 = (\pi_1, s_1)$ and $q_3 = (\pi_0, s_1)$ as accepting. We then add edges between any states $q = (\pi, s)$ and $q' = (\pi', s')$ where $\pi \rightarrow \pi'$ and $(s, g, a, s') \in E$. Here, we note that the state $q_5 = (\pi_1, s_0)$ is not reachable from any other state (since entering state π_1 evokes the transition to s_1). We can therefore simplify the product by removing q_5. The resulting BWTS is illustrated in Fig. 2.2b.

Algorithm 1 Dijkstra Algorithm with weight mapping $\hat{w}_G$ as cost function.

```
procedure DIJSKTRA(P)
    Q = set of states, q_0 = initial state, SearchSet = q_0, d(q, q') = weight of
    transition q ⇝ q' in P
    if q = q_0 then
        dist(q) = 0
    else
        dist(q) = ∞
    end if
    for q ∈ Q do
        pred(q) = ∅
    end for
    while pathFound = False do
        pick q ∈ SearchSet s.t. q = arg min(dist(q))
        if q ∈ F then
            pathFound = True
        else
            find all q' s.t. q ⇝ q'
            for every q' do
                dist^step = ŵ_G(q, q')
                if dist(q') > dist(q) + dist^step then
                    update dist(q') and pred(q')
                    add q' to SearchSet
                    remove q from SearchSet
                end if
            end for
        end if
    end while
    r_d^min = q
    while q ≠ q_0 do
        use pred(q) to iteratively form the path back to q_0:
        r_d^min = [pred(q), r_d^min]
    end while
end procedure and return r_d^min, dist
```



(a) A simple WTS consisting of 2 states with bi-directional transitions.



(b) The automata product of the WTS in Fig. 2.2a and the TBA in Fig. 2.1.

Figure 2.2: WTS (a) and BWTS (b) in Example 2.2

Chapter 3

Abstraction with Time Constraints

In this chapter we consider the abstraction of an agent following affine dynamics in a square workspace into a WTS, where the weights are the maximum required transition times. The work in this chapter is part of [55].

3.1 Introduction

The main motivation for this chapter is to construct a timed abstraction which simulates the behaviour of the original system, and which can be used in the control synthesis framework. The final goal is to have a discrete system which we can use to find a sequence of control inputs yielding a desired behaviour from the original system. For this to work, it must be true that any discrete path in the abstraction has a corresponding continuous path in the original system. That is, the abstraction can not include any transition that does not exist in the original system. Furthermore, the transition times of the abstraction (i.e. the weights of the WTS) must be greater or equal to the times of the corresponding transitions in the real system. By constructing this abstraction, we can use discrete methods to find a plan for the WTS which can then be translated into a continuous plan, instead of having to use more complicated continuous methods directly on the original system.

The suggested abstraction is based on the work presented in [52], which considered time bounds on facet reachability for a continuous-time multi-affine single agent system. Here, we consider multi-agent systems and suggest an alternative time estimation and provide a proof for its validity. The contribution of this chapter is a method to construct a WTS from affine dynamics. When applied in the control synthesis framework, this allows a user to give the system affine dynamics and consider linear control input, rather than using the more common assumption that a WTS already exists and using single integrator controllers.

3.2 Problem Statement

Consider an agent in a bounded workspace $W \subset \mathbb{R}^n$, governed by the dynamics

$$\dot{x} = Ax + Bu \tag{3.1}$$

$$x(0) = x^0, x \in W \tag{3.2}$$

The problem then becomes:

Problem 3.1. Construct a WTS which simulates the dynamics and workspace expressed by (3.1)-(3.2).

3.3 Constructing a Weighted Transition System

The definition of a WTS was given in Chapter 2. To construct the WTS we first use the workspace to create a partitioning into rectangles (Chapter 3.3.1). We then find controllers for transitioning between rectangles and suitable weights by considering the dynamics (Chapter 3.3.2). Finally, the components of the WTS can be determined (Chapter 3.3.3).

3.3.1 Workspace Abstraction

Following the idea of [52], we begin by dividing the state space W into p-dimensional rectangles, defined as in 3.1.

Definition 3.1. A *p-dimensional rectangle* $R_p(a, b) \subset \mathbb{R}^p$ is characterized by two vectors a, b, where $a = (a_1, a_2, ..., a_p)$, $b = (b_1, b_2, ..., b_p)$ and $a_i < b_i$, $\forall\, i = 1, 2, ..., p$. The rectangle is then given by

$$R_p(a, b) = \{x \in \mathbb{R}^p : a_i \leq x_i \leq b_i, \forall i \in \{1, 2, .., p\}\} \tag{3.3}$$

Here, we construct the rectangles such that formula (3.4) is satisfied for each rectangle, i.e, such that each atomic proposition in the set AP is either true at all points within a rectangle $R_p(a, b)$ or false at all points within the rectangle, i.e.

$$\begin{aligned} &\forall ap_i \in AP \text{ and } \forall R_p(a, b) : \\ &\qquad ap_i = (\top,\ \forall x \in R_p(a, b)) \text{ or} \\ &\qquad ap_i = (\bot,\ \forall x \in R_p(a, b)). \end{aligned} \tag{3.4}$$

3.3.2 Control Design and Upper Time Limits

For a transition to be defined in the WTS a control input must exist such that the agent is guaranteed to move in between the corresponding rectangles, that is the edge which the rectangles shares must be reachable and no other edges can be reached during the transition. These conditions on control inputs are required both

to ensure that the synthesized path is followed and to guarantee that the following time estimation holds. A suggested low-level controller for a transition $\pi_k \to \pi_l$ in direction i, based on [52], is given by

$$\begin{aligned} \max_{u \in U} \quad & \dot{x}_i \\ \text{s. t.} \quad & \dot{x}_i \geq \epsilon > 0, \\ & \dot{x}_j \leq -\epsilon, \forall j \neq i, j = \{1, ..., p\}, \text{if } [x]_j = b_j^k, \\ & \dot{x}_j \geq \epsilon, \forall j \neq i, j = \{1, ..., p\}, \text{if } [x]_j = a_j^k. \end{aligned} \tag{3.5}$$

where $U = [-u_{max}, u_{max}]$ is some bound on u and ϵ is a robustness parameter. The idea is to maximize the transition speed, under the conditions that the speed in direction j is negative at the edge with norm direction j, where j is not the transition direction.

The maximum transition times (minimum time which guarantees that the transition has occurred if the correct control input is given) can then be found according to Theorem 3.1 below. The theorem depends on the assumption $Bu = B_1 x + B_2$, where B_1 and B_2 are matrices of dimension $\dim W \times \dim W$ and $\dim W \times 1$ respectively, where W is the workspace. The assumption corresponds to u being affine.

Theorem 3.1. *The maximum time $T^{max}(\pi_k, \pi_l)$ required for the transition $\pi_k \to \pi_l$ to occur, where $R_p(a^k, b^k)$ and $R_p(a^l, b^l)$ share the edge e_{kl}, $\overline{e_{kl}}$ is the edge located opposite to e_{kl} in $R_p(a^k, b^k)$, i is the direction of the transition, and assuming that e_{kl} is reachable from all points within π_k, is defined as:*

$$T^{max}(\pi_k, \pi_l) = \ln \left(\frac{(A_{ii}^* x^1 + C^* + B_i^*)}{(A_{ii}^* x^0 + C^* + B_i^*)} \right) \frac{1}{A_{ii}^*} \tag{3.6}$$

where

$$C^* = \sum_{\substack{j=1 \\ j \neq i}}^{n} \min_{x_j \in \pi_k} \left(A_{ij}^* x_j \right),$$

$$\min_{x_j \in \pi_k} \left(A_{ij}^* x_j \right) = \begin{cases} A_{ij}^* a_j^k & \text{if } A_{ij}^* > 0 \\ A_{ij}^* b_j^k & \text{if } A_{ij}^* < 0 \end{cases}, \tag{3.7}$$

and $x^0 \in \overline{e_{kl}}$, $x^1 \in e_{kl}$ (note that x^0, x^1 are the ith coordinate of the initial and final positions of the transition), $A^* = A + B_1$ and $B^* = B_2$, where $\dot{x} = Ax + Bu = Ax + B_1 x + B_2$.

See Fig. 3.1 for illustration of the variables of Theorem 3.1 in 2 dimensions.

Proof of Theorem 3.1. T^{max} - the maximum transition time for $\pi_k \to \pi_l$ in a system following the linear dynamics (4.1) is determined by considering the minimum

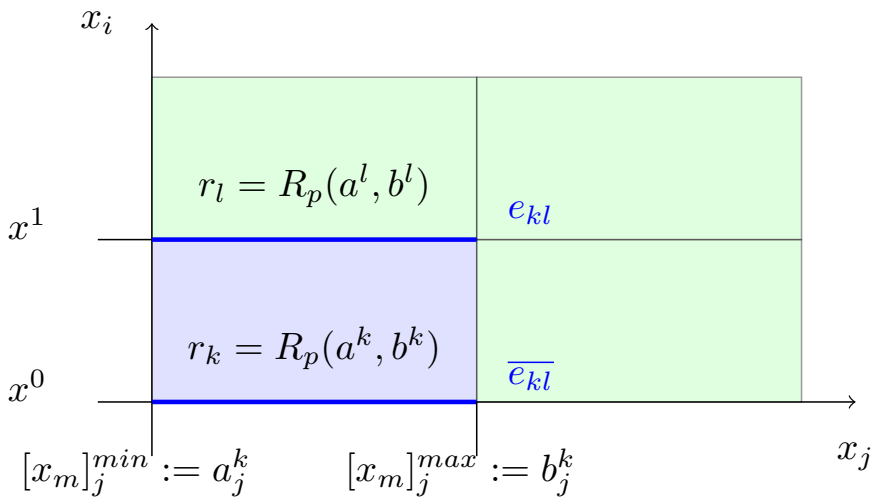


Figure 3.1: Illustration of the variables in Theorem 3.1 in 2 dimensions.

transition speed. Consider the dynamics projected onto the direction of the transition i, i.e

$$\begin{aligned} \dot{x}_i &= [Ax + Bu]_i, \\ x(0)_i &= x_i^0 = x^0, \\ x(t_1)_i &= x_i^1 = x^1, \end{aligned} \tag{3.8}$$

where x^0 is the ith coordinate of some point on the edge $\overline{e}_{kl}$, and x^1 is the ith coordinate of some point on the edge e_{kl}. Since $Bu = B_1 x + B_2$, system (3.8) can be rewritten to (3.9), by introducing $A^* = A + B_1$ and $B^* = B_2$.

$$\begin{aligned} \dot{x}_i &= A_{ii}^* x_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} A_{ij}^* x_j + B_i^* \\ & x(0)_i = x^0 \\ & x(t_1)_i = x^1 \end{aligned} \tag{3.9}$$

The maximum transition time is determined by solving (3.9) for t_1. The equation can be solved by separating x_i from t, if and only if $A_{ij}^* x_j$ is a constant $\forall j$. Since A_{ij}^* is a constant this holds if and only if $\dot{x}_j = 0$ or $A_{ij}^* = 0$. Otherwise, the maximum transition time can be overestimated by considering the minimum transition speed $\dot{x}_i^{min}$, at every point in π_k, which can be determined by considering the limits of x_j

in π_k, namely a_j^k and b_j^k

$$\min_{x_j \in \pi_k} \left(A_{ij}^* x_j\right) = \begin{cases} A_{ij}^* a_j^k & \text{if } A_{ij}^* > 0 \\ A_{ij}^* b_j^k & \text{if } A_{ij}^* < 0 \end{cases} \tag{3.10}$$

The maximum transition time, denoted T^{max}, can then be overestimated as the solution to

$$\begin{aligned} \dot{y} &= A_{ii}^* y + C^* + B_i^* \\ y(0) &= x^0 \\ y(T^{max}) &= x^1 \end{aligned} \tag{3.11}$$

where

$$C^* = \min_{x_j \in \pi_k} \sum_{\substack{j=1 \\ j \neq i}}^{n} A_{ij}^* x_j = \sum_{\substack{j=1 \\ j \neq i}}^{n} \min_{x_j \in \pi_k} A_{ij}^* x_j$$

the latter can be solved as follows:

$$\begin{aligned} \frac{dy}{dt} &= A_{ii}^* y + C^* + B_i^* \implies \\ \int dt &= \int \left(\frac{1}{A_{ii}^* y + C^* + B_i^*} \right) dy \implies \\ t + t_c &= \frac{\ln(A_{ii}^* y + C^* + B_i^*)}{A_{ii}^*} \end{aligned} \tag{3.12}$$

Now, $y(0) = x^0$ yields

$$t_c = \frac{\ln(A_{ii}^* x^0 + C^* + B_i^*)}{A_{ii}^*} \tag{3.13}$$

and $y(T^{max}) = x^1$ yields

$$T^{max} + t_c = \frac{\ln(A_{ii}^* x^1 + C^* + B_i^*)}{A_{ii}^*} \tag{3.14}$$

and hence

$$T^{max} = \ln \left(\frac{A_{ii}^* x^1 + C^* + B_i^*}{A_{ii}^* x^0 + C^* + B_i^*} \right) \frac{1}{A_{ii}^*} \tag{3.15}$$

□

Remark 3.1. If $C^* = 0$ or $\dot{x}_j = 0 \; \forall j$, then T^{max} is the maximal time required for the transition to occur. Otherwise T^{max} is an over-approximation.

3.3.3 Determining the Components of the WTS

- The set of states $\Pi = \{\pi_0, \pi_1, ..., \pi_{m_\Pi}\}$ of the WTS is defined as the set of rectangles $\mathcal{R} = \{R_p(a^0, b^0), R_p(a^1, b^1), ..., R_p(a^{m_\Pi}, b^{m_\Pi})\}$, fulfilling the requirements described in Chapter 3.3.1.
- The definition of the initial state Π_{init}, and labelling L follows directly:

$$\Pi_{init} = \{\pi_i \in \Pi | x_m^0 \in R_p(a^i, b^i)\} \tag{3.16}$$

$$L(\pi_i) = a_j \in 2^{AP} : ap_k = True \ \ \forall x \in R_p(a^i, b^i), \forall ap_k \in a_j \tag{3.17}$$

- Transitions between states are determined based on shared edges and whether a control input which enforce the necessary motion could be found by following (3.5) in Chapter 3.3.2, and the actions are determined as said control inputs:

$$\begin{aligned} \pi_i \to \pi_j \qquad & \text{iff } R_p(a^i, b^i) \text{ and } R_p(a^j, b^j) \text{ have a common edge, and} \\ & \exists \sigma : \forall x(t) \in R_p(a^i, b^i), x(t + \Delta t) \in R_p(a^j, b^j) \text{ for some } \Delta t, \\ \Sigma = \{\sigma = u(\pi_i, \pi_j), \pi_i, \pi_j \in \Pi\} \end{aligned}$$

- The weights are defined as the maximum transitions times found in Chapter 3.3.2:

$$w(\pi_i, \pi_j) = T^{max}(\pi_i, \pi_j) \text{ where } (\pi_i, \sigma, \pi_j) \in \to . \tag{3.18}$$

  for $\sigma = u(\pi_i, \pi_j)$.

3.4 Simulations

Two agents moving in the workspace illustrated in Figure 3.2a (6 rooms and a corridor) was considered. The resulting partitioning is presented in Figure 3.2b. (The encircled numbers represent initial positions and states.) For the remainder of this example we will consider only agent 1.

Assuming that the agent follows the dynamics in (3.19), the resulting WTS is illustrated by Figure 3.3.

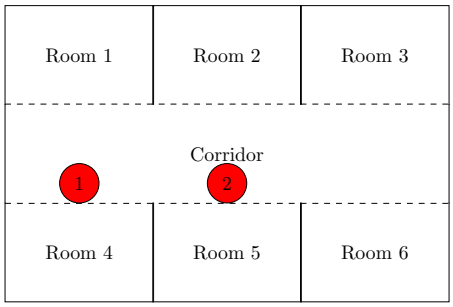
$$\dot{x} = \begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix} x + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} u \tag{3.19}$$

$$|u_i| \leq 20 \qquad i = 1, 2 \tag{3.20}$$

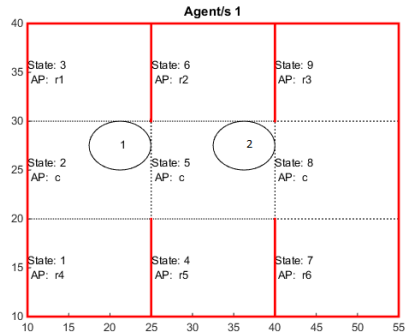
The values of u_i, $\forall i = 1, ..., 16$ and w_{ij}, $\forall i, j = 1, ..., 9$ in Figure 3.3 are given in Table 3.1. The complete construction of the WTS were performed in Matlab (applying the method suggested in Chapter 3.3) on a laptop with a Core i7-6600U 2.80 GHz processor, the runtime was 0.653s.

Table 3.1: Values of u_i, w_{jk} and the maximum absolute values of the components of u_i (to compare with $u_{max} = 20$) where $\pi_j \rightarrow^{u_i} \pi_k$

i	u_i	$\max(\lvert u_{i1}\rvert)$	$\max(\lvert u_{i2}\rvert)$	j	k	w_{jk}
1	$\begin{bmatrix} -20.33 & 7.36 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 9.45 \\ -20 \end{bmatrix}$	19.31	20	3	2	0.077075
2	$\begin{bmatrix} -13.28 & 6.39 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 14.52 \\ -20 \end{bmatrix}$	19.41	20	6	5	0.077075
3	$\begin{bmatrix} -6.43 & 4.24 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 2.78 \\ -20 \end{bmatrix}$	19.85	20	9	8	0.077075
4	$\begin{bmatrix} -16.72 & 6.20 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 17.77 \\ 20 \end{bmatrix}$	17.83	20	1	2	0.043506
5	$\begin{bmatrix} -12.70 & 0.12 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 35.50 \\ 20 \end{bmatrix}$	15.17	20	4	5	0.043506
6	$\begin{bmatrix} -5.91 & -0.76 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 18.54 \\ 20 \end{bmatrix}$	15.47	20	7	8	0.043506
7	$\begin{bmatrix} -20.66 & 6.53 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 19.78 \\ 20 \end{bmatrix}$	18.80	20	2	3	0.040021
8	$\begin{bmatrix} -17.22 & 6.34 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 12.92 \\ -20 \end{bmatrix}$	17.44	20	2	1	0.066766
9	$\begin{bmatrix} -13.86 & 7.01 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 22.19 \\ 20 \end{bmatrix}$	19.21	20	5	6	0.040021
10	$\begin{bmatrix} -12.55 & 7.11 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 16.74 \\ -20 \end{bmatrix}$	19.24	20	5	4	0.066766
11	$\begin{bmatrix} -7.62 & 1.71 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 20.86 \\ 20 \end{bmatrix}$	17.64	20	8	9	0.040021
12	$\begin{bmatrix} -16.26 & 5.08 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 59.44 \\ -20 \end{bmatrix}$	19.80	20	8	7	0.066766
13	$\begin{bmatrix} 0 & 0 \\ 1.33 & -12.75 \end{bmatrix} x + \begin{bmatrix} 20 \\ 25.92 \end{bmatrix}$	20	11.01	2	5	0.58892
14	$\begin{bmatrix} 0 & 0 \\ 5.69 & -17.15 \end{bmatrix} x + \begin{bmatrix} 20 \\ 19.54 \end{bmatrix}$	20	17.70	5	8	0.52680
15	$\begin{bmatrix} 0 & 0 \\ 7.60 & -23.95 \end{bmatrix} x + \begin{bmatrix} -20 \\ 33.55 \end{bmatrix}$	20	19.28	5	2	0.143841
16	$\begin{bmatrix} 0 & 0 \\ 8.12 & -23.33 \end{bmatrix} x + \begin{bmatrix} -20 \\ 18.06 \end{bmatrix}$	20	19.44	8	5	0.202733



(a) Illustration of the simulation example.



(b) Partition constructed by the MATLAB simulation. The circles represents the initial states of each agent.

Figure 3.2

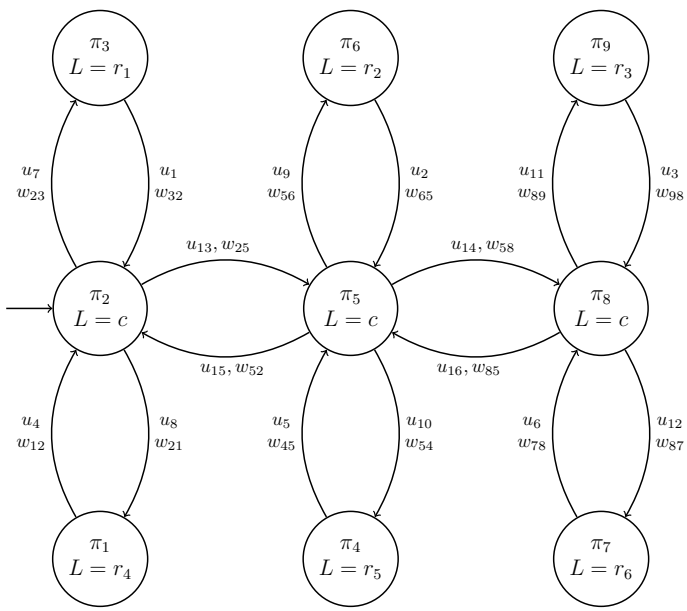


Figure 3.3: Resulting WTS for agent 1.

3.5 Conclusion

In this chapter we presented a method to abstract a rectangular workspace and affine dynamics into a WTS with weights corresponding to the minimal guaranteed transition time. The method was first suggested as part of [55]. We will use the abstraction method throughout the thesis.

Chapter 4

Control Synthesis for Multi-Agent Systems under Hard MITL Tasks

In this chapter we consider a basic control synthesis problem for a multi-agent system under MITL specifications. We consider both individual tasks and cooperative tasks, and suggest a centralized automata-based solution. The work is part of [55].

4.1 Introduction

The control synthesis problem for multi-agent systems, has been widely researched in [6, 49–51, 54] among others. In this chapter we consider finite tasks under time constraints which are both individual and cooperative. We express the tasks with MITL formalism and assumes that each MITL formula can be translated into a TBA. The translation has been discussed in several papers ([29], [28], [30], [31] etc.) and it has been concluded that all MITL formulas can be translated into a TBA. In this work the translation itself has been performed manually. We will use motion planning as an example throughout, but the theory and methods only require a known state space which suits many other implementations. Considering the case of motion planning the atomic propositions used in the temporal logic specifications include labels of regions and cooperative tasks such as multiple agents meeting at a specific region or visiting different regions at the same time etc.

The approach to solution suggested in this paper follows similar principles as in [6], but here we present alternative definitions of the local BWTS, the product BWTS and the global BWTS. The definitions suggested here requires a smaller number of states and hence, a lower computational demand. The drawback of the suggested definitions is an increased risk of a false negative result and a required modification to the applied graph-search-algorithm. However, this will have no effect on the fact that the method is correct-by-construction. The method, in its entirety, has been implemented in simulations, demonstrating the satisfaction of the specifications through the resulting controller. The contribution of this chapter can be summarized as follows; it provides for a less computationally demanding

alternative than previous methods, and simulation results which support the claims are included.

4.2 Problem Statement

Consider N agents in a bounded workspace $W \subset \mathbb{R}^n$, governed by the dynamics

$$\begin{aligned} \dot{x}_i &= A_i x_i + B_i u_i, \;\; i = 1, ..., N, \\ x_i(0) &= x_i^0, x_i \in W \end{aligned} \tag{4.1}$$

The problem considered in this chapter consists in synthesizing N controllers, $u_i, i = 1, .., N$, such that each agent satisfies a local individual MITL formula ϕ_i over the set of atomic propositions AP_i. At the same time, the team of agents should satisfy a team specification MITL formula ϕ_G over the set of atomic propositions $AP_G = AP_1 \times ... \times AP_N$. Before we can formally state the problem we need to define the concept of a collective run. In motion-planning, the movement of an agent can be described by a timed run. For the multi-agent case, the movement of all agents can be collectively described by a collective run. The definition is

Definition 4.1. [6] The *collective timed run* $r_G = (r_G(0), \tau_G(0))(r_G(1), \tau_G(1))...$ of N agents, is defined as follows

- $(r_G(0), \tau_G(0)) = (r_1(0), ..., r_N(0), \tau_G(0))$
- $(r_G(i+1), \tau_G(i+1)) = (r_1(j_1), ..., r_N(j_N), \tau_G(i+1))$, for $i \geq 0$ where $(r_G(i), \tau_G(i)) = (r_1(i), ..., r_N(i), \tau_G(i))$ and

$$\begin{aligned} & l = \underset{k \in I}{\operatorname{argmin}}\{\tau_k(i_k + 1)\}, \\ & \tau_G(i+1) = \tau_l(i_l + 1), \\ & r_k(j_k) = \begin{cases} r_l(i_l + 1), & \text{if } k = l \\ r_k(i_l), & \text{otherwise.} \end{cases} \end{aligned}$$

We can now state the problem as:

Problem 4.1. Synthesize a sequence of individual timed runs $r_1^t, ..., r_N^t$ such that:

$$(r_G \models \phi_G) \wedge \left(r_1^t \models \phi_1 \wedge ... \wedge r_N^t \models \phi_N\right) \tag{4.2}$$

where r_G is the collective run.

Remark 4.1. Initially it might seem that if a run r_G that satisfies the conjunction of the local formulas i.e., $r_G \models r_1^t \wedge \ldots \wedge r_N^t$ can be found, then the Problem 4.1 is solved in a straightforward centralized way. This does not always hold. A counterexample was given in [6] showing that:

$$r_G^t \models \bigwedge_{k \in \mathcal{I}} \varphi_k \not\Leftrightarrow r_1^t \models \varphi_1 \wedge \ldots \wedge r_N^t \models \varphi_N. \tag{4.3}$$

4.3 Control Strategy

The suggested solution approach is:

1. For each agent, abstract the continuous-time linear system (4.1) into a WTS as described in Chapter 3.

2. For each agent, construct a local BWTS out of its WTS and a TBA representing the local MITL specification. The accepting timed runs of the local BWTS satisfy the local specification (Chapter 4.3.1).

3. Construct a product BWTS out of the local BWTSs. The accepting timed runs of the product BWTS satisfy all local specifications (Chapter 4.3.2).

4. Construct a global BWTS out of the product BWTS and the TBA representing the global MITL specification. The accepting runs of the global BWTS satisfy both the global specification and all local specifications (Chapter 4.3.3).

5. Determine the control input by applying a graph-search algorithm to find an accepting run of the global BWTS and projecting this accepting run onto the individual WTSs (Chapter 4.3.4).

4.3.1 Büchi Weighted Transition System

The definition of the product automaton of a WTS and a TBA was given in Definition 2.6 in Chapter 2. We use that definition directly to construct a BWTS. We denote an accepting run of a BWTS to be a sequence of states: $q_0 ... q_m$ such that $q_0 \in Q^{init}$, $q_m \in \mathcal{F}$, and $(q_i, q_{i+1}) \in \rightsquigarrow \forall i = 0, ..., m-1$.

It follows from the construction that the projection of an accepting run of a BWTS onto the TBA used in the product, is an accepting run of the TBA. It also follows that a run of a BWTS is accepting if the projection onto the TBA is accepting. The later holds since the WTS has no impact on acceptability.

4.3.2 Product Büchi Weighted Transition System

The definition of a product BWTS, the product between several BWTSs, is given below in Defintion 4.2.

Definition 4.2. Given N BWTSs $P_1, ..., P_N$, defined as in 2.6, the *product BWTS* is defined as: $P_L = P_1 \otimes ... \otimes P_N = (Q_1, Q_1^{init}, \mathcal{F}_1, \hat{\Sigma}_1, \mathcal{AP}_1, \mathcal{L}_1, \mathcal{X}_1, I_{\mathcal{X}_1}, \rightsquigarrow_1, \hat{w}_1) \otimes ... \otimes (Q_N, Q_N^{init}, \mathcal{F}_N, \hat{\Sigma}_N, \mathcal{AP}_N, \mathcal{L}_N, \mathcal{X}_N, I_{\mathcal{X}_N}, \rightsquigarrow_N, \hat{w}_N) = (Q_L, Q_L^{init}, \mathcal{F}_L, \hat{\Sigma}_L, \mathcal{AP}_L, \mathcal{L}_L, \mathcal{X}_L, I_{\mathcal{X}_L}, \rightsquigarrow_L, \hat{w}_L)$, where $Q_L \subseteq Q_1 \times ... \times Q_N$ is a set of states, $Q_L^{init} = Q_1^{init} \times ... \times Q_N^{init}$ is a set of initial states, $\mathcal{F}_L = \{q_L = (q_1, ..., q_N) \in Q_L$ s.t. $q_k \in \mathcal{F}_k, \forall k = 1, ..., N\}$ is a set of accepting states, $\hat{\Sigma}_L = \hat{\Sigma}_1 \times ... \times \hat{\Sigma}_N$ is a set of actions, $\mathcal{AP}_L = \mathcal{AP}_1 \times ... \times \mathcal{AP}_N$ is a set of atomic propositions, $\mathcal{L}_L((q_1, ..., q_N)) = \mathcal{L}_1(q_1) \times ... \times \mathcal{L}_N(q_N)$ is a labelling function mapping states to actions, $\mathcal{X} = \{\mathcal{X}_1, ..., \mathcal{X}_N\}$ is a

set of clocks, $I_{\mathcal{X}}(q_1, ..., q_N) = \bigcup_{k=1}^{N} I_{\mathcal{X}_{\|}}(q_k)$ is a a mapping of clock constraints onto states, $\rightsquigarrow_L$ is a set of transitions where $(q_L, q'_L) \in \rightsquigarrow_L$ iff

- $q_L = (q_1, ..., q_N), q'_L = (q'_1, ..., q'_N) \in Q_L$, and
- $(q_k, q'_k) \in \rightsquigarrow_k, \forall\ k = 1, ..., N$,

and $\hat{w}_L(q_L, q'_L) = \max_{i=1,...,N}(\hat{w}_i(q_i, q'_i))$, if $(q_L, q'_L) \in \rightsquigarrow_L$,, where $q_L = (q_1, ..., q_N)$ and $q'_L = (q'_1, ..., q'_N)$ is a weight function mapping weight constants onto transitions.

We denote an accepting run of a product BWTS to be a sequence of states: $q_L^0, ..., g_L^m$ such that $q_L^0 Q_L^{init}$, $g_L^m \in \mathcal{F}_L$, and $(q_L^i, q_L^{i+1}) \in \rightsquigarrow_L\ \forall i = 0, ..., m-1$.

It follows from the construction that the projection of an accepting run of the product BWTS onto any of the BWTSs used in the product, is an accepting run of that BWTS. However, it can not be concluded that a run of a product BWTS is accepting based on the knowledge that the projection of the run onto one of the used BWTSs is an accepting run of said BWTS. This is due to the fact that the remainder of the BWTSs affects the acceptability. It can be concluded that a run of a product BWTS is accepting iff the projections of the run onto each BWTS are accepting.

4.3.3 Global Büchi Weighted Transition System

Two definitions of a global BWTS, a product of a product BWTS and a global TBA, are given below in Definitions 4.3 and 4.4. The first definition is taken directly from our paper [55], with some updated notation. The second definition is the one we suggest using if considering finite tasks, which we do in this thesis. It is a simplified version of the first, where some content which is unnecessary when finite tasks are considered has been removed. In particular, this includes the resetting of clocks (Z) and the need of multiple visits to an acceptable state (l).

We use global TBA to denote that the TBA represents a global MITL formula. The global TBA is hence defined as the ordinary TBA, however the set of actions AP_G (also known as the set of atomic propositions considered by the MITL specification) must correlate to the labelling of the environment of all agents, while the previously considered TBAs (or local TBAs) only consider the labelling of the environment of the corresponding agent.

Definition 4.3. Given a product BWTS $P_L = (Q_L, Q_L^{init}, \mathcal{F}_L, \hat{\Sigma}_L, \mathcal{AP}_L, \mathcal{L}_L, \mathcal{X}_L, I_{\mathcal{X}_L}, \rightsquigarrow_L, \hat{w}_L)$ and a global TBA $A_G = (S_G, S_G^{init}, F_G, AP_G, X_G, I_{X_G}, E_G)$ their *global BWTS* is defined as: $P_G = P_L \otimes A_G = (Q_G, Q_G^{init}, \mathcal{F}_G, \hat{\Sigma}_G, \mathcal{AP}_G, \mathcal{L}_G, \mathcal{X}_G, I_{\mathcal{X}_G}, \rightsquigarrow_G, \hat{w}_G)$ where $\hat{Q}_G = Q_L \times S_G \times Z_0 \times ... \times Z_N \times \{1, 2\}$ is a set of states, where $Z_i = \{z_1^i, ..., z_{m_{\mathcal{X}_i}}^i\}$ for $i = 1, ..., N$ and $Z_0 = \{z_1^0, ..., z_{m_{X_G}}^0\}$, $\hat{Q}_G^{init} = Q_G^{init} \times S_G^{init} \times \{1, .., 1\} \times ... \times \{1, ..., 1\}$ is a set of initial states, where $\{1, ..., 1\} \times ... \times \{1, ..., 1\}$ consists of $N + 1$ sets, where the first set contains m_{X_G} ones, and the remaining sets contains $m_{\mathcal{X}_i}$ ones each,

$\mathcal{F}_G = \{(q_L, s, Z_0, ..., Z_N, 1) \in \hat{Q}_G : q_L \in \mathcal{F}_L \text{ and } s \in F_G\}$ is a set of accepting states, $\hat{\Sigma}_G = \hat{\Sigma}_L$ is a set of actions, $\mathcal{AP}_G = \mathcal{AP}_L$ is a set of atomic propositions, $\mathcal{L}_G(q_L, s, Z_0, ..., Z_N, l) = \mathcal{L}_L(q_L)$ is a labelling function, $\mathcal{X}_G = \mathcal{X}_L \times X_G$ is a set of clocks, $I_{\mathcal{X}_G}(q_L, s, Z_0, ..., Z_N, l) = I_{\mathcal{X}_L}(q_L) \cup I_{X_G}(s)$ is a mapping of clock constraints onto states, $\rightsquigarrow_G$ is a set of transitions where $(q_G, q'_G) \in \rightsquigarrow_G$ iff

- $q_G = (q_L, s, Z_0, ..., Z_N, l), q'_G = (q'_L, s', Z'_0, ..., Z'_N, l') \in \hat{Q}_G$,
- $(q_L, q'_L) \in \rightarrow_L$,
- $\exists g_L, a$ s.t.

  $(s, g, a, s') \in E_G$ where $a = \mathcal{L}_L(q'_L)$,
  For all $i \in \{1, ..., N\}$, Z_i and Z'_i are such that

$$\begin{array}{ll} z_k^i = 0 \text{ and } z_k^{i'} = 1, & \text{if } (q, q') \in c_k^i \\ z_k^{i'} = z_k^i, & \text{otherwise} \end{array}$$

  Z_0 and Z'_0 are such that

$$z_k^0 = \begin{cases} 0 & \text{if } x_k \in R \\ 1 & \text{otherwise} \end{cases}$$

$$z_k^{0'} = \begin{cases} 1 & \text{if } x_k \in R \\ z_k^0 & \text{otherwise} \end{cases}$$

$$l' = \begin{cases} 1, & \text{if } l = 1 \text{ and } q \in F_G \\ & \text{or } l = 2 \text{ and } s \in \mathcal{F}_G \\ 2, & \text{otherwise} \end{cases}$$

and $\hat{w}_G((q_L, s, Z_0, ..., Z_N), (q'_L, s', Z'_0, ..., Z'_N)) = \hat{w}_L(q_L, q'_L)$ iff $((q_L, s, Z_0, ..., Z_N), (q'_L, s, Z'_0, ..., Z'_N)) \in \rightsquigarrow_G$ is a weight function mapping a weight constant onto each transition.

Definition 4.4. Given a product BWTS $P_L = (Q_L, Q_L^{init}, \mathcal{F}_L, \hat{\Sigma}_L, \mathcal{AP}_L, \mathcal{L}_L, \mathcal{X}_L, I_{\mathcal{X}_L}, \rightsquigarrow_L, \hat{w}_L)$ and a global TBA $A_G = (S_G, S_G^{init}, F_G, AP_G, X_G, I_{X_G}, E_G)$ their *global BWTS* is defined as: $P_G = P_L \otimes A_G = (Q_G, Q_G^{init}, \mathcal{F}_G, \hat{\Sigma}_G, \mathcal{AP}_G, \mathcal{L}_G, \mathcal{X}_G, I_{\mathcal{X}_G}, \rightsquigarrow_G, \hat{w}_G)$ where $\hat{Q}_G = Q_L \times S_G$ is a set of states, $\hat{Q}_G^{init} = Q_G^{init} \times S_G^{init}$ is a set of initial states, $\mathcal{F}_G = \{(q_L, s) \in \hat{Q}_G : q_L \in \mathcal{F}_L \text{ and } s \in F_G\}$ is a set of accepting states, $\hat{\Sigma}_G = \hat{\Sigma}_L$ is a set of actions, $\mathcal{AP}_G = \mathcal{AP}_L$ is a set of atomic propositions, $\mathcal{L}_G(q_L, s) = \mathcal{L}_L(q_L)$ is a labelling function, $\mathcal{X}_G = \mathcal{X}_L \times X_G$ is a set of clocks, $I_{\mathcal{X}_G}(q_L, s) = I_{\mathcal{X}_L}(q_L) \cup I_{X_G}(s)$ is a mapping of clock constraints onto states, $\rightsquigarrow_G$ is a set of transitions where $(q_G, q'_G) \in \rightsquigarrow_G$ iff

- $q_G = (q_L, s), q'_G = (q'_L, s') \in \hat{Q}_G$,
- $(q_L, q'_L) \in \rightarrow_L$,

- $\exists g, a$ s.t. $(s, g, a, s') \in E_G$ where $a = \mathcal{L}_L(q'_L)$,

and $\hat{w}_G((q_L, s), (q'_L, s')) = \hat{w}_L(q_L, q'_L)$ iff $((q_L, s), (q'_L, s)) \in \rightsquigarrow_G$ is a mapping of weights onto transitions.

We denote an accepting run of a global BWTS to be a sequence of states: $q_G^0, ..., g_G^m$ such that $q_G^0 \in Q_G^{init}$, $q_G^m \in \mathcal{F}_G$, and $(q_G^i, g_G^{i+1}) \in \rightsquigarrow_G \ \forall i = 0, ..., m-1$.

It follows that the projection of an accepting run of a global BWTS onto the product BWTS or onto the global TBA is accepting. As for the product BWTS, it holds that it can only be concluded that a run of a global BWTS is accepting if both the projection onto the product BWTS and the projection onto the global TBA are accepting.

4.3.4 Control Design by Graph Search and Projection

Given a global BWTS, a Dijkstra algorithm (Alg. 1) can be applied to find an accepting run. If the cost function of the Dijkstra algorithm is defined as the weights $\hat{w}_G$, it follows that the first accepting run to be found corresponds to the minimum weight summation, i.e. the minimum time which guarantees that all agents have completed their runs. If no accepting run is found, it can *not* be concluded that there doesn't exist any set of paths which would satisfy all specifications. This is due to the overestimations which have been used throughout the synthesis to enforce the guarantee that a run that is produced is satisfying. These overestimations include method to determine control inputs and transition times in the abstraction as well as the definitions of the weight functions of the product BWTS and the global BWTS.

If an accepting run of the global BWTS is found, it can be projected onto the product BWTS, to determine an accepting run of the product BWTS which can be projected onto the BWTSs, and so on until a set of discrete paths of the WTSs have been determined. It then holds that all MITL specifications will be satisfied if each agent follows its discrete path.

4.4 Simulations

We consider two agents with the dynamics:

$$\dot{x} = \begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix} x + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} u \tag{4.4a}$$

$$\dot{x} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x + \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} u \tag{4.4b}$$

Moving around the workspace described in Chapter 3. Note that agent 1 is the example used in the simulation for the abstraction.

Each agent is assigned with the local MITL formula $\phi_L = \Diamond_{0.1} r_2 \wedge r_2 \rightarrow \Diamond_{0.3} r_6$ ('Eventually, within 0.1 time units, the agent must be in room 2, and if the agent enters room 2 it must then enter room 6 within 0.3 time units.'). Furthermore, they

Table 4.1: The maximum transition times (T^{max}) which were approximated as in Chapter 3, and the actual transition times (T). The actual times are defined as $T = \max(T_1, T_2)$, where T_i is the time agent i requires to complete the transition.

Position[1]	Agent 1	Agent 2	T^{max}	T
0	2	5	0	0
1	5	6	0.0589	0.0368
2	6	6	0.04	0.026[2]
3	5	5	0.0771	0.0212
4	8	8	0.0645	0.0403
5	7	7	0.0668	0.0551
6	8	8	0.0465	0.0151
7	5	5	0.2027	0.1115
8	2	6	0.1438	0.1366
9	3	6	0.04	0.027[2]

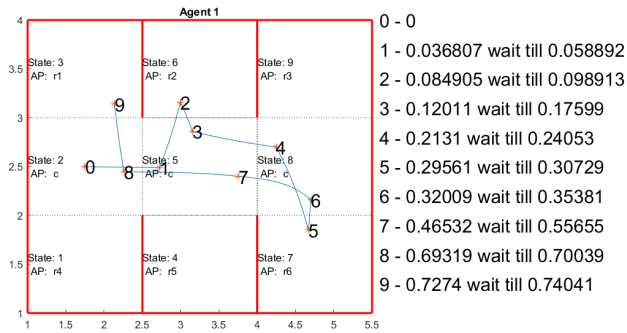
are assigned with the global MITL formula $\phi_G = \Diamond_1(a_1 = r_1 \wedge a_2 = r_2)$ ('Eventually, within 1 time units, agent 1 must be in room 1 and agent 2 must be in room 2, at the same time.').

The WTS for agent 1 was determined in Chapter 3 , and the WTS of agent 2 has an identical structure, the differences being the initial state and the values of the control inputs and weights due to the change of dynamics. The resulting WTSs have 9 states. The local MITL formulas can be represented by TBAs of 4 states, and the resulting BWTSs hence have 36 states each. The product BWTS consists of $(|Q_1| \cdot |Q_2|) = 1296$ states while the global BWTS consists of $2 \cdot (|Q_{pBTWS}| \times |Q_{gTBA}| \times 2^{m_{X_1}} \times 2^{m_{X_2}} \times 2^{m_{X_G}}) = 248832$ states.

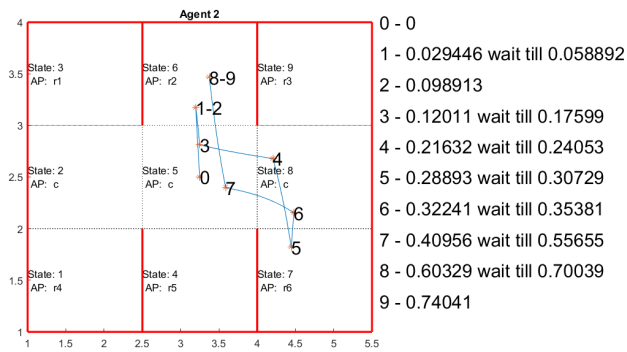
The projection of the found accepting run onto each WTS, yielded $[2, 5, 6, 5, 8, 7, 8, 5, 2, 3]$ and $[5, 6, 6, 5, 8, 7, 8, 5, 6, 6]$, for the respective agent. The result is visualized in Figure 4.1, which shows the evolution of each closed-loop system for the given initial positions. The figure was constructed by implementing the built-in function *ode45* for the determined closed-loop system in each state with the initial position equal to the last position of the former transition. The switching between controllers is performed based on the position of the agent; namely the switching from controller u_{ij} to u_{jk} is performed when the agent has entered into state j and been there for 5 iterations of *ode45*. The estimated time distances for each joined transition are given in Table 4.1.

[1] Numbered in order of transitions, see Figure 4.1.

[2] These transitions require agent 2 to stay in place, hence the actual time is here defined as the time agent 1 requires to complete the transition.



(a) Agent 1



(b) Agent 2

Figure 4.1: Illustration of the paths of each agent in the example. The numbers 0-9 represent the end of each joined transition. The actual arrival time at each location as well as the time the agent is required to wait till the worst case transition time has been reached (and it is guaranteed that all other agents have transitioned), is noted to the right of the figure. The time the agent has to wait till corresponds to the worst case estimation of the required transition time and is due to the requirement that the agents make transitions simultaneously. It is notable that both agents finish all transitions on less time than the worst case estimation. Hence, the waiting time can be further cut by allowing the agents to communicate to each other when a transition is done.

That is, the worst case transition times yields;

At position 0	Agent 1 and Agent 2 begins at their respective initial position in the corridor
At position 1	Agent 2 enters room 2 within 0.0589 time units from start
At position 2	Agent 1 enters room 2 within 0.0989 time units from start
At position 5	Agent 1 and Agent 2 enters room 6 within 0.2084 and 0.2484 time units respectively from entering room 2
At position 9	Agent 1 is in room 1 while Agent 2 is in room 2 within 0.7404 time units from start.

From this, it is clear that the given path will satisfy the MITL formulas.

4.4.1 Complexity

The proposed framework requires at most

$$|P_G| = \prod_{i=1}^{N} (|T_i| \times |A_i| \times 2^{M_i}) \times |A_G| \times 2^{M_G} \times 2 \tag{4.5}$$

where $M_i = |X_i|$ and $M_G = |X_G|$, i.e. the number of clocks associated with each formula.

The simulation presented in this Chapter was run in MATLAB on a laptop with a Core i7-6600U 2.80 GHz processor, the runtime was approximately 30min (including the abstraction).

4.5 Conclusion

A correct-by-construction framework to synthesize a controller for a multi-agent system following continuous linear dynamics such that some local MITL formulas as well as a global MITL formula are satisfied, has been presented. The method is supported by result of the simulations in the MATLAB environment. Future work includes communication constraints between the agents.

The framework does not guarantee that any existing solution is found, and it does not provide any feedback on why a solution is missing or how one could be found. In the upcoming chapters this issue will be investigated.

Chapter 5

Control Synthesis for Single-Agent Systems under Soft MITL Tasks

In this Chapter we aim to resolve the issue from the previous Chapter, i.e. what can we do when we're not able to find an accepting run. We approach the issue by considering the specifications as soft constraints and suggest a framework for finding a least violating run by introducing a novel metric, hybrid distance, which quantifies the violation. The content is part of the work we presented in [56].

5.1 Introduction

The idea of control synthesis is to use the methods from model checking to design controllers which are guaranteed to satisfy some safety constraints. While achieving this guarantee is the very purpose behind the approach, it is also one of the main disadvantages with it, in the sense that a failure to find a satisfying solution gives no guarantees. More specifically, no feedback is given when a failure occur, and hence we can not conclude whether a solution exists which we failed to find (due to overestimations) or if the task is infeasible (no solution exists). There's also no information given regarding in what way the specification was infeasible. As a consequence, applying the standard synthesis leaves us with no clear ideas of how to continue when a failed result is given. As mentioned before there are three mayor ways of handling a lack of a solution; i) minimize the overestimations by reconsidering the abstraction [32], ii) change the formula to something that is feasible [33], or iii) relax the notion of satisfiability [7, 35, 36]. Here, we take the third route by introducing the metric hybrid distance and considering the solution with the lowest hybrid distance as the least violating run. It follows that if the hybrid distance is non-zero, we will lose the guarantee of complete satisfiability. However, by evaluating the value of the distance we receive information of how big the violation is and of what type it is, more on this below.

In this chapter we also introduce a human-in-the-loop to the system. The interaction and cooperation between human and robot is crucial in any situation

where they are either present in the same workspace or the tasks performed would benefit from co-piloting. To design systems which would fit in society (for commercial use, industry work etc) the co-existing with humans has to be considered on some level. Here, we introduce the human in a simple cooperation framework where the human gives feedback on the type of violation of suggested path by direct input to the hybrid distance function.

The contribution of this chapter can be summarized in three parts; i) introducing the novel metric hybrid distance which quantifies the violation of a run with respect to a MITL formula, ii) constructing a method to use the hybrid distance in the control synthesis framework, allowing us to find a least violating plan, and iii) introducing a simple human-in-the-loop interface where the human can give direct feedback on violation. The methods are validated in simulations.

5.2 Preliminaries and Notation

Before we can construct a problem formulation we need to expand on our notation and preliminary knowledge. In this Chapter we will consider some categorization of MITL formulas, and introduce hybrid distance and how it can be added to a TBA.

5.2.1 Categorization of Operators

The operators $\mathcal{U}_I$, $\Diamond_I$ and $\Box_I$, are bounded by the interval I. $I = [a, b]$, indicates that the operator should be satisfied from time a to time b. If $b \neq \infty$, this implies that the operator is subject to some deadline. We will denote these as *temporally bounded* operators. All operators that are not included in the set of temporally bounded operators, will be called *non-temporally bounded* operators from here on. The operator $\mathcal{U}_I$ is temporally bounded but contains a non-temporally bounded part in the sense that a deadline is associated with the second part of the formula (it must be true at some time $t \in I$), while no such deadline exists for the first part (it must hold at least until the second part becomes true). When we use the term *violating non-temporally bounded operators*, we refer to the non-temporally bounded part of an operator being violated. An example of this is $\phi = A\mathcal{U}_{\leq T}B$, indicating that A must hold until B holds, and that B must hold within T time units. Here, the non-temporally bounded operator is violated if $\neg A$ becomes true before B has become true, while the temporally bounded operator is violated if time T is exceeded before B becomes true. A formula ϕ which contains a temporally bounded operator will be called a temporally bounded formula. The same holds for non-temporally bounded formulas. An MITL specification ϕ can be written as $\phi = \bigwedge_{i\in\{1,2,\ldots,n\}} \phi_i = \phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n$ for some $n > 0$ and some sub formulas ϕ_i. In this paper, the notation sub formulas ϕ_i of ϕ, refers to the set of sub formulas which satisfies $\phi = \bigwedge_{i\in\{1,2,\ldots,n\}} \phi_i$ for the largest possible choice of n such that $\phi_i \neq \phi_j \ \forall i \neq j$. For each sub formula ϕ_i, there are 3 possible temporal outcomes if ϕ_i is temporally bounded: ϕ_i is satisfied, ϕ_i is violated, or the satisfaction of ϕ_i is uncertain.

Table 5.1: Operators categorized according to the *temporally bounded/non-temporally bounded* notation and Definition 5.1.

Operator	$b = \infty$	$b \neq \infty$
$\Box_{[a,b]}$	Non-temporally bounded, type II	Temporally bounded
$\Diamond_{[a,b]}$	Non-temporally bounded, type I	Temporally bounded
$\mathcal{U}_{[a,b]}$	Non-temporally bounded, type I	Temporally bounded

Example 5.1. During a run, the formula $\phi_i = \Diamond_I A$ is satisfied if A holds at some $t \in I$, violated if $\neg A$ holds for all $t \in I$ where $I = [a, b]$, and uncertain if $\neg A$ holds for all $t \leq \tau$ where $\tau \in I$ is the current clock valuation.

If ϕ_i is non-temporally bounded there are only two possible temporal outcomes, either satisfied/uncertain or violated/uncertain as illustrated by examples 5.2 and 5.3.

Example 5.2. Consider $\phi_i = \Diamond_{[0,\infty]} A$. During a run, the formula is; satisfied if A holds at some t, and uncertain if $\neg A$ holds for all $t \leq \tau$ where τ is the current clock valuation.

Example 5.3. Consider $\phi_i = \Box_{[0,\infty]} A$. During a run, the formula is: violated if $\neg A$ holds for some t, and uncertain if A holds for all $t \leq \tau$ where τ is the current clock valuation.

To separate these different types of non-temporally bounded formulas we introduce *Type I* and *Type II* formulas:

Definition 5.1. A non-temporally bounded formula ϕ is said to be of *Type I* if ϕ cannot be concluded to be violated at any time during a run, since there remains a possibility for it to be satisfied in the future, or of *Type II* if ϕ cannot be concluded to be satisfied during a run, since there remains a possibility for it to be violated in the future.

The categorization of all operators are given in Table 5.1.

5.2.2 Hybrid Distance

We now introduce the hybrid distance as a function of a MITL formula and a timed run:

Definition 5.2. The hybrid distance d_h is a satisfaction metric with respect to a MITL formula ϕ and a timed run $r^t = (\pi_0, \tau_0), (\pi_1, \tau_1), ..., (\pi_m, \tau_m)$, defined as:

$$d_h = hd_c + (1 - h)d_d \tag{5.1}$$

where d_c and d_d are the *continuous and discrete distances* between the run and the satisfaction of ϕ:

$$d_c = \sum_{i \in X} T_i^c \qquad\qquad d_d = \sum_{j=0,1,\ldots,m} T_j^d$$

X is the set of clocks (given next in Definition 5.4), T_i^c is the time which the run violates the deadline expressed by clock i, and T_j^d is defined as:

$$T_j^d = \begin{cases} \tau_{j+1} - \tau_j & \text{if } (r^t, j) \nvDash \phi_i \in \phi \\ 0 & \text{otherwise,} \end{cases}$$

where ϕ_i is any non-temporally bounded subformula of ϕ and $h \in [0, 1]$ is a weight assigning constant which determines the priority between continuous and discrete violations, where $h = 0.5$ yields equal importance.

To be able to calculate d_h we define its derivative:

Definition 5.3. $\Phi_H = (\dot{d}_c, \dot{d}_d)$, is a tuple, where $\dot{d}_c \in \{0, ..., n_c\}$ and $\dot{d}_d \in \{0, 1\}$, and $n_c = |X|$ is the number of time bounds associated with the MITL specification (or number of clocks).

5.2.3 Timed Automaton with Hybrid Distance

We can now expand the definition of the TBA to include hybrid distance by considering its derivatives:

Definition 5.4. [56] A *Timed Automaton with hybrid distance* (TAhd) is a tuple $A_H = (S, S_0, AP, X, F, I_X, I_H, E, H, \mathcal{L})$ where $S = \{s_i : i = 0, 1, ...m\}$ is a finite set of locations, $S_0 \subseteq S$ is the set of initial locations, 2^{AP} is the alphabet (i.e. set of actions), where AP is the set of atomic propositions, $X = \{x_i : i = 1, 2, ..., n_c\}$ is a finite set of clocks (n_c is the number of clocks), $F \subseteq S$ is a set of accepting locations, $I_X : S \to \Phi_X$ is a map from location to clock constraints, $H = (d_c, d_d)$ is the hybrid distance, $I_H : S \to \Phi_H$ is a map from location to hybrid distance derivative (labelling each location with some derivatives, $\dot{d}_d$ and $\dot{d}_c$), where I_H is such that $I_H(s) = (d_1, d_2)$ where d_1 is the number of temporally bounded operators violated in s, and $d_2 = 0$ if no non-temporally bounded operators are violated in s and $d_2 = 1$ otherwise, $E \subseteq S \times \Phi_X \times 2^{AP} \times S$ is a set of edges, and $\mathcal{L} : S \to 2^{AP}$ is a labelling function mapping each location to a set of actions.

The notation $(s, g, a, s') \in E$ is used to state that there exists an edge from s to s' under the action $a \in 2^{AP}$ where the valuation of the clocks satisfy the guard $g = I_X(s) \in \Phi_X$. The expressions $d^c(s)$ and $d^d(s)$ are used to denote the hybrid distance derivatives $\dot{d}_c$ and $\dot{d}_d$ assigned to s by I_H.

To simplify future notation when considering the hybrid distance, we introduce the automata timed run:

Definition 5.5. An *automata timed run* $r^t_{A_H} = (s_0, \tau_0), ..., (s_m, \tau_m)$ of a TAhd, A_H, corresponding to the timed run $r^t = (\pi_0, \tau_0), ..., (\pi_m, \tau_m)$ of a WTS T, is a sequence where $s_0 \in S_0$, $s_j \in S$, and $(s_j, g_{j+1}, a_{j+1}, s_{j+1}) \in E$ $\forall j \geq 1$ such that i) $\tau_j \models g_j$, $j \geq 1$, and ii) $L(\pi_j) \in \mathcal{L}(s_j)$, $\forall j$.

It follows from Definitions 5.4 and 5.5, that the continuous violation for the automata timed run is $d_c = \sum_{i=0,...,m-1} d^c(s_i)(\tau_{i+1} - \tau_i)$, and similarly, the discrete violation for the automata timed run is $d_d = \sum_{i=0,...,m-1} d^d(s_i)(\tau_{i+1} - \tau_i)$, and hence the hybrid distance, d_h, as defined in Definition 5.2, is equivalently given with respect to *an automata timed run* as

$$d_h(r^t_{A_H}, h) = \sum_{i=0}^{m-1} (hd^c(s_i) + (1-h)d^d(s_i))(\tau_{i+1} - \tau_i) \tag{5.2}$$

5.3 Problem Statement

The problem considered in this Chapter is to find the plan which violates a given MITL specification the least, for some human preference. We define how much a plan violates a specification in accordance with the hybrid distance, where $d_h = 0$ corresponds to complete satisfaction, and a greater hybrid distance corresponds to a greater violation. The human preference is indicated by the choice of h. This can be formulated as two sub problems:

Problem 5.1. Given a WTS T and an MITL specification ϕ, find the timed run r^t of T that corresponds to the automata timed run $r^t_{A_H}$ that satisfies:

$$r^t_{A_H} = \arg\min_{r^t_{A_H}} d_h(r^t_{A_H})$$

where A_H is the TAhd that corresponds to ϕ.

Problem 5.2. Given a human feedback $f \in \mathbb{F}$, update c such that the new solution of Problem 5.1, r^t_{new}, satisfies $eval(r^t_{old}, r^t_{new}, f) > 0$, where r^t_{old} is the previously found solution, if such a solution exists.

5.4 Control Strategy

The solution to Problems 5.1 and 5.2, is inspired by the standard 3 steps procedure for single agent control synthesis; i) expressing the temporal logic specification as an automaton, ii) constructing the product of the automaton and the transition system describing the dynamics and environment, and iii) implementing graph search to find the shortest path.

The suggested control synthesis framework follows the steps below (step 1-3 solves Problem 5.1 and step 4 solves Problem 5.2):

1. Construct a Timed Automaton with Hybrid Distance ($TAhd$) which represents the MITL specification.

2. Construct a Product Automaton as the product of the $TAhd$ and a WTS which represents the dynamics of the robot.

3. Find the least violating path by finding the shortest path with respect to the hybrid distance, d_h, and a given h.

4. Update h in accordance with human feedback and repeat steps 3 and 4 until the human ends the process either by approving the result or aborting.

The details of the proposed solution are further described below.

5.4.1 Soft MITL Tasks to Timed Automaton with Hybrid Distance

In this section we consider the construction of a TAhd representing an MITL formula. The construction can be performed be following Algorithm 2 below. The details of how to defines locations, clocks (and hybrid distance) and edges are given in Algorithms 3-5 respectively. The construction is roughly based on the LTL to automata translation described in [1].

Remark 5.1. This procedure has been improved since the publication of [56]. In this chapter we describe the method which was suggested in [56], it should be noted however that the approach used in the upcoming chapter 6 (where both hard and soft constraints are considered) can be applied here (with some small modification to the edge construction).

Algorithm 2 Construction of a TAhd from a MITL specification

1: **procedure** MAIN(ϕ)
2: $[S, S_0, F]$ =SETLOCATIONS(ϕ) ▷ see Algorithm 3
3: $AP = \{ap : ap \in \phi\}$
4: $[X, I_X, H, I_H]$ =SETCLOCKS(ϕ, S) ▷ see Algorithm 4
5: E =SETEDGES($\phi, I_X, 2^{AP}, S$) ▷ see Algorithm 5
6: **if** $\exists\, g \in \Phi_X$ $s.t.$ $(s, g, a, s) \in E$ **then**
7: $\mathcal{L} : s \to a \in 2^{AP}$
8: **end if**
9: **end procedure** and **return** $A_H = (S, S_0, AP, F, X, I_X, H, I_H, E, \mathcal{L})$

Considering the set of locations, the formula ϕ can be partitioned into sub formulas ϕ_i such that $\phi = \bigcap_{i \in \{1,\ldots,n\}} \phi_i$ for some $n > 0$. Each sub formula ϕ_i can

be evaluated as $\phi_i^{state} \in \varphi_i$, where

$$\varphi_i = \begin{cases} \{\phi_i^{vio}, \phi_i^{sat}, \phi_i^{unc}\} & \text{if } \phi_i \text{ is temporally bounded} \\ \{\phi_i^{sat}, \phi_i^{unc}\} & \text{if } \phi_i \text{ is non-temporally bounded of Type I} \\ \{\phi_i^{vio}, \phi_i^{unc}\} & \text{if } \phi_i \text{ is non-temporally bounded of Type II} \end{cases}$$

Based on this we introduce $\Psi = \prod_{i \in \{1,\dots,n\}} \varphi_i$, and construct the set of locations such that there exists a location s for each possible $\psi \in \Psi$. The initial location is then defined as the location where each sub formula is uncertain. Considering the MITL formula, this corresponds to no progress being made. The accepting location is defined as the location where each temporally-bounded sub formula and each non-temporally bounded sub formula of Type I are satisfied, while all non-temporally bounded sub formulas of Type II are uncertain. Considering the MITL formula, this corresponds to complete satisfaction.

Algorithm 3 Construct set of locations S, initial location S_0 and accepting location F of a TAhd

1: **procedure** SETLOCATIONS(ϕ)
2: $\Phi = \{\phi_i : \phi = \bigcap_i \phi_i\}$
3: **for** $\phi_i \in \Phi$ **do**
4: **if** ϕ_i is temporally bounded **then**
5: $\varphi_i = \{\phi_i^{sat}, \phi_i^{vio}, \phi_i^{unc}\}$
6: **else if** ϕ_i is Type I **then**
7: $\varphi_i = \{\phi_i^{sat}, \phi_i^{unc}\}$
8: **else**
9: $\varphi_i = \{\phi_i^{vio}, \phi_i^{unc}\}$
10: **end if**
11: **end for**
12: $\Psi = \prod_i \varphi_i$
13: $S = \{s_i : i = 0, ..., n\}$, where n is the number of $\psi \in \Psi$
14: $S_0 = s_0$, where s_0 corresponds to $\psi_0 = \bigcap_i \phi_i^{unc}$
15: $F = s_F$, where s_F corresponds to $\psi_F = \bigcap_{i \in I} \phi_i^{sat} \cap \bigcap_{j \in J} \phi_j^{unc}$, where $i \in I$ are the indexes of sub formulas that are either temporally bounded or of Type I, and $j \in J$ are the indexes of sub formulas that are of Type II
16: **end procedure** and **return** S, S_0, F

The mapping of clock constraints and hybrid distance derivatives are constructed according to Algorithm 4. Recall from Definition 5.4, that the set of clocks is defined such that each temporally bounded operator corresponds to one clock. The mapping of clock constraints is constructed such that a location that corresponds to a temporally bounded operator is mapped to the corresponding clock constraint. The mapping of hybrid distance derivatives is constructed such that a location s

corresponding to ψ is mapped to $\dot{d}_c = k$ if a number of k temporally bounded operators are violated in ψ, and $\dot{d}_d = 0$ if no non-temporally bounded operators are violated and $\dot{d}_d = 1$ if at least one is violated.

Algorithm 4 Construct the map of clock constraints I_X, and the map of hybrid distance derivatives I_H.

1: **procedure** SETCLOCKS(ϕ, S)

2: $I_X : s \to \Phi_X$ where s corresponds to ψ and for each $\Phi_x \in \Phi_X$, there exists a temporally bounded sub formula ϕ_i, such that either $\phi_i^{vio} \in \psi$ or $\phi_i^{sat} \in \psi$, where Φ_x is the time constraint of $\neg\phi_i$ in the former case and ϕ_i in the latter

3: $I_H : s \to \Phi_h = (\dot{d}_c, \dot{d}_d)$, where $\Phi_h = (0, 0)$ if s corresponds to ψ which does not contain any ϕ_i^{vio}, (i.e. if ϕ is satisfied or uncertain), $\dot{d}_c = n$ if s corresponds to ψ which contains n number of violated temporally bounded operators ϕ_i^{vio}, and

$$\dot{d}_d = \begin{cases} 1 & \text{if } s \text{ corresponds to } \psi \text{ which contains at} \\ & \text{least one non-temporally bounded} \\ & \text{subformula } \phi_i^{vio}; \\ 0 & \text{otherwise} \end{cases}$$

4: **end procedure** and **return** I_X, I_H

The edges are constructed in five sets; E_1, E_2, E_3, E_4, E_5, in Algorithm 5. The first set, E_1, corresponds to progress of the MITL formula, both negative and positive. It is defined such that $(s, a, g, s') \in E_1$ if an action a under the guard g taken in the location s yields that ψ' is satisfied, where s' corresponds to ψ'. The second set, E_2, contains edges from locations where $\dot{d}_d = 1$, i.e. locations which corresponds to discrete violations. The remaining properties of the edges, i.e. actions, guards and goal locations, are copies of other edges in E_1, where the goal location share the same evaluation of violated and satisfied temporally-bounded sub formulas as s. That is, $(s, a, g, s') \in E_2$ if some temporally bounded sub formula/formulas which is uncertain in s is violated or satisfied by the action a under the guard g *or* if a non-temporally bounded sub formula changes state (becomes violated or satisfied, or stops being violated or satisfied), and s' is a copy of s with respect to all evaluations of sub formulas except the one/ones effected by (a, g). One can hence consider the edges in E_2 as alternative routes to the ones in E_1, where some non-temporally bounded sub formula/formulas are violated at some point/points. The third set, E_3, contains edges from locations with $\dot{d}_c > 0$, i.e. locations which corresponds to continuous violations. If a location has outgoing transitions to two locations, one where a temporally bounded sub formula ϕ_i is violated, and one where it is satisfied, then a transition must exists from the violating location to the satisfying location under the same guard and action as the transition from the original location to the satisfying location. This ensures that the satisfying location will be reached when the time bounded action finally occurs, even after the deadline is exceeded

Algorithm 5 Construct edges E

1: **procedure** SETEDGES(ϕ,S,2^{AP}, I_X, I_H)
2: **if** the action a performed under the guard g in s leads to ψ' being satisfied, where s' corresponds to ψ' **then**
3: $(s, g, a, s') \in E_1$
4: **end if**
5: **if** $(s'', g', a', s) \in E_1$ & $\phi_i^{vio} \in \psi$ where s corresponds to ψ and ϕ_i is non-temporally bounded & $(s'', g, a, s') \in E_1$ where s' corresponds to ψ' & it holds $\forall$ temporally bounded ϕ_j that

$$\begin{array}{ll} \phi_j^{vio} \in \psi' \text{ if} & \phi_j^{vio} \in \psi \\ \phi_j^{sat} \in \psi' \text{ if} & \phi_j^{sat} \in \psi \end{array}$$

then
6: $(s, g, a, s') \in E_2$
7: **end if**
8: **if** $(s'', g', a', s) \in E_1$ & $\phi_i^{vio} \in \psi$ where s corresponds to ψ and ϕ_i is temporally bounded & $(s'', g, a, s') \in E_1$ where s' corresponds to ψ' and $\phi_i^{sat} \in \psi'$ **then**
9: $(s, g, a, s') \in E_3$
10: **end if**
11: **if** $\exists(s_i, g_i, a_i, s) \in E_1 \cup E_2 \cup E_3$ for some s_i **then**
12: $(s, g, a, s) \in E_4$ where $(g, a) = \bigcup_i (g_i, a_i)$
13: **end if**
14: **if** $s = s_0$ & $\exists(s_0, g_i, a_i, s_i) \in E_1 \cup E_2 \cup E_3$ for some s_i **then**
15: $(s, g, a, s) \in E_4$ where $(g, a) = \Phi_X \times 2^{AP} \setminus \bigcup_i (g_i, a_i)$
16: **end if**
17: **if** $\exists(s', g', a', s) \in E_1$, where s corresponds to ψ and s' corresponds to ψ' & $\phi_i^{vio} \in \psi$ is non-temporally bounded & $\phi_i^{unc} \in \psi'$ & it holds for all temporally bounded ϕ_j that

$$\begin{array}{ll} \phi_j^{vio} \in \psi' \text{ if} & \phi_j^{vio} \in \psi \\ \phi_j^{sat} \in \psi' \text{ if} & \phi_j^{sat} \in \psi \end{array}$$

& $\exists(s', g_i, a_i, s_i) \in E_1 \cup E_2 \cup E_3 \cup E_4$ for some s_i **then**
18: $(s, g, a, s') \in E_5$ where $(g, a) = \Phi_X \times 2^{AP} \setminus \bigcup_i (g_i, a_i)$
19: **end if**
20: $E = E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5$
21: **end procedure** and **return** E

(however with non-zero continuous distance). The fourth set, E_4, corresponds to self-loops, i.e. transitions from and to the same location. They are defined such that all combinations of actions $a \in 2^{AP}$ and guards $g \in \Phi_X$ present in the ingoing edges are handled by outgoing edges. This ensures that there are no deadlocks in the automaton. The last set, E_5, contains the inverse of the edges in E_1 which ended in locations where $\dot{d}_d = 1$. That is, E_5 contains the edges which corresponds to going back when discrete violations stop. This set is constructed last in order to determine the actions and guards of the edges based on the other sets.

Remark 5.2. The motivation behind the subsets becomes clearer from the construction shown in Algorithm 5, since the construction of the latter sets are dependent on the former.

An example of a TAhd construction is given below, where the edges are added set by set. The example also provides a comparison with a TBA (where the specification is considered to be a hard task) to illustrate the differences.

Example 5.4. Consider the MITL specification $\Box\neg a \wedge \Diamond_{t<5} b$. The subformulas ϕ_i are $\phi_1 = \Box\neg a$ and $\phi_2 = \Diamond_{t<5} b$. The first subformula is non-temporally bounded type II and the second one is temporally bounded. Hence the first subformula can be evaluated as violated or uncertain, and the second as violated, uncertain or satisfied. Using Algorithm 3 we can then determine that the TAhd should have 6 locations. The initial location corresponds to both subformulas being evaluated as uncertain, denoted as $s_0 \sim \phi_1^{unc} \wedge \phi_2^{unc}$. The accepting location corresponds to the first subformula being evaluated as uncertain and the second as satisfied, here denoted $s_4 \sim \phi_1^{unc} \wedge \phi_2^{sat}$. For the remainder of the locations we will denote $s_1 \sim \phi_1^{vio} \wedge \phi_2^{unc}$, $s_2 \sim \phi_1^{vio} \wedge \phi_2^{vio}$, $s_3 \sim \phi_1^{unc} \wedge \phi_2^{vio}$, $s_5 \sim \phi_1^{vio} \wedge \phi_2^{sat}$.

In this example there are two clock constraint; $t < 5$ which is associated with locations corresponding to ϕ_2^{sat}, and $t \geq 5$ which is associated with ϕ_2^{vio}. The first clock constraint is then mapped to s_4 and s_5, and the second to s_2 and s_3. The hybrid distance derivatives are mapped such that $s_0 \to (0,0)$ (no violation), $s_1 \to (0,1)$ (violation of non-temporally bounded operator), $s_2 \to (1,1)$ (violation of both non-temporally bounded operator and one temporally bounded operator), $s_3 \to (1,0)$ (violation of one temporally bounded operator), $s_4 \to (0,0)$ (no violation), and $s_5 \to (0,1)$(violation of non-temporally bounded operator).

The edges can now be constructed following Algorithm 5. The result is illustrated by Figs. 5.1a to 5.1e. The final TAhd is illustrated in Fig. 5.2a and the corresponding TBA (if the specification had been considered as a hard task) is illustrated in Fig. 5.2b.

By comparing the first figure (with edges E_1) to the TBA, we note that the edges are almost the same, but the TAhd is lacking the self-loops. Next, we construct E_2 which contains edges from locations where the non-temporally bounded subformula ϕ_1 is violated to locations where it is uncertain, and the two locations share a common predecessor. For E_3 we add edges from locations where ϕ_2 is violated to locations where it is satisfied, where the two locations share a common predecessor.

E_4 consists of the self-loops, which if disregarding the actions and guards are identical to the self-loops of the TBA. The actions and guards associated with the self-loops of the TAhd will be different since a subset of the ones used in the TBA will correspond to other edges here. Finally, E_5 consist of edges from locations where ϕ_1 is violated to locations where it is uncertain, where the second location is a predecessor of the first.

5.4.2 Product Automata

The construction of a product of a WTS and a TAhd is similar to the product of a WTS and a TBA. Definitions of the latter are given in [55] and [6]. The former is given below. The difference is that the product of a WTS and a TAhd requires that the mappings of the hybrid distance derivatives are taken into consideration through simple projection.

Definition 5.6. Given a weighted transition system $T = (\Pi, \Pi_{init}, \Sigma, \rightarrow, AP, L, d)$ and a timed automaton with hybrid distance $A_H = (S, S_0, AP, X, F, I_X, I_H, E, H, \mathcal{L})$ their *Product Automaton* (P) is defined as $P = T \otimes A_H = (Q, Q^{init}, \rightsquigarrow, \hat{w}, \mathcal{F}, AP, \mathcal{L}^p, I_X^p, I_H^p, X, H)$, where $Q \subseteq \{(\pi, s) \in \Pi \times S : L(\pi) \in \mathcal{L}(s)\} \cup \{(\pi, s) \in \Pi_{init} \times S_0\}$ is the set of states, $Q^{init} = \Pi_{init} \times S_0$ is the set of initial states, $\rightsquigarrow$ is the set of transitions defined such that $q \rightsquigarrow q'$ if and only if

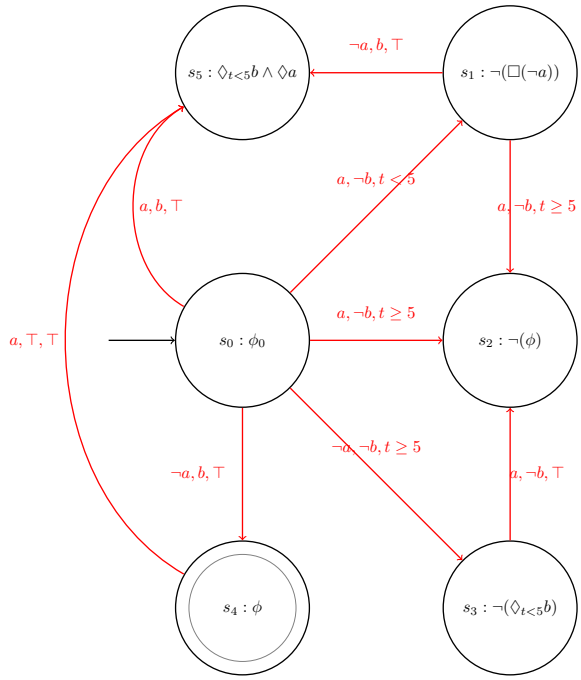
- $q = (\pi, s)$, $q' = (\pi', s') \in Q$
- $(\pi, \pi') \in \rightarrow$ and
- $\exists\, g, a$, s.t. $(s, g, a, s') \in E$,

$\hat{w}(q, q') = w(\pi, \pi')$ if $(q, q') \in \rightsquigarrow$, is a positive weight assignment map, $\mathcal{F} = \{(\pi, s) \in Q : s \in F\}$, is the set of accepting states, $\mathcal{L}^p(q) = L(\pi)$ is an observation map, $I_X^p(q) = I_X(s)$ is a map of clock constraints, and $I_H^p(q) = I_H(s)$ is a map of hybrid distance derivative constraints.

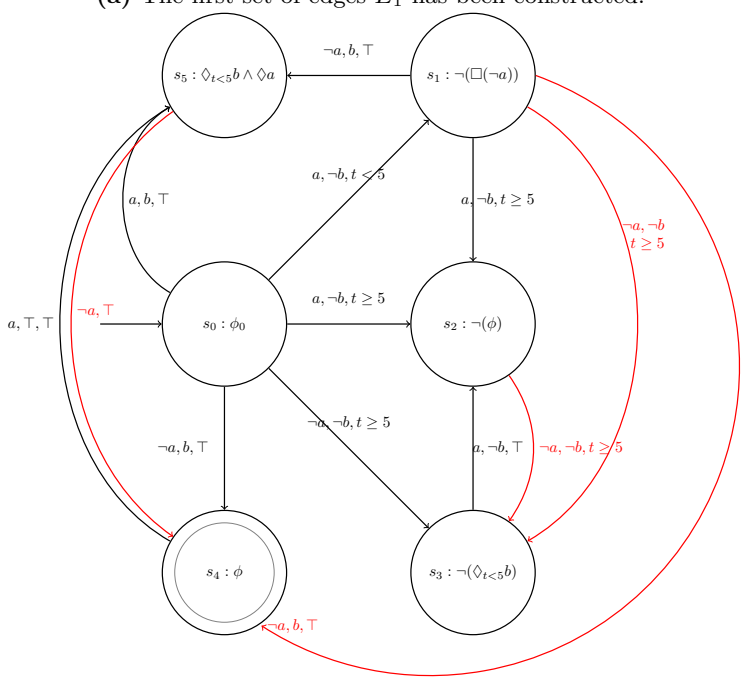
Given this definition we can conclude that we will always be able to find a least-violating path if one exists, and furthermore least-violating path always exists under the assumption that the labelling of the WTS and the actions of the MITL specification match:

Theorem 5.1. *If $\exists$ some $\pi \in \Pi$ for every $a \in 2^{AP}$ that are considered by a reachability operator in the MITL specification ϕ, such that $a \in L(\pi)$, and if π is reachable from Π_{init}, then Algorithm 6 will always have a solution. Here, a is a word (a combination of atomic propositions), and the reachability operators are eventually and until, the operators which requires a word to be reached at some point.*

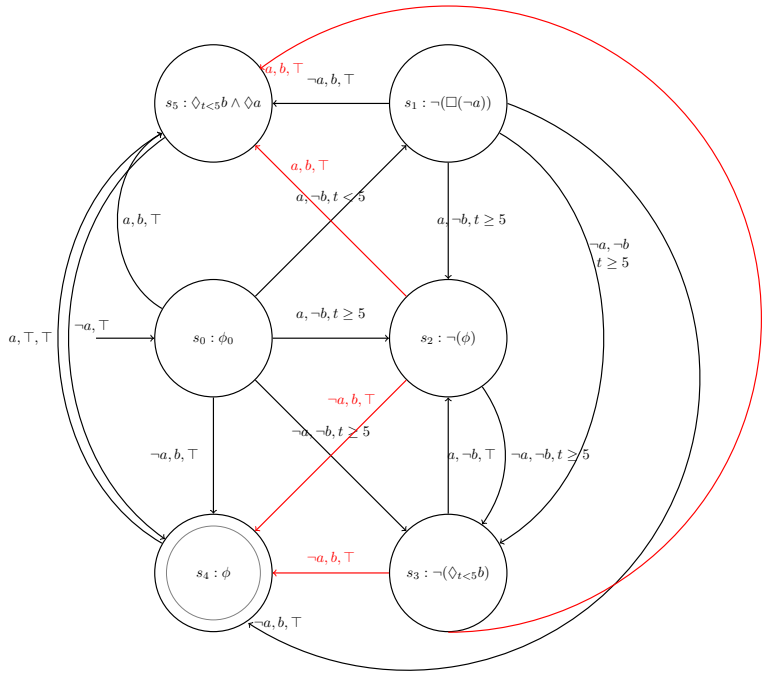
Proof. If $\exists$ some $\pi \in \Pi$ for a reachability operator such that $a \in L(\pi)$, then it follows that $\exists\, q \in Q$ such that $q = (\pi, s)$, where s is a location in the $TAhd$ corresponding to the satisfaction of the reachability operator. Furthermore, q is reachable from



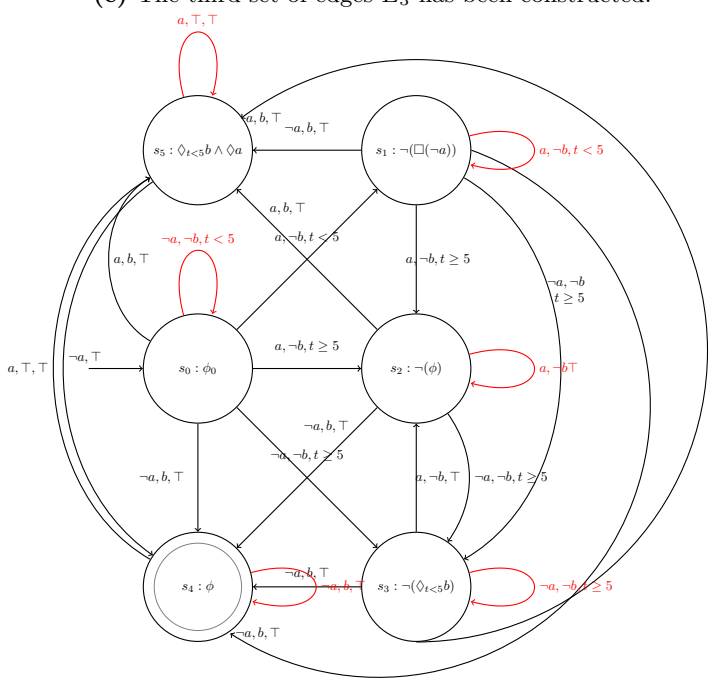
(a) The first set of edges E_1 has been constructed.



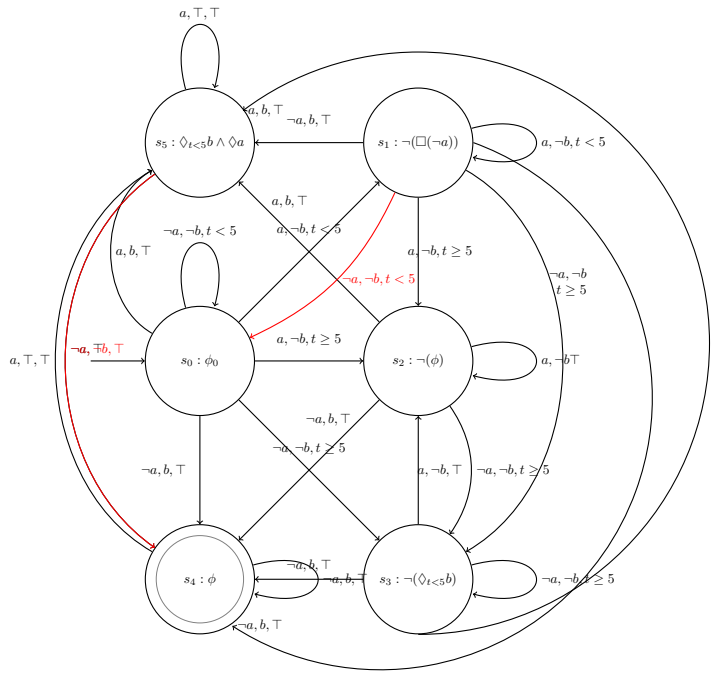
(b) The second set of edges E_2 has been constructed.



(c) The third set of edges E_3 has been constructed.



(d) The fourth set of edges E_4 has been constructed.



(e) The fifth set of edges E_5 has been constructed.

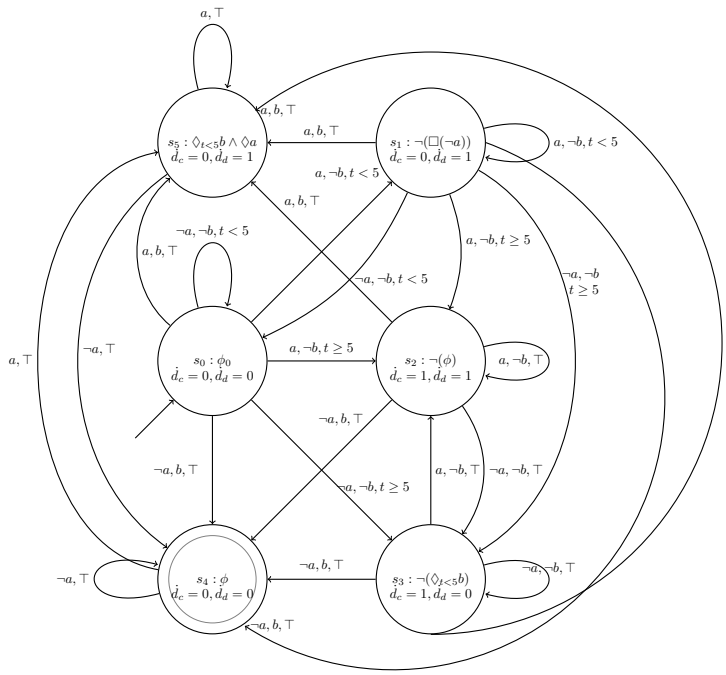
Figure 5.1: The steps of constructing edges for the TAhd that corresponds to the soft MITL task $\Box\neg a \wedge \Diamond_{t<5} b$

Q^{init} if π is reachable from Π^{init}. It follows that $\exists$ a state $q' = (\pi', s') \in Q$, which is reachable from Q^{init}, where s' corresponds to the satisfaction of all reachability operators in ϕ. By definition $q' \in \mathcal{F}$, and hence Algorithm 6 will have a solution. $\Box$

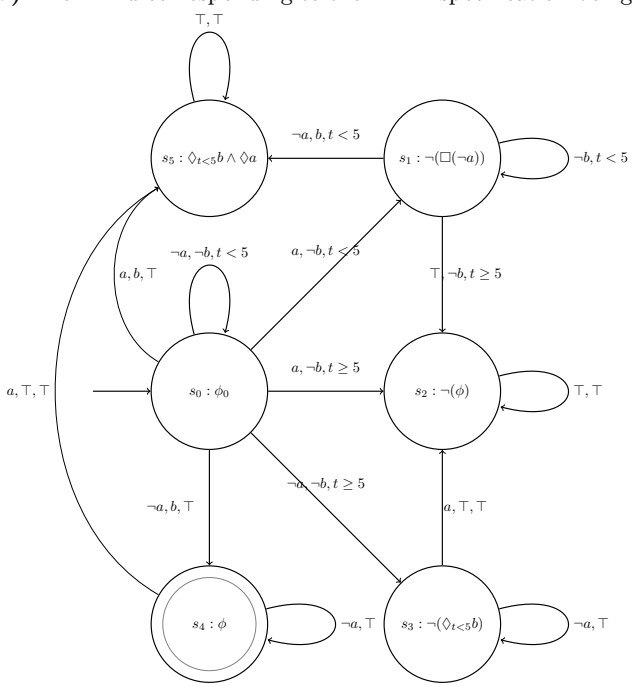
5.4.3 Graph Search Algorithm with Violation Preference

To find the path of the PA which corresponds to the smallest value of the hybrid distance a Dijkstra algorithm which uses the hybrid distance as the cost function is suggested. The overview of the suggested algorithm is given in Algorithm 6.

Remark 5.3. It is notable that d_h^{step} in Algorithm 6 considers the hybrid distance derivative of the previous state q rather than the successor q'. The reason for this is that we assume that the agent will move from q to q' at the end of the transition time.



(a) The TAhd corresponding to the MITL specification being a soft task.



(b) The TBA corresponding to the MITL specification being a hard task.

Figure 5.2: The TAhd and TBA corresponding to the MITL specification $\Box\neg a \wedge \Diamond_{t<5} b$.

Algorithm 6 Dijkstra Algorithm with Hybrid Distance as cost function.

```
procedure DIJSKTRAHD(P, h)
    Q =set of states, q_0 =initial state, SearchSet = q_0, ŵ(q, q') =weight of
    transition q ⇝ q' in P
    if q = q_0 then
        dist(q) = 0, d_h(q) = 0, d_c(q) = 0, d_d(q) = 0
    else
        dist(q) = d_h(q) = d_c(q) = d_d(q) = ∞
    end if
    for q ∈ Q do
        pred(q) = ∅
    end for
    while pathFound = False do
        pick q ∈ SearchSet s.t. q = arg min(d_h(q))
        if q ∈ F then
            pathFound = True
        else
            find all q' s.t. q ⇝ q'
            for every q' do
                d_h^step = (h d̈_c(q) + (1 − h) d̈_d(q)) ŵ(q, q')
                if d_h(q') > d_h(q) + d_h^step then
                    update dist(q'), d_h(q'), d_c(q'), d_d(q') and pred(q')
                    add q' to SearchSet
                    remove q from SearchSet
                end if
            end for
        end if
    end while
    r_hd^min = q
    while q ≠ q_0 do
        use pred(q) to iteratively form the path back to q_0:
        r_hd^min = [pred(q), r_hd^min]
    end while
end procedure and return r_hd^min, d_h, d_c, d_d
```

5.4.4 Human Feedback Framework

In order to incorporate the human feedback in the system it must be translated into a deterministic response. The response should be such that d_c^+ (preference to lower discrete violation with a cost of increasing the continuous violation) leads to a decrease in d_d, and d_c^- (preference to lower continuous violation with a cost to increase discrete violation) leads to a decrease in d_c. The remaining feedback, d_c^0 and *abort*, should lead to the response of ending the synthesis, either to run the determined plan or to do nothing.

In this paper, we suggest that the system responds as described in Algorithm 7. Since $d_h = hd_c + (1-h)d_d$, it follows that d_c has a greater impact on d_h if c is chosen large and that d_d has a greater impact if h is chosen small. Hence, increasing h results in a suggested path that has less continuous violation (if it exists), and similarly decreasing c results in slower discrete violation. Algorithm 7 implements this by decreasing h with δ if the feedback d_c^+ is given, while increasing h with δ if the feedback d_c^- is given. Limits on how much h can be increased or decreased is set in accordance with $h \in [0,1]$. The increment $\delta > 0$ should be chosen small enough to avoid that possible paths are missed. However, decreasing δ will result in a greater number of runs of Algorithm 6.

5.5 Simulations

To illustrate the suggested framework, simulations have been performed in MATLAB. The simulations consider a single agent following the dynamics given in (5.3), moving in the environment illustrated in 5.3a, the abstraction of the dynamics was performed as in Chapter 3.

$$\begin{aligned} \dot{x} &= \begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix} x + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} u \\ & x_0 = (2.5, 3.5) \\ & x_1 \in [1,6], x_2 \in [1,4] \\ & |u| \in [-20, 20] \end{aligned} \tag{5.3}$$

The MITL task $\phi = \Box_{[0,\infty]}\neg a \wedge \Diamond_{[0,0.01]} b$, i.e. avoid a and reach b within 0.01s, was given as input. Hence, there is only one word which is considered by a reachability operator $\{\neg a, b\}$, and since there exists states in the environment where b holds and a doesn't, it follows that the control synthesis will give at least one suggested path. The construction of the product automaton was performed in 3s, and the graph search in 47ms on a laptop with a Core i7-6600U 2.80 GHz processor. The increment δ was set to 0.01. Three different paths were suggested based on the human feedback. In the first case, the human feedback was d_c^0 and hence $h = 0.5$. In the second case, the human feedback was d_c^+, indicating that the human favours a low value of d_d or little discrete violations. In the last case, the human feedback was d_c^-, indicating

Algorithm 7 Algorithm for handling feedback from human user

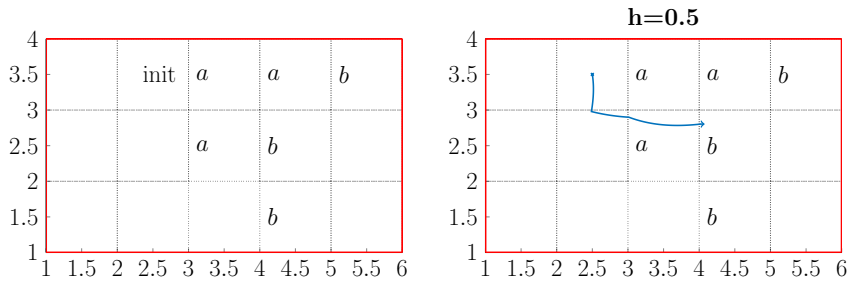
```
1:  procedure INTERFACE(feedback)
2:      Parameter to set: δ
3:      if feedback=d_c^0 then
4:          Implement controller
5:      else if feedback=abort then
6:          Ask for a new task
7:      else if  feedback=d_c^+ then
8:          while  no new path is found and h ≥ 0 do
9:              h = h − δ
10:             dijkstraHD(P, h)
11:         end while
12:         if  h < 0 then
13:             No path with smaller d_d exists - inform human
14:         else
15:             New path was found - suggest it to human user
16:         end if
17:     else if  feedback=d_c^− then
18:         while  no new path is found and h ≤ 1 do
19:             h = h + δ
20:             dijkstraHD(P, h)
21:         end while
22:         if  h > 1 then
23:             No path with smaller d_c exists - inform human
24:         else
25:             New path was found - suggest it to human user
26:         end if
27:     end if
28: end procedure and return h, path
```

that the human favours less violation of deadlines. The resulting values of h and the distances are given in Table 5.2. Neither increasing h above 0.55 nor decreasing it below 0.34, gives any new suggested path. The paths are illustrated in Figs. 5.3 and 5.4, where the determined control sequences where implemented. The switches between the controllers were performed based on position, i.e. on the edge between states, rather than time.

In Figure 5.3b, the framework suggests that the agent moves through one state labelled a and misses the deadline by 0.09s. That is, both the discrete and the continuous part of the formula is violated. In the second case (Figure 5.4a), the agent takes a longer path to go around all states labelled by a, however it misses the deadline with 0.16s. In the last case (Figure 5.4b), the agent takes the quickest path to the goal in order to violate the deadline as little as possible, however this is

Table 5.2: Estimated and actual hybrid distance for the suggested paths.

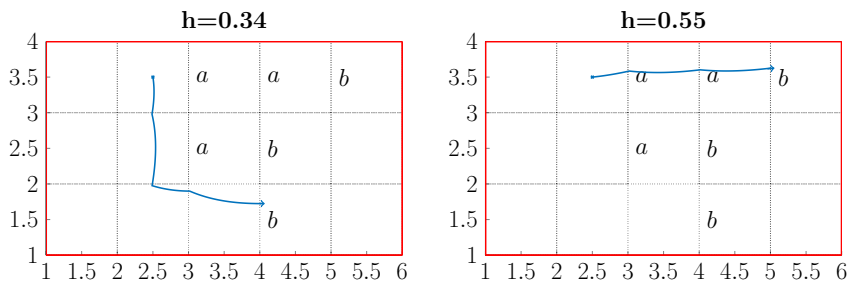
Path	h	d_c	d_d	Estimated d_h	Real d_h
1	0.5	0.09	0.036	0.064	0.042
2	0.34	0.16	0	0.055	0.041
3	0.55	0.067	0.067	0.067	0.062



(a) The environment the agent is allowed to move through.

(b) Final path for feedback d_c^0, i.e. initial path.

Figure 5.3: Environment and suggested paths satisfying ϕ as close as possible with respect to the hybrid distance for initial h



(a) Final path for feedback d_c^+.

(b) Final path for feedback d_c^-.

Figure 5.4: Suggested paths satisfying ϕ as close as possible with respect to the hybrid distance under human feedback.

at the expense of moving through two states labelled a. Hence, the differences of the suggested paths follows what the human indicated with her feedback.

5.6 Conclusion

In this Chapter a novel hybrid distance metric was introduced, and associated to deriving the least violating path with respect to an MITL formula given by a human user. A framework for finding the path with the lowest value for this metric with respect to human feedback was suggested. The presented case study illustrates that the framework gives reasonable estimations of the metric, and that the resulting

path suggestion follows the expected behaviour. Hence we have eliminated the issue discussed in Chapter 4; not finding an accepting run. However, we have also limited our system to only consider tasks which could be violated, loosing our guarantee that some tasks are completely satisfied by the resulting run. We will consider this issue in the next chapter.

The methods presented in this chapter were limited to the single-agent case. However, it could easily be extended to the multi-agent case by i) using the approach we suggested in Chapter 4 with small modifications to the product BWTS and the global BWTS to include the hybrid distance, or ii) by applying the decentralized framework which will be described in Chapter 7.

Chapter 6

Control Synthesis for Single-Agent Systems under Hard and Soft MITL Tasks

In this chapter we will bind the methods in the previous chapters together to consider tasks which contain both hard and soft constraints. The work was first presented in [57].

6.1 Introduction

In the previous chapters we have considered the standard control synthesis problem using hard constraints, and a framework for soft constraints which allowed us to find solutions when the tasks was infeasible or existing solutions were hidden by overestimations. We concluded that the first approach was lacking in the sense that no feedback was produced, while the second lost the guarantees which control synthesis was created to achieve. To solve these issues we will consider hard and soft constraints together in this chapter. By designing a controller that is guaranteed to satisfy the hard constraints, while minimizing the violation of the soft constraints we get the advantages of both approaches.

The concept of hard and soft constraints was also addressed in [58], where we weighed the satisfaction of a soft LTL task against a control input limitation. It is a commonly known concept, but is rarely related to control synthesis and formal methods since these areas are so strongly connected to guaranteed satisfaction. By implementing it in the manner we have in both [58] and [57], we attempt to widen the implementation possibilities of control synthesis into non-ideal environments where solutions can not always be found and some constraints are less important than others. The main contribution of this chapter is i) a method to construct a timed automaton with hybrid distance for MITL specifications with hard and soft constraints, complete with a systematic algorithm, and ii) placing it into a control synthesis framework.

6.2 Problem Statement

The problem considered in this thesis is to find the plan which violates the given soft constraint the least, for some human preference, while satisfying the given hard constraint. The input of each agent is assumed to be bounded with $|u_i| \leq u_{max}$, $\forall i \in \{1, ..., N\}$.

The hybrid distance (d_h) is used as the measurement of violation, where $d_h = 0$ corresponds to complete satisfaction. The human preference is indicated by the value of h. The problem then becomes:

Problem 6.1. Given a WTS T and an MITL specification $\phi = \phi^{hard} \wedge \phi^{soft}$, find the timed run r^t of T that corresponds to the automata timed run $\hat{r}^t_{A_H}$ that satisfies: $\hat{r}^t_{A_H} = \arg\min_{r^t_{A_H}} d_h(r^t_{A_H}, h)$, where A_H is the TAhd that corresponds to ϕ and $h = 0.5$. That is, find the control policy which guarantees the satisfaction of ϕ^{hard}, and maximizes the satisfaction of ϕ^{soft}, given the preference h.

6.3 Control Strategy

The solution to Problem 6.1 is performed offline and follows the outline we suggested in Chapter 5. It is inspired by the standard 3 steps procedure for single agent control synthesis; i) expressing the temporal logic specification as an automaton, ii) constructing the product of the automaton and the transition system, and iii) implementing graph search to find the shortest path. The planning follows the steps:

1. Construct a Timed Automaton with Hybrid Distance (TAhd) which represents the MITL specification.

2. Construct a Product Automaton as the product of the TAhd and a WTS representing the system dynamics.

3. Find the least violating path, i.e, the shortest path with respect to the hybrid distance, d_h, for $h = 0.5$.

The difference between the solution suggested here and the solution presented in Chapter 5 appears in step 1, i.e. the construction of the TAhd, where we now consider hard constraints as well as soft constraints. The details are given below.

6.3.1 Hard and Soft MITL Tasks to Timed Automaton with Hybrid Distance

In this section we consider the construction of a TAhd when both soft and hard constraints are given. The same approach can be used to construct a TAhd from a soft constraint (by assuming the hard constraint is empty), or a hard constraint (by assuming the soft constraint is empty).

We start by describing the construction of locations. To do so we formally define the evaluation sets φ and the set of subformula evaluation Ψ which were briefly introduced in Chapter 5:

Definition 6.1. A evaluation set φ_i of a subformula ϕ_i contains the possible evaluations of the subformula:

$$\begin{array}{ll} \{\phi_i^{unc}, \phi_i^{vio}, \phi_i^{sat}\} & \text{if } \phi_i \text{ is temporally bounded} \\ \{\phi_i^{unc}, \phi_i^{sat}\} & \text{if } \phi_i \text{ is non-temporally bounded, type I} \\ \{\phi_i^{unc}, \phi_i^{vio}\} & \text{otherwise} \end{array}$$

Definition 6.2. A subformula evaluation ψ of a formula ϕ is one possible outcome of the formula, i.e. a conjunction of elements of the evaluation sets:

$$\psi = \bigwedge_i \phi_i^{state},\ \phi_i^{state} \in \varphi_i$$

We will use $\Psi = \prod_{i \in \{1,..,n\}} \varphi_i$ to denote the set of all subformula evaluations ψ of a formula ϕ, i.e. all possible outcomes of ϕ at any time.

We can now construct the location set $S = \{s_i : i = 1, ..., |\Psi|\}$. Then $S_0 = s_j$ where $\psi_j = \bigwedge_i \phi_i^{unc}$, and $F = s_k$ where $\psi_k = \bigwedge_{i \in I} \phi_i^{sat} \wedge \bigwedge_{j \in J} \phi_j^{unc}$, where $I \cap J = \{1, .., |\Psi|\}$ and J contains the indexes of all ϕ_j which are non-temporally bounded type II (i.e. cannot be evaluated as satisfied). The set of clocks X must include at least one clock for each temporally bounded ϕ_i, two if there is both a lower and an upper bound. I_X is easily constructed such that $s \to \Phi_X \in I_X$ if $\phi_i^{vio} \notin \psi$ where ϕ_i is temporally bounded by Φ_X. The hybrid distance derivative mapping $I_H(s) = (d_1, d_2)$ is then constructed such that

$$d_1 = \begin{cases} |\Phi_X| & \text{if } \exists \phi_i^{vio} \in \psi \text{ that is temporally bounded, and } \Phi_X \\ & \text{is the set of clock constraints that bounds } \phi_i^{vio} \in \psi \\ 0 & \text{otherwise} \end{cases} \quad (6.1)$$

$$d_2 = \begin{cases} 1 & \text{if } \exists \phi_i^{vio} \in \psi \text{ that is non-temporally bounded} \\ 0 & \text{otherwise} \end{cases} \quad (6.2)$$

To construct the edges we first introduce some new definitions and notation:

Definition 6.3. The distance set of two subformula evaluations ψ and ψ' is defined as $|\psi - \psi'| = \{\phi_i : \phi_i^{state'} \neq \phi_i^{state}\}$. That is, it consists of all subformulas ϕ_i which are evaluated differently in the subformula evaluations.

We use $(\psi, g, a) \to \psi'$ to denote that all subformulas $\phi_i \in |\psi - \psi'|$ are i) evaluated as uncertain in ψ (i.e. $\phi_i^{unc} \in \psi$) and ii) re-evaluated to either satisfied or violated in ψ'(i.e. $\phi_i^{state'} \in \psi'$, where $state' \in \{vio, sat\}$) if action a occurs at time t satisfying guard g.

The edges can now be constructed in 4 steps:

i) Construct all edges corresponding to progress in regards of the specification (i.e. the edges a TBA would have) such that: $(s, g, a, s') \in E$ if $(\psi, g, a) \rightarrow \psi'$.

ii) Construct edges which correspond to non-temporally bounded soft constraint/s no longer being violated such that: $(s, g, a, s') \in E$ if

- $\forall \phi_i \in |\psi - \psi'|$, $\phi_i \in \phi^{soft}$ and is non-temporally bounded, and $\phi_i^{vio} \in \psi$,
- $(s'', g, a, s') \in E$ for some s'' where $|\psi - \psi'| = |\psi - \psi''|$

or

- $\forall \phi_i \in |\psi - \psi'|$, $\phi_i \in \phi^{soft}$ and is non-temporally bounded, and $\phi_i^{vio} \in \psi$
- $(s', g, a', s) \in E$, where a' contain all atomic propositions in a except the one/s associated with ϕ_i

iii) Construct edges which correspond to temporally-bounded soft constraint/s no longer being violated such that: $(s, g, a, s') \in E$ if

- $\exists \phi_i \in |\psi - \psi'|$, such that $\phi_i \in \phi^{soft}$ is temporally bounded, and $\phi_i^{vio} \in \psi$, $\phi_i^{sat} \in \psi'$, $\phi_i^{unc} \in \psi''$,
- $(s'', g', a, s') \in E$ and $(s'', g, a, s) \in E$, where g contain all clock constraints of g' except the one/s associated with any subformula ϕ_i such that $\phi_i^{unc} \in \psi'$ and $\phi_i^{vio} \in \psi$, and
- $\nexists\ \phi_i \in |\psi - \psi'|$ such that $\phi_i \in \phi^{hard}$

iv) Construct self-loops such that $(s, g, a, s) \in E$ if $\exists\ (g, a)$ such that

- $g \subseteq g'$ and $a \subseteq a'$, where $g' = \cap_i g_i$ and $a' = \cap_i a_i$ such that $\exists (s', g_i, a_i, s) \in E$ (i.e. g and a belongs to g' and a': the set of all guards and actions mapped to incoming edges), and
- $g \nsubseteq g''$ and $a \nsubseteq a''$ where $g'' = \cap_i g_i$ and $a'' = \cap_i a_i$ such that $\exists (s, g_i, a_i, s') \in E$ (i.e. g and a does not belong to g'' and a'': the sets of guards and actions mapped to outgoing edges).

To illustrate the TAhd constructed for hard and soft constraints we give a small example.

Example 6.1. We consider the formula $\phi_{hard} \wedge \phi_s oft$ where $\phi_{hard} = \Box \neg a$ and $\phi_{soft} = \Diamond_{t<5} b \wedge \Box d$. We recognize the hard formula and the first part of the soft formula as the formula from Example 5.4. The resulting TAhd is illustrated in Fig. 6.1.

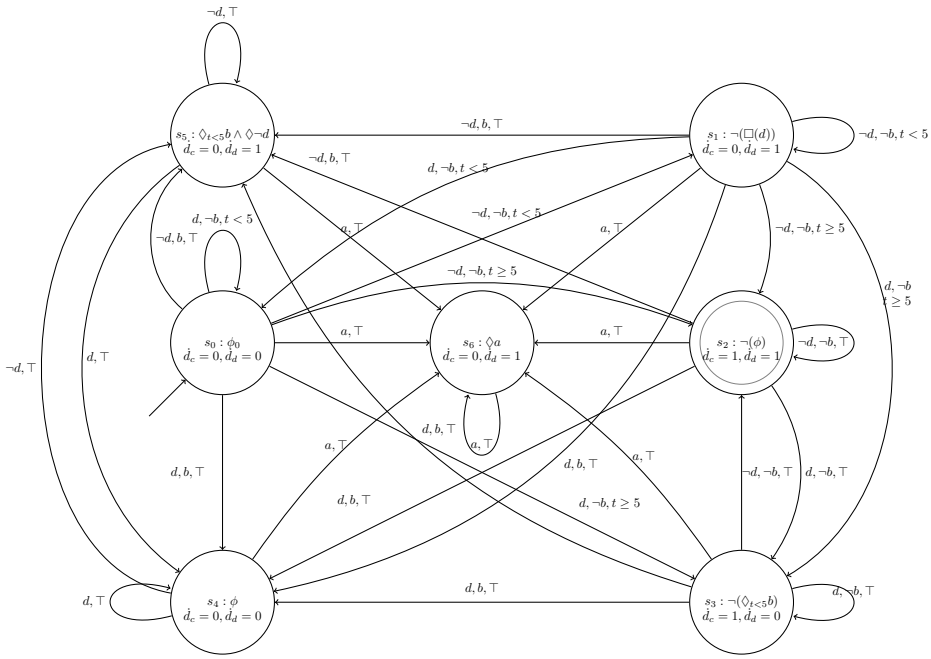


Figure 6.1: The TAhd corresponding to the MITL specification $\phi_{hard} \wedge \phi_{soft}$ where $\phi_{hard} = \Box\neg a$ and $\phi_{soft} = \Diamond_{t<5} b \wedge \Box d$.

6.3.2 Product Automaton and Control Synthesis

The product automaton of a WTS and a TAhd was given in Definition 5.6 in Chapter 5. The least violating path with respect to the soft constraint and the human feedback constant h, which satisfies the hard constraint, can now be determined by constructing the product automaton, applying the modified Dijkstra algorithm (Alg. 6), and projecting the found accepting run onto the WTS.

In Chapter 5 we showed that a solution to the algorithm is always found under the assumption that the temporally bounded part of the MITL formula is feasible on the given WTS when deadlines are disregarded. This result is however based on the fact that the TAhd was constructed to represent a soft constraint alone and does no longer apply when hard constraints are applied as well. The result can however be relaxed by adding the assumption that the hard constraint is feasible and does not contradict any eventually or until operators of the soft constraints when deadlines of the soft constraints are disregarded. With this extra assumption we can conclude that we will always find a solution.

6.4 Conclusion

In this chapter we have suggested a control synthesis framework for single-agents under hard and soft MITL constraints, given some human preference. Hence, we take into consideration the issue discussed in Chapter 4 (how to handle not finding a completely satisfying run) without limiting the system to soft constraints as we did in Chapter 5. The result is a framework which will always find a path if the hard constraint can be satisfied, allowing us to keep the guarantees while taking advantage of knowledge regarding which tasks can be relaxed.

The methods described in this chapter consider the single-agent case, however as with the approach of Chapter 5, it can be extended to the multi-agent case by either applying the remainder of the approach of Chapter 4 or the approach of the upcoming Chapter 7.

Chapter 7

Human-in-the-Loop Control with Preference Learning and Collision Avoidance for Multi-Agent Systems

In this chapter we take a step back to consider the implementation of the previously discussed methods. More specifically, we consider i) how to apply the single-agent control synthesis to a decentralized multi-agent system, ii) how to allow the human a more direct impact on the system by giving her input directly to the controller, without losing guarantees, iii) how to convert said input into a preference value which can be used by the modified Dijkstra algorithm, and iv) how to avoid collisions. The content of this chapter is part of the work presented in [58] and [57].

7.1 Introduction

One of the main disadvantages of centralized control is the issue of computational requirements. In the case of automata based control synthesis this corresponds to the number of states, commonly denoted as state explosion. To avoid this, we here consider a decentralized solution where each agent creates its own plan, and add a collision avoidance algorithm to the online planning to make up for the lack of joint planning.

Another aspect of the previous chapters which we aim to improve here is the role the human play in the system. Previously this was limited to decreasing or increasing the value of a weight constant which impacted the hybrid distance function. Humans generally want to obtain as much control as they can, and hence a successful system should maximize the humans control without losing the guarantee for safety. Here, we aim to do this by using a mixed-initiative controller [41–43]. That is, the human gives one part of the controller and a function is used to determine if the human control input is used in full, dampened or completely ignored. The function is designed to maximize the impact of the human while ensuring that the hard constraints are not violated. As a consequence, we must also consider how to determine the weight

constant which was previously given directly by the human. We do this by assuming that the human will use her input to steer the robot according to her preference. We can then assume that the resulting path is preferable to the previously planned path, and use the set of paths in an IRL algorithm [44, 45] to systematically learn what the value of the weight constant should be.

7.2 Problem Statement

The problem considered in this chapter is to, for each agent in a multi-agent system, i) find the plan which violates the given soft constraint the least, for some human preference, while satisfying the given hard constraint, ii) learn the human preference concerning the type of violation of the soft constraints based on human control input, and iii) avoid collisions with other agents by re-planning when the next target region is occupied. The input of each agent is assumed to be bounded with $|u_i| \leq u_{max}$, $\forall i \in \{1, ..., N\}$.

The hybrid distance (d_h) is used as the measurement of violation, where $d_h = 0$ corresponds to complete satisfaction. The human preference is indicated by the value of h. This can be expressed as four sub-problems:

Problem 7.1. Initial plan: Given a WTS T and an MITL specification $\phi = \phi^{hard} \wedge \phi^{soft}$, find the timed run r^t of T that corresponds to the automata timed run $\hat{r}^t_{A_H}$ that satisfies: $\hat{r}^t_{A_H} = \arg\min_{r^t_{A_H}} d_h(r^t_{A_H}, h)$, where A_H is the TAhd that corresponds to ϕ and $h = 0.5$. That is, find the control policy which guarantees the satisfaction of ϕ^{hard}, and maximizes the satisfaction of ϕ^{soft}, given the preference h.

Problem 7.2. Safety: Design a control law such that the input from the human (u_h) can not cause the agent to violate the hard constraints.

Problem 7.3. Learning preference and updating plan: Given a human control input u_h, update the estimation of h such that the resulting trajectory (up until this point in time) is optimal with respect to the hybrid distance. Given the updated value of h, find a new plan r^t_{new} (for the remainder of the task) such that $d_h(r^t_{A_H}, h)$ is minimized by the corresponding automata timed run $\overline{r}^t_{A_H}$. Assuming that the human has a value of h in mind and acts accordingly, the updated solution should thus satisfy $d_h(\overline{r}^t_{A_H}, h) < d_h(\hat{r}^t_{A_H}, h)$.

Problem 7.4. Collision avoidance: Given the location of all other agents in the system, find a new plan which does not include occupied states and otherwise follows the preferences of the human, if the imminent part of the trajectory crosses the location of another agent.

Remark 7.1. Note that Problem 7.1 is identical to Problem 6.1 which was solved in Chapter 6.

7.3 Control Strategy

Our strategy is two-parted; i) Find an initial plan through offline control synthesis by applying the approach in Chapter 6 on each agent for $h = 0.5$, and ii) design controller with human input in the form of mixed-initiative and perform online learning to update h and re-plan based on new knowledge of h and positions of neighbouring agents.

Below we will go into details on the online part.

7.3.1 Human Input through Mixed-Initiative Control

To allow the human direct input, while keeping the guarantees of not violating the hard constraints we implement a mixed-initiative controller:

$$u = u_r(x, \pi_s, \pi_g) + \kappa(x, \Pi)u_h(t) \tag{7.1}$$

for each transition $(\pi_s, \pi_g) \in \rightarrow$, where u_r is the control input from the system designed to follow the current plan (i.e. the initial plan found from Chapter 6 if no re-planning has occurred), and u_h is the human input. The problem then becomes to design κ such that ϕ^{hard} is never violated. To solve the problem we follow the same idea as in [58], namely to design κ such that:

i) it is zero if the agent is on the limit of entering an area which would violate ϕ^{hard},

ii) it is one if the agent is outside of said limit with a given safety margin,

iii) it is in the interval $[0, 1]$ if the agent is outside of the limit but inside of the safety margin, and

iv) in the area in between the limit and the safety margin it decreases when nearing the limit and increases when nearing the safety margin.

In [58], where we considered LTL specifications (i.e. no time bounds), this was done by choosing:

$$\kappa(x, \mathcal{O}_t) = \frac{\rho(d_t - d_s)}{\rho(d_t - d_s) + \rho(\varepsilon + d_s - d_t)} \tag{7.2}$$

where d_t is the minimum distance between the agent and any region within $\mathcal{O}_t$, $\rho(s) = e^{-1/s}$ for > 0 and $\rho(s) = 0$ for $s \leq 0$, $d_s > 0$ and $\varepsilon > 0$ are design parameters for safety, and $\mathcal{O}_t$ contains all regions $\pi \in \Pi$ which corresponds to a violating state $q \in Q$.

In [57] we considered MITL specifications and unlike in [58], we also had to consider the time constraints of ϕ^{hard}. If ϕ^{hard} includes temporally bounded operators it can be violated in almost all states $\pi \in \Pi$ of the WTS for some time t. More specifically, only the states which satisfy the spacial task of the temporally bounded

operator are safe for all time. For example, if the task is $\Diamond_I a$, then only states π such that $a \in L(\pi)$ are safe. Hence, if we do not take time into consideration in the design of κ, almost all states will belong to $\mathcal{O}_t$. It follows that, if we apply the solution in [58] directly it would be too conservative, resulting in $\kappa = 0$ in almost all states. As a result, the learning algorithm wouldn't have enough data and the human wouldn't have enough impact.

To solve this problem we use the set Q_T (containing all states which cannot reach an accepting state) which we constructed in the previous section, and construct a new set $Q_T^t = \{(q,t) : q \in Q_T, t = \min(x \in I_X^P(q))\}$ containing all states corresponding to the violation of ϕ^{hard} paired with the corresponding violated deadline (i.e. the minimum time required to enter the state). We then redefine:

$$d_t = \min_{(q,t)\in Q_T^t} dist(x,(q,t)) \tag{7.3}$$

$$dist(x,(q,t)) = \begin{cases} \|x - proj(q,T)\| & \text{if } t_0 + \hat{w}(\pi_0, proj(q,T)) > t \\ \infty & \text{otherwise} \end{cases} \tag{7.4}$$

where t_0 and π_0 are the time and state of the WTS at the time of calculation, and $\|x - proj(q,T)\| = \min_{x' \in proj(q,T)} \|x - x'\|$. That is, $dist(x,(q,t))$ is the distance between the current location and the region corresponding to q if the deadline is violated (and the transition would lead to the violation of ϕ^{hard}), and ∞ otherwise. The resulting d_t is then the minimum distance to a violating state, and hence equation (7.2) can be applied without the aforementioned issue.

7.3.2 Online Re-Planning

To re-plan the path when new knowledge is given (either an update of h through the human input or of the neighbouring agents positions), we modify the previously suggested search algorithm by adding some initial settings. More specifically, we allow for a initial time and initial values of the distances (hybrid, continuous and discrete) to be non-zero. The result is described by Algorithm 8.

Learning the Human Preference

In this chapter we consider how h can be determined based on u_h. The suggested method is an inverse reinforcement learning (IRL) [44] approach and the estimated value of h is iteratively improved when new knowledge is given in the form of human input (i.e. when $u_h \neq 0$) under the assumption that the human is trying to help the system. That is,

Assumption 7.1. The human input u_h is chosen such that d_h is optimized for the true value of h

$$Cost(r_P^{t,*}, h^*) = \min_{r_P^t} Cost(r_P^t, h^*) \tag{7.5}$$

Algorithm 8 Dijkstra Algorithm with Hybrid Distance as cost function and initial distances as input for Re-Planning implementation.

```
1: procedure DIJSKTRAHD2(P, h, τ_0, d_c^0, d_d^0, d_h^0)
2:     Q =set of states, q_0 =initial state, SearchSet = q_0, d(q, q') =weight of
       transition q ⇝ q' in P
3:     if q = q_0 then
4:         dist(q) = τ_0, d_h(q) = d_h^0, d_c(q) = d_c^0, d_d(q) = d_d^0
5:     else
6:         dist(q) = d_h(q) = d_c(q) = d_d(q) = ∞
7:     end if
8:     for q ∈ Q do
9:         pred(q) = ∅
10:    end for
11:    while pathFound = False & SearchSet ≠ ∅ do
12:        pick q ∈ SearchSet s.t. q = arg min(d_h(q))
13:        if q ∈ F then
14:            pathFound = True
15:        else
16:            find all q' s.t. q ⇝ q'
17:            for every q' do
18:                d_h^step = (h d̈_c(q) + (1 − h) d̈_d(q)) d(q, q')
19:                if d_h(q') > d_h(q) + d_h^step then
20:                    update dist(q'), d_h(q'), d_c(q'), d_d(q') and pred(q')
21:                    add q' to SearchSet
22:                    remove q from SearchSet
23:                end if
24:            end for
25:        end if
26:    end while
27:    if pathFound = True then
28:        r_hd^min = q
29:        while q ≠ q_0 do
30:            use pred(q) to iteratively form the path back to q_0:
31:            r_hd^min = [pred(q), r_hd^min]
32:        end while
33:        return r_hd^min, d_h, d_c, d_d
34:    else
35:        return r_hd^min = ∅
36:    end if
37: end procedure
```

where $r_P^{t,*}$ is the timed run of P which the human guides the agent through, h^* is the true value of h, and $Cost$ is the cost function which should be minimized by the optimal run.

We start by designing the cost function, which should be a function of the hybrid distance such that minimization of one is equivalent to minimization of the other. We suggest the following:

$$Cost(r_P^t, h) = d_h(proj(r_P^t, A_H), h) \tag{7.6}$$

where $proj(r_P^t, A_H)$ is the projection of the timed run of the product automaton P onto the TAhd A_H as defined below (Definition 7.1). We also define the projection onto the WTS T for later use.

Definition 7.1. The projections of a timed run of a product automaton $r_P^t = (\pi_1, s_1)(\pi_2, s_2), ..., (\pi_m, s_m)$ onto a TAhd A_H and a WTS T are defined as:

$$proj(r_P^t, A_H) = s_1, s_2, ..., s_m, \text{ and} \tag{7.7}$$

$$proj(r_P^t, T) = \pi_1, \pi_2, ..., \pi_m. \tag{7.8}$$

To determine the kth estimate of h we suggest solving

$$h_k = \underset{h \in [0,1]}{\arg\min} \sum_{i=1}^{k} p(Cost(r_P^{t,h}, h) - Cost(r_P^{t,i}, h)) \tag{7.9}$$

$$p(x) = \begin{cases} x & \text{if } x \leq 0 \\ \infty & \text{if } x > 0 \end{cases} \tag{7.10}$$

$$r_P^{t,h} = \underset{r_P^t \in R_P^t}{\arg\min} Cost(r_P^t, h) \tag{7.11}$$

where $r_P^{t,i}$, $i = 1, .., k$ are the previously suggested paths (i.e. $r_P^{t,1}$ is the initial plan and the outcome of Chapter 5 or Chapter 6), $R_P^t = \{r_P^t = q_1, q_2, ...q_m : r_P^{t,0} = q_1, q_2, ..., q_l, l \leq m\}$ and $r_P^{t,0}$ is the timed run of P which has been followed from start up until the time of the human input. That is, R_P^t is the set of timed runs of P which can be followed given the up-to-date trajectory. The function $p(x)$ is used to ensure that

$$Cost(r_P^{t,h_k}, h_k) \leq Cost(r_P^{t,i}, h_k), \ \forall \ i = 1, ..., k.$$

This removes any solutions h_k for which a previously suggested (and rejected) path $r_P^{t,i}$ has a lower value of d_h than the new path r_P^{t,h_k}. No loss of correct solutions occurs due to Assumption 7.1. The solution to (7.9) is the h which maximizes how much d_h decreases due to the human input.

The optimal timed run with respect to hybrid distance and h_k is then

$$r_P^{t,k+1} = \underset{r_P^t \in R_P}{\arg\min} Cost(r_P^t, h_k). \tag{7.12}$$

That is, the timed run $r_P^{t,h}$ calculated in (7.11) for $h = h_k$. The new path to follow is then found by the projection of $r_P^{t,k+1}$ onto the WTS, i.e, $r_{A_H}^{t,k+1} = proj(r_P^{t,k+1}, A_H)$. The solution to (7.9) and (7.12) can be found by implementing Algorithm 9.

Algorithm 9 Finds the h which maximizes the decrease of d_h if the path enforced by the human is followed instead of the previously suggested paths: h_k, the optimal path to continue to follow given h_k: $r_P^{t,k+1}$ and the projection of said path onto the TAhd A_H: $r_{A_H}^{t,k+1}$

```
procedure IRL4H((d_c(r_P^{t,0}), ..., d_c(r_P^{t,k})),(d_d(r_P^{t,0}), ..., d_d(r_P^{t,k})), P)
                    ▷ r_P^{t,0} : timed run of P followed from start to current state
                ▷ r_P^{t,i}(∀i > 0) : number i of previously suggested timed runs of P
    δ =design parameter for the step size of the optimization
    update Q^{init}, τ_0, d_c^0, d_d^0 and d_h^0
    for h = 0, δ, 2δ, ..., 1 do
        r_P^{t,h} and d_h ←DIJKSTRAHD2(P, h, τ_0, d_c^0, d_d^0, d_h^0)          ▷ Alg. 8
        Cost(r_P^{t,h}, h) = d_h
        for i = 1, ..., k do
            Cost(r_P^{t,i}, h) = h d_c(r_P^{t,i}) + (1 − h) d_d(r_P^{t,i})
            p(i, h) = Cost(r_P^{t,0,h}, h) − Cost(r_P^{t,i}, h)
            if p(i, h) > 0 then
                p(i, h) = ∞
            end if
        end for
    end for
    h_k = arg min_h Σ_{i=1}^{k} p(i, h)
    r_P^{t,k+1} and d_h ←DIJKSTRAHD2(P, h, τ_0, d_c^0, d_d^0, d_h^0)            ▷ Alg. 8
    r_{A_H}^{t,k+1} = proj(r_P^{t,k+1}, A_H)                                    ▷ Def. 7.1
end procedure and return h_k, r_P^{t,k+1}, r_{A_H}^{t,k+1}
```

Remark 7.2. Under the assumption that the transition times of the WTS are over-approximations, it is possible that a re-planned path has a lower hybrid distance than an initial path. The reason for this is that the already performed progress required less time than estimated. In this case the re-planning will have a start time which is lower than the estimated progress time at the corresponding state used by the offline algorithm. Based on this, one could argue that it would be better if each agent re-planned multiple times online regardless of whether an imminent state is occupied. The downside of this is the loss of performance time (due to standing still while re-planning).

Collision Avoidance

In this chapter we suggest a method for online collision avoidance given knowledge of the neighbouring agents position. We do so by considering the occupied regions and constructing a set of forbidden states (states which cannot reach the accepting states), which we will denote as Q_T. Q_T can be determined indirectly by first finding $Q_T^{-1} = Q \backslash Q_T$ (the set of states which an accepting state can be reached from). Q_T^{-1} is found iteratively by: $q \in Q_T^{-1}$ if $q \rightsquigarrow q'$ and $q' \in Q_T^{-1}$, where initially $Q_T^{-1} = \mathcal{F}$. We can now apply the collision avoidance algorithm described in Algorithm 10, where we have made use of second part of Definition 7.1.

Remark 7.3. Algorithm 8 attempts to implement the time limit T_{wait}^{max} as a maximum time the agent can stay in its current region before moving out of the way (to a neighbouring region) to let other agents (which are located in its path) pass. However, it is possible that this fails. This will be the case if all neighbouring regions are unsafe or occupied. If this happens, the agent will be forced to continue to wait until one of the safe neighbouring regions is free, and the maximum wait time will be exceeded.

Remark 7.4. Further improvement of the collision avoidance could be done by finding a way to update the product automaton such that the occupied state is disregarded only in the near future. This would allow the agent to find paths where the occupied state is used further in the future, when it may be free.

7.4 Simulations

A simulation with two agents, each following the dynamics in eq. (7.13), has been performed. Agent 1 is partially controlled by a human user, i.e. u_1 follows eq. (7.1), while agent 2 is fully autonomous (see eq. (7.14)).

$$\dot{x}_i = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix} x_i + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} u_i, \; i = 1, 2 \tag{7.13}$$

$$u_1 = u_r(x, \pi_s, \pi_g) + \kappa(x, \Pi) u_h(t) \tag{7.14}$$

$$u_2 = u_r(x, \pi_s, \pi_g) \tag{7.15}$$

Agent 1 is tasked with visiting areas c and d, while agent 2 is tasked with visiting areas e and f, both with soft deadlines. Both agents should also try to avoid areas marked b while they are strictly forbidden to enter area a. The resulting MITL specifications are $\phi_1 = \phi_1^{hard} \wedge \phi_1^{soft} = (\Box\neg a) \wedge (\Box\neg b \wedge \Diamond_{\leq t_1} c \wedge \Diamond_{\leq t_2} d)$ and $\phi_2 = \phi_2^{hard} \wedge \phi_2^{soft} = (\Box\neg a) \wedge (\Box\neg b \wedge \Diamond_{\leq t_3} e \wedge \Diamond_{\leq t_4} f)$. The assumption $t_1 \geq t_2$ and $t_3 \geq t_4$ has been made to minimize the number of edges of the resulting TAhd's. The assumption can be made without loss of generality since the labels c, d, e and f can be chosen such that the assumption holds. To clarify this we give a small example;

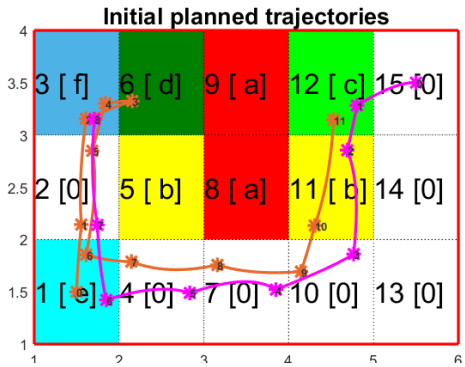
Algorithm 10 *collAv*() Collision Avoidance of agent i

```
1: procedure COLLAV(x = x_1, x_2, ..., x_k, p_c = q_c, q_{c+1}, ..., q_{goal}, P,h)
                    ▷ x =positions of all agents close enough to risk collision
            ▷ p_c = current discrete plan (from current state to goal state) of P
2:     PathFound = False
3:     while pathFound = False do
4:         for j = 1 : k do
5:             if x_j ∈ π_{c+1} where π_{c+1} = proj(q_{c+1}, T) then
6:                 Occupied = True
7:             end if
8:         end for
9:         if Occupied == True then
10:            update τ_0, d_c^0, d_d^0, d_h^0 and Q^{init}
11:            set d(q, q') = ∞ ∀q if proj(q', T) = π_{c+1}
12:            r_P^{t,k+1} ←DIJKSTRAHD2(P, h, τ_0, d_c^0, d_d^0, d_h^0)        ▷ Alg. 8
13:            if r_P^{t,k+1} ≠ ∅ then
14:                pathFound = True
15:            end if
16:        end if
17:        if pathFound = True then
18:            break
19:        else▷ Wait before a new attempt is made (to see if region becomes free)
20:            wait for t = ΔT
21:            set T_{wait} = T_{wait} + ΔT
22:        end if
23:        if T_{wait} > T_{wait}^{max} then                    ▷ Move out of the way
24:            update τ_0
25:            set q ∈ F if proj(q, T) = π where π ∈ Π_{neighboors}, and q ∉ Q_T
26:            r_P^{t,k+1} ←DIJKSTRAHD2(P, h, τ_0, d_c^0, d_d^0, d_h^0)        ▷ Alg. 8
27:            if r_P^{t,k+1} ≠ ∅ then
28:                pathFound = True
29:            end if
30:        end if
31:    end while
32: end procedure and return r_P^{t,k+1}
```

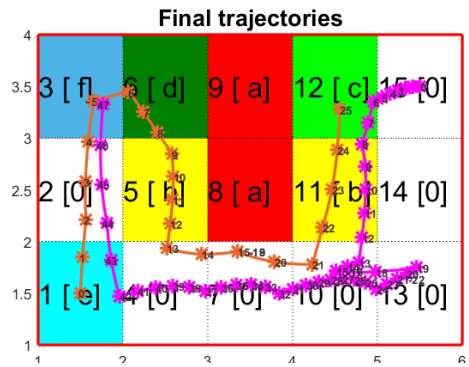
Example 7.1. Assume the MITL task $\Diamond_{\leq t} a \wedge \Diamond_{\leq t'} a'$ has been given and we have assumed $t \leq t'$. Since a and a' symbolizes some atomic propositions, we can arbitrarily choose which one should symbolize which property to ensure that the assumption hold. For instance, if a and a' symbolizes two rooms R_1 and R_2, and the deadlines are such that the agent must visit R_1 before 5 time units have passed and R_2 before 2 time units have passed, we simply pick $a = R_2$ and $a' = R_1$ since $t = 2 \leq t' = 5$.

The control input from the system, i.e. u_r, is determined following the method we suggested in Chapter 3, where for each transition in the WTS a controller is found such that: i) the velocity of the agent is positive in the direction of the transition for every x in the start region, and ii) the velocity of the agent is negative in the direction of any other edge on the given edge. That is, the controllers are designed to steer the agent towards the desired edge, while stopping the agent from exiting the starting region through any other edge. The transition times of the WTS's were also determined following Chapter 3 and are the maximum times required for each transition to be guaranteed. It follows that the transitions may actually be faster. Since the violation distances used during planning consider the transition times, it follows that they are worst case approximations, and that they may be smaller in implementation when the planned paths are followed. The real violation distances are used in the online re-planning when evaluating the followed trajectories to determine the current d_h. The estimates are used to plan the future path.

The workspace and the initial plans for each agent is illustrated in Figure 7.1a. During the online run the human user has a chance to apply control input every 0.015 time steps. Figure 7.1b illustrates one possible outcome. Here, the human decided to apply the control input $u_h = (0, -u_{max})$ during time $[0.09, 0.135]$ (step 6-9), at which point agent 1 was steered into region 5 instead of region 3. At this point the agent applied the Algorithm 9 to determine that the optimal h was 1, and re-planned accordingly. The next interesting event occurred at step 15 (or time= 0.225) where agent 1 and 2 blocked each other by being in the others goal region. This was solved by the agents applying the collision avoidance algorithm (Alg. 10), resulting in agent 2 moving out of the way (into region 13) and agent 1 waiting until region 10 was free. At step 21 (time= 0.315) agent 2 was once again blocked by agent 1, which hadn't left region 10 yet. Agent 2 once again applied Algorithm 10, resulting in agent 2 waiting until region 10 was free. The estimated values of the violation distances (used during planning) and the real violation distances (calculated from the final trajectories) are given in Table 7.1. As expected the real distances are smaller than the estimates (due to the fact that the agents move faster than estimated by the abstraction). Comparing the initial plans with the final plans, the hybrid distance is decreased for agent 1 while it is increased for agent 2. The improvement of agent 1 is due to the path being planned with the correct value of h. The negative change for agent 2 is caused by the collision avoidance forcing the agent to take a longer route.



(a) Initial plans of the 2 agents with dynamics according to (7.13), tasked with the MITL specifications ϕ_1 and ϕ_2, where $t_1 = 0.5$, $t_2 = 0.9$, $t_3 = 0.01$ and $t_4 = 0.03$. For the initial planning both agents use $h = 0.5$. Agent 1 follows the orange trajectory and agent 2 follows the magenta trajectory. Agent 1 starts in region 1 and agent 2 starts in region 15.



(b) Final trajectories of agent 1 and 2, where human input has been given to agent 1 to change the transition $6 \to 3$ to $6 \to 5$, indicating that keeping deadlines is more important than avoiding b, i.e. $h = 1$. Collision avoidance has been applied by both agents at step 15, and again by agent 2 at step 21.

Figure 7.1: Initial planned and final trajectories of the agents in the case study. Each number/star along the trajectories indicates one iteration where the human had a chance to change her control input, the time step in between is 0.015 time units. The tasks are to avoid the red areas and preferably avoid the yellow areas, while agent 1 should visit the green areas and agent 2 should visit the blue areas.

7.5 Conclusion

In this chapter we have presented a decentralized control synthesis framework for a multi-agent system under hard and soft constraints given as MITL specifications. The framework uses mixed initiative control to allow a human-in-the-loop to affect the trajectories of the agents while keeping the guarantees of satisfaction for the hard constraints. The human input is used in an IRL approach to learn the value of a weight assigning constant which indicates the human preference considering the manner of violation of the soft constraints. A collision avoidance algorithm is used to ensure safety. The result is a control policy which guarantees satisfaction of hard constraints and maximizes the satisfaction of soft constraints with respect to human preference, while avoiding collisions.

Table 7.1: Values of the violation distances: d_c, d_d and d_h for agent 1 and agent 2 in the case study. The table shows the estimated values used when constructing the initial and final plans and the real values calculated from the resulting trajectories when the plans had been followed. For agent 1 we have used $h = 1$ (the learnt human preference), and for agent 2, $h = 0.5$ (the initial setting) was used.

		Initial Plan/Trajectory		Final Plan/Trajectory	
		Estimates	Real Values	Estimates	Real Values
Ag. 1	d_c	0.0418	0.0405	0	0
	d_d	0.0418	0.0405	0.1137	0.1037
	d_h	0.0418	0.0405	0	0
Ag. 2	d_c	0.9351	0.7573	1.6571	1.2795
	d_d	0.0719	0.0664	0.0759	0.0668
	d_h	0.7099	0.4118	0.8665	0.6731

Chapter 8

Summary and Future Work

8.1 Summary

In this thesis we presented results on the control synthesis problem for single-agent and multi-agent systems under MITL specifications. We considered tasks which are individual and cooperative, and include hard and soft constraints. When soft constraints are considered, we apply a human-in-the-loop set-up either by a directly given constant indicating a violation preference, or by a mixed-initiative controller from which the system can learn the preference. Both centralized and decentralized approaches have been considered for the multi-agent case.

In Chapter 3 we presented an abstraction method to construct a WTS for a single-agent system under affine dynamics located in a square workspace. The weights of the WTS are determined as an over-approximation of the minimum time required to guarantee a transition, which is calculated as a function of the agents dynamics including the transition control input. The suggested transition control inputs were determined through an optimization problem to achieve every transition as fast as possible while guaranteeing that no other regions are entered. The results has been used continuously throughout the remainder of the thesis to abstract the agent-dynamics.

In Chapter 4 we presented a control synthesis framework for a multi-agent system under hard constraints consisting of both individual and cooperative tasks. The approach is centralized and automata-based, and produces a set of control sequences which, when applied in an event-triggered fashion for each agent, guarantee the satisfaction of the constraints. The method uses automata products, a graph search algorithm and projection to determine an accepting run, which can then be connected to a set of control input sequences through the WTS of each agent. If no accepting run is found, we can not conclude that it doesn't exist since this may be due to the overestimations.

In Chapter 5 we presented a solution to the issue of not finding existing accepting runs, which the approach in Chapter 4 had. We did this by considering the specifications as soft constraints and introducing a novel metric which quantifies

the violation, namely the notion of hybrid distance. The hybrid distance consists of a continuous and a discrete distance, quantifying the violation of deadlines and spacial constraints, and a weight constant which determines the relative importance of the two distances. The weight constant h is denoted as the human preference constant, and is assumed to be given. By including the hybrid distance derivatives in the TBA and using it in the cost function of the search algorithm a least violating run can be found. The framework will always find a run if the reachability operators are feasible disregarding the deadlines, where the reachability operators are $\Diamond$ and $\mathcal{U}$. That is, if all actions on the edges along one path from the initial location to an accepting location of the TAhd have corresponding labels on some states in the WTS, and those state are reachable from the initial state, then a least violating run exists. If there exist accepting runs, one of them will be found. The method produces a discrete path in the WTS corresponding to a least violating run, a sequence of control inputs which make the agent follow the suggested path and the estimated value of the hybrid distance. The estimated value of the hybrid distance is an over-approximation, under the assumption that the weights of the WTS are over-approximations of the transition times. That is, the produced hybrid distance approximation is a value which we can guarantee that the real hybrid distance doesn't exceed. Here, the real hybrid distance refers to the value of the metric calculated from an online run or simulation. The difference between the approximations and the real values were studied and discussed by comparing the result from the control synthesis framework to the result from applying the control input sequence in an event-triggered fashion on the agent-dynamics in a Matlab simulation. This chapter focused on the single-agent case, but it could easily be extended to the multi-agent case by following the remaining steps of Chapter 4, and making some small modifications to the products.

In Chapter 6 we built on the work of Chapter 5 and extended it to include both hard and soft constraints. The result is a framework which finds the least violating run of the soft constraint, which is accepting of the hard constraint. The advantage of this approach is that we keep the guarantees of the approach in Chapter 4, while maximizing our chances of finding a solution by allowing for the relaxation of the soft constraint which we achieved in Chapter 5. This does however not exclude the possibility of not finding existing solutions, since there may still be overestimations which make runs that satisfy the hard constraint appear to violate it.

In Chapter 7 we presented a decentralized approach to an adaptive online execution of the plan found by the methods in the previous chapters. The approach included a mixed-initiative controller which allows us to keep our guarantees of the satisfaction of the hard constraint while giving the human direct input to the system. It is assumed that the human is trying to help guide the system to act according to her preference and the human input is therefore used in an IRL method to determine the value of the human preference constant h. It is also assumed that each agent gets updated knowledge of the neighbouring agents position. Any new knowledge, either of neighbours position or the value of h, induces an online re-planning to ensure that the current plan is adapted to the value of h, and to avoid collisions.

8.2 Future Work

There are several possibilities for extending and continuing the work presented in this thesis. The extensions are progressively presented for the content of each chapter below.

The approach in Chapter 3 could be extended to include a wider range of dynamics by determining a function for the maximum required time that is more general. It would also be interesting to improve the method of finding the overestimated time, by considering how the speed changes throughout the regions, in order to eliminate the difference between the overestimation and the actual worst case transition time.

Chapter 4-6, build on each other, and we will therefore consider the extension together. The implementation of the hybrid distance could be extended to include the violation of the lower time-limits of the time-intervals related to the MITL formula. The extension is not straight forward since it may require the system to make a decision of what violation type to consider which affects the future behaviour. Namely, the agent has to make a decision when an action occur early to either view it as a violation of the lower-time limit and continue with the remainder of the task, or to redo the action within the time limits (or after the upper-time limit) and risk other type of violations. There may also be a need to determine whether a lower or upper time-limit is violated if the desired action occurs multiple times. Another possible extension is to construct a tool for automatic translation of a MITL formula into a TBA and a TAhd.

Chapter 7 should be extended by deriving a proof of convergence for the h-learning algorithm. It would also be interesting to consider a leader-follower structure of the multi-agent system, where agents under mixed-initiative controllers are considered leaders and self-steering agents are considered followers. The approach used for collision avoidance could be extended to include other temporary tasks, allowing the human user to add smaller assignments along the way.

Extensions to the work in its entirety include performing experiments on robotic platforms to validate and demonstrate the methods which have been suggested, and developing an abstraction-free framework with the hybrid distance implemented directly on the continuous dynamics.

Bibliography

[1] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, Cambridge, MA, 1st edition, 2008.

[2] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. *Formal Methods for Discrete-time Dynamical Systems*, volume 89. Springer, 2017.

[3] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, Dec 2009.

[4] Georgios E. Fainekos, Antoine Girard, Hadas Kress-Gazit, and George J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343 – 352, 2009.

[5] Alexandros Nikou, Jana Tumova, and Dimos V. Dimarogonas. Cooperative task planning of multi-agent systems under timed temporal specifications. *CoRR*, abs/1509.09137, 2015.

[6] Alexandros Nikou, Dimitris Boskos, Jana Tumova, and Dimos V Dimarogonas. Cooperative planning for coupled multi-agent systems under timed temporal specifications. *American Control Conference (ACC)*, 2017.

[7] Jie Fu and Ufuk Topcu. Computational methods for stochastic control with metric interval temporal logic specifications. *CoRR*, abs/1503.07193, 2015.

[8] Amit Bhatia, Matthew R Maly, Lydia E Kavraki, and Moshe Y Vardi. Motion planning with complex goals. *IEEE Robotics & Automation Magazine*, 18(3):55–64, 2011.

[9] Meng. *Hybrid Control of Multi-robot Systems under Complex Temporal Tasks*. PhD thesis, EES, KTH Royal Institute of Technology, 2016.

[10] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Translating structured english to robot controllers. *Advanced Robotics*, 22(12):1343–1359, 2008.

[11] Georgios E Fainekos and George J Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.

[12] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Where is waldo? sensor based temporal logic motion planning. *mag*, 2007.

[13] Wongpinomsarn, Topcu, and Murray. Receding horizon control for temporal logic specifications. In *2010 International Conference on Hybrid Systems: Computational and Control (HSCC)*.

[14] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M Murray. Synthesis of control protocols for autonomous systems. *Unmanned Systems*, 1(01):21–39, 2013.

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

[16] Deepak Souza and Pavithra Prabhakar. On the expressiveness of mtl in the pointwise and continuous semantics. *International Journal on Software Tools for Technology Transfer*, 9(1):1–4, 2007.

[17] Patricia Bouyer. From qualitative to quantitative analysis of timed systems. *Mémoire d'habilitation, Université Paris*, 7:135–175, 2009.

[18] Joël Ouaknine and James Worrell. On the decidability of metric temporal logic. In *Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on*, pages 188–197. IEEE, 2005.

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

[20] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M Murray, and Sanjit A Seshia. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 239–248. ACM, 2015.

[21] Xiaoqing Jin. Verification and validation of hybrid systems. 2013.

[22] Alexandre Donzé and Oded Maler. *Robust satisfaction of temporal logic over real-valued signals*. Springer, 2010.

[23] Alexandre Donzé, Thomas Ferrere, and Oded Maler. Efficient robust monitoring for stl. In *Computer Aided Verification*, pages 264–279. Springer, 2013.

[24] 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.

[25] David C Conner, Hadas Kress-Gazit, Howie Choset, Alfred A Rizzi, and George J Pappas. Valet parking without a valet. In *2007 IEEE/RSJ international conference on intelligent robots and systems*, pages 572–577. IEEE, 2007.

[26] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[27] Rajeev Alur. Timed automata. In *Computer Aided Verification*, pages 8–22. Springer, 1999.

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

[29] Dejan Ničković and Nir Piterman. From mtl to deterministic timed automata. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 152–167. Springer, 2010.

[30] Oded Maler, Dejan Nickovic, and Amir Pnueli. From mitl to timed automata. In *Formal Modeling and Analysis of Timed Systems*, pages 274–289. Springer, 2006.

[31] Thomas Brihaye, Morgane Estiévenart, and Gilles Geeraerts. On mitl and alternating timed automata. In *Formal Modeling and Analysis of Timed Systems*, pages 47–61. Springer, 2013.

[32] Pierre-Jean Meyer and Dimos V. Dimarogonas. Compositional abstraction refinement for control synthesis. *Nonlinear Analysis: Hybrid Systems*, 2017. to appear.

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

[34] M Lahijanian and M Kwiatkowska. Specification revision for markov decision processes with optimal trade-off. In *Decision and Control (CDC), 2016 IEEE 55th Conference on*, pages 7411–7418. IEEE, 2016.

[35] Antoine Girard and George J Pappas. Approximation metrics for discrete and continuous systems. *IEEE Transactions on Automatic Control*, 52(5):782–798, 2007.

[36] Georgios E Fainekos and George J Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.

[37] S. Carr, N. Jansen, R. Wimmer, J. Fu, and U. Topcu. Human-in-the-loop synthesis for partially observable markov decision processes. In *2018 Annual American Control Conference (ACC)*, pages 762–769, June 2018.

[38] Rachel Schlossman, Minkyu Kim, Ufuk Topcu, and Luis Sentis. Toward achieving formal guarantees for human-aware controllers in human-robot interactions. *arXiv preprint arXiv:1903.01350*, 2019.

[39] M. Cao, A. Stewart, and N. E. Leonard. Integrating human and robot decision-making dynamics with feedback: Models and convergence analysis. In *2008 47th IEEE Conference on Decision and Control*, pages 1127–1132, Dec 2008.

[40] V. Okunev, T. Nierhoff, and S. Hirche. Human-preference-based control design: Adaptive robot admittance control for physical human-robot interaction. In *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, pages 443–448, Sept 2012.

[41] Wenchao Li, Dorsa Sadigh, S. Shankar Sastry, and Sanjit A. Seshia. Synthesis for human-in-the-loop control systems. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 470–484, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[42] S. G. Loizou and V. Kumar. Mixed initiative control of autonomous vehicles. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1431–1436, April 2007.

[43] Li Wang, Aaron D Ames, and Magnus Egerstedt. Multi-objective compositions for collision-free connectivity maintenance in teams of mobile robots. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 2659–2664. IEEE, 2016.

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

[45] S. Natarajan, G. Kunapuli, K. Judah, P. Tadepalli, K. Kersting, and J. Shavlik. Multi-agent inverse reinforcement learning. In *2010 Ninth International Conference on Machine Learning and Applications*, pages 395–400, Dec 2010.

[46] iRobot's robot vacuum cleaner Roomba. `https://www.irobot.se/`. Accessed: 2019-04-28.

[47] Coyote III search and rescue robot developed by university of bremen and dlr. `https://robotik.dfki-bremen.de/en/research/robot-systems/coyote-iii.html`. Accessed: 2019-04-28.

[48] KUKA robot working with humans in the Volkswagen factory. `https://www.kuka.com/en-de/press/news/2016/10/vw-commits-to-human-robot-collaboration-in-wolfsburg`. Accessed: 2019-04-28.

[49] M. Kloetzer, X. C. Ding, and C. Belta. Multi-Robot Deployment from LTL Specifications with Reduced Communication. *50th IEEE Conference on Decision and Control (CDC 2011)*, pages 4867–4872, 2011.

[50] M. Guo and D. Dimarogonas. Multi-Agent Plan Reconfiguration Under Local LTL Specifications. *The International Journal of Robotics Research*, 34(2):218–235, 2015.

[51] Y. Kantaros and M. Zavlanos. A Distributed LTL-Based Approach for Intermittent Communication in Mobile Robot Networks. *American Control Conference (ACC), 2016*, pages 5557–5562, 2016.

[52] Ebru Aydin Gol and Calin Belta. Time-constrained temporal logic control of multi-affine systems. *Nonlinear Analysis: Hybrid Systems*, 10:21–33, 2013.

[53] Y. Zhou, D. Maity, and John S. Baras. Timed Automata Approach for Motion Planning Using Metric Interval Temporal Logic. *European Control Conference (ECC 2016)*, 2016.

[54] S. Karaman and E. Frazzoli. Vehicle Routing Problem with Metric Temporal Logic Specifications. *47th IEEE Conference on Decision and Control (CDC 2008)*, pages 3953–3958, 2008.

[55] Sofie Andersson, Alexandros Nikou, and Dimos V Dimarogonas. Control synthesis for multi-agent systems under metric interval temporal logic specifications. *20th World Congress of the International Federation of Automatic Control (IFAC WC 2017)*, 2017.

[56] S. Andersson and D. V. Dimarogonas. Human in the loop least violating robot control synthesis under metric interval temporal logic specifications. *European Control Conference (ECC)*, 2018.

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

[58] Meng Guo, Sofie Andersson, and Dimos V Dimarogonas. Human-in-the-loop mixed-initiative control under temporal tasks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6395–6400. IEEE, 2018.

[59] Paulo Tabuada. Event-triggered real-time scheduling of stabilizing control tasks. *Automatic Control, IEEE Transactions on*, 52(9):1680–1685, 2007.

[60] Meng Guo and Dimos V Dimarogonas. Reconfiguration in motion planning of single-and multi-agent systems under infeasible local ltl specifications. In *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, pages 2758–2763. IEEE, 2013.

[61] Sofie Andersson. Automatic control design synthesis under metric interval temporal logic specifications, 2016.