Optimal Planning with Temporal Logic Specifications

by

Sertac Karaman

Submitted to the Department of Mechanical Engineering in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2009

Author .... [illegible] ......
Department of Mechanical Engineering
May 18, 2009

Certified by .... [illegible] ......
Emilio Frazzoli
Associate Professor
Thesis Supervisor

Certified by .... [illegible] ....
John Leonard
Professor
Mechanical Engineering Faculty Thesis Reader

Accepted by .... [illegible] ....
David E. Hardt
Chairman, Department Committee on Graduate Students
Mechanical Engineering Department

Optimal Planning with Temporal Logic Specifications

by

Sertac Karaman

Submitted to the Department of Mechanical Engineering
on May 18, 2009, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Most of the current uninhabitated Aerial Vehicles (UAVs) are individually monitored, commanded and controlled by several operators of different expertise. However, looking forward, there has been a recent interest in multiple-UAV systems, in which the system is only provided with the high-level goals and constraints, called the "mission specifications," and asked to navigate the UAVs such that the mission specifications are fulfilled. A crucial part in designing such multiple-UAV systems is the development of coordination and planning algorithms that, given a set of high-level mission specifications as input, can synthesize provably correct and possibly optimal schedules for each of the UAVs.

This thesis studies optimal planning problems in a multiple-UAV mission planning setting, where the mission specifications are given in formal languages. The problem is posed as a novel variant of the Vehicle Routing Problem (VRP), in which temporal logics and process algebra are utilized to represent a large class of mission specifications in a systematic way. The thesis is structured in two parts. In the first part, two temporal logics that are remarkably close to the natural language, namely the linear temporal logic LTL_{-X} and the metric temporal logic (MTL), are considered for specification of a large class of temporal and logical constraints in VRPs. Mixed-integer linear programming based algorithms, which solve these variants of the VRP to optimality, are presented. In the second part, process algebra is introduced and used as a candidate for the same purpose. A tree search based anytime algorithm is given; this algorithm is guarranteed to find a best-first feasible solution in polynomial time and improve it to an optimal one in finite time.

Thesis Supervisor: Emilio Frazzoli
Title: Associate Professor

Mechanical Engineering Faculty Thesis Reader: John Leonard
Title: Professor

Acknowledgments

First and foremost, I would like to thank my thesis advisor, Emilio Frazzoli, without whose support not only this particular thesis, but learning all that I learned throughout graduate school would have been virtually impossible. Together with Emilio Frazzoli, I also would like to acknowledge Jonathan How, John Leonard, Nancy Lynch, Asuman Ozdaglar, Seth Teller, and Alan Willsky, among many MIT faculty, all of whom have been great mentors and most importantly a memorable inspiration for my life after graduation. Knowing that it is far down the road, I wish to become a researcher and a teacher like them one day in my career.

Apart from my thesis, I have worked in various projects with Jonathan How, John Leonard, and Seth Teller. They all expanded my knowledge and my vision to different horizons. I am very glad, I have had their support. John Leonard has also been the mechanical engineering faculty thesis reader for this thesis, for which I am very grateful.

Parts of this thesis was written in the Air Force Research Laboratory (AFRL) in Dayton, OH, where I have been a visiting scholar for a couple of months. A large part of this thesis could not have been written, if I have not been to the AFRL in the summer of 2008. I am very grateful to Corey Schumacher and Siva Banda, who had put a great effort to give me a place in the AFRL. I am also very grateful to Steven Rasmussen, Derek Kingston, Ray Holsapple, Philip Chandler, and various other coworkers in the AFRL. This thesis was shaped by their input, suggestions, and long-lasting discussions with them.

I also would like to acknowledge Anouck Girard of the University of Michigan and all her group. Their valuable comments and input to my work has been a major driver for my research.

Even though it is impossible to list them all, I would like to thank my friends I met at MIT and in Boston as well as my office mates both in the Laboratory for Information and Decision Systems and the Computer Science and Artificial Intelligence Laboratory. Some of them that have and will continue to have a remarkable place in my life include Christiana Athanasiou, Amit Bhatia, Andrew Correa, Michelangelo Graziano, Yola Katsargyri, Marco Pavone, Georgios Papadopoulos, Navid Sabbaghi, Ariana, Christine, and Ricardo Sanfelice, Ketan Savla, Courtney and Tom Temple, Kyle Treleaven, Matthew Walter, Beracah Yankama, and Yetkin Yilmaz. I have shared my ideas with them, gotten their feedback, and shaped my thesis accordingly. I am very grateful to them for their input and their friendship.

Finally, I would like to thank my parents and my brother. I know that my mother deserves to see me graduate more than I do.

Contents

List of Figures

Chapter 1

Introduction

Most of the current Uninhabited Aerial Vehicle (UAV) systems require several human operators with different expertise to operate a single UAV. However, looking forward, multiple-UAV systems operated by a single operator are becoming a reality due to the recent developments in the UAV technology. Since a single operator cannot oversee and command all the UAVs individually, but can only provide the system with a set of high-level goals, multiple-UAV systems have to be supplied with effective planning algorithms, which automatically generate trajectories for UAVs so as to fulfill these high-level goals. Moreover, one of the foremost advantages of multiple-UAV systems has been the significant performance improvement through cooperation; algorithms that can optimally exploit this property are favored in practice. Such multiple-UAV systems can provide several advantages in both military and civilian applications, especially when the infrastructure is weak and a diversity of capabilities (in terms of sensory and weapon systems) is required. Optimal planning algorithms, which can handle complex constraints and objectives, can contribute also to future logistics systems, where the variety and complexity of operations play a crucial role. These logistics problems involve scheduling of several deliveries, most of which take place in highly dependent stages. The dependencies between the stages of a logistic operation may include temporal constraints such as ordering of different events or logical constraints such as having the chance to choose between different options. In such a scenario, planning algorithms equipped with natural interfaces can render specification of the mission/operation fast and easy.

A large class of multiple-UAV missions, as well as other logistics problems, can be modeled as Vehicle Routing Problems (VRP). Informally speaking, a VRP is defined by a set of customers and a fleet of vehicles, and the objective is to find a path for each vehicle in the fleet to visit all the customers so as to minimize a given cost function. In the classical sense, the VRP formulations do not allow specification of temporal constraints or high-level goals in a natural and formal manner. In this thesis, temporal logics such as Linear Temporal Logic (LTL), Metric Temporal Logic (MTL), as well as process algebras are studied for natural specification of constraints in vehicle routing problems, with a special emphasis to applications in UAV mission planning problems.

Linear temporal logics (e.g., LTL and MTL) allow mission specifications that combine temporal and logical constraints: some simple examples of such temporal reasoning include specifications like those appearing in "pick-up and delivery" problems ("service *Customer 1* and then *Customer 2*, in this order"), or priority constraints ("only service *Customer 3* if either *Customer 4* or *Customer 5* has already been serviced"). In the UAV mission planning context, these constraints can model in a very natural way rules of engagement and multi-platform coordination ("first track a target, then engage it, then assess the damage"), where different tasks must be carried out in a specified order, possibly by different vehicles, as well as conditional plan execution, such as "approach the target only if the SAM site protecting it has been neutralized."

An alternative to linear temporal logics is process algebra, which can handle specification of ordering and logical constraints. Using process algebra, a smaller set of constraints and objectives can be specified, however, the planning algorithms can be computationally more effective. Moreover, in the Artificial Intelligence literature, process algebras are generally associated with graphical models (such as in Hierarchical Task Networks [21]), which make them easier to visualize.

This thesis is structured in two main parts. In the first part, temporal logics are considered as a specification language to specify complex constraints of vehicle routing problems in a natural and unifying way. This part considers two widely used temporal logics: linear temporal logic LTL_{-X} and the Metric Temporal Logic (MTL). After formally introducing these languages, the vehicle routing problem with temporal logics is formalized and, subsequently, mixed integer linear programming formulations of the resulting problems are studied. In the second part of the thesis, process algebras are employed to define and solve a similar problem in a computationally more effective manner. In this case, a tree search based algorithm is proposed to solve the problem.

1.1 Vehicle Routing Problem

The Vehicle Routing Problem (VRP) can be stated informally as follows (a formal definition will be given in Chapter 2). Given a set of vehicles, a depot, a set of customers, and the cost of traveling from one customer to another, determine a tour for each vehicle in such a way that every tour starts from and ends at the depot, every node is visited exactly once, and the total cost of the tours is minimized. Since the seminal paper by Dantzig [19] (see also [18]), several extensions of the VRP have been proposed and applied to many engineering problems of practical interest. Even though the decision problem for the VRP is NP-complete [26, 43], several exact (but computationally expensive) solution approaches were proposed [42, 58]; for large-scale problems, heuristic approaches to the problem were considered [14, 15, 59]. Using these algorithms real-world problem instances were solved (see for example [5, 46, 51]).

The VRP has also been one of the foremost candidates for satisfying the need for an algorithmic basis towards solving optimal cooperative control problems involving multiple-UAVs. Several authors have employed Mixed-Integer Linear Programming (MILP) formulations of the VRP and demonstrated the effectiveness of state-of-the-art MILP solvers in handling such problems [9, 3, 37, 50, 54, 55, 60]. Two important extensions of the standard VRP that have found applications in multiple-UAV mission planning problems are the time window and relative timing constraints [3, 60], both of which take a step towards handling more complex scenarios. The focus of this thesis is on another related problem, namely, the formal specification of such temporal constraints in a unifying way using formal methods such as temporal logics and process algebras. With the help of formal methods, not only relative timing constraints, but also several other problem objectives can be addressed in a systematic way; to the author's best knowledge, such constraints and objectives have neither been dealt within the multiple-UAV Mission Planning application domain nor considered in a more general VRP setting.

1.2 Temporal Logics

Temporal logics were first studied by philosophers as a form of modal logic. The seminal paper by Pnueli [47] discussed its applications in computer science to reason about temporal behavior of concurrent computer programs (see also [44]). Since then, a large body of literature was developed especially on the algorithmic aspects of the model checking problem, which amounts to checking computer programs for satisfaction of desired temporal behavior. More precisely, given a set of concurrent software and a temporal logic formula encoding the desired temporal behavior, the model checking problem asks to either prove that all executions of the software satisfy the formula or find an execution that violates it [17]. One of the various algorithmic procedures that solve this problem is to construct a finite-state machine (in particular, a Büchi automaton [16]), called the language generator, that runs on infinite words and accepts a given word if and only if it satisfies the temporal logic formula (see for example [63] for a tutorial). The Büchi automaton is used in conjunction with the model of the software system to decide the model checking problem.

Different classes of temporal logics were utilized for planning purposes in artificial intelligence [25] as well as control theory [24, 57, 22, 38, 39, 56]. Early approaches to proving temporal properties of control systems using temporal logics appeared in as early as the eighties [24, 57]. However, algorithmic approaches to the problem of designing controllers that satisfy a given temporal logic specification by construction were not considered only until recently [22, 38, 39, 56]. This recent literature mainly employs Linear Temporal Logics (LTL) and its variants for specification of temporal properties and uses the language generator automaton with a discrete abstraction of the continuous state space to design trajectories, which in turn are realized by appropriate control systems.

This thesis provides algorithms that automatically synthesize equivalent provably correct MILP formulations for temporal logic specifications. In that regard, it should be noted that Integer Programming (IP) formulation of propositional logic operators were studied before [27, 30, 61, 62]. Using similar IP formulations, propositional logic was used in control theory to represent a broad class of systems in a unifying way [10]. This thesis also adopts a formulation that is similar to the one in [62] for formulating the propositional logic operators. None of these references, however, consider formulation of temporal logic operators, which makes the approach in this thesis novel in that context.

1.3 Process Algebras

While temporal logics are very expressive, they generally require computationally intensive algorithms, especially when utilized for optimal planning purposes. In fact, the model checking problem for, e.g., LTL, is PSPACE-complete [53]. Moreover, MILP-based optimal planning algorithms generally run in exponential time and are not guaranteed to return even a feasible solution within a polynomial time bound. On the other hand, one can consider languages that, at the cost of a more limited expressive power, allow efficient algorithms for, e.g., model checking. For example, in [41], a fragment of LTL was used for specification of temporal properties in reactive robotic motion planning problems, leading to polynomial-time algorithms. In a similar spirit, the second part of the thesis will concentrate on Process Algebra [45, 6, 23] as a specification language: while Process Algebra, in general, is not as expressive as LTL, it is amenable to efficient planning algorithms, while preserving the ability to describe a broad class of mission specifications of practical interest.

Process algebra was first proposed to reason about concurrent software and prove their correctness properties (see also [7] and the references therein), and successively used in several application contexts (see, for example, [11, 52, 1]). However, to the best of author's knowledge, this thesis constitutes the first application in UAV cooperative control problems.

The main advantage in using PA as a mission specification language is that a feasible plan (i.e., a plan that meets the mission specification) can be constructed efficiently, in polynomial time.

1.4 Contributions

The contributions of this thesis are as follows. First, in the first part of the thesis, which spans Chapters 2 to 5, two novel variants of the vehicle routing problem called the Vehicle Routing Problem with Linear Temporal Logic Specifications (VRPLTL) and the Vehicle Routing Problem with Metric Temporal Logic Specifications (VRPMTL), are introduced. Instances of these problems can represent the constraints that appear in several variants of the VRP in a natural and unifying way. These two problems are similar except that MTL is a strictly more expressive language than LTL. Hence, VRPMTL is a strict superset of VRPLTL. Both problems, however, are introduced and extensively studied noting that computational properties of the two problems may differ, even though they are not studied as in this thesis.

To solve VRPLTL, we propose a novel systematic procedure, which converts a given LTL_{-X} formula into a set of mixed-integer linear inequalities that are feasible if and only if there exists a finite execution that satisfies the LTL_{-X} formula. Moreover, those variables that render the inequalities feasible can naturally be converted to an execution that satisfies the LTL_{-X} formula. Using this procedure, two MILP formulations are developed. Both formulations solve the VRPTL to optimality. Two examples of multiple-UAV mission planning problems, relevant for military applications, are pointed out. In a similar spirit, to solve the VRPMTL, the MTL specification is formulated in a MILP setting and incorporated with one of the MILP formulations of the standard VRP.

In the second part of the thesis, which consists of Chapters 6 and 7, we propose process algebras for the specification of a large class of constraints in vehicle routing problems. This part of the thesis is presented more towards the application domain and multiple-UAV mission planning problems are emphasized. A computationally effective algorithm is proposed to solve the optimal planning problem with process algebra specifications. The computational effectiveness of the algorithm stems from the fact that it returns a feasible plan in polynomial time and is guaranteed to return the optimal solution in finite time.

Parts of this thesis were published in [31, 32, 33, 35]. Some related publications of the author that does not appear in this thesis include [36].

1.5 Organization

This thesis is organized as follows. In Chapter 2, the vehicle routing problem is introduced and two mixed integer linear programming formulations of the standard VRP are provided. In Chapter 3, both the linear temporal logic LTL_{-X} and the metric temporal logic are introduced. In Chapters 4 and 5, the standard vehicle routing problem is extended to address constraints specified in LTL_{-X} and MTL, respectively. Chapter 6 is devoted to the introduction of process algebra. In Chapter 7 the vehicle routing problem with process algebra specifications is introduced and a computationally effective algorithm to solve the problem is proposed. Finally, in Chapter 8, the thesis is concluded with discussions and remarks.

Chapter 2

The Vehicle Routing Problem

This chapter introduces the Vehicle Routing Problem (VRP) [59]. First, the VRP is introduced as a graph-theoretic problem. Then, two MILP formulations, namely the network flow formulation and the set covering formulation, are provided. Multiple-UAV mission planning applications of the two formulations are also discussed. The MILP formulations of the VRP are later extended to include various types of temporal logic specifications in Chapters 4 and 5, to which this chapter provides the preliminaries.

The rest of this chapter is organized as follows. In Section 2.1, preliminary definitions introducing graphs are presented and, in Section 2.2, the VRP is defined as a graph theoretic problem. Then, in Sections 2.3 and 2.4, the network flow and set covering formulations of the VRP are given.

2.1 Preliminaries on Sequences and Graphs

An infinite sequence σ on a set S is a map from the natural numbers $\mathbb{N}$ to S; σ is is also denoted as $(\sigma(1), \sigma(2), \dots)$. A finite sequence σ is a partial function defined from a finite subset of $\mathbb{N}$ to S such that if n largest number that σ is defined for, then σ is defined also for all $n' \leq n$, i.e., $\sigma : \{0, 1, \dots, n\} \to S$. Given an infinite sequence σ, a finite prefix ρ of σ is also a map from the first $L \in \mathbb{N}$ natural numbers to S, which agrees with σ on the domain that it is defined, i.e., $\rho(l) = \sigma(l)$ for

$l \in \{0, 1, \ldots, L\}$. A weighted directed graph G is defined by the triple $G = (V, E, w)$, where $V = \{0, 1, \ldots, N\}$ is the set of vertices, $E \subseteq V \times V$ is the set of edges between these vertices, and $w : E \to \mathbb{R}^+$ is a function associating a non-negative weight to each of the edges. Given an edge $e = (u, v) \in E$, let $\mathrm{Src}(e) = u$ and $\mathrm{Dest}(e) = v$. An edge e_j is said to be adjacent to another edge e_j if $\mathrm{Dest}(e_i) = \mathrm{Src}(e_j)$. A path is a sequence $(e_1, e_2, \ldots, e_n)$ of edges such that e_i is adjacent to e_{i-1} for all $i = 2, 3, \ldots, n$; the integer n is the length of the path and the sum the weights of all edges in the path is the weight of the path. A path $p = (e_1, e_2, \ldots, e_n)$ is said to visit a vertex v if $v \in \mathrm{Src}(e_1) \cup_{i=1}^{n} \mathrm{Dest}(e_i)$. Finally, p is called a circuit if $\mathrm{Dest}(e_n) = \mathrm{Src}(e_1)$.

2.2 VRP as a Graph-Theoretic Problem

The VRP can be defined as a graph theoretic problem as follows.

Problem 2.2.1 (Vehicle Routing Problem) *Given a weighted directed graph $G = (V, E, w)$, a depot $v_0 \in V$, and an integer $K \geq 1$ (the number of vehicles), find K circuits such that:*

- *All circuits visit the depot vertex v_0.*
- *All vertices in V are visited by at least one circuit.*
- *The sum of the weights of the circuits is minimized.*

In this case, each vehicle is associated with a weighted directed graph, in which every city is represented by a vertex. Each graph conveys the information regarding the cost of traveling from a given city to another for the vehicle that the graph is associated with. One can safely assume, without loss of any generality, that the set of vertices are fixed across all the weighted graphs. Note that if a given vehicle can not visit one of the cities in the problem, then the cost of traveling to the vertex corresponding to that particular city can be set to infinity, or equivalently the same vertex can be removed from the weighted directed graph.

There are many variations of the VRP in which, for example, there is a limit to the number of vertices that can be visited by a single vehicle, or there are constraints on the time at which certain vertices must be visited [59]; problem 2.2.1 can be extended to such cases.

2.3 Network Flow Formulation of the Vehicle Routing Problem

Problem 2.2.1 can be formulated in a form amenable to mathematical programming, e.g., as a a network flow problem, in several ways. In this section, we study the three-index formulation of the VRP [59]. Extensions of this formulation were recently applied to solve multiple-UAV mission planning problems [60] (see also [37, 54, 55] for similar MILP formulations).

Let $\mathcal{K}$ denote a set of vehicles and let c_{ijk} denote the cost of traversing the edge from vertex i to vertex j for vehicle k. Let us define the binary decision variables x_{ijk} for all $i, j \in V$ and $k \in \mathcal{K}$ such that the variable x_{ijk} is equal to 1 if vehicle k travels from vertex i to vertex j, and zero otherwise. Then, Problem 2.2.1 can be formulated as a MILP as follows:

$$\begin{aligned}
&\text{minimize} && \textstyle\sum_{i\in V}\sum_{j\in V}\sum_{k\in\mathcal{K}} c_{ijk}x_{ijk}, && && (2.1)\\
&\text{subject to} && \textstyle\sum_{j\in V, j\neq v_0} x_{v_0 jk} = 1, && \forall k \in \mathcal{K}, && (2.2)\\
& && \textstyle\sum_{k\in\mathcal{K}}\sum_{j\in V, i\neq j} x_{ijk} = 1, && \forall i \in V, && (2.3)\\
& && \textstyle\sum_{i\in V, i\neq h} x_{ihk} - \sum_{j\in V, j\neq h} x_{hjk} = 0, && \forall h \in V, \quad \forall k \in \mathcal{K}, && (2.4)
\end{aligned}$$

where constraints (2.2) and (2.3) ensure that all the circuits visit the depot vertex v_0 and all the vertices in V are visited by exactly one circuit, respectively. Constraints (2.4) ensure that every non-depot vertex that is visited is subsequently left. Let us note that the formulation given by (2.1-2.4) may produce more than one circuit for one vehicle. Such a solution is said to have sub-tours. A solution with a single circuit for each vehicle can be obtained by adding "sub-tour elimination constraints" to the formulation and solving the resulting MILP, until no sub-tours remain [59]. As an example, the following set of constraints is one of the many that eliminate the subtours (see [59] for other types of subtour elimination constraints):

$$\textstyle\sum_{i\in S}\sum_{j\in S} x_{ijk} \leq |S|-1, \quad \forall S \subset V \setminus \{v_0\}, \quad |S| \geq 2, \quad \forall k \in \mathcal{K}. \tag{2.5}$$

Note that Constraints (2.5) are exponentially many with respect to the number of vertices. For practical implementations, these constraints are added to the formulation as necessary. That is, if the solution has a subtour involving a set S of the vertices and a vehicle k, then the subtour elimination constraint corresponding to that particular S and k pair is added to the formulation. This procedure is repeated until no subtours remain.

A widely-used extension of the standard VRP is the vehicle routing problem with time windows [59]. Let the parameters t_{ijk} denote the time it takes for vehicle k to travel from vertex i to vertex j. Also, let the continuous decision variables t_i and t_{v_0k} be the time that a vehicle reaches vertex i and the time that vehicle k completes its circuit, respectively.[1] Let the continuous decision variables s_{ik} denote the idling time vehicle k spends at vertex j. Then, the following constraints define a feasible flow of time and make sub-tours infeasible [59]:

$$t_{v_0} = 0, \tag{2.6}$$

$$t_i + t_{ijk} + s_{ik} - M\left(1 - x_{ijk}\right) \leq t_j, \quad \forall i \in V, \quad \forall j \in V - \{v_0\}, \quad \forall k \in \mathcal{K}, \tag{2.7}$$

$$t_i + t_{iv_0k} + s_{ik} - M(1 - x_{iv_0k}) \leq t_{v_0k}, \quad \forall i \in V, \quad \forall k \in \mathcal{K}, \tag{2.8}$$

where M is a big enough number. Placing upper and lower bound constraints (time-windows) on the decision variables t_i enforces the vertex i to be visited in its own time-window. Note also that, using the new set of variables, one can minimize $\sum_{k=1}^{K} c_k t_{v_0k}$, where c_k denotes the cost of employing vehicle k for unit time. For mission planning applications, c_k generally indicates a risk factor for employing vehicle k in the mission.

[1]Here, we make an abuse of notation and distinguish t_i, t_{v_0k} and t_{ijk} with their number of indices.

Now let us introduce another extension to the problem which enables us to consider two different types of depots: starting depots and the ending depots. In this case, instead of finding a circuit, one has to find a path, which starts from a vehicle specific starting depot and ends in one of the ending depots. To formulate this extension, let $\mathcal{L}$ and C denote the sets of starting and ending depots, respectively. Also, let $\mathcal{N}$ denote set of all the vertices except the depots, i.e., $\mathcal{N} = V - (\mathcal{L} \cup C)$. Let us further define the sets $\mathcal{I}$ and $\mathcal{J}$ of departing and approaching nodes as $\mathcal{I} = \mathcal{L} \cup \mathcal{N}$ and $\mathcal{J} = \mathcal{N} \cup C$, respectively. Then, the multi-depot extension of the previous MILP formulation is given as follows.

$$\begin{aligned}
&\text{minimize} && \textstyle\sum_{j \in C} \sum_{k=1}^{K} c_k t_{jk}, && && (2.9)\\
&\text{subject to} && \textstyle\sum_{i \in \mathcal{L}} \sum_{j \in \mathcal{N}} x_{ijk} = 1, && \forall k \in \mathcal{K}, && (2.10)\\
& && \textstyle\sum_{k=1}^{K} \sum_{j \in \mathcal{J}, j \neq i} x_{ijk} = 1, && \forall i \in \mathcal{N}, && (2.11)\\
& && \textstyle\sum_{i \in \mathcal{I}, i \neq h} x_{ihk} - \sum_{j \in \mathcal{J}, j \neq h} x_{hjk} = 0, && \forall h \in \mathcal{N}, \quad \forall k \in \mathcal{K}, && (2.12)\\
& && t_i = 0, && \forall i \in \mathcal{L}, && (2.13)\\
& && t_i + t_{ijk} + s_{ik} - M(1 - x_{ijk}) \leq t_j, && \forall i \in \mathcal{I}, \quad \forall j \in \mathcal{N}, j \neq i \quad \forall k \in \mathcal{K} && (2.14)\\
& && t_i + t_{ijk} + s_{ik} - M(1 - x_{ijk}) \leq t_{jk}, && \forall i \in \mathcal{N}, \quad \forall j \in C, \quad \forall k \in \mathcal{K}. && (2.15)
\end{aligned}$$

Another extension of the problem that is of interest to this thesis is the following. Let r_k denote the maximum amount of time that vehicle k can be employed; the number k, in this case, indicates a range constraints. Then, the following constraints ensure that the vehicles do not travel along paths longer than their maximum range:

$$t_{jk} \leq r_k, \quad \forall j \in C, \quad \forall k \in \mathcal{K}. \tag{2.16}$$

2.4 Set Covering Formulation of Vehicle Routing Problem

The set covering formulation for the VRP was first presented in [8]. This formulation was later strengthened and used for developing exact algorithms for both the standard VRP and the VRP with time-windows in [2] and [20], respectively. In fact, MILP formulations based on set covering were shown to be quite general (see, for instance, [28] for a crew scheduling example). Recently, multiple-UAV mission planning applications of the formulation have also been considered both for centralized and distributed implementations [9, 34].

The set-covering formulation of the VRP has two main advantages. Firstly, it allows formulating more complicated cost functions. From a multiple-UAV mission planning point of view, the set-covering formulation can handle cost functions that may nonlinearly depend on the target priorities, UAV capabilities etc. [9]. Secondly, LP relaxation of the set-covering formulation is known to be tight (see [20] for experimental evidence and [13] for theoretical results under some special conditions).

The work in [9] was later strengthened in [3] to include loitering constraints. For notational simplicity, this paper studies the formulation presented in [9]. However, the results can be extended to the case presented in [3].

The set-covering formulation of the VRP has a presolve phase, in which all the possible circuits for all the vehicles are computed. Let $\mathcal{K}$ be the set of vehicles and $J(k)$ be the set of all feasible circuits for vehicle $k \in \mathcal{K}$. Let c_{jk} denote the cost of the circuit $j \in J(k)$ when traversed by vehicle k. Let us define the parameters D_{ijk} for all $i \in V$, $j \in J(k)$, and $k \in \mathcal{K}$ such that D_{ijk} is equal to one if circuit j of vehicle k traverses vertex i, and zero otherwise. Let us also define the binary decision variables, x_{jk} for all $j \in J(k)$ and $k \in \mathcal{K}$, such that x_{jk} equal to one if and only if vehicle k executes circuit j. Then, Problem 2.2.1 can be formulated as follows.

$$\begin{aligned} &\text{minimize} && \textstyle\sum_{k\in\mathcal{K}}\sum_{j\in J(k)} c_{jk}x_{jk}, && && (2.17)\\ &\text{subject to} && \textstyle\sum_{k\in\mathcal{K}}\sum_{j\in J(k)} D_{ijk}x_{jk} = 1, && \forall i \in V-\{v_0\}, && (2.18)\\ & && \textstyle\sum_{j\in J(k)} x_{jk} = 1, && \forall k\in\mathcal{K}, && (2.19)\end{aligned}$$

where constraints (2.18) and (2.19) ensure that each vertex is visited exactly once and each vehicle is issued a circuit, respectively. Notice that the cost of each circuit can be any function of the problem parameters in the set-covering formulation.

Chapter 3

Temporal Logics

This chapter introduces two different temporal logics: the Linear Temporal Logic LTL_{-X} and the Metric Temporal Logic (MTL). In Chapters 4 and 5, LTL_{-X} and MTL will be used as natural and formal specification tools in vehicle routing problems.

This chapter is organized as follows. First, in Section 3.1, the formal definition of a transition system is given as a preliminary for the rest of the chapter. Then, the temporal logics LTL_{-X} and MTL are presented, in Sections 3.2 and 3.3, respectively.

3.1 Transition Systems

In several applications of computer science, computer software is modeled as a transition system, which has a finite number of states, a set of transition rules, and a set of observables.

The states in a transition system generally represent the internal information of the software, e.g., the values of the variables. Indeed, it is common to define the 'variables' of a software system first, and then derive the set of states from those variables (see for example the formal definition of the transition system in [44]). The transition rules, on the other hand, relate two states such that a transition from the former one to the latter one is possible. Equivalently, they define the allowable changes in the variables given the current values of the variables. Finally, observables, also referred to as the atomic propositions, group states with a similar property into classes. In this case, each state is associated with a set of atomic propositions that it satisfies. Similarly, those states that satisfy a given atomic proposition fall into the same class. In several applications of logics and formal methods, the user is not interested in reasoning about the state itself, but the specification is given in terms of the atomic propositions. One of the main motivations is that the user only observes the atomic propositions, not the states of the software, since states mostly refer to the internal information. Moreover, the state space is generally very large, whereas the set of atomic propositions is generally a smaller set.

The transition system model has been the main abstract model of computer programs for model checking algorithms [44]. However, the definition of the transition system is quite general. Indeed, it will be used as an abstract model for the VRP in Chapter 4.

Transition system is formally defined as follows.

Definition 3.1.1 (Transition System) *A Transition System is a tuple*

$$\mathcal{TS} = (Q, Q_0, \rightarrow, \Pi, L),$$

where Q is a set of states, $Q_0 \subseteq Q$ is a set of initial states, $\rightarrow \subseteq Q \times Q$ is a transition relation, Π is a set of atomic propositions, and $L : Q \rightarrow 2^{\Pi}$ is a labeling function.

As mentioned earlier, the complex reasoning with temporal logics is carried out via the atomic propositions, instead of the states of the system. More precisely, atomic propositions are the building blocks of complex reasoning for temporal logics.

Informally speaking, an *atomic proposition* is a predicate, i.e., a declarative sentence, which is either `True` or `False` at a given time. The states of the transition system are labeled by the atomic propositions that they satisfy. In other words, if a state q is labeled with an atomic proposition p, then p evaluates to true whenever the system is in state q.

A transition system is a set of rules identifying a particular class of sequences Q, describing the evolution of the states of the system. These sequences are called runs, and are formally defined as follows.

Definition 3.1.2 (Infinite Run) *An* Infinite Run *σ on a given transition system $\mathcal{TS}$ is an infinite sequence of states, $\sigma = (q_1, q_2, \ldots)$, such that (i) $q_i \in Q$, for all $i \in \mathbb{N}$, (ii) $q_1 \in Q_0$, and (iii) $(q_i, q_{i+1}) \in \rightarrow$, for all $i \in \mathbb{N}$.*

Given a transition system $\mathcal{TS}$ on a weighted directed graph G, the set $\Sigma_{\mathcal{TS}}$ of all runs is defined as the set of all infinite runs and all their finite prefixes, called the *finite runs*.

3.2 The LTL$_{-X}$ Language

The linear temporal logic LTL is one of the most widely-used temporal logics [44]. Its variant LTL$_{-X}$ eliminates the *next* operator from the language and is generally used in problems that take place in a continuous time setting rather than discrete time. Recently, LTL$_{-X}$ was used for formal specification of continuous-time control systems [38, 39]. This section introduces the language LTL$_{-X}$ with its syntax and semantics.

3.2.1 Syntax of LTL$_{-X}$

The syntax of the LTL$_{-X}$ language can be defined recursively as follows. Every atomic proposition $p \in \Pi$ is an LTL$_{-X}$ formula, and if ϕ and ψ are formulas then so are $\neg\phi$, $\phi \wedge \psi$, and $\phi\mathcal{U}\psi$. That is, the LTL$_{-X}$ grammar in Backus-Naur Form (BNF) is

$$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid \phi\mathcal{U}\phi \tag{3.1}$$

where p is an atomic proposition, ϕ is a formula, $\neg$ is the *negation* operator, $\vee$ is the *disjunction* operator, and $\mathcal{U}$ is the *until* operator. The *negation* operator implies that the property it binds is false. The disjunction asserts that at least one of the properties it binds is true. The temporal operator *until*, when used as in $p\mathcal{U}q$, implies that q eventually becomes true and until then p keeps being true.

Given the operators *negation* and *disjunction*, the operators *conjunction* ($\wedge$), *implication* ($\Rightarrow$), and *equivalence* ($\Leftrightarrow$) can be defined as $\phi_1 \wedge \phi_2 = \neg(\neg\phi_1 \vee \neg\phi_2)$, $\phi_1 \Rightarrow \phi_2 = \neg\phi_1 \vee \phi_2$, and $\phi_1 \Leftrightarrow \phi_2 = (\phi_1 \Rightarrow \phi_2) \wedge (\phi_1 \Rightarrow \phi_2)$ respectively. Finally, using all these operators with *until* we can define *eventually* ($\Diamond$) and *always* ($\Box$) as $\Diamond\phi = \top\mathcal{U}\phi$ and $\Box\phi = \neg\Diamond\neg\phi$. It is convenient to also define the operator *Unless* as $p\mathcal{W}q = (\Box p) \vee (p\mathcal{U}q)$. *Unless* is slightly weaker that the until operator in the sense that q does not have to be true eventually, in which case p holds to be true forever. This operator is also referred to as the *weak until* operator.

3.2.2 Semantics of LTL$_{-X}$

Let $\Phi_{LTL_{-X}}$ be the set of all LTL$_{-X}$ formulae and $\Sigma_{\mathcal{TS}}$ be the set of all runs on a transition system $\mathcal{TS}$. The semantics of LTL$_{-X}$ is defined as a satisfaction relation, which I will denote as $\models \subseteq (\Sigma_{\mathcal{TS}} \times \mathbb{N}) \times \Phi_{LTL_{-X}}$. Given an atomic proposition p, a run $\sigma \in \Sigma$ of $\mathcal{TS} = (Q, Q_0, \rightarrow, \Pi, L)$, and a time instance $j \in \mathbb{N}$, σ is said to satisfy p at j, i.e., $((\sigma, j), p) \in \models$, (also denoted as $(\sigma, j) \models p$) if and only if $p \in L(q_j)$. Let p be an atomic proposition, ϕ and ψ be any two formulas in LTL$_{-X}$. Then, the semantics of LTL$_{-X}$ is defined as the smallest $\models$ which satisfies the following.

$$(\sigma, j) \models p \quad \text{iff} \quad p \in L(s_j), \tag{3.2}$$

$$(\sigma, j) \models \neg\phi \quad \text{iff} \quad (\sigma, j) \not\models \phi, \tag{3.3}$$

$$(\sigma, j) \models \phi \vee \psi \quad \text{iff} \quad (\sigma, j) \models \phi \text{ or } (\sigma, j) \models \psi, \tag{3.4}$$

$$(\sigma, j) \models \phi\mathcal{U}\psi \quad \text{iff} \quad \exists k \geq j \text{ such that } (\sigma, k) \models \psi \tag{3.5}$$

$$\text{and for } \forall i, j \leq i \leq k \text{ there holds } (\sigma, k) \models \phi.$$

Even thought the semantic rules above is adequate to define the language LTL$_{-X}$, semantic rules for the derived operators can also be formalized as follows.

$$(\sigma, j) \vDash \phi \wedge \psi \quad \text{iff} \quad s_j \vDash \phi \text{ and } s_j \vDash \psi, \tag{3.6}$$

$$(\sigma, j) \vDash \Box\phi \quad \text{iff} \quad (\sigma, k) \vDash \phi \text{ for } \forall k \geq j, \tag{3.7}$$

$$(\sigma, j) \vDash \Diamond\phi \quad \text{iff} \quad \exists k \geq j \text{ such that } (\sigma, k) \vDash \phi. \tag{3.8}$$

If a run σ satisfies a formula ϕ at the initial time instance 1, i.e., $(\sigma, 1) \vDash \phi$, then σ is simply said to satisfy ϕ, denoted by $\sigma \vDash \phi$ with a slight abuse of notation.

Each LTL$_{-X}$ formula can be associated with a set of its subformulae and, consequently, a height. In the rest of this section, a more precise definition of an LTL$_{-X}$ formula, followed by the definitions of a subformula and height of a formula are given. These definitions will be useful in Chapter 4.

Definition 3.2.1 (LTL$_{-X}$ Formula) *An LTL$_{-X}$ formula on a set Π of atomic propositions is a sentence which consists of atomic propositions from Π and LTL$_{-X}$ operators and obeys the syntax of LTL$_{-X}$.*

Definition 3.2.2 (LTL$_{-X}$ Subformula) *Given an LTL$_{-X}$ formula ϕ, a subformula of ϕ is any formula which is strictly included in ϕ and satisfies the syntax of LTL$_{-X}$.* [1]

Notice that a formula is not a subformula of itself. Even though an atomic proposition is a formula by itself, an atomic proposition has no subformula.

Definition 3.2.3 (Height of an LTL$_{-X}$ Formula) *The height of an atomic proposition is zero. The height of any other LTL$_{-X}$ formula ϕ is the largest number n such that*

- *ψ_1 is a subformula of ϕ*
- *ψ_k is a subformula of ψ_{k-1} for $\forall k \in \{2, \ldots, n\}$*

[1]The formal definition of a subformula requires definition of parse trees, which is not employed in the first part of the thesis. Interested reader is referred to [29] for a more formal definition of subformula through parse trees of LTL formulae.

Notice that all formulae with only one operator have height one and atomic propositions have height zero. Finally, given a formula ϕ, a subformula ψ of ϕ is called an immediate subformula of ϕ if ψ has height greater than or equal to every other subformula of ϕ.

Throughout the thesis, only a finite set Π of propositions and formulas with only a finite number of operators are considered. Consequently, these formulas always have finite height.

3.3 Metric Temporal Logic

This section introduces the Metric Temporal Logic (MTL). The temporal logic LTL_{-X} assumed discrete time instances, at each of which a transition occurred. These time instances were all natural numbers. MTL, on the other hand, extends the runs of a transition system with the time that the transition occurs. That is, in MTL the underlying time is defined as continuous in the sense that transitions can occur in time instances represented as real numbers. Essentially, using a dense time coordinate, MTL is able to express qualitative properties time such as deadlines or time-windows, which can not be expressed in LTL. The temporal logics that assume continuous time are called real-time logics [40].

The rest of this section is organized as follows. First, in Section 3.3.1, timed executions of transition systems are discussed. Then, in Sections 3.3.2 and 3.3.3, syntax and semantics of MTL are presented.

3.3.1 Preliminary Definitions

This section, introduces definitions that allow formal definition of MTL semantics. The notation introduced in this section is similar to the one in [4].

An interval is any convex subset of the real line $\mathbb{R}$. The left and the right limits of an interval I are defined as $l(I) = \inf_{x \in I} x$ and $r(I) = \sup_{x \in I} x$, respectively. An interval I is said to be left closed if $l(I) \in I$, and right closed if $r(I) \in I$. Two intervals

I_1 and I_2 are said to be adjacent if the right limit of I_1 is equal to the left limit of I_2. Given $t \in \mathbb{R}_{\geq 0}$, the interval, which has its left and right limits equal to $t + l(I)$ and $r(I) + t$ and which is left closed if I is left closed and right closed if I is right closed, is denoted as $t + I$. The notation $I - t$ will be used in a similar way.

Definition 3.3.1 (State and Interval Sequences) *A state sequence $\sigma = (s_0, s_1, s_2, \dots)$ is a possibly infinite sequence of states with $s_i \in Q$ for $\forall i \geq 0$, $s_0 \in Q_0$ and $(s_i, s_{i+1}) \in \rightarrow$ for $\forall i \geq 0$. Similarly an interval sequence $\kappa = (I_0, I_1, I_2)$ is a possibly infinite sequence of intervals which satisfy the following*

- *I_0 is left closed and $l(I) = 0$;*
- *for all $i \geq 0$, the intervals I_i and I_{i+1} are adjacent;*
- *every time $t \in \mathbb{R}_{\geq 0}$ belongs to an interval in κ.*

Notice that the definition of state sequences coincide with the that of the runs presented in the previous section. The following definition extends the runs presented in the previous section, so that each state sequence is accompanied with an interval sequence of the same length.

Definition 3.3.2 (Timed State Sequence) *A timed state sequence $\omega = (\sigma, \kappa)$ is a pair consisting of a state sequence σ and an interval sequence κ.*

Informally, a timed state sequence indicates the time interval during which a state of the transition system has been active, i.e., the state s_i has been active in the interval I_i for $\forall i$. Let me use the function $s(t) : \mathbb{R}_{\geq 0} \rightarrow Q$ to denote the state that is active at a given time t, i.e., $s(t) = s_i$ if and only if $t \in I_i$.

Given a timed state sequence $\omega = (\sigma, \kappa)$ with $\sigma = (s_1, s_2, \dots)$ and $\kappa = (I_1, I_2, \dots)$, the atomic propositions take unique values at any given time $t \in \mathbb{R}_{\geq 0}$. The value of the atomic proposition $p \in \Pi$ at time $t \in \mathbb{R}_{\geq 0}$ will be denoted by $s(t)[p_i]$.

Finally, one can introduce the formal definition for a suffix of a timed state sequence, which will be used to define the semantics of MTL.

Definition 3.3.3 (Suffix) *Let $\omega = (\sigma, \kappa)$ be a timed state sequence. For some $t \in I_i$, the suffix ω^t at time t is the timed state sequence $\omega' = (\sigma', \kappa')$ where $\sigma' = (s_i, s_{i+1}, s_{i+2}, \dots)$ and $\kappa = (I_i - t, I_{i+1} - t, I_{i+2} - t, \dots)$.*

3.3.2 Syntax of MTL

The formulae of MTL are constructed using the atomic propositions along with the operators of MTL, which are to be defined shortly. Every formula of MTL is a proposition by itself, i.e., it is either `True` or `False` at some given time represented with a real number.

MTL includes the usual operators of propositional logic together with a set of temporal operators, which are bound with an interval. The syntax of MTL is defined as follows:

$$\phi ::= p|\neg\phi|\phi_1 \wedge \phi_2|\phi_1 \mathcal{U}_I \phi_2, \tag{3.9}$$

where I is an interval, $p \in \Pi$ is an atomic proposition, ϕ, ϕ_1, and ϕ_2 are MTL formulae. $\neg$ is the negation operator, $\wedge$ is the conjunction operator and $\mathcal{U}_I$ is the until operator. Informally speaking, the formula $\phi_1 \mathcal{U}_I \phi_2$ is true at time t if there is some time $t' \in t + I$ for which ϕ_2 is `True` and ϕ_1 holds to be true within the interval (t, t').

Even though the operators above are adequate to fully represent the MTL language, we define the following operators as well to improve readability of MTL formulae. Given the negation and conjunction operators, the operators *disjunction* ($\vee$), *implication* ($\Rightarrow$), and *equivalency* ($\Leftrightarrow$) can be defined as $\phi_1 \vee \phi_2 = \neg(\neg\phi_1 \wedge \neg\phi_2)$, $\phi_1 \Rightarrow \phi_2 = \neg\phi_1 \vee \phi_2$, and $\phi_1 \Leftrightarrow \phi_2 = (\phi_1 \Rightarrow \phi_2) \wedge (\phi_2 \Rightarrow \phi_1)$ respectively. Together with these operators given the temporal operator *until*, the temporal operators *eventually* ($\Diamond_I$), *always* ($\Box_I$) and *unless* $\mathcal{W}_I$ can be defined as $\Diamond_I \phi = \top \mathcal{U}_I \phi$, $\Box_I \phi = \neg\Diamond_I \neg\phi$, and $\phi_1 \mathcal{W}_I \phi_2 = \neg(\neg\phi_1 \mathcal{U}_I \neg\phi_2)$. Informally speaking, the formula $\Diamond_I \phi$ is `True` at some time t if ϕ holds to be `True` at some point time in the interval $t + \mathcal{I}$. Similarly the formula $\Box_I \phi$ is `True` at some time t if ϕ holds to be `True` at all points in the interval $t + \mathcal{I}$. Finally, the formula $\phi_1 \mathcal{W}_I \phi_2$ holds at time t if and only if either ϕ_1 is `True` throughout the interval or there exists a $t' > t$ for which ϕ_2 is `True` at t' and ϕ_2 is `True` during the interval $[t', t] \cap I$.

3.3.3 Semantics of MTL

The formulae of MTL are interpreted over the timed state sequences. Let Φ_{MTL} denote the set of MTL formulae, and $\Delta_{\mathcal{TS}}$ denote the set of timed state sequences on a given transition system $\mathcal{TS}$. The semantics of MTL is defined by the satisfaction relation $\models \subseteq (\Delta \times \mathbb{R}_{\geq 0}) \times \Phi_{MTL}$, which is defined recursively as follows.

$$\begin{aligned}
(\omega, 0) \models p \quad &\text{iff} \quad p \in L(s_0); && (3.10)\\
(\omega, 0) \models \neg\phi \quad &\text{iff} \quad (\omega, 0) \not\models \phi; && (3.11)\\
(\omega, 0) \models \phi_1 \wedge \phi_2 \quad &\text{iff} \quad (\omega, 0) \models \phi_1 \text{ and } (\omega, 0) \models \phi_2; && (3.12)\\
(\omega, 0) \models \phi_1 \mathcal{U}_I \phi_2 \quad &\text{iff} \quad \exists t \in \mathcal{I}, (\omega, t) \models \phi_2 \\
&\qquad \text{and } \forall t' \in (0, t), (\omega, t') \models \phi_1; && (3.13)\\
(\omega, t) \models \Phi \quad &\text{iff} \quad (\omega^t, 0) \models \Phi. && (3.14)
\end{aligned}$$

A timed state sequence ω is said to satisfy the formula ϕ if and only if $(\omega, 0) \models \phi$. Even thought the semantic rules above define the language MTL, the semantics of *disjunction, always* and *eventually* can be given as follows.

$$\begin{aligned}
(\omega, 0) \models \phi_1 \vee \phi_2 \quad &\text{iff} \quad (\omega, 0) \models \phi_1 \text{ or } (\omega, 0) \models \phi_2; && (3.15)\\
(\omega, 0) \models \Diamond_I \phi \quad &\text{iff} \quad \exists t \in I, (\omega, t) \models \phi; && (3.16)\\
(\omega, 0) \models \Box_I \phi \quad &\text{iff} \quad \forall t \in I, (\omega, t) \models \phi. && (3.17)
\end{aligned}$$

Chapter 4

Vehicle Routing with Linear Temporal Logic Specifications

In this chapter, a new variant of the VRP, called the Vehicle Routing Problem with Linear Temporal Logic specifications (VRPLTL), is introduced. VRPLTL extends the standard vehicle routing problem (see Problem 2.2.1 defined in Chapter 2) by asking to find routes satisfying linear temporal logic specifications instead of visiting all the cities and employing all the vehicles. VRPLTL is defined using a vehicle-routing transition system, which is a transition system model for instances of vehicle routing problems. The semantics of LTL is well defined on a vehicle-routing transition system, which, in turn, renders the VRPLTL well defined.

Following the problem definition, two MILP-based algorithms, both of which solve VRPLTL instances to optimality, are presented. One of these algorithms is based on the network flow formulation of the VRP, whereas the other one extends the set-covering formulation (cf. Chapter 2). Both MILP formulations of the VRPLTL are structured around the same idea: first converting the LTL_{-X} specification into a set of linear constraints in mixed integer and continuous variables, and then incorporating those constraints into the standard MILP formulations of the VRP.

This chapter is organized as follows. First, the vehicle-routing transition systems are defined in Section 4.1, followed by the formal problem definition in Section 4.2. Then, in Section 4.3, a systematic procedure, which converts LTL_{-X}

formulae into a set of linear constraints is introduced. Finally, in Section 4.4, this formulation is combined with the standard MILP formulations of the vehicle routing problem to solve VRPLTL to optimality. Section 4.4 also highlights the multiple-UAV mission planning applications of VRPLTL.

4.1 Vehicle Routing Transition Systems

In Chapter 3, it was mentioned that a transition system is a model used for reasoning about computer software. In this section, a transition system is adopted as a model for VRP instances. For this purpose, let us introduce the definition of a Vehicle-Routing Transition System, after the definition of a Single-Vehicle-Routing Transition System.

Definition 4.1.1 (Single-Vehicle-Routing Transition System) *A single-vehicle-routing transition system $(Q, Q_0, \rightarrow, \Pi, L)$ on a weighted directed graph $G = (V, E, w)$, with depot $v_0 \in V$, is such that*

- *$Q = V \times \mathbb{R}_{\geq 0}$, i.e., the state $q = (\theta, v)$ is composed by a variable $\theta \in \mathbb{R}_{\geq 0}$ which represents a time instance, and a variable $v \in V$ which indicates the position of the vehicle on the graph;*
- *$Q_0 = \{(v_0, 0)\}$, i.e., the vehicle is in its depot initially;*
- *$(v_i, \theta_i) \rightarrow (v_j, \theta_j)$ if and only if $(v_i, v_j) \in E$ and $\theta_j - \theta_i \geq w(v_i, v_j)$;*
- *Π is such that there exists $p_i \in \Pi$ for each $v_i \in V$;*
- *$L : (v_i, \theta) \mapsto \{p_i\}$ for all $\theta \in \mathbb{R}$ and all $v_i \in V$.*

In essence, the Vehicle-Routing Transition System is defined as a parallel composition of a fixed number of Single-Vehicle-Routing Transition Systems as follows.

Definition 4.1.2 (Vehicle-Routing Transition System) *A (multiple-) Vehicle-Routing transition system $(Q, Q_0, \rightarrow, \Pi, L)$ for K vehicles on K weighted directed graphs $G^k = (V^k, E^k, w^k)$, with depots $v_0^k \in V^k$ for $k = 1, \ldots, K$ is such that*

- $Q = \prod_{k=1}^{K}(V^k \times \mathbb{R}_{\geq 0})$, *i.e., the state* $q = (q^1, q^2, \ldots, q^K)$, *where* $q^k = (v^k, \theta^k)$, *is composed by the individual vehicle states on their respective single-vehicle transition system;*

- $Q_0 = \Big((0, v_0^1), \ldots, (0, v_0^K) \Big)$, *i.e., each vehicle is initially at its own depot;*

- $\Big((v_i^1, \theta_i^1), \ldots, (v_i^K, \theta_i^K) \Big) \to \Big((v_i^1, \theta_i^1), \ldots, (v_i^{k-1}, \theta_i^{k-1}), (v_j^k, \theta_j^k), (v_i^{k+1}, \theta_i^{k+1}), \ldots (v_i^K, \theta_i^K) \Big)$ *if and only if* $(v_i^k, v_j^k) \in E$, $\theta_j^k \geq \theta_i^l$ *for all* $l \in \{1, \ldots, K\}$, *and* $\theta_j^k - \theta_i^k \geq w^k(v_i^k, v_j^k)$.

- $\Pi = \cup_{k=1,\ldots,K}\Pi^k$ *such that there exists* $p_i^k \in \Pi^k$ *for each* $v_i \in V$.

- $L : \Big((v^1, \theta^1), \ldots, (v^K, \theta^K) \Big) \mapsto \{p^1, \ldots, p^K\}$ *for all* $\theta^k \in \mathbb{R}_{\geq 0}$, $k \in \{1, \ldots, K\}$ *and all* $v \in V$.

The semantics of LTL$_{-X}$, introduced in Section 3.2.2, are well defined on a given vehicle-routing transition system $\mathcal{TS}$. That is, given (i) an LTL$_{-X}$ formula ϕ defined on the atomic propositions of $\mathcal{TS}$, (ii) a run σ of $\mathcal{TS}$, and (iii) a time instance $j \in \mathbb{N}$, the relation $\models$ is well-defined, i.e., $\Big((\sigma, j), \phi\Big) \in \models$.

Below is an example of how one can specify several interesting properties in a vehicle routing problem using the linear temporal logic LTL$_{-X}$.

Example Consider, again, a VRP instance with two customers and a single vehicle. Let p_1, p_2 be atomic propositions indicating Customers 1 and 2 being serviced, respectively.

A simple *reachability* specification is, for instance, the proposition that states *Customer 1 will eventually be serviced,* which can be expressed in LTL$_{-X}$ as $\Diamond p_1$. A *safety* specification example is *Customer 1 will not be serviced,* which can be expressed with the LTL$_{-X}$ formula $\Box\neg p_2$. Such safety specifications may arise when a vehicle is not capable of servicing a particular customer. An example combining reachability and safety conditions is as follows: *if Customer 2 is not going to be serviced then do not service Customer 1 either,* which can be expressed in LTL$_{-X}$ as $(\neg\Diamond p_2) \implies (\Box\neg p_3)$.

A final example is the *ordering* specifications. In this case, one would like Customer 1 to be serviced, for instance, before Customer 2. The corresponding LTL$_{-X}$

specification is $(\neg p_2)\mathcal{U}p_1$. Carefully analyzing this specification, one can recognize that it enforces Customer 1 to be serviced eventually but not Customer 2. The following specification ensures servicing both of the customers: $[(\neg p_2)\mathcal{U}p_1] \wedge (\Diamond p_2)$. Notice that this last specification combines reachability and ordering properties. One may also desire that servicing both of the customers is optional, but if Customer 2 will be serviced than Customer 1 must be serviced before Customer 2, in which case $[(\neg p_2)\mathcal{U}p_1] \vee (\Box\neg p_2)$ (or in a more compact notation $(\neg p_2)\mathcal{W}p_1$) is the corresponding LTL_{-X} specification. Notice that this last specification combines safety and ordering properties.

More complex specifications can be built with negations, conjunctions, and disjunctions of several of the examples above. ▌

Below are examples to make the definition of an LTL_{-X} formula and other related definitions more concrete in the vehicle routing setting. Later in this chapter these definitions will be used to formalize the algorithms.

Example Consider a VRP with two customers and a single vehicle. Notice that the objective which states that *Eventually either Customer 1 or Customer 2 will be serviced by Vehicle 1* is indeed a temporal formula. Let p_1 (resp. p_2) denote the atomic proposition which states that Customer 1 (resp. 2) is serviced by Vehicle 1. Then, one can express the temporal formula as

$$\phi = \Diamond(p_1 \vee p_2).$$

Subformulas of ϕ are $p_1 \vee p_2$, p_1, and p_2. In this case, p_1 and p_2 have heights zero, $p_1 \vee p_2$ has height one, and ϕ has height two. $p_1 \vee p_2$ is the unique immediate subformula of ϕ. The formula $p_1 \vee p_2$ has two immediate subformulae: p_1 and p_2. ▌

One aspect of the vehicle routing problem that is different than model checking computer software is that one is concerned with finding an optimal runs rather than finding a single run, or showing that none exists. In order to search for an execution of a vehicle-routing transition system with optimum cost, however, each and every run must be associated with a well defined cost. The following definitions serve this purpose.

Definition 4.1.3 (Time schedule) *The* running time Θ *of an infinite run* $\sigma = (q_1, q_2, \ldots)$ *of the vehicle-routing transition system* $\mathcal{TS}$ *is a sequence* $\Theta = (\Theta_1, \Theta_2, \ldots)$ *on* $\mathbb{R}_{\geq 0}$ *such that*

$$\Theta_i = \max_{k = \in \{1, \ldots, K\}} \theta_i^k.$$

Each run is associated with a well defined cost as follows.

Definition 4.1.4 (Cost) *The* running cost C *of a run* $\sigma = (q_1, q_2, \ldots)$ *of the vehicle-routing transition system* $\mathcal{TS}$ *is a sequence* $C = (C_1, C_2, \ldots)$ *on* $\mathbb{R}_{\geq 0}$ *such that*

$$C_i = \sum_{i=1}^{K} c^k \Theta_i^k,$$

where $\{c^1, c^2, \ldots, c^K\}$ *are given non-negative constants. The* cost C_∞ *of* σ *is defined as*

$$C_\infty = \lim_{i \to \infty} C_i.$$

Note that the cost of a sequence is always well defined, since the running cost is a non-decreasing sequence.

Let $\sigma = (q_1, q_2, \ldots, q_L)$ be a finite run. Notice that its time schedule can be defined as $\Theta = (\Theta_1, \Theta_2, \ldots, \Theta_L)$ where Θ_i is defined as before. Similarly, the definition of cost can be extended to finite runs in the natural way by letting $C_\infty = C_L$.

4.2 Problem Definition

This section is devoted to the formal definition of the Vehicle Routing Problem with Linear Temporal Logic specifications (VRPLTL). First, a graph theoretic definition of the VRPTL will be presented. Then, a mathematical programming based formulation will be outlined. The former definition is as follows.

Problem 4.2.1 (Vehicle Routing Problem with Linear Temporal Logic Specifications) *Given a vehicle-routing transition system $\mathcal{TS} = (Q, Q_0, \rightarrow, \Pi, L)$ for K vehicles on K weighted directed graphs $G^k = (V^k, E^k, c^k)$, with depots $v_0^k \in V^k$ for $k \in \{1, 2, \dots, K\}$, and an LTL$_{-X}$ formula ϕ defined on Π, find a run σ on $\mathcal{TS}$ such that the following hold:*

- *σ satisfies the LTL$_{-X}$ specification ϕ, i.e., $\sigma \models \phi$.*
- *The cost C_∞ is minimized.*

When compared to the standard VRP (Problem 2.2.1), the VRPLTL features the following fundamental difference. In VRPLTL, one does not have to have all the vertices visited exactly once, instead, has to satisfy an LTL$_{-X}$ formula. A formal proof to be given later in this section, it is worth mentioning here that the VRPLTL refers to a more general class of routing problems than does the standard VRP. Note also that the vehicles in a VRPLTL instance may need to travel an infinite route to satisfy the specification by visiting some of the vertices infinitely often. In a multiple-UAV mission planning context, this resembles the case of a surveillance mission where the UAVs visit possibly a subset of the targets persistently and come back to the base to refuel as necessary. Even though LTL$_{-X}$ can be used to specify the temporal properties of such infinite routes, this chapter focuses only on routes that end in an ending depot. In the rest of this chapter, only a subclass of Problem 4.2.1 will be treated. More precisely, it is assumed that the following assumption holds.

Assumption 4.2.2 *The specification ϕ of Problem 4.2.1 is of the form*

$$\bigwedge_{k \in \{1,\dots,K\}} \left(\Diamond \Box p_0^k\right) \wedge \psi$$

where ψ is an LTL$_{-X}$ formula.

In other words, after some finite time all the vehicles should go back to their depots and stay there forever. Hence, the transition system abstraction of the VRP instance reaches to a final state. Notice that even with this assumption Problem

2.2.1 is in the class of problems that can be modeled as a VRPLTL. Indeed, letting $\psi = \wedge_{i\in\{1,\dots,n\}} \left(\vee_{k\in\{1,\dots,K\}} \Diamond p_i^k\right)$ where $|V^k| = n + 1$, reduces Problem 4.2.1 to Problem 2.2.1, which indicates that VRPLTL is a generalization of the standard VRP, even under Assumption 4.2.2.

With this assumption, the UAV mission planning problems studied in this chapter will be such that all the UAVs land at ending depots in the end of the mission, as generally assumed in the existing literature (see for example [3, 9, 37, 54, 55, 60]). Therefore, even though this assumption arises as a technicality for the computational method presented in Section 4.3 to solve the VRPLTL, it is natural in the VRP setting. Before proceeding with a mathematical programming formulation of the VRPLTL, note the following lemma as a consequence of Assumption 4.2.2.

Lemma 4.2.3 *Let the vehicle-routing transition system $\mathcal{TS}$ be an instance of the VRPTL and ϕ be an LTL_{-X} specification that satisfies Assumption 4.2.2. Then, any run σ that satisfies ϕ is finite.*

In the rest of this section, the network-flow and the set-covering MILP formulations of the VRPTL are outlined. Let $\mathcal{TS}$ denote the vehicle-routing transition system model of the problem, and ϕ be an LTL_{-X} specification that satisfies Assumption 4.2.2, i.e., $\phi = \bigwedge_{k\in\{1,\dots\}} \Diamond\Box v_0^k \wedge \psi$. Recall that the VRPTL is to find a run $\sigma = (q_1, q_2, \dots, q_T)$ such that $\sigma \models \phi$. Let us define a set $\{1, \dots, T\}$ of finite time instances, where T is a large enough integer. Let θ^k be continuous decision variables defined for $k \in \{1, \dots, K\}$ and let $\boldsymbol{\theta}$ denote the vector $\boldsymbol{\theta} = [\theta^1, \theta^2, \dots, \theta^T]$. For all atomic propositions $p_i \in \Pi$ and $t \in \{1, \dots, T\}$, let us define the binary slack variables $P_{p_i}^t$ such that $P_{p_i}^t$ is equal to 1 if $(\sigma, t) \models p_i$ and 0 otherwise. Similarly, let us also define the binary slack variables P_ψ^t which are equal to 1 if $(\sigma, t) \models \psi$ and 0 otherwise. Let $\boldsymbol{P}_{p_i}$ denote the vector $[P_{p_i}^1, P_{p_i}^2, \dots, P_{p_i}^T]$. This vector will be called the *evolution vector* of the atomic proposition p_i. The evolution vector of any LTL_{-X} formula φ is defined similarly. Then, Problem can be formulated as follows.

$$\begin{aligned}
\text{minimize} \quad & f(y), && && (4.1)\\
\text{subject to} \quad & y \in \mathcal{Y}, && && (4.2)\\
& \boldsymbol{P}_{p_i} \in \mathcal{F}_{p_i}(y, \boldsymbol{\theta}), && \forall p_i \in \Pi, && (4.3)\\
& \boldsymbol{P}_{\psi} \in \mathcal{G}_{\psi}(\boldsymbol{P}_{p_1}, \ldots, \boldsymbol{P}_{p_m}), && && (4.4)\\
& P^1_{\psi} = 1, && && (4.5)\\
& \theta^t \leq \theta^{t+1}, && \forall t \in \{1, 2, \ldots T-1\}, && (4.6)
\end{aligned}$$

where y corresponds to the continuous and binary variables used to model the VRP as presented in (2.9-2.16) or in (2.17-2.19), $\mathcal{Y}$ is the feasible set of the variables y, i.e., a slightly relaxed version of the feasible sets given by (2.11 - 2.16) or (2.19) which will be outlined shortly, $\mathcal{F}_{p_i}(y, \boldsymbol{\theta})$ constrains the binary variables vector $\boldsymbol{P}_i$ such that $P^t_i = 1$ if and only if $(\sigma, t) \models p_i$, and $\mathcal{G}_{\psi}$ is the set of all vectors $\boldsymbol{P}_{\psi}$ for which the corresponding evolution of the proposition satisfies the formula ψ. Finally, $P^1_{\psi} = 1$ states that the proposition ψ is `True` at the time 1, i.e., the specification ψ is satisfied at the initial time. Notice that $\bigwedge_{k \in \{1,\ldots\}} \Diamond \Box v^k_0$ is enforced by the constraint set $\mathcal{Y}$, since all the vehicles are constrained to end their route in the depot or not start a route at all by staying at the depot at all times.

As mentioned earlier, VRPTL differs from the VRP in the sense that it requires neither all the vertices to be visited nor all the vehicles to be employed. Computationally, this generalization is realized by slightly relaxing the MILP formulations presented in Chapter 2. In particular, the network-flow formulation given by (2.9-2.16) is generalized by relaxing Constraints (2.10) and (2.11), respectively, as:

$$\textstyle\sum_{i \in \mathcal{L}} \sum_{j \in \mathcal{N}} x_{ijk} \leq 1, \quad \forall k \in \mathcal{K}, \qquad (4.7)$$

$$\textstyle\sum_{k=1}^{K} \sum_{j \in \mathcal{J}, j \neq i} x_{ijk} \leq 1, \quad \forall i \in \mathcal{N}, \qquad (4.8)$$

and adding the following constraint into the formulation,

$$\sum_{i\in\mathcal{L}}\sum_{j\in\mathcal{N}} x_{ijk} - \sum_{i\in\mathcal{N}}\sum_{j\in C} x_{ijk} = 0, \quad \forall k \in \mathcal{K}. \tag{4.9}$$

Notice that the constraints (4.7) ensure that each vehicle starts from at most one starting depot, the constraints (4.8) require every vertex to be visited at most once, and the constraints (4.9) makes each vehicle that starts a route get to an ending depot eventually.

Similarly, the set-covering formulation given by (2.17-2.19) is generalized by relaxing Constraints (2.18) and modifying Constraints (2.19) to

$$\sum_{j\in J(k)} x_{jk} \leq 1, \quad \forall k \in \mathcal{K}, \tag{4.10}$$

which implies that each vehicle can execute at most one route.

Given the generalizations, the set $\mathcal{Y}$ can be formulated with constraints (2.12-2.15,4.7-4.9) for the network-flow formulation, and with (4.10) for the set-covering formulation.

In this section, the MILP formulation of $\mathcal{Y}$ was shown. In Sections 4.3 and 4.4, the sets $\mathcal{G}_\phi$ and $\mathcal{F}_i$ are formulated in a MILP framework.

4.3 MILP Formulation of LTL$_{-X}$ Specifications

This section is devoted to the definition of $\mathcal{G}_\psi(\boldsymbol{P}_{p_1},\ldots,\boldsymbol{P}_{p_m})$. Formally, $\mathcal{G}_\psi(\boldsymbol{P}_{p_1},\ldots,\mathcal{P}_{p_m})$ is a set of evolution vectors $\boldsymbol{P}_\psi$ parametrized by the formula ψ and the evolution vectors $\boldsymbol{P}_{p_i}$ for all $p_i \in \Pi$. The salient property of this set is that $\boldsymbol{P}_\psi \in \mathcal{G}_\psi(\boldsymbol{P}_{p_1},\ldots,\boldsymbol{P}_{p_m})$ holds if and only if the run $\sigma = (q_1, q_2, \ldots, q_T)$ with $L(q_t) = \{p_i \,|\, P^t_{p_i} = 1\}$ satisfies ϕ at the initial time, i.e., $(\sigma, 1) \models \phi$. In the remainder of this section, $\mathcal{G}_\psi(\boldsymbol{P}_{p_1},\ldots,\mathcal{P}_{p_m})$ is first constructed for LTL$_{-X}$ formulae with height one, i.e., formulae with only one operator. Then, the procedure is generalized to LTL$_{-X}$ formulae with arbitrary height using a recursive algorithm.

Consider the height one-formula $\psi = \neg p_1$, the semantics of which were provided by Equation (3.3). Notice that the set inclusion $\boldsymbol{P}_{\psi} \in \mathcal{G}_{\psi}(\boldsymbol{P}_{p_1})$ holds if and only if the following inequalities are satisfied:

$$P^t_{\psi} = (1 - P^t_{p_1}), \quad t \in \{1, \ldots, T\}; \tag{4.11}$$

$$0 \le P^t_{\psi} \le 1, P^t_{\psi} \in \mathbb{R}, \quad t \in \{1, \ldots, T\}. \tag{4.12}$$

Hence, by construction, if the value of p is `True` at a time instance t, i.e., $P^t_{p_1} = 1$, then the value of P^t_{ϕ} is `False`, and vice versa. The set $\mathcal{G}_{\psi}(\boldsymbol{P}_{p_1})$ can be written also as

$$\mathcal{G}_{\psi}(\boldsymbol{P}_{p_1}) = \left\{\boldsymbol{P}_{\psi} \,|\, P^t_{\psi} = (1 - P^t_{p_1}), 0 \le P^t_{\psi} \le 1, P^t_{\psi} \in \mathbb{R} \text{ for all } t \in \{1, \ldots, T\}\right\}.$$

Just for the sake of brevity, however, formulations of $\mathcal{G}_{\psi}$ for other operators will only be given as the set of constraints that $\boldsymbol{P}_{\psi}$ needs to satisfy.

Recall the semantics of the conjunction operator given by Equation (3.6). To make the MILP formulation more compact, consider a more general operator where conjunction of two or more propositions can be written as $\psi = \bigwedge_{i=1}^{k} p_i$. The semantics of this operator is defined by recursive application of the usual conjunction operator, i.e., $\bigwedge_{i=1}^{k} p_i = (((p_1 \wedge p_2) \wedge p_3) \wedge \ldots p_k)$. Then, the set inclusion $\boldsymbol{P}_{p_{\psi}} \in \mathcal{G}_{\psi}(\boldsymbol{P}_{p_1} \ldots \boldsymbol{P}_{p_k})$ holds if and only if $\boldsymbol{P}_{\psi}$ satisfies the following inequalities:

$$P^t_{\psi} \le P^t_{p_i}, \qquad i \in \{1, \ldots, k\}, \quad t \in \{1, \ldots, T\}; \tag{4.13}$$

$$P^t_{\psi} \ge \textstyle\sum_{i=1}^{k} P^t_{p_i} - (k - 1), \quad t \in \{1, \ldots, T\}; \tag{4.14}$$

$$0 \le P^t_{\psi} \le 1, P^t_{\psi} \in \mathbb{R}, \quad t \in \{1, \ldots, T\}. \tag{4.15}$$

The disjunction operator defined by the semantics equivalencies (3.4) can also be generalized to represent disjunction of k propositions as $\phi = \bigvee_{i=1}^{k} p_i = (((p_1 \vee$

$p_2) \vee p_3) \vee \ldots p_k)$. Then, $\boldsymbol{P}_{p_\psi} \in \mathcal{G}_\psi(\boldsymbol{P}_{p_1} \ldots \boldsymbol{P}_{p_k})$ can be characterized as follows:

$$P^t_\psi \leq \textstyle\sum_{i=1}^k P^t_{p_i}, \qquad t \in \{1,\ldots,T\}; \tag{4.16}$$

$$P^t_\psi \geq P^t_{p_i}, \qquad i \in \{1,\ldots,k\}, \quad t \in \{1,\ldots,T\}; \tag{4.17}$$

$$0 \leq P^t_\psi \leq 1, P^t_\psi \in \mathbb{R} \quad t \in \{1,\ldots,T\}. \tag{4.18}$$

Noting the semantics of the eventually operator in (3.8), the set inclusion $\boldsymbol{P}_{p_\psi} \in \mathcal{G}_\psi(\boldsymbol{P}_{p_1})$ for the formula $\psi = \Diamond p_1$ can be characterized as follows:

$$P^t_\psi \leq \textstyle\sum_{\tau=t}^T P^\tau_{p_1} \qquad t \in \{1,\ldots,T\}; \tag{4.19}$$

$$P^t_\psi \geq P^\tau_{p_1}, \qquad \tau \in \{t,\ldots,T\}, \quad t \in \{1,\ldots,T\}; \tag{4.20}$$

$$0 \leq P^t_\psi \leq 1, P^t_\psi \in \mathbb{R}, \quad t \in \{1,\ldots,T\}. \tag{4.21}$$

For the formula $\phi = \Box p_1$, the semantics given by Equation (3.7) leads to the following characterization of the set inclusion $\boldsymbol{P}_{p_\psi} \in \mathcal{G}_\psi(\boldsymbol{P}_{p_1})$:

$$P^t_\psi \leq P^\tau_{p_1} \qquad \tau \in \{t,\ldots,T\}, \quad t \in \{1,\ldots,T\}; \tag{4.22}$$

$$P^t_\psi \geq \textstyle\sum_{\tau=t}^T P^\tau_{p_1} - (T-t) \quad t \in \{1,\ldots,T\}; \tag{4.23}$$

$$0 \leq P^t_\psi \leq 1, P^t_\psi \in \mathbb{R} \quad t \in \{1,\ldots,T\}. \tag{4.24}$$

Let $\psi = p\mathcal{U}q$ and recall the semantics of the until operator given in (3.5). To characterize the set inclusion $\boldsymbol{P}_{p_\psi} \in \mathcal{G}_\psi(\boldsymbol{P}_{p_1}, \boldsymbol{P}_{p_2})$ as linear constraints, let us define the extra continuous variables α_{tj}. Then, the following is a characterization of the set:

$$\begin{aligned}
\alpha_{tj} &\geq P^j_{p_2} + \textstyle\sum_{\tau=t}^{j} P^\tau_{p_1} - (j-t+2), && j \in \{t+1,\dots,T\}, \quad t \in \{1,\dots,T-1\}; && (4.25)\\
\alpha_{tj} &\leq P^j_{p_2}, && j \in \{t+1,\dots,T\}, \quad t \in \{1,\dots,T-1\}; && (4.26)\\
\alpha_{tj} &\leq P^\tau_{p_1}, && \tau \in \{t,\dots,j\}, \quad j \in \{t+1,\dots,T\}, &&\\
& && t \in \{1,\dots,T-1\}; && (4.27)\\
\alpha_{tt} &= P^t_{p_1}, && t \in \{1,\dots,T\}; && (4.28)\\
P^t_\phi &\leq \textstyle\sum_{j=t}^{T} \alpha_{tj}, && t \in \{1,\dots,T\}; && (4.29)\\
P^t_\phi &\geq \alpha_{tj}; && t \in \{1,\dots,T\}, \quad j \in \{t,\dots,T\}; && (4.30)\\
0 \leq P^t_\phi &\leq 1, P^t_\phi \in \mathbb{R}, && t \in \{1,\dots,T\}. && (4.31)
\end{aligned}$$

The following lemma holds for the constraints (4.11 - 4.31).

Lemma 4.3.1 *Given an LTL_{-X} formula ψ of height one defined on the set of atomic propositions $\Pi = \{p_1,\dots,p_m\}$, and a finite run $\sigma = (q_1, q_2, \dots, q_T)$, let $P^t_{p_i} = 1$ if $p_i \in L(q_t)$ and $P^t_{p_i} = 0$ otherwise for all $t \in \{1,\dots,T\}$, and $\boldsymbol{P}_\psi \in \mathcal{G}_\psi(\boldsymbol{P}_{p_1},\dots,\boldsymbol{P}_{p_m})$ hold. Then, for all $t \in \{1,\dots,T\}$, $(\sigma,t) \models \phi$ holds if and only if $P^t_\psi = 1$ holds.*

Given an LTL_{-X} formula Ψ, let $\varphi_1, \varphi_2, \dots, \varphi_k$ be all its immediate subformulae. Note that Ψ must be in one of the following forms: (i) $\neg\varphi_1$, (ii) $\bigwedge_{i=1,\dots,k}\varphi_i$, (iii) $\bigvee_{i=1,\dots,k}\varphi_i$, (iv) $\Diamond\varphi_1$, (v) $\Box\varphi_1$, or (vi) $\varphi_1\mathcal{U}\varphi_2$. For each of the six cases, let `BasisConstraints`$(\Psi, \varphi_1, \varphi_2, \dots, \varphi_m)$ denote the set of evolution vectors $\boldsymbol{P}_\Psi$, which satisfy Constraints (4.11), (4.13-4.14), (4.16-4.17), (4.19-4.20), (4.22-4.23), and (4.25-4.30), respectively, where ϕ_i are substituted instead of p_i, respectively, for all i.

The recursive procedure that constructs $\mathcal{G}_\Psi$ for Ψ with arbitrary height is given in Algorithm 1. If Ψ is an atomic proposition, then Algorithm 1 returns a set of constraints that ensures P^t_Ψ takes a binary value for all $t \in \{1,2,\dots,T\}$ (Line 2). Otherwise, it first constructs the sets $\mathcal{G}_{\varphi_i}(\boldsymbol{P}_{p_1},\dots,\boldsymbol{P}_{p_m})$ for all immediate subformulae φ_i of Ψ by calling itself with different parameters (Line 6), then constructs the necessary basis constraints that couple the evolution vectors of ϕ_i with Ψ (Line 9), and returns the set $\mathcal{G}_\Psi(\boldsymbol{P}_{p_1},\dots,\boldsymbol{P}_{p_m})$, which is a combination of all these constraints (see Line 10).

```
1  if $\Psi \in \{p_1, p_2, \ldots, p_m\}$ then
2      return $\mathcal{G}_\Psi(\boldsymbol{P}_{p_1}, \ldots, \boldsymbol{P}_{p_m}) = \left\{\boldsymbol{P}_\Psi \,\middle|\, P^t_\Psi \in \{0,1\}\, \textit{for all } t \in \{1, \ldots, T\}\right\}$
3  else
4      Let $\varphi_1, \varphi_2, \ldots, \varphi_k$ be all immediate subformulae of $\Psi$
5      for $i = 1, 2, \ldots, k$ do
6          $\mathcal{G}_{\varphi_i}$ := ConstructConstraints($\varphi_i, p_1, \ldots, p_m$)
7          Define slack variables $\boldsymbol{P}_{\varphi_i}$
8      end
9      $\mathcal{G}'_\Psi(\boldsymbol{P}_{\varphi_1}, \ldots, \boldsymbol{P}_{\varphi_k})$ = BasisConstraints($\Psi, \varphi_1, \ldots, \varphi_k$)
10     return $\mathcal{G}_\Psi(\boldsymbol{P}_{p_1}, \ldots, \boldsymbol{P}_{p_m}) = \left\{\boldsymbol{P}_\Psi \,\middle|\, \boldsymbol{P}_\Psi \in \mathcal{G}'_\Psi(\boldsymbol{P}_{\varphi_1}, \ldots, \boldsymbol{P}_{\varphi_k}), \boldsymbol{P}_{\varphi_i} \in \mathcal{G}_{\varphi_i}(\boldsymbol{P}_{p_1}, \ldots, \boldsymbol{P}_{p_m}),\ 0 \le P^t_{\varphi_i} \le 1, P^t_{\varphi_i} \in \mathbb{R} \text{ for all } i \in \{1, \ldots, k\} \text{ and } t \in \{1, \ldots, T\}\right\}$
11 end
```

Algorithm 1: ConstructConstraints ($\Psi, p_1, p_2, \ldots, p_m$)

Example Consider the following example with $\Psi = \Diamond(p_1 \vee p_2)$. To construct the constraints $\mathcal{G}_{\Diamond(p_1 \vee p_2)}(\boldsymbol{P}_{p_1}, \boldsymbol{P}_{p_2})$, Algorithm 1 is called with ConstructConstraints ($\Diamond(p_1 \vee p_2), p_1, p_2$). Notice that the unique immediate subformula of $\Diamond(p_1 \vee p_2)$ is $\varphi_1 = (p_1 \vee p_2)$. Hence, Algorithm 1 calls itself with ConstructConstraints(($p_1 \vee p_2, p_1, p_2$)). Noting the immediate subformulae of $p_1 \vee p_2$ are $\varphi_2 = p_1$ and $\varphi_3 = p_2$, this second call also calls itself with ConstructConstraints(p_1, p_1, p_2) and ConstructConstraints(p_2, p_1, p_2), which return

$$\mathcal{G}_{p_1}(\boldsymbol{P}_{p_1}, \boldsymbol{P}_{p_2}) = \left\{\boldsymbol{P}_{\varphi_2} \,\middle|\, P^t_{\varphi_2} \in \{0,1\} \text{ for all } t \in \{1, \ldots, T\}\right\},$$

$$\mathcal{G}_{p_2}(\boldsymbol{P}_{p_1}, \boldsymbol{P}_{p_2}) = \left\{\boldsymbol{P}_{\varphi_3} \,\middle|\, P^t_{\varphi_3} \in \{0,1\} \text{ for all } t \in \{1, \ldots, T\}\right\},$$

respectively.

Then, the execution returns back to the call ConstructConstraints($p_1 \vee p_2, p_1, p_2$), which calls `BasisConstraints`$(p_1 \vee p_2, p_1, p_2)$, which, in turn, returns

$$\mathcal{G}'_{p_1 \vee p_2}(\boldsymbol{P}_{\varphi_2}, \boldsymbol{P}_{\varphi_3}) = \Big\{\boldsymbol{P}_{\varphi_1} \;\Big|\; P^t_{\varphi_1} \leq P^t_{\varphi_i}, \text{ for all } i \in \{2,3\} \text{ and } t \in \{1,\dots,T\}; \\ P^t_{\varphi_1} \geq P^t_{\varphi_2} + P^t_{\varphi_3} - 1, \text{ for all } t \in \{1,\dots,T\}\Big\}.$$

Finally, `ConstructConstraints`$(p_1 \vee p_2, p_1, p_2)$ merges the all the constraints given above to return

$$\mathcal{G}_{p_1 \vee p_2}(\boldsymbol{P}_{p_1}, \boldsymbol{P}_{p_2}) = \Big\{\boldsymbol{P}_{\varphi_1} \;\Big|\; \boldsymbol{P}_{\varphi_1} \in \mathcal{G}'_{\varphi_2 \vee \varphi_3}(\boldsymbol{P}_{p_1}, \boldsymbol{P}_{p_2}); \boldsymbol{P}_{\varphi_2} \in \mathcal{G}_{p_1}(\boldsymbol{P}_{p_1}, \boldsymbol{P}_{p_2}); \boldsymbol{P}_{\varphi_3} \in \mathcal{G}_{p_2}(\boldsymbol{P}_{p_1}, \boldsymbol{P}_{p_2}); \\ 0 \leq P^t_{\varphi_i} \leq 1, P^t_{\varphi_i} \in \mathbb{R} \text{ for all } i \in \{2,3\} \text{ and } t \in \{1,\dots,T\}\Big\},$$

which yields the execution back to the call `ConstructConstraints`$(\Diamond(p_1 \vee p_2))$. Going though a similar computational procedure, first, `BasisConstraints`$(\Diamond\varphi_1, \varphi_1)$ is constructed as

$$\mathcal{G}'_{\Diamond\varphi_1}(\boldsymbol{P}_{\varphi_1}) = \Big\{\boldsymbol{P}_{\Psi} \;\Big|\; P^t_{\psi} \leq \sum_{\tau=t}^{T} P^t_{\varphi_1}, \text{ for all } t \in \{1,\dots,T\}; \\ P^t_{\psi} \geq P^{\tau}_{p_1}, \text{ for all } \tau \in \{t,\dots,T\} \text{ and } t \in \{1,\dots,T\}\Big\},$$

then, `ConstructConstraints`$_{\Diamond(p_1 \vee p_2)}(\boldsymbol{P}_{p_1}, \boldsymbol{P}_{p_2})$ returns

$$\mathcal{G}_{\Diamond(p_1 \vee p_2)}(\boldsymbol{P}_{p_1}, \boldsymbol{P}_{p_2}) = \Big\{\boldsymbol{P}_{\Psi} \;\Big|\; \boldsymbol{P}_{\Psi} \in \mathcal{G}'_{\Diamond\varphi_1}(\boldsymbol{P}_{\varphi_1}); \boldsymbol{P}_{\varphi_1} \in \mathcal{G}_{p_1 \vee p_2}(\boldsymbol{P}_{p_1}, \boldsymbol{P}_{p_2}); \\ 0 \leq P^t_{\varphi_1} \leq 1, P^t_{\varphi_1} \in \mathbb{R} \text{ for all } t \in \{1,\dots,T\}\Big\},$$

which is the desired set of constraints. ∎

Hence, the call `ConstructConstraint`$(\psi, p_1, \dots, p_m)$ to Algorithm 1 returns the set $\mathcal{G}_{\psi}(\boldsymbol{P}_{p_1}, \dots, \boldsymbol{P}_{p_m})$ of constraints. The algorithm defines slack variables before each recursive call (cf. Line 7). Notice that the slack variables need not be constrained as binary variables, which enhances the computational effectiveness of the method. It is easy to see that the algorithm terminates. Indeed, it can be shown that the number of recursive calls is linear with respect to the size of the formula, which implies that the number of constraints is also linear in the size of the formula. It can also be shown that the number of constraints and the slack variables is at most quadratic with the length of the time horizon.

The following theorem states the completeness of the algorithm.

Theorem 4.3.2 *Given an LTL_{-X} formula ψ of arbitrary finite height defined on the set of atomic propositions $\Pi = \{p_1, \ldots, p_m\}$, and a finite run $\sigma = (q_1, q_2, \ldots, q_T)$, let $P^t_{p_i} = 1$ if $q_t[p_i] = \texttt{True}$ and $P^t_{p_i} = 0$ otherwise for all $t \in \{1, \ldots, T\}$, and $\mathbf{P}_\psi \in \mathcal{G}_\psi(\mathbf{P}_{p_1}, \ldots, \mathbf{P}_{p_m})$. Then, for all $t \in \{1, \ldots, T\}$, $(\sigma, t) \models \phi$ if and only if $P^t_\psi = 1$.*

4.4 Applications to Mission Planning Problems

In this section, the sets $\mathcal{F}_{p_i}$ (see Equation (4.3)) are formulated within the multiple-UAV mission planning application domain. The network-flow and the set-covering MILP formulations of the VRPTL are fully developed in Sections 4.4.1 and 4.4.2, respectively. Each section is provided with an example mission planning scenario of practical interest.

4.4.1 The Network Flow Formulation

This section presents the MILP formulation of the atomic propositions followed by modeling and solution of a complex multiple-UAV mission planning example.

Formulation of Propositions for Network Flow Formulation

Recall the definition the vehicle-routing transition system. The set of atomic propositions was composed of p_i^k, where i and k were indices over the sets of customers and vehicles, respectively. By definition, the atomic proposition p_i^k is true if Customer i is being serviced by Vehicle k.

To make the MILP formulation more compact, the set $\mathcal{F}_{p_i^k}$ is constructed such that $P^t_{p_i^k}$ is false if Customer i is not serviced by Vehicle k as of time instance t and is true if it has been visited at some time before t. This approach makes the big-M formulations of MILP possible and practical as shown in the examples below. Notice that this formulation does not incur any loss of any generality, since each p_i^k can be true only once throughout the execution, i.e., each customer can be visited no more than once by the definition of VRPTL.

Let $\bar{T}$ be a big enough real number. In practical terms, $\bar{T}$ can be chosen as an upper bound on the completion time of each meaningful execution. Notice that the set inclusion $\boldsymbol{P}_{p_i^k} \in \mathcal{F}_{p_i^k}(y, \boldsymbol{\theta})$ (see Equation (4.3)) holds if and only if the following constraints hold:

$$P^t_{p_{ik}} \geq \frac{1}{\bar{T}}(\theta_t - t_i) - \left(1 - \sum_{j \in \mathcal{J}} x_{ijk}\right), \quad \forall t \in \{1, \dots, T\}; \tag{4.32}$$

$$P^t_{p_{ik}} \leq \frac{1}{\bar{T}}(t_i - \theta_t), \quad \forall t \in \{1, \dots, T\}; \tag{4.33}$$

$$P^t_{p_{ik}} \leq \sum_{j \in \mathcal{J}} x_{ijk}, \quad \forall t \in \{1, \dots, T\}; \tag{4.34}$$

$$P^t_{p_{ik}} \in \{0, 1\}, \quad \forall t \in \{1, \dots, T\}. \tag{4.35}$$

Using the LTL$_{-X}$ language with the atomic propositions p_i^k, several other properties can be expressed. Consider, for example, the case when a customer can be serviced one of the several vehicles indexed over the set $\kappa \subseteq \{1, 2, \dots, K\}$. The corresponding LTL$_{-X}$ specification is $\diamond \vee_{k \in \kappa} p_i^k$. For computational effectiveness, the proposition $\vee_{k \in \kappa} p_i^k$ can also be represented in the MILP framework, just like p_i^k and treated as an atomic proposition even though it does not have the atomicity property. Let us represent $\vee_{k \in \kappa} p_i^k$ with p_i^κ with a slight abuse of notation. Then, $\mathcal{F}_{p_i^\kappa}$

can be characterized as follows:

$$P^t_{p_{i\kappa}} \geq \frac{1}{\bar{T}}(\theta_t - t_i) - \left(1 - \sum_{k\in\kappa}\sum_{j\in\mathcal{J}} x_{ijk}\right), \quad \forall t \in \{1,\ldots,T\}; \tag{4.36}$$

$$P^t_{p_{i\kappa}} \leq \frac{1}{\bar{T}}(t_i - \theta_t), \quad \forall t \in \{1,\ldots,T\}; \tag{4.37}$$

$$P^t_{p_{i\kappa}} \leq \sum_{k\in\kappa}\sum_{j\in\mathcal{J}} x_{ijk}, \quad \forall t \in \{1,\ldots,T\}; \tag{4.38}$$

$$P^t_{p_{i\kappa}} \in \{0,1\}, \quad \forall t \in \{1,\ldots,T\}. \tag{4.39}$$

Similarly, if $\vee_{k=1,\ldots,K} v^k_i$, denoted by q_i, is of concern, there is even a more compact characterization, which is as follows:

$$P^t_{q_i} \geq \frac{1}{\bar{T}}\left(\theta^t - t_i\right), \quad \forall t \in \{1,\ldots,T\}; \tag{4.40}$$

$$P^t_{q_i} \leq 1 - \frac{1}{\bar{T}}\left(t_i - \theta^t\right), \quad \forall t \in \{1,\ldots,T\}; \tag{4.41}$$

$$P^t_{q_i} \in \{0,1\}, \quad \forall t \in \{1,\ldots,T\}. \tag{4.42}$$

Another special case is when v_j is an ending depot, i.e., $j \in C$. Denoting this atomic proposition by s^k_j, the corresponding characterization is as follows:

$$P^t_{s_{jk}} \geq \frac{1}{\bar{T}}\left(\theta^t - t_{jk}\right) - \left(1 - \sum_{i\in\mathcal{N}} x_{ijk}\right), \quad \forall t \in \{1,\ldots,T\}; \tag{4.43}$$

$$P^t_{s_{jk}} \leq 1 - \frac{1}{\bar{T}}\left(t_{jk} - \theta^t\right), \quad \forall t \in \{1,\ldots,T\}; \tag{4.44}$$

$$P^t_{s_{jk}} \leq \sum_{i\in\mathcal{N}} x_{ijk}, \quad \forall t \in \{1,\ldots,T\}; \tag{4.45}$$

$$P^t_{s_{jk}} \in \{0,1\}, \quad \forall t \in \{1,\ldots,T\}. \tag{4.46}$$

Similarly, one might only be interested in Vehicle k having landed on an ending depot, i.e., $s^k = \vee_{j\in C} v_j^k$, which results in the following more compact formulation:

$$P_{s_k}^t \geq \frac{1}{\bar{T}}\left(\theta^t - \sum_{j\in C} t_{jk}\right), \quad \forall t \in \{1,\ldots,T\}; \tag{4.47}$$

$$P_{s_k}^t \leq 1 - \frac{1}{\bar{T}}\left(\sum_{j\in C} t_{jk} - \theta^t\right), \quad \forall t \in \{1,\ldots,T\}; \tag{4.48}$$

$$P_{s_k}^t \in \{0,1\}, \quad \forall t \in \{1,\ldots,T\}. \tag{4.49}$$

Only for further illustration, in what follows, the set $\mathcal{F}_{q_{ik}}$ is explicitly given for proposition q_k.

$$\mathcal{F}_{q_i} = \left\{(P_{q_i}^1,\ldots,P_{q_i}^t) \,\middle|\, P_{q_i}^t \geq \frac{1}{\bar{T}}\left(\theta^t - t_i\right), P_{q_i}^t \leq 1 - \frac{1}{\bar{T}}\left(t_i - \theta^t\right), P_{q_i}^t \in \{0,1\}, \text{ for all } t \in \{1,\ldots,T\}\right\}$$

For all the other propositions, the corresponding sets can be obtained in a similar manner.

Let Π' be the set of all propositions treated as atomic propositions in the formula. Obviously, $\Pi \subseteq \Pi'$. Having defined the extra slack propositions, the following assumption complements these definitions and improves the computational effectiveness.

Assumption 4.4.1 *The specification ψ is such that, for any subformula of ψ in the form of $\Diamond\varphi_1$, $\Box\varphi_1$, $\varphi_1\mathcal{U}\varphi_2$, or $\varphi_1\mathcal{W}\varphi_2$, the following hold*

- $\varphi_1 \in \Pi'$ *or* $\neg\varphi_1 \in \Pi'$,
- $\varphi_2 \in \Pi'$ *or* $\neg\varphi_2 \in \Pi'$.

This assumption implies that the temporal operators eventually, always, until and unless are only applied to either propositions in Π' or their negations, hence rendering the specification ψ a combination of conjunctions, disjunctions, and negations of several temporal specifications of the form above. With this assumption, the number of time instances required to solve the problem can be fairly small. However, the assumption is not very constraining in the application domain. Motivational examples of complex specifications that satisfy Assumption 4.4.1 will follow in the next Section. Moreover, in some cases, formula that involve nesting of operators inside temporal operators is equivalent to formula that satisfy Assumption 4.4.1. The following are examples for formulae of height two.

- $[(\Diamond\phi) \vee (\Diamond\psi)] \Leftrightarrow [\Diamond(\phi \vee \psi)]$,
- $[(\Box\phi) \wedge (\Box\psi)] \Leftrightarrow [\Box(\phi \wedge \psi)]$,
- $[(\phi\mathcal{U}\psi) \vee (\phi\mathcal{U}\varphi)] \Leftrightarrow [\phi\mathcal{U}(\psi \vee \varphi)]$,
- $[(\phi\mathcal{U}r) \wedge (\psi\mathcal{U}\varphi)] \Leftrightarrow [(\phi \wedge \psi)\mathcal{U}\psi]$,

where ϕ, ψ and φ are any LTL$_{-X}$ formula. Moreover, the following implications can also be employed, leading to a conservative solution,

- $[(\Box\phi) \vee (\Box\psi)] \Rightarrow [\Box(\phi \vee \psi)]$,
- $[(\phi\mathcal{U}r) \vee (\psi\mathcal{U}\varphi)] \Rightarrow [(\phi \vee \psi)\mathcal{U}\psi]$.

It is possible to relax Assumption 4.4.1 within the MILP framework. One straightforward way of doing this is to allow only one proposition to change its value at every time instance. For this purpose, let us define a continuous variable $\beta_i^t \in \mathbb{R}$ with $0 \leq \beta_i^t \leq 1$ for every proposition $p_i \in \Pi$ and each time instance $t \in \{1, 2, \ldots, T\}$. This variable needs to be one if p_i made a transition from `False` to `True` between time instances t and $t+1$, and 0 if no such transition occurs. The following constraints ensure this property.

$$\beta_i^t \geq P_{P_i}^{t+1} - P_{P_i}^t, \quad \forall p_i \in \Pi, \quad \forall t \in \{1,2,\ldots,T\}; \tag{4.50}$$

$$\beta_i^t \geq P_{P_i}^t - P_{P_i}^t, \quad \forall p_i \in \Pi, \quad \forall t \in \{1,2,\ldots,T\}; \tag{4.51}$$

$$\beta_i^t \leq P_{P_i}^t + P_{P_i}^{t+1}, \quad \forall p_i \in \Pi, \quad \forall t \in \{1,2,\ldots,T\}; \tag{4.52}$$

$$\beta_i^t \leq 2 - (P_{P_i}^t + P_{P_i}^{t+1}), \quad \forall p_i \in \Pi, \quad \forall t \in \{1,2,\ldots,T\}. \tag{4.53}$$

Finally, the following constraint allows no more than one transition between two consequent time instances:

$$\sum_{i \in \mathcal{P}} \beta_i^t \leq 1, \quad \forall t \in \{1, \ldots, T\}. \tag{4.54}$$

This method, however, introduces several extra constraints and variables. Moreover, one has to define as many time instances as the number of atomic propositions. Even though the number of variables increase polynomially with the number of atomic propositions, the computation time increases drastically since the variables that represent the atomic propositions are binary variables.

Examples

Consider a mission planning example and observe the changes in the resulting mission plan for slight changes in the scenario. The example considered in this section is motivated by a complex military operation, loosely inspired by the events narrated in [12]. Consider an urban setting, in which a friendly unit is pinned down by enemy units and needs to be rescued. There are three groups of enemy infantry, denoted as T1, T2 and T3, two of which are protected by Surface-to-Air Missile (SAM) units S1 and S2. SAM unit S1 protects only T1 and S2 protects only T2. There are four friendly UAVs with different capabilities, which are to be outlined shortly. The UAVs are launched from one launch site L1 and can land on one of the two landing sites C1 and C2. See Figure 4-1 for a map of the scenario, in which the precise positions of the units are shown. The friendly UAVs V1,V2, V3, and V4 travel with 25mph, 25mph, 40mph and 12mph respectively. Time to fly from one node to another one is computed using the manhattan distance, since it is believed to be a good indicator of distance in urban environments [60]. Each target also has a servicing time of 0.25 hours, which is incorporated into the cost. The capabilities of the UAVs are as follows: V1 and V2 can attack the targets, but they can not attack the SAM sites; V1 is vulnerable to S1, whereas V2 is vulnerable to both S1 and S2; V3 can not engage the targets T1 and T2, even though it can engage T3 as well as the SAM sites S1 and S2; V4 is an autonomous road vehicle in this scenario, which

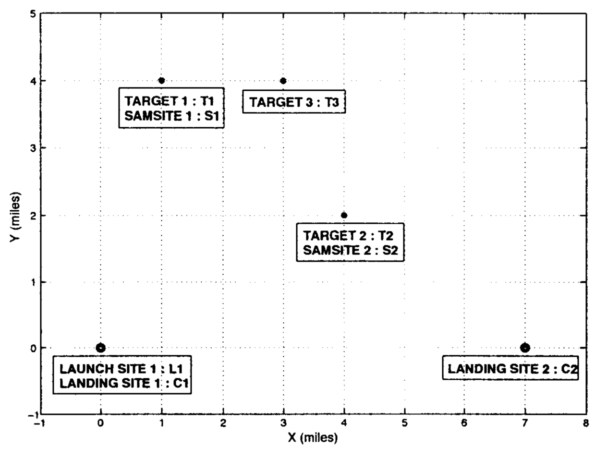


Figure 4-1: Map of the "Black-Hawk-Down" scenario

travels with relatively low speed, but it can engage any of the SAM sites or targets.

The mission objective is to destroy either T1 and T3, or T2 and T3. Thus making a way through for the infantry from the T1 side to base C1 or from the T2 side to base C2, respectively. Finally, if the infantry units escape from the T2 side and get to base C2, then V4 must meet them at base C2, i.e., V4 must go to base C2 even if it does not attack any of the targets.

The LTL$_{-X}$ mission specification is a combination of mission objectives and mission constraints. Recall that the mission objective is to eventually destroy T3 along with either T1 or T2. This specification can be expressed in LTL$_{-X}$ as $(\Diamond q_{T1} \vee \Diamond q_{T2}) \wedge (\Diamond q_{T3})$. Recall that if T2 is destroyed then V4 has to get to C2, which can be expressed in LTL$_{-X}$ as $(\Diamond q_{T2}) \rightarrow (\Diamond s_{C2,V4})$. Moreover, if T1 is serviced by V1 or V2 then S1 has to be destroyed beforehand, i.e., $(\neg p_{T1,\{V1,V2\}}) \mathcal{W} (q_{S1})$. Similarly, for T2, we have that $(\neg p_{T2,V2}) \mathcal{W} (q_{S2})$. Finally, recall that V1 and V2 can not destroy the SAM sites, which is equivalent to $(\Box \neg p_{V1,S1}) \wedge (\Box \neg p_{V1,S2}) \wedge (\Box \neg p_{V2,S2}) \wedge (\Box \neg p_{V2,S2})$. Similarly, noting that V3 can not destroy the targets T1 and T2 results in $(\Box \neg p_{V3,T1}) \wedge (\Box \neg p_{V3,T2})$.

Then, the overall LTL$_{-X}$ specification is the conjunction of all these, i.e.,

$$\begin{aligned}\phi = \;& [(\Diamond q_{T1} \vee \Diamond q_{T2}) \wedge (\Diamond q_{T3})] \wedge [(\Diamond q_{T2}) \rightarrow (\Diamond s_{C2,V4})] \\ & \wedge[(\Box\neg p_{T1,\{V1,V2\}}) \vee (\neg p_{T1,\{V1,V2\}})\mathcal{U}(q_{S1})] \\ & \wedge[(\Box\neg p_{T2,V2}) \vee (\neg p_{T2,V2})\mathcal{U}(q_{S2})] \\ & \wedge[(\Box\neg p_{V1,S1}) \wedge (\Box\neg p_{V1,S2}) \wedge (\Box\neg p_{V2,S2}) \wedge (\Box\neg p_{V2,S2})] \\ & \wedge[(\Box\neg p_{V3,T1}) \wedge (\Box\neg p_{V3,T2})]. \qquad (4.55)\end{aligned}$$

Recall the cost function given in Equation (2.9). This cost function represents the total risk of employing all the UAVs in the mission. More precisely, the parameter c_k represents the risk of employing UAV k in the mission per unit time. The larger the c_k is, the bigger the risk of utilizing vehicle k will be, which will discourage the algorithm from using k in the mission. For the numerical example, let $c_k = 1$ for $k \in \{V1, V2, V3, V4\}$ associating equal risk to all the UAVs. The optimal solution is depicted in Figure 4-2(a). The resulting mission plan is to use only V4 and destroy T3 and T1 respectively. Notice that S1 was not engaged, since V4 is not vulnerable to any of the SAM sites.

The optimal solution to many combinatorial optimization problems varies drastically in structure as the problem parameters varied slightly. To demonstrate an example, consider the case in which V4 is 2mph slower. That is V4 travels with 10mph speed. This time the solution is given in Figure 4-2(b). In this case, the solution is to destroy T1 by V2, and S1 and T3 by V3. SAM unit S1 is destroyed by V3 before V1 engages T1, since V2 can be hit by S1.

In addition to the modification in V4's speed, consider the case for which T1 and S1 are 4 miles to the north. The optimal solution for this scenario is depicted in Figure 4-2(c). This time, it is more advantageous to engage T2 and T3 with V4. Notice that V4 finally gets to C2 to deliver the necessary cargo.

To demonstrate a final example, let V4 travel with 8mph, i.e., 2mph even more slower, in addition to all the other changes in parameters. In this case, the resulting optimal mission plan is depicted in Figure 4-2(d). In the optimal solution, T2 and

T3 are engaged by V1. Note that V1 is not vulnerable to S2. Notice also that V4 lands on C2 just to deliver the necessary cargo without destroying any of the targets.

The cost function that was considered so far was an indication of the total risk. However, in some cases, it is fairly important to complete the mission in minimum time. To model this cost function in the MILP framework, let us define the continuous decision variable t_{max} and introduce the following set of constraints to the MILP formulation: $t_{jk} \leq t_{max}$ for $\forall j \in C$ and $\forall k \in \mathcal{K}$. The cost function is t_{max}, which is minimized to complete all the mission requirements in minimum time. Consider the scenario described before without the parameter modifications. Figure 4-2(e) shows the optimal solution, where V1 destroys T2, V3 destroys T3, and V4 heads out to base C2 to meet the rescued infantry without engaging any of the targets. Notice that the minimum time solution employs as many vehicles as possible in order to accomplish the mission as soon as possible. But, the solution risks all three of the UAVs whereas the minimum risk solution has employed only one UAV (see Figure 4-2(a)). Indeed, the amount of risk taken, i.e., the total time that the UAVs have been employed, was 1.92 hours in the minimum risk solution whereas the same function happens to be 2.62 hours in the minimum time solution. On the other hand, the mission time, i.e., the time that the mission was completed was 1.92 hours for the minimum risk solution. But, with the minimum time solution, this comes down to 0.94 hours.

Noting the considerable difference of risk and time between the minimum time and minimum risk solutions, let us employ a cost function which is a convex combination of the previous two. Let parameter α be a real number such that $0 \leq \alpha \leq 1$ and consider the cost function $\alpha \sum_{j \in C} \sum_{k \in \mathcal{K}} c_k t_{ck} + (1 - \alpha) t_{max}$. For $c_k = 1$ for all $k \in \mathcal{K}$ and $\alpha = 0.5$, Figure 4-2(f) depicts the optimal mission plan, where V3 first destroys S1 and then T3 and comes back to base C1 whereas the relatively faster vehicle V1 destroys S1. Note that, destruction of S1 occurs before the destruction of T1 allowing V1 to engage T1. In this plan, interestingly, the total time that the UAVs are employed is 2 hours and the mission time is 1.1 hours. Note that this solution is pretty close to both the risk of the minimum risk solution and the mission time of minimum time solution satisfying the mission specifications in a reasonable amount of time while taking a relatively small amount of risk.

4.4.2 The Set-covering Formulation

In the previous section, the network-flow formulation of the VRPTL was developed along with a military multiple-UAV mission planning example. This section briefly outlines the set-covering formulation of the VRPTL and presents an illustrative example.

Formulation of Propositions

Recall that a set Π' of propositions were defined in the previous section to denote a set of proposition which are directly formulated within the MILP framework. The same methodology is explored in this section as well. Before presenting the propositions for the application domain, let us introduce two new matrices: V and W, which are computed in the presolve phase of the set-covering formulation. The time that Customer i is reached in Permutation j of Vehicle k, denoted as V_{ijk}, is computed for $\forall i \in V$, $\forall j \in J(k)$ and $\forall k \in \mathcal{K}$. If kth vehicle never reaches the ith target in its jth permutation, then V_{ijk} is equal to $\tilde{T}$, which is a big enough real number such that $\max_{i,j,k} V_{ijk} = \tilde{T}$. The time that vehicle k reaches its ith customer in its jth permutation, denoted as W_{ijk}, is computed for $\forall i \in V$, $\forall j \in J(i)$ and $\forall k \in \mathcal{K}$.

Let $\bar{T}$ be a real number which is slightly larger than $\tilde{T}$. Let p_{ik} denote the proposition, which states that *"Customer i has been serviced by vehicle k"*. Then, the set inclusion $\boldsymbol{P}_{p_{ik}} \in \mathcal{F}_{p_{ik}}(y, \theta)$ holds if and only if the following inequalities hold:

$$P^t_{p_{ik}} \geq \frac{1}{\bar{T}}\left(\theta^t - \sum_{j=1}^{J(i)} V_{ijk}x_{ij}\right), \quad \forall t \in \{1,\ldots,T\}; \tag{4.56}$$

$$P^t_{p_{ik}} \leq 1 - \frac{1}{\bar{T}}\left(\sum_{j=1}^{J(i)} V_{ijk}x_{ijk} - \theta^t\right), \quad \forall t \in \{1,\ldots,T\}; \tag{4.57}$$

$$P^t_{p_{ik}} \in \{0,1\}, \quad \forall t \in \{1,\ldots,T\}. \tag{4.58}$$

As an other example, let q_k be the proposition stating that *"Customer k has been serviced"*. Then, a more compact characterization of the set $\mathcal{F}_{q_k}(y,\theta)$ can be given as follows:

$$P^t_{q_k} \geq \frac{1}{\tilde{T}}\left(\theta^t - \sum_{i=1}^{M}\sum_{j=1}^{J(i)} V_{ijk}x_{ij} + (N-1)\tilde{T}\right), \quad \forall t \in \{1,\dots,T\}; \tag{4.59}$$

$$P^t_{q_k} \leq 1 - \frac{1}{\tilde{T}}\left(\sum_{i=1}^{M}\sum_{j=1}^{J(i)} V_{ijk}x_{ij} - \theta^t - (N-1)\tilde{T}\right), \quad \forall t \in \{1,\dots,T\}; \tag{4.60}$$

$$P^t_{q_k} \in \{0,1\}, \quad \forall t \in \{1,\dots,T\}. \tag{4.61}$$

Consider the following final example. Let r_{kz} be an atomic proposition which stands for the statement *Vehicle k has serviced more than z customers*. Then, the following set of constraints is a characterization of the corresponding set $\mathcal{F}_{r_{kz}}(y,\theta)$:

$$P_{r_{kz}} \geq \frac{1}{N}\left(\theta^t - \sum_{j=1}^{J(i)} W_{ijk}x_{ij}\right), \quad \forall t \in \{1,\dots,T\}; \tag{4.62}$$

$$P_{r_{kz}} \leq 1 - \frac{1}{N}\left(\sum_{j=1}^{J(i)} W_{ijk}x_{ij} - \theta^t\right), \quad \forall t \in \{1,\dots,T\}; \tag{4.63}$$

$$P_{r_{kz}} \in \{0,1\}, \quad \forall t \in \{1,\dots,T\}. \tag{4.64}$$

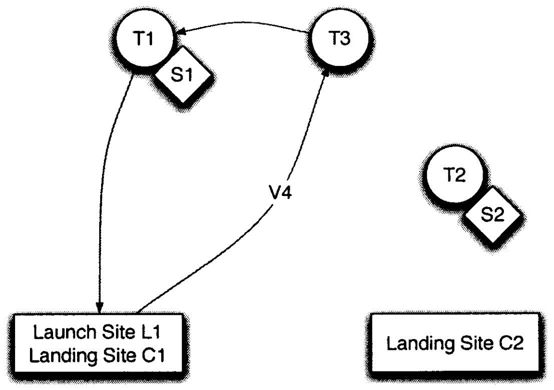
Simulations of the Set-covering Formulation

Consider a simple illustrative mission with two UAVs: V1, V2, and three targets: T1, T2 and T3. The mission objective is to engage target T3 as well as to engage either T1 or T2. Also, if T2 is engaged then the destruction of T3 has to be verified by visiting T3 again after its destruction. But if T1 is engaged then there is no need for such a verification. Let us will denote the destruction of T3 by T3a and its verification by T3b.

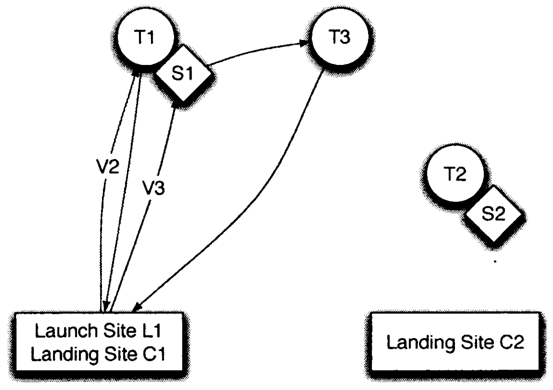
The objective of the mission can be denoted by the LTL$_{-X}$ specification $\Diamond q_{T1} \vee \Diamond q_{T2}$, which is equivalent to stating that either T1 or T2 has to be destroyed eventually. One of the mission constraints can be represented as $(\neg q_{T3b})\mathcal{U}q_{T3a}$, which implies that T3 can only be verified after its destruction. Note that this specification also implies that the destruction of T3 (denoted by T3a) will eventually happen. Recall that if T1 is not visited then destruction of T3 must be verified, which can be expressed as $(\neg\Diamond q_{T1}) \Rightarrow (\Diamond q_{T3b})$. Finally, to make the problem more interesting, let us impose the condition that V2 can not be used for destruction of T3, and V1 can not be used for the verification of the destruction. These conditions can be specified as $\Box\neg p_{T3a,V2}$ and $\Box\neg p_{T3a,V1}$ respectively. Then, the LTL$_{-X}$ mission specification is

$$\begin{aligned} \phi = \; & [\Diamond q_{T1} \vee \Diamond q_{T2}] \wedge [(\neg q_{T3b}) \mathcal{U} q_{T3a}] \\ & \wedge [(\neg\Diamond q_{T1}) \Rightarrow (\Diamond q_{T3b})] \wedge [\Box\neg p_{T3a,V2}] \wedge [\Box\neg p_{T3b,V1}]. \end{aligned} \tag{4.65}$$

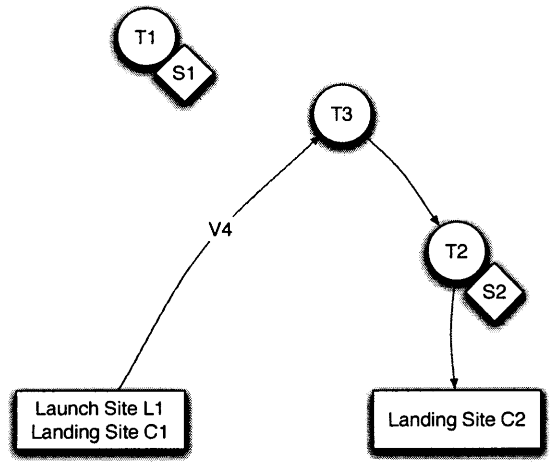
A map of this mission is given in Figure 4-3(a); and the optimal solution for this scenario is illustrated in Figure 4-3(b). Notice that, in the optimal mission plan, T1 is not destroyed. Instead, the optimal plan is to destroy T3 and then verify the destruction by V1. Furthermore, V2 engages T2 in order to satisfy the mission specification. 0.1 hours after mission starts, the destruction of T3 takes place, which is followed by the verification after 0.01 hours after the destruction. The destruction of T2 takes place 0.17 hours after the mission start.



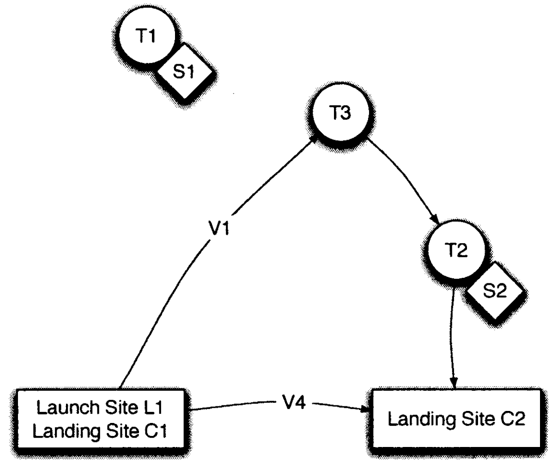
(a) Simulation 1



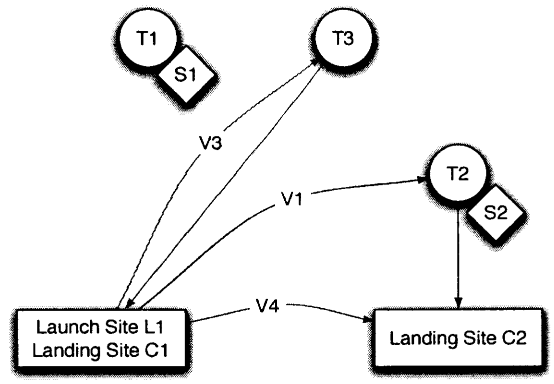
(b) Simulation 2



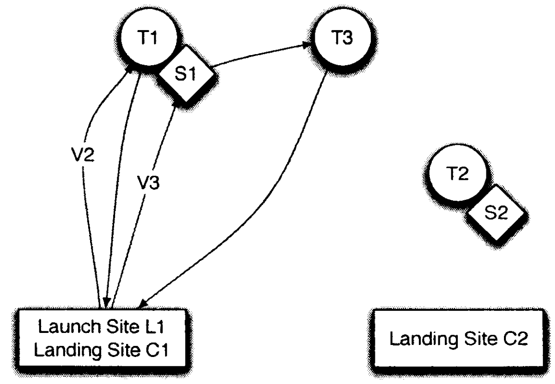
(c) Simulation 3



(d) Simulation 4

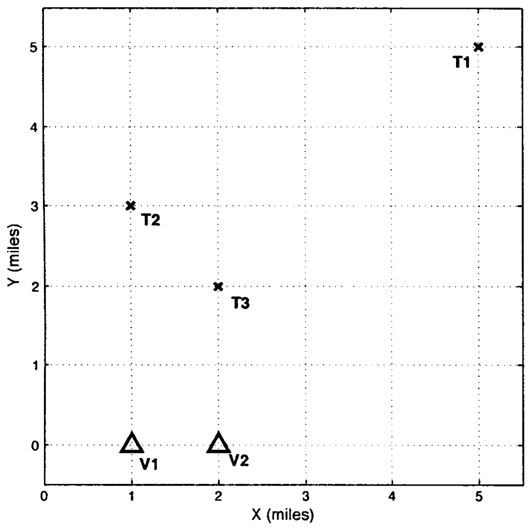


(e) Simulation 5

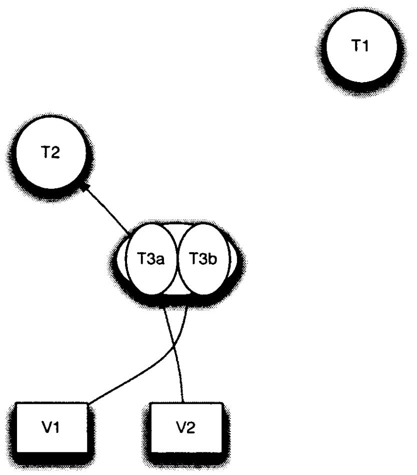


(f) Simulation 6

Figure 4-2: Simulations of the military scenario



(a) Map of the battle damage assessment scenario



(b) Solution for the battle damage assessment scenario

Figure 4-3: Simulation results for the battle damage assessment scenario

Chapter 5

Vehicle Routing with Metric Temporal Logic Specifications

In this chapter, the real-time temporal logic MTL is considered as a candidate for specification of temporal and logical constraints in a vehicle routing setting. MTL allows specification of qualitative properties of time such as deadlines and time windows. In this regard, it differs from the linear temporal logics discussed in the earlier chapters and is a strict superset of LTL_{-X}. In the spirit of the previous chapter, the problem is first formalized, and then a mixed-integer linear programming formulation is developed to solve it to optimality.

5.1 Preliminary Definitions

Recall the definition of a vehicle-routing transition system from Chapter 4. To model instances of the vehicle routing problem with metric temporal logic specifications, vehicle-routing transition systems can also be adopted. Recall also that the semantics of MTL are defined on timed state sequences of transition systems, rather than their runs. Hence, one needs to define the timed state sequences of a given vehicle routing transition system to formally define the VRPMTL. Notice, however, that the vehicle-routing transition system implicitly conveys this information in a given run. More precisely, the states contain the information regarding the exact time of the state transition, using which a timed state sequence can be derived given a run.

Formally speaking, each run of a vehicle-routing transition system map to a timed state sequence on the vehicle routing transition system as follows.

Definition 5.1.1 (Timed State Sequence of a Run) *Given a vehicle-routing transition system $\mathcal{TS}$, a run $\sigma = \big((v_1, \theta_1), (v_2, \theta_2), \ldots, (v_n, \theta_n)\big)$, the corresponding timed state sequence ω of the run σ on $\mathcal{TS}$ is defined as follows.*

$$\omega = (\sigma, \kappa),$$

$$\kappa = (I_1, I_2, \ldots, I_n),$$

where the left limit $l(I_i)$ of I_i is equal to θ_i for all $i = 1, 2, \ldots, I_n$ and the right limit $r(I_n)$ is equal to infinity.

Noting that the subsequent intervals in κ are all adjacent, this definition suggests that I_i is indeed the time interval, in which all the vehicles are in the same vertex, for all $i = 1, 2, \ldots, n$.

Notice that given a run σ, the corresponding timed state sequence ω is uniquely defined for all finite σ according to Definition 5.1.1. Recall also that given a finite run on a vehicle-routing transition system, the cost of σ is well defined according to Definitions 4.1.3 and 4.1.4.

5.2 Problem Definition

A formal definition of the VRPMTL is the following.

Problem 5.2.1 *Given a vehicle-routing transition system $\mathcal{TS} = (Q, Q_0, \rightarrow, \Pi, L)$ for K vehicles on K directed graphs $G^k = (V^k, E^k, c^k)$ with depots $v_0^k \in V^k$ for $k \in \{1, 2, \ldots, K\}$, and an MTL formula ϕ defined on Π, find a run σ on $\mathcal{TS}$ such that the following hold:*

- *the timed state sequence ω that corresponds to σ according to Definition 5.1.1, satisfies the MTL specification, i.e., $\omega \models \phi$,*

- *the cost C_∞ is minimized.*

Notice the similarity between the Problems 4.2.1 and 5.2.1. The difference, however, stems from the fact that the element of reasoning (timed state sequences, in this case), convey information about the exact time that the events happen. Hence, in the latter problem, this information can be specified via the formula, whereas in the former problem the reasoning problem is based, rather, on ordering of events. For this reason, MTL allows incorporating real time constraints including but not limited to deadlines and time windows, in a unifying manner.

The rest of this chapter is devoted to the introduction of a MILP-based algorithm for solving VRPMTL instances under some assumptions on the structure of the MTL formula ϕ.

5.3 A MILP-based Formulation of VRPMTL

In this section, a MILP based formulation of the VRPMTL is developed. The MILP formulation of the VRPMTL is in the same sprit with the that of the VRPLTL; first, the MTL formula is converted into a set of equivalent mixed-integer linear constraints and then the resulting constraints are incorporated into one of the MILP formulations of the standard VRP.

5.3.1 MILP Formulation of Atomic Propositions

Let us associate the following two variables to the evolution of an atomic proposition p: a binary variable ξ_p and a continuous variable τ_p. $\xi_p = 1$ if the atomic proposition p stays `False` forever; $\xi_p = 1$ if p becomes `True` at some point in time. Then, τ_p indicates the earliest time that p switches to `True`, if $\xi_p = 1$; it is meaningless if $\xi_p = 0$.

In the rest of this section, some examples of atomic propositions will be provided in the application domain. Consider, for example, the atomic proposition p, which states that "Target j is serviced by UAV k". In this case, the constraints on ξ_p and τ_p become

$$\xi_p = \sum_{i \in \mathcal{I}} x_{ijk}; \quad \tau_p = t_j.$$

Next, consider the atomic proposition p which states that "Target j is serviced". Notice that the parameters $\xi_p \in \{0,1\}$ and $\tau_p \in \mathbb{R}_{\geq 0}$ can be defined using the following linear constraints over the variables of the VRP formulation presented in Chapter 2.

$$\xi_p = \sum_{i \in \mathcal{I}} \sum_{k \in \mathcal{K}} x_{ijk}; \quad \tau_p = t_j.$$

Another example is the atomic proposition p, which states that "UAV k landed on landing site j". For this atomic proposition the linear constraints turn out to be

$$\xi_p = \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{C}} x_{ijk}; \quad \tau_p = t_{jk}.$$

Let us note the following final example. This time, the atomic proposition p states that "UAV k is launched from launch site i", for which the constraints on ξ_p and τ_p can be written as

$$\xi_p = \sum_{j \in \mathcal{J}} x_{ijk}; \tau_p = s_{ik}.$$

Notice that this proposition uses the loitering time at the launch site which corresponds to the launch time of the UAV.

5.3.2 MILP Formulation of MTL Formulae

This section is devoted to the introduction of a systematic procedure which constructs a set of linear inequalities over the variables ξ_{p_i} and τ_{p_i} for any given MTL formula ϕ under some assumption on the structure of ϕ. Given some timed state sequence $\omega = (\sigma, \kappa)$, where $\sigma = (s_1, s_2, \dots)$ and $\kappa = (I_1, I_2, \dots)$, let $\xi_p = 1$ and $\tau_{p_i} \leq t$ if $s(t)[p_i] = \texttt{True}$, and $\xi_p = 0$ if $s(t)[p_i] = \texttt{False}$, for $\forall p_i \in \Pi$. Then, for any given timed state sequence ω, the aforementioned set of linear constraints are satisfied if and only if $\omega \models \phi$, where ϕ satisfies the following assumption.

Assumption 5.3.1 *The temporal operators of MTL are applied only to atomic propositions and their negations.*

Even though this assumption seems constraining, many interesting complex temporal specifications already satisfy this assumption.

The algorithm, which generates the set of MILP constraints from the MTL specification, runs in two phases. In the first phase, it eliminates all the temporal operators from the formula, whereas in the second phase it considers the remaining non-temporal formula to construct the MILP constraints.

More precisely, in the first phase of the algorithm, for each temporal operator and the propositions it binds, a set of constraints is generated. Noting that, by Assumption 5.3.1, each temporal operator is either applied to an atomic proposition p or its negation $\neg p$, this set of constraints can be defined considering the four cases of the binary operator *until*, and two cases of the unary operators *eventually* and *always*. Let us first consider the *until* operator. Let $p_1, p_2 \in \Pi$ be the set of atomic propositions with their corresponding variables ξ_{p_1}, τ_{p_1} and ξ_{p_1}, τ_{p_1}, respectively. Then, $p_1 \mathcal{U}_I p_2$ with $I = [a, b]$ is satisfied at time $t = 0$ if and only if $\left(\xi_{p_2} = 1, \tau_{p_2} \leq b\right) \wedge \left(\left(\tau_{p_2} \leq a\right) \vee \left(\xi_{p_1} = 1, \tau_{p_1} < \tau_{p_2}\right)\right)$. For $\neg p_1 \mathcal{U}_I p_2$, the constraints are $\left(\xi_{p_2} = 1, \tau_{p_2} \leq b\right) \wedge \left(\left(\tau_{p_2} \leq a\right) \vee \left(\tau_{p_2} < \tau_{p_1}\right)\right)$. For the cases $p_1 \mathcal{U}_I \neg p_2$ and $\neg p_1 \mathcal{U}_I \neg p_2$ the only constraint is $\tau_{p_2} \geq a$, which is same for both since both force p_2 to be `False` throughout the interval.

Given the interval $I = [a, b]$, the constraints corresponding to the formula $\Diamond_I p_1$ are $\xi_{p_1} = 1, \tau_{p_1} \leq b$, whereas the ones corresponding to the formula $\Diamond_I \neg p_1$ are $\tau_{p_1} > a$.

Considering the always operator, the constraints corresponding to the formula $\Box_I p_1$ are $\xi_{p_1} = 1, \tau_{p_1} \leq a$, and the constraints corresponding to the formula $\Box_I \neg p_1$ are $\tau_{p_1} \geq b$.

Every constraint above is of the form $a_j z \leq b_j$. In the next step, each such constraint is bound with a binary variable $y_j \in \{0, 1\}$ which satisfies $y_j \leq 1 -$

$\frac{1}{\bar{M}}\left(a_j z - b_j\right)$; $y_j \geq -\frac{1}{\bar{M}}\left(a_j z - b_j\right)$, where $\bar{M}$ is a big enough number. Notice that $y_j = 1$ if $a_j z \leq b_j$ holds and $y_j = 0$ otherwise.

In the second phase, all the temporal subformulae are replaced with the corresponding binary variables y_j in the formula ϕ, and we apply the following rules recursively to the remaining formula ϕ, composed of negations, conjunctions, and disjunctions of y_i. Initially, $l = m + 1$ and then,

- for any negation of a single variable y_j a slack variable $y_l \in \mathbb{R}$, $0 \leq y_l \leq 1$ is defined, which satisfies $y_l = 1 - y_j$;
- for any conjunction of m single variables $y_1, \ldots, y_J$, a slack variable $y_l \in \mathbb{R}$, $0 \leq y \leq 1$ is defined which satisfies $y_l \leq y_j$, $j \in \{1, \ldots, J\}$; $y_l \geq \sum_{j=1}^{J} y_j - (J-1)$;
- for any disjunction of m single variables $y_1, \ldots, y_J$, a slack variable $y_l \in \mathbb{R}$, $0 \leq y \leq 1$ is defined which satisfies $y_l \leq \sum_{j=1}^{J} y_j$; $y_l \geq y_j$, $j \in \{1, \ldots, J\}$.

Then, at the end of each step, the variable y_l is substituted in the formula instead of variables and the operator it represents; and l is incremented by one. This recursive procedure is continued until ϕ is becomes a single variable, as in $\phi = y_L$. Finally, the following constraint is added to the formulation: $y_L = 1$, which states that the formula is True at the initial time, i.e., for any feasible solution of the VRP instance the corresponding timed state sequence satisfies $\omega^0 \models \phi$.

5.4 Multi-UAV Mission Planning Applications

In this section we present a simple example in order to illustrate the framework presented in the paper. Let us consider a scenario with three UAVs, one launch site, two landing sites, and five targets. The spatial distribution of the targets and the sites are shown in Figure 5-1. The three vehicles K_1, K_2, and K_3 can travel at 15, 18, and 20 mph respectively. The servicing time of each of the targets is 0.05 hours. When calculating the time required to from a node to another we consider rectilinear distances which is assumed to be a good model for urban environments [60].

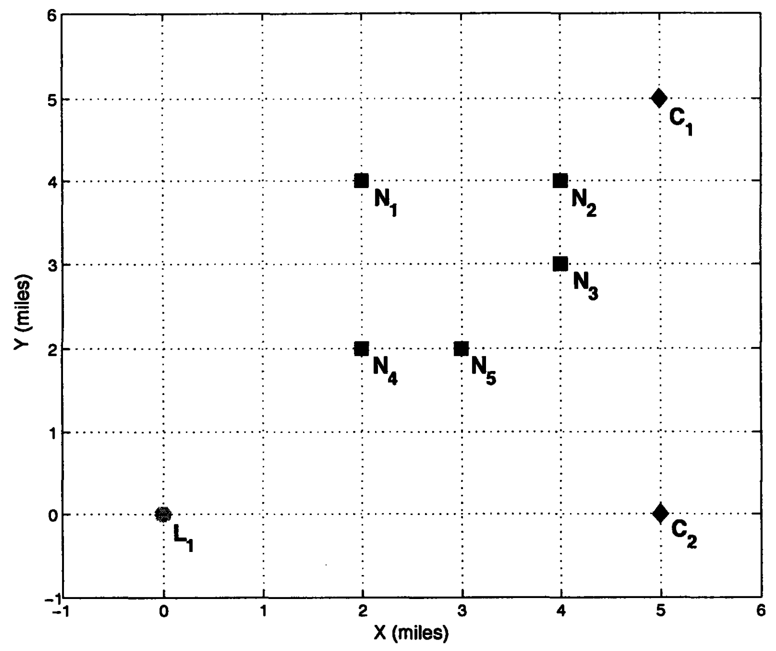


Figure 5-1: Map of the example mission planning scenario.

The objective in this mission is to either service targets N_1, N_4, N_5 within the first 1.5 hours, or to service targets N_2, N_3 within 0.7 hours. If the former option is taken, then eventually K_1 must land on the landing site C_2, after target N_4 is serviced, even if it does not service any of the targets. Furthermore, target N_4 can only be serviced after 0.4 from the beginning of the mission only by the UAV K_2. If the latter option is taken, servicing of the second target must always be avoided in the first 0.6 hours. Moreover, target N_3 must be serviced before target N_2. To model this problem in MTL, let us define the following atomic propositions,

- p_1: Target N_1 is serviced;
- p_2: Target N_2 is serviced by UAV K_2;
- p_3: Target N_3 is serviced by UAV K_3;
- p_4: Target N_4 is serviced by UAV K_2;
- p_5: Target N_5 is serviced;
- p_6: UAV K_1 is on landing site C_2,

using which the specification can be written as

$$\begin{aligned}\phi = \ & (\Diamond_{[0,1.5]}p_1 \wedge \Diamond_{[0,1.5]}p_4 \wedge \Diamond_{[0,1.5]}p_5 \\ & \wedge \Diamond_{[0,1.5]}p_6 \wedge \Box_{[0,0.4]}\neg p_4 \wedge (\neg p_6)\mathcal{U} p_4) \\ & \vee(\Diamond_{[0,0.7]}p_2 \wedge (\neg p_2)\mathcal{U}_{[0,0.7]}p_3 \wedge \Box_{[0,0.6]}\neg p_2)\end{aligned}$$

The optimal solution for the mission is shown in Figure 5-2. Notice that K_2 services the targets N_1, N_2, and N_3 and K_1 directly flies to landing site C_2, even though it does not service any of the targets. The landing times of K_1 and K_2 are 0.4334 and 0.8667 hours, and the servicing times of targets N_5, N_4, N_1 are 0.3278, 0.4334, and 0.5945, respectively. K_1 loiters around the launch site L_1 before starting its route for 0.0501 hours. Notice that the solution also satisfies the constraints that the target N_4 is not serviced within the first 0.4 hours of the mission. Notice also that the optimal solution makes UAV K_1 loiter for a while at the launch site in order to delay its arrival to C_2 and satisfy the constraint that UAV V_1 is not landed on C_2 before target N_4 is serviced.

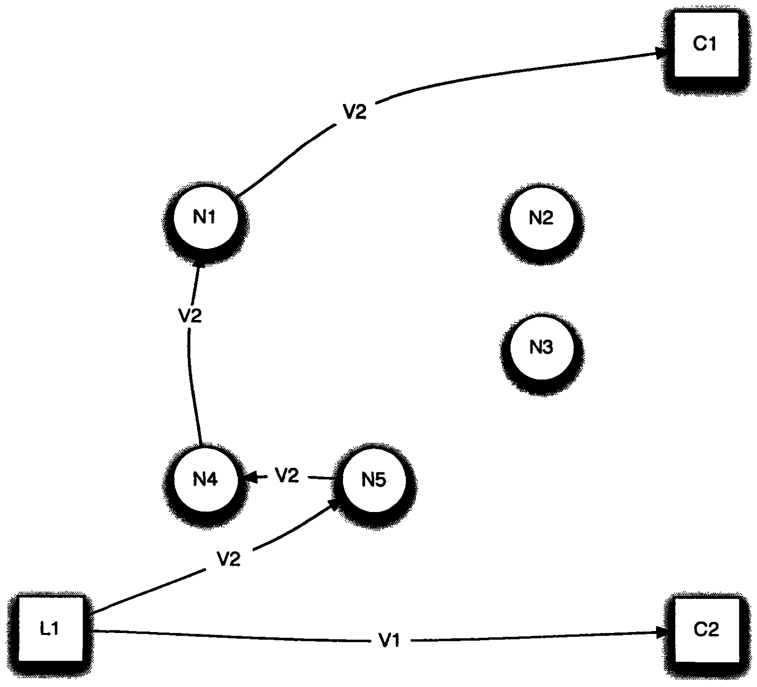


Figure 5-2: Optimal scheduling for MTL specification ϕ

As a slightly different example let us consider a scenario in which VRPMTL instance is the same except that the MTL specification changed to ϕ' which differs from ϕ by requiring that N_4 can not serviced within the first 0.5 hours of the mission instead of 0.4 hours. The corresponding optimal solution of the mission is shown in Figure 5-3. Landing times of UAVs K_2 and K_3 are 0.7611 and 0.6, the servicing times of the targets N_2 and N_3 are 0.6 and 0.4, respectively. In this solution UAV K_2 loiters at the launch location L_1 for 0.1056 hours before starting the mission.

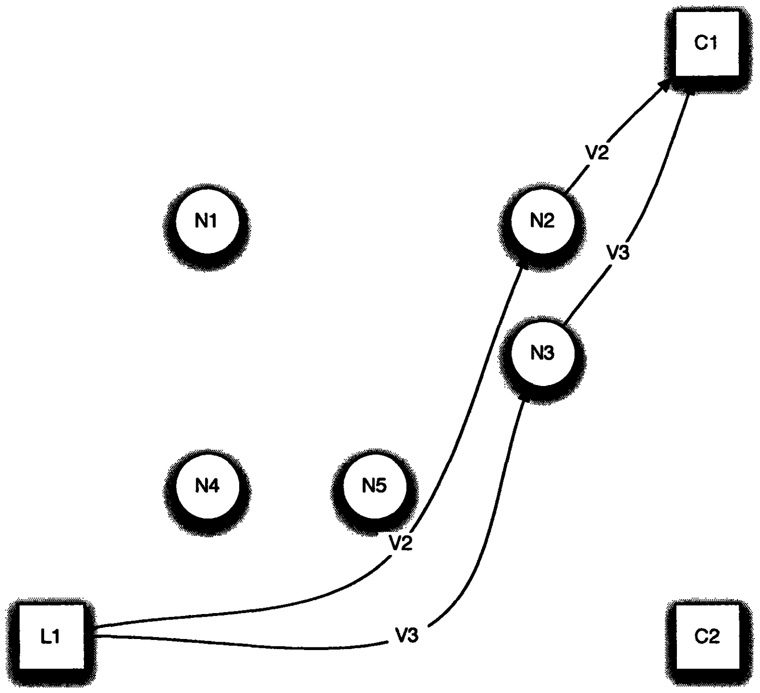


Figure 5-3: Optimal scheduling for MTL specification ϕ'

Chapter 6

Process Algebra

The second part of the thesis addresses problems in which the specification is given by a process algebra term instead of a temporal logic formula. The main motivation behind studying process algebra is its nice computational properties: A feasible execution that satisfies a given process algebra term can be found in time polynomial with respect to the size of the term, whereas the model checking problem for, e.g., LTL is known to be PSPACE-complete [53].

In this chapter, several definitions that introduce process algebra are provided. These definitions constitute preliminaries for the material that will be presented in the next chapter. Subsequently, in the next chapter, the motivation for employing process algebra specifications in multiple-UAV mission planning problems is further discussed.

This chapter is organized as follows. In Section 6.1, some preliminaries are introduced; these definitions will be used throughout Chapters 6 and 7. The syntax and semantics of process algebra are introduced in Sections 6.2 and 6.3, respectively.

6.1 Preliminaries on Sequences and Trees

This section presents preliminary definitions on sequences, graphs, and trees, in subsequent sections. Note that sequences and graphs were defined previously as necessary. In this section, those definitions are reminded and extended.

6.1.1 Sequences

Given a set S, a (finite) sequence σ of (distinct) elements from S is an injective function, which maps $\{1, 2, \ldots, K\}$ to S, where $K \in \mathbb{N}$. The number K, denoted by $|\sigma|$, is called size of the sequence σ. The empty sequence has size 0 and is denoted by Λ. For notational convenience, σ will be denoted as $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(K))$.

An element $s \in S$ is said to appear in the sequence σ, denoted by $s \in \sigma$ with an abuse of notation, if there exists some k such that $\sigma(k) = s$. Element s_1 is said to precede element s_2 in σ, denoted by $s_1 \prec_\sigma s_2$, if $s_1, s_2 \in \sigma$ and $\sigma^{-1}(s_1) < \sigma^{-1}(s_2)$. Given a sequence $\sigma = (\sigma(1), \ldots, \sigma(K))$, any sequence $\tilde{\sigma} = (\sigma(1), \ldots, \sigma(l))$, where $l < K$, is called a prefix of σ. Given two sequences $\sigma_1 = (\sigma_1(1), \sigma_1(2), \ldots, \sigma_1(K))$ and $\sigma_2 = (\sigma_2(1), \sigma_2(2), \ldots, \sigma_2(K'))$, their concatenation $\sigma' = (\sigma_1(1), \ldots, \sigma_1(K), \sigma_2(1), \ldots, \sigma_2(K'))$ is denoted by $\sigma_1 \bullet \sigma_2$. Given a set S, the set of all sequences that can be defined on S, which includes Λ, is denoted by Σ_S.

6.1.2 Graphs

A graph $G = (N, E)$ consists of a set N of nodes and a set $E \subseteq N \times N$ of edges. The edge e is said to be an incoming (or outgoing) edge of a node n if $e = (n', n)$ (or $e = (n, n')$) for some $n' \in N$. Two edges $e, e' \in E$ are said to be adjacent if $e = (n, n')$ and $e' = (n', n'')$ for some $n, n', n'' \in N$. A path on G is a sequence $p = (e_1, e_2, \ldots, e_l)$ of edges such that $e_i \in E$ for all $i \in \{1, 2, \ldots, l\}$ and e_i and e_{i+1} are adjacent, for all $i \in \{1, 2, \ldots, l-1\}$. The graph G is said to include a path p, if p is a path on G. The path $p = (e_1, e_2, \ldots, e_l)$ is said to start at node n_s and end at node n_s if $e_1 = (n_s, n')$ and $e_l = (n'', n_e)$ for some $n', n'' \in N$. A path is called a cycle if it starts and ends at the same node.

6.1.3 Trees

A tree $T = (N, E)$ is a graph that does not include any cycles. It can be shown that every tree has a node, called the root node, that has no incoming edges. Given a tree T, the unique root node of T will be denoted by $\mathtt{Root}(T)$. It can also be shown that all other nodes in the tree have exactly one incoming edge each. Let n be a node in tree other than the root. Let $e = (n', n)$ be the unique incoming edge of n. The node n' is called the parent of n and will be denoted by $\texttt{Parent}_T(n)$. Let $e_1 = (n, n_1), e_2 = (n, n_2), \ldots, e_l = (n_l, n)$ be all the outgoing edges of n. Then, the set $\{n_1, n_2, \ldots, n_l\}$, which will be denoted as $\texttt{Children}_T(n)$, is the set of children of n. If a node n has no outgoing edge, then it is called a leaf node. The function $\texttt{Leaf}_T(n)$ is used to decide this case; it returns `True` if n is a leaf node and `False` otherwise.

A labeled tree is a tree $T = (N, E)$, in which each node in N is associated with a label from an alphabet Σ_N and each edge is associated with a label from another alphabet Σ_E. Given a node n (or an edge e), its label will be denoted by $\texttt{Label}_T(n)$ (or $\texttt{Label}_T(e)$).

6.1.4 Binary Trees

A binary tree $B = (N, E)$ is a tree, each node of which has either no outgoing edge or exactly two outgoing edges. If a node has two outgoing edges, then exactly one of them is labeled as *right* and the other one as *left*. Let n be a node and $e_l = (n, n_l)$ and $e_r = (n, n_r)$ be its outgoing edges labeled with left and right, respectively. Then, n_l is called the left child and n_r is called the right child of n, denoted by $\texttt{LeftChild}_B(n)$ and $\texttt{RightChild}_B(n)$.

A labeled binary tree is a binary tree $B = (N, E)$, in which each node in N is associated with a label from an alphabet Σ_N. Given a node n, its label will be denoted by $\texttt{Label}_B(n)$.

These definitions will be useful throughout the paper. In particular, a labeled tree structure, called the assignment tree, will be introduced in the next chapter and it will be denoted by $\mathcal{T} = (\mathcal{N}, \mathcal{E})$; a labeled binary tree structure, denoted by $\mathcal{B} = (\mathcal{M}, \mathcal{F})$, called the parse tree, will be presented in Section 6.4.

6.2 Syntax of Process Algebra

The syntax of *Process Algebra* (PA) is defined by a *set of terms*. More formally,

Definition 6.2.1 (Terms of Process Algebra) *Given a finite set A of actions, the set $\mathbb{T}$ of BPA terms is the least set that satisfies the following:*

- *each action $a \in A$ is in $\mathbb{T}$;*
- *if $p, p' \in \mathbb{T}$, then $p + p' \in \mathbb{T}$, $p \cdot p' \in \mathbb{T}$, and $p \,||\, p' \in \mathbb{T}$.*

Each element of $\mathbb{T}$ is called a PA term.

The compositions $p+p'$, $p \cdot p'$, and $p \,||\, p'$ of processes p and p' are called the alternative, sequential, and parallel compositions, respectively. Intuitively, the process that is composed only of the term $a \in \mathbb{T}$, where $a \in A$, can execute a and then terminate. The process $p + p'$, behaves either like the process p or process p', i.e., $p + p'$ executes either a behavior of p or one of p', which introduces a logical coupling between the actions. The process $p \cdot p'$, on the other hand, first executes a behavior of process p and then, after p terminates, executes a behavior of p'; the process $p \cdot p'$ terminates when p' terminates. Finally, $p \,||\, p'$ executes the actions of p and p' in an interleaving manner and terminates when they both terminate. A process which has terminated and can execute no further actions will be denoted by $\surd$.

The size of a term is defined as the number of actions and operators in it. Without loss of any generality, we assume that each action appears at most once in a term.

6.3 Semantics of Process Algebra

In process algebra, each term represents a process, and thus a set of behaviors. A behavior is formalized as a sequence of actions, referred to as a *trace*, that a process can execute. Indeed, each process can be associated with a set of traces, which describes all its behaviors, i.e., all the traces that the process can execute. Unlike temporal logic formulae, semantics of process algebra terms are not defined using transition system models. Instead, a function that maps each one of the processes in $\mathbb{T}$ to a set of traces that it can execute is defined using a *process graph.*

Definition 6.3.1 (Process Graph) *A* process graph $p_0 \in \mathbb{T}$ *is a labeled transition system* $(Q, A, \Rightarrow)$ *together with a root state* $q_0 \in Q$ *and a function* $\pi : Q \rightarrow \mathbb{T}$*, where*

- Q *is a set of states;*
- A *is a finite set of actions;*
- $\Rightarrow \subset Q \times A \times Q$ *is a ternary relation;*
- π *associates each state* $q \in Q$ *with a term* $t \in \mathbb{T}$ *such that* q_0 *is mapped to* p_0.

On the process graph of a given process, a transition from a state q_1 to another state q_2 with action $a \in A$, denoted by $q_1 \overset{a}{\Longrightarrow} q_2$, exists if and only if $(q_1, a, q_2) \in \Rightarrow$. If there is a transition $q_1 \overset{a}{\Longrightarrow} q_2$ such that $\pi(q_1) = p_1$, $\pi(q_2) = p_2$, then p_1 is said to evolve to p_2 after executing action a, denoted as $p_1 \xrightarrow{a} p_2$. If there is a set of actions $a_1, \ldots, a_k \in A$ and a set of processes $p_0, \ldots, p_k \in \mathbb{T}$ such that $p_{i-1} \xrightarrow{a_i} p_i$ for all $i \in \{1, \ldots, k\}$, then process p_0 is said to evolve to process p_k after executing the sequence $(a_1, a_2, \ldots, a_k)$ of actions. The corresponding sequence of transitions is denoted as $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \ldots \xrightarrow{a_k} p_k$, or equivalently, $p_0 \xrightarrow{(a_1, a_2, \ldots, a_k)} p_k$.

Definition 6.3.2 (Trace) *A* trace *of a PA term* p_1 *is a sequence* $\gamma = (a_1, a_2, \ldots, a_n)$ *of actions, for which a corresponding sequence* $(p_2, p_3, \ldots, p_{n+1})$ *of processes exists such that* $p_i \in \mathbb{T}$ *and* $p_i \xrightarrow{a_i} p_{i+1}$ *for all* $i \in \{1, 2, \ldots, n\}$. *The set of all traces of a term* p *is denoted as* Γ_p.

Note that if the process graph of a process p is known, then the set Γ_p is well-defined. Thus, the next step is to define the process graph for each process. Obviously, given a finite set A of actions, the process graph for all the processes in $\mathbb{T}$ can be defined by giving the labeled transition system $(Q, A, \rightarrow)$, root state q_0, and function π, explicitly, for all $p \in \mathbb{T}$. Instead of this exhaustive approach, however, *operational semantics* can be used to define the process graph of each process with a finite set of *transition rules*.

A transition rule is defined as follows.

Definition 6.3.3 (Transition Rule) *A transition rule ρ is an expression $\frac{H}{\pi}$, where H is a set of transitions $p \xrightarrow{a} p'$, called the* positive premisses *of ρ, and π is a transition $p \xrightarrow{a} p'$, called the conclusion of ρ, where $p, p' \in \mathbb{T}$.*

Intuitively, if the set H of premisses are possible transitions, then so is π. A set of transition rules will be referred to as a *transition system specification*. The transition system specification of PA is given as follows:

Definition 6.3.4 (Operational Semantics of PA) *The operational semantics of the process algebra is given by the following transition system specification:*

$$\frac{}{a \xrightarrow{a} \surd}$$

$$\frac{p_1 \xrightarrow{a} \surd}{p_1 + p_2 \xrightarrow{a} \surd} \quad \frac{p_1 \xrightarrow{a} p_1'}{p_1 + p_2 \xrightarrow{a} p_1'} \quad \frac{p_2 \xrightarrow{a} \surd}{p_1 + p_2 \xrightarrow{a} \surd} \quad \frac{p_2 \xrightarrow{a} p_2'}{p_1 + p_2 \xrightarrow{a} p_2'}$$

$$\frac{p_1 \xrightarrow{a} \surd}{p_1 \cdot p_2 \xrightarrow{a} p_2} \quad \frac{p_1 \xrightarrow{a} p_1'}{p_1 \cdot p_2 \xrightarrow{a} p_1' \cdot p_2}$$

$$\frac{p_1 \xrightarrow{a} \surd}{p_1 \| p_2 \xrightarrow{a} p_2} \quad \frac{p_1 \xrightarrow{a} p_1'}{p_1 \| p_2 \xrightarrow{a} p_1' \| p_2} \quad \frac{p_2 \xrightarrow{a} \surd}{p_1 \| p_2 \xrightarrow{a} p_1} \quad \frac{p_2 \xrightarrow{a} p_2'}{p_1 \| p_2 \xrightarrow{a} p_1 \| p_2'}$$

where $a \in A$ and $p_1, p_1', p_2, p_2' \in \mathbb{T}$

The definition above yields a process graph for each process in $\mathbb{T}$. By the first rule, the process $a \in \mathbb{T}$ where a is an action, i.e., $a \in A$, can only execute the action a and then terminate. The rest of the rules obey the intuition presented after the syntax of process algebra.

6.4 Parse Trees of Process Algebra Terms

Recall that each process algebra term is composed of other terms, called the *subterms*. The parse tree of a process algebra term p defined on a set A of actions is a labeled binary tree, denoted as $\mathcal{B}_p = (\mathcal{M}_p, \mathcal{F}_p)$, in which the nodes are labeled with the alphabet $\{+, \cdot, \|\} \cup A$. The labels are assigned such that only the leaf nodes are labeled with actions from A, and all other nodes are labeled with operators from set $\{+, \cdot, \|\}$. More formally, the parse tree of p is defined recursively as follows:

- if $p = a$, then $\mathcal{B}_p = (\mathcal{M}_p, \mathcal{F}_p)$ is such that $\mathcal{M}_p = \{n\}$ and $\mathcal{F}_p = \emptyset$ with $\texttt{Label}_{\mathcal{B}_p}(n) = a$.
- if $p = p_l \diamond p_r$ with $\diamond \in \{+, \cdot, \|\}$, then $\mathcal{B}_p = (\mathcal{M}_p, \mathcal{F}_p)$ is constructed as follows. Let $\mathcal{B}_{p_l} = (\mathcal{M}_{p_l}, \mathcal{F}_{p_l})$ and $\mathcal{B}_{p_r} = (\mathcal{M}_{p_r}, \mathcal{F}_{p_r})$ be the parse trees of p_l and p_r, respectively. Then, $\mathcal{M}_p = \{n\} \cup \mathcal{M}_{p_l} \cup \mathcal{M}_{p_r}$ and $\mathcal{F}_p = \{e_l, e_r\} \cup \mathcal{F}_{p_l} \cup \mathcal{F}_{p_r}$, where $\texttt{Label}_{\mathcal{B}_p}(n) = \diamond$, $e_l = (n, n_l)$ and $e_r = (n, n_r)$ are the left and the right outgoing edges of n, respectively, with $n_l = \texttt{Root}(\mathcal{B}_{p_l})$ and $n_r = \texttt{Root}(\mathcal{B}_{p_r})$.

For an illustration, consider the following example.

Example Consider, for example, the term $p = (a + (b \cdot c))\|d$, where $a, b, c, d \in A$. The corresponding parse tree is given in Figure 6-1. As more easily seen from the parse tree, $p_1 = (a + (b \cdot c))$ and $p_2 = d$ are subterms of p, and are bound using the operator $\|$. Similarly, $p3 = a$ and $p_4 = (b \cdot d)$ are subterms of p_1, and $p_5 = b$ and $p_6 = d$ are subterms of p_4. ▮

Given a node n in the parse tree $\mathcal{B}_p$ of a process algebra term p, let N_l be the set of all leaf nodes, to which there exists a path on $\mathcal{B}_p$ starting from n. Noting that each leaf is labeled with an action, the set $\{\texttt{Label}(n') \mid n' \in N_l\}$ of labels will be denoted as $\texttt{ChildrenActions}_{\mathcal{B}_p}(n)$.

Several algorithms that operate on the parse tree of a given process algebra term will be introduced in the next chapter, where these definitions will prove to be useful.

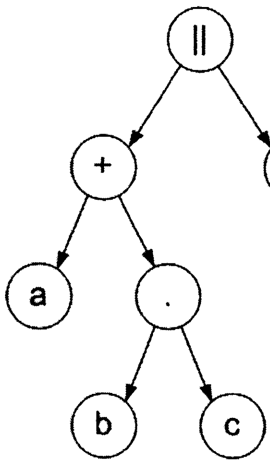


Figure 6-1: The parse tree of the term $(a + (b \cdot c))||d$.

Chapter 7

Vehicle Routing with Process Algebra Specifications

In the previous chapter, the syntax and the semantics of process algebra were introduced. This chapter is devoted to the definition of a vehicle routing problem, where the specification is given a process algebra term. Unlike the first part of the thesis, the presentation of the material in this chapter is oriented more towards the multiple-UAV mission planning application domain.

This chapter is organized as follows. In Section 7.1, several definitions presented in the previous section are adapted to multiple-UAV mission planning problems. Then, in Section 7.2, a formal definition of the problem is provided. An algorithm that solves the problem to optimality is presented in Section 7.3 and an example scenario is investigated in Section 7.4

7.1 Objectives and Specificaitons

Recall the definitions related to process algebra presented in the previous chapter. In this chapter, several definitions that were presented in the previous chapter are utilized to reason about temporal and logical constraints in vehicle routing problems, or, more specifically, its application in solving multiple-UAV mission planning problems. Accordingly, the actions defined in the previous chapter will refer to the individual tasks such as classifying a target, a sector search, or a reconnaissance mission, that are performed by exactly one UAV, in this chapter. Each mission plan will be associated with a behavior, i.e., a temporal ordering of individual tasks. Finally, each process will denote a mission specification. Of course, the actions in UAV missions are not necessarily done instantaneously as they may have a start time and a completion time as in an area search task. In the next section, the necessary theory to introduce a well-defined problem definition will be given. This section presents preliminaries and a couple of examples demonstrating multiple-UAV mission planning problems specified with process algebra. For this purpose, in this section, the aforementioned individual tasks are formalized under the name of atomic objectives, first; then, atomic objectives are combined into represent more complex objectives in the mission using process algebra.

7.1.1 Atomic Objectives

An atomic objective is, in essence, an abstraction of a generic individual task [48]. Each atomic objective is associated with two spatial parameters and one temporal parameter: the initial and final points, and the duration of the task, respectively. Intuitively, the initial point is a coordinate that the UAV has to get to in order to start executing the task. Similarly, the final point refers to the coordinate that the UAV gets to after completing the task. The duration, on the other hand, is the execution time of the task. Finally, each atomic objective is associated with exactly one UAV, which is capable of executing the task[1]. Formally, an atomic objective o refers to the tuple

$$\left(x_o^i, x_o^f T_o^e, U_o\right),$$

where x_o^i, x_o^f are the initial and final coordinates, $T_o^e \in \mathbb{R}_{\geq 0}$ is the duration, and $U_o \in \mathcal{U}$ is a UAV.

From a practical point of view, the initial and final points may vary depending

[1]In practice, there might be a number of UAVs that can accomplish a given atomic objective. Yet, for reasons to be clear in the next section, assuming that a single vehicle can execute a given task does not incur any loss of generality.

on the current position of the UAV. Moreover, the atomic objectives can be terminal action in the sense that the UAV executing it may not be able to execute any other task. The approach presented here can easily be extended to cover all these cases.

Examples of atomic objectives include the classification, attack, and verification tasks as well as object tracking, reconnaissance, area search, and communications relay (cf. [48]).

7.1.2 Complex Objectives

A formal definition to be provided in the next section, this section informally introduces a specification and points out a hierarchical methodology for constructing the mission specification, through two small examples.

The Target Engagement Scenario

Consider a target engagement scenario with n targets and 2 UAVs, where each target has to be (i) classified, (ii) attacked, and (iii) verified for damage inflicted, in this order. The problem is to assign these tasks to UAVs in such a way that (i) all three tasks are executed on each target and (ii) the time that the mission is completed is minimized.

Let $o_{u,t,c}$, $o_{u,t,a}$, and $o_{u,t,v}$ be the atomic objectives for classify, attack, and verify tasks on Target t performed by UAV u. Then, the set of atomic objectives O is identified as

$$O = \Big\{o_{u_1,t_1,c}, o_{u_1,t_1,a}, o_{u_1,t_1,v}, o_{u_2,t_1,c}, o_{u_2,t_1,a}, o_{u_2,t_1,v} \dots,$$
$$o_{u_1,t_n,c}, o_{u_1,t_n,a}, o_{u_1,t_n,v}, o_{u_2,t_n,c}, o_{u_2,t_n,a}, o_{u_2,t_n,v}\Big\}.$$

Even though this scenario is simple enough to write the specification immediately from the atomic objectives, let me consider two middle layers. In the first layer let me define the complex objectives of classifying, attacking, and verifying the damage inflicted on target t_i as $o_{t_i,c}, o_{t_i,a}, o_{t_i,v}$, respectively. Using process algebra terms, $o_{t_i,c} = o_{u_1,t_i,c} + u_2 c_i$, $o_{t_i,c} = o_{u_1,t_i,a} + o_{u_2,t_i,a}$, and $o_{t_i,v} = o_{u_1,t_i,v} + o_{u_2,t_i,v}$. In the second layer, let the term $o_{t_i} = o_{t_i,c} \cdot o_{t_i,a} \cdot o_{t_i,v}$ denote engaging target t_i. Finally, the specification is

$$p = o_{t_1} \| o_{t_2} \| \dots \| o_{t_n}. \tag{7.1}$$

A hierarchically specified task can be modified quickly in different levels, as in similar hierarchical task networks type of specifications [21]. Consider, for example, the case in which there exists an attack option with a powerful UAV that does not require a verification of the damage inflicted. However, assume that this attack option applies to only some of the targets. Let me denote this attack option with $o_{u_3,t_i,\bar{a}}$, which is also an atomic objective. Introducing a new objective $o_{\bar{t}_i}$ such that the corresponding term is defined as $o_{\bar{t}_i} = o_{t_i,c} \cdot (o_{t_i,a} \cdot o_{t_i,v} + o_{t_i,\bar{a}})$, the specification p can be rearranged as: $p = o_{t_1} \| o_{t_2} \| \dots \| o_{t_k} \| o_{\bar{t}_{k+1}} \| \dots \| o_{\bar{t}_n}$.

The Rescue Mission

In this section a scenario similar to the one depicted in Chapter 4 is presented. Recall that in this scenario, a friendly infantry unit is pinned down by enemy in a certain location. There are three targets, T1, T2, and T3, that needs to be neutralized for the friendly unit be rescued. Two of the targets, T1 and T2, are protected by missile sites, S1 and S2, respectively. There are two friendly bases: Base B1 and Base B2. There are three types of friendly UAVs: the first type, V1, can only engage the targets, the second type, V2, can only engage the missile sites, and the third one, V3, can engage any of them, but it is a relatively low-speed asset. A map of this scenario can be seen in Figure 5-1. In order to successfully accomplish this mission, the infantry unit can either escape to Base B1, by the targets T1 and T3 being neutralized, or to Base B2 after the UAVs neutralize targets T2 and T3. Moreover, if the latter option is chosen a V3 type UAV has to head to B2 with necessary health equipment, even if it does not engage any target along the way.

Let p be the main objective, i.e., the mission specification, which states that the infantry unit must be rescued by either escaping to Base B1 or Base B2, i.e., $p = o_{B1} + o_{B2}$, where o_{B1} and o_{B2} are complex objectives. Noting that both targets $T1$ and $T3$ have to be engaged in order to accomplish the objective o_{B1}, one can write $o_{B1} = o_{T1} \| o_{T3}$, where o_{T1} and o_{T3} denote the objectives of neutralizing targets T1 and T3 respectively. Similarly, $o_{B2} = o_{T2} \| o_{T3} \| o_{V3@B2}$, where o_{T2} is the objective of neutralizing Target T2 and $o_{V3@B2}$ is the objective of moving a V3 type vehicle to Base B2 with the necessary health equipment. Even though $o_{V3@B2}$ is an atomic objective, notice that o_{T1}, o_{T2}, and o_{T3} are still complex objectives that need to be identified in terms of atomic objectives. As noted before there are two possible ways of neutralizing Target $T1$: (i) either the missile site $S1$ can be attacked by a vehicle of type $V2$ after the neutralization of $T1$ by a vehicle of type $V1$, (ii) or $T1$ can be neutralized directly using a vehicle of type $V3$. Hence, $o_{T1} = (o_{V2@S1} \cdot o_{V1@T1}) + o_{V3@T1}$. Similarly, $o_{T2} = (o_{V2@S2} \cdot o_{V1@T2}) + o_{V3@T2}$. Noting that Target T3 is not protected by any missile sites, it is the case that $o_{T3} = o_{V1@T3} + o_{V3@T3}$. To summarize:

$$\begin{aligned} p &= o_{B1} + o_{B2} = (o_{T1} \| o_{T3}) + (o_{T2} \| o_{T3} \| o_{V3@B2}) \\ &= ((o_{V2@S1} \cdot o_{V1@T1}) + o_{V3@T1} \| o_{V1@T3} + o_{V3@T3}) \\ &+((o_{V2@S2} \cdot o_{V1@T2}) + o_{V3@T2} \| o_{V1@T3} + o_{V3@T3} \| o_{V3@B2}). \end{aligned} \tag{7.2}$$

7.2 Problem Definition

The previous section provided a brief introduction to specification of multiple-UAV missions using process algebra terms. This section first formalizes the definition of specification and then presents the problem definition.

Let me fix a set $\mathcal{U}$ of UAVs and a set O of atomic objectives. The problem parameters consist of the time required to execute the atomic objective o, denoted by $T^e_o \in \mathbb{R}_{\geq 0}$, and the time required for a UAV u to travel from the exit point of atomic objective o_1 to the entry of atomic objective o_2, denoted by $T^t_{u,o_1,o_2} \in \mathbb{R}_{\geq 0}$, for all $u \in \mathcal{U}$ and all $o, o_1, o_2 \in O$.

The mission planning problem to be outlined shortly asks to schedule a subset of the atomic objectives to satisfy a given specification. Hence, the solution to the problem assigns a set of atomic objectives together with their respective execution times to each UAV. This assignment is formalized in the definitions to follow.

Definition 7.2.1 (Single-UAV Schedule) *A* single-UAV schedule σ_u *for UAV* $u \in \mathcal{U}$ *is a sequence of pairs* (τ, o)*, where* $\tau \in \mathbb{R}_{\geq 0}$ *denotes an absolute time,* $o \in O$ *is an atomic objective such that* σ_u *satisfies the following constraint:*

- *for all* $(o, \tau) \in \sigma_u$ *with* $o = (x^i_o, x^f_o, T^e_o, U_o)$*, there holds* $U_o = u$*,*
- *for all* $l \in \{1, \dots, |\sigma_u| - 1\}$*, there holds* $\tau_l + T^e_{o_l} + T^t_{i,o_l,o_{l+1}} \leq \tau_{l+1}$*, where* $\sigma_u(l) = (\tau_l, o_l)$ *and* $\sigma_u(l+1) = (\tau_{l+1}, o_{l+1})$*.*

Intuitively, the first condition on σ_u states that UAV u can be assigned only those atomic objectives that it is capable of executing; the second condition ensures that before executing an atomic objective o_{l+1}, UAV u must be given enough time to execute the previous atomic objective o_l in its schedule and travel to the entry point of o_{l+1}. A complete schedule is then formalized as follows.

Definition 7.2.2 (Complete Schedule) *A complete schedule is a set* $\mathcal{S}$ *of single-UAV schedules, which includes exactly one single-UAV schedule* σ_u *for each UAV* $u \in \mathcal{U}$*.*

An atomic objective is said to be scheduled in a complete schedule $\mathcal{S}$ if there exists a $\sigma_u \in \mathcal{S}$ and a $t \in \mathbb{R}_{\geq 0}$ such that (o, t) appears in σ_u.

To employ the techniques presented in Chapter 6, one would like to associate complete schedules with sequences of atomic objectives and check whether the corresponding sequences are traces of the specification. Formalizing this intuition, the following definition leads to a simple and intuitive way to define the behavior of a schedule.

Definition 7.2.3 (Observation) *Given a complete schedule* $\mathcal{S}$*, an* observation *of* $\mathcal{S}$ *is a sequence* $\pi = (o_1, o_2, \dots, o_l)$ *of atomic objectives which satisfies the following constraints:*

- o_i *appears in* π *if and only if it is scheduled in* $\mathcal{S}$*.*

- *there exists $\tau_1, \tau_2, \ldots, \tau_l \in \mathbb{R}_{\geq 0}$ such that (i) $\tau_i \leq \tau_{i+1}$ for all $i \in \{1, \ldots, l-1\}$, (ii) $t_i \leq \tau_i \leq t_i + T_i$, where t_i is the execution time of o_i, for all $i \in \{1, \ldots, l\}$ and (iii) $o_i \prec_\pi o_j$ implies $\tau_i \leq \tau_j$ for all $i, j \in \{1, 2, \ldots, l\}$.*

The set of all observations of $\mathcal{S}$ is denoted by $\Pi_{\mathcal{S}}$.

Hence, every schedule $\mathcal{S}$ is associated with a set $\Pi_{\mathcal{S}}$ of observations, which, in essence, describes the behavior of $\mathcal{S}$. More precisely, the observations $\pi \in \Pi_{\mathcal{S}}$ together represent the ordering of the atomic objectives in $\mathcal{S}$ without referring to the time interval of execution of the atomic objectives that are scheduled in $\mathcal{S}$. Importantly, observations make the following formal definition of the specification possible.

Definition 7.2.4 (Specification) *Given a set O of atomic objectives, a* specification *defined on O is a process algebra term p that is defined on O. A complete schedule $\mathcal{S}$ is said to satisfy a specification p if and only if every observation of $\mathcal{S}$ is a trace of p, i.e., $\Pi_{\mathcal{S}} \subseteq \Gamma_p$.*

In the following, these ideas are illustrated with an example.

Example Recall the target engagement scenario described in Section 7.1.2. Consider an instance of this scenario with two UAVs and two targets, each of which have to be classified, attacked, and verified in this order. Let $\mathcal{U} = \{u_1, u_2\}$ be the set of UAVs. Furthermore, let the atomic objectives $o_{u,t,c}$, $o_{u,t,a}$, and $o_{u,t,v}$, respectively, denote the classification, attack, and verification tasks on target t performed by UAV u. Regarding the parameters of the problem, let the execution times of the atomic objectives be $T^e_{u_1,j,k} = 3$, $T^e_{u_2,j,k} = 2$ for all $j \in O$, all $k \in \{c, a, v\}$ and the traveling times be $T^t_{u,0,o_{1,k,u}} = 6$, $T^t_{u,0,o_{2,k,u}} = 11$, $T^t_{u,o_{1,k,u},o_{2,l,u}} = T^t_{u,o_{2,k,u},o_{2,l,u}} = 4$ for all $k, l \in \{c, a, v\}$ and all $u \in \mathcal{U}$. An example complete schedule is $\mathcal{S} = \{\sigma_1, \sigma_2\}$, where

$$\sigma_{u_1} = ((11, o_{u_1,t_2,c}), (14, o_{u_1,t_2,a}))\,,$$

$$\sigma_{u_2} = ((6, o_{u_2,t_1,c}), (8, o_{u_2,t_1,a}), (10, o_{u_2,t_1,v}), (17, o_{u_2,t_2,v}))\,.$$

The complete schedule $\mathcal{S}$ is depicted in Figure 7-1 on a timeline. Notice that the observations of $\mathcal{S}$ are

$$\pi_1 = (o_{u_2,t_1,c}, o_{u_2,t_1,a}, o_{u_2,t_1,v}, o_{u_1,t_2,c}, o_{u_1,t_2,a}, o_{u_1,t_2,v}),$$

$$\pi_2 = (o_{u_2,t_1,c}, o_{u_2,t_1,a}, o_{u_1,t_2,c}, o_{u_2,t_1,v}, o_{u_1,t_2,a}, o_{u_1,t_2,v}),$$

both of which are traces of the specification of the target engagement scenario (cf. Equation (7.1)). ∎

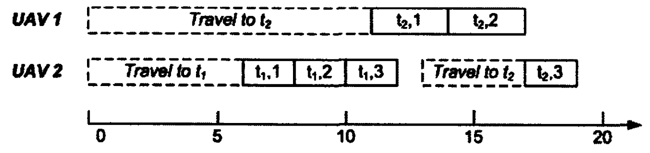


Figure 7-1: Example schedules. Pairs of tasks and targets indicate the task being executed at the corresponding time. The periods of traveling to the target position are also shown.

Given a single-UAV schedule $\sigma_u = ((o_1, t_1), \ldots, (o_l, t_l))$, let τ_{σ_u} denote the completion time of σ_u, i.e., $\tau_{\sigma_u} = t_l + T^e_{o_l}$, where $T^e_{o_l}$ is the duration of o_l. A complete schedule $\mathcal{S}$ can be associated with a cost in several ways, using the variables τ_{σ_u} for all $\sigma_u \in \mathcal{S}$. Two widely-used cost functions are

$$\sum_{\sigma_u \in \mathcal{S}} \tau_{\sigma_u}, \qquad \max_{\sigma_u \in \mathcal{S}} \tau_{\sigma_u}. \tag{7.3}$$

The former one can be interpreted as a type of risk, since it denotes the total amount time each vehicle was employed in the mission, whereas the latter one directly addresses the mission completion time. Minimizing the former cost function will enforce the optimal schedule to employ as few vehicles as possible and treat the mission completion time as a secondary objective. Minimizing the second cost function on the other hand will lead to a schedule that completes the mission as quickly as possible, perhaps by utilizing every possible UAV there exists. Another interesting cost function is obtained as a convex combination of the the two given in Equation (7.3) to model a combination of risk and mission time and represent the trade-off between the two with one parameter. Recall that similar cost functions appeared in Chapters 4 and 5. Such cost functions all share common properties, which will be formalized in the next section.

The Task Assignment Problem for Complex UAV Operations is formally defined as follows:

Problem 7.2.5 *Given a set $\mathcal{U}$ of vehicles, a set O of atomic objectives, and a specification p, the is to find a plan $\mathcal{S}$ such that:*

- *$\mathcal{S}$ satisfies p,*
- *the cost of $\mathcal{S}$, given by (7.3), is minimized.*

In the next section, a state-space search algorithm that solves Problem 7.2.5 will be presented.

7.3 Tree Search Based Optimal Planning Algorithm

This section is devoted to the presentation of an algorithm that solves Problem 7.2.5. Inspired by the tree-search algorithm in [49], the algorithm presented in this section constructs a minimum cost complete schedule by searching a finite tree structure. A more formal definition to follow shortly, for any given specification p defined on a set O of atomic objectives, there exists a labeled tree such that each node is labeled with a process algebra term that p can evolve to and each edge is labeled with an atomic objective in O. Moreover, each path starting from the root node and ending at a leaf node is associated with a complete schedule that satisfies the given specification p. It is guaranteed that at least one of them achieves the minimum cost. In the rest of this section, these two claims in the rest of this section are formalized and proved.

The rest of this section is organized as follows. First, in Section 7.3.1, the structure of the tree is presented and the algorithm that constructs the corresponding complete schedule is given in Sections 7.3.2 and 7.3.3. Then, in Section 7.3.4, this schedule construction algorithm is used effectively by utilizing the tree search methods outlined in [49] to arrive at a close-to-optimal solution quickly and an optimal one eventually. Afterward, it is formally proven, in Section 7.3.5, that the complete schedules associated with the leaf nodes are feasible, i.e., they all satisfy p, and at least one of those schedules is optimal.

7.3.1 Structure of the Tree

Given a specification p, the *assignment tree* of p is a labeled tree $\mathcal{T}_p = (\mathcal{N}_p, \mathcal{E}_p)$, in which every node is labeled with a process algebra term and every edge is labeled with an atomic proposition as follows[2]:

- the root node n_r is labeled with the specification p,
- for any node $n' \in \mathcal{N}_p$, if n' is labeled with a process algebra term p and there exists an atomic objective o and a process algebra term p'' such that $p' \xrightarrow{o} p''$, then there exists a node $n'' \in \mathcal{N}_p$ labeled with p'' such that $(n', n'') \in E$ and the edge (n', n'') is labeled with the atomic proposition o.

This definition has the following implications. Firstly, by definition, the root node is labeled with the specification p. Secondly, each leaf node is labeled with the terminated process $\surd$. Finally, the following lemma holds.

Proposition 7.3.1 *Let $\gamma = (o_1, o_2, \ldots, o_l)$ be a sequence of atomic objectives. The sequence γ is a trace of p if and only if there exists a path $(e_1, e_2, \ldots, e_l)$ on the tree $\mathcal{T}_p$ such that (i) the path starts at the root node and ends in one of the leaf nodes, (ii) for all $i = \{1, 2, \ldots, l\}$, the edge e_i is labeled with the atomic objective o_i.*

In the following example, the tree structure is illustrated.

[2]The assignment tree $\mathcal{T}_p = (\mathcal{N}_p, \mathcal{E}_p)$ of p should not be confused with the parse of p, which was denoted by $\mathcal{B}_p = (\mathcal{M}_p, \mathcal{F}_p)$

Example Consider the rescue mission example of Section 7.1.2. Recall that the set O of atomic objectives was

$$O = \{o_{V1@T1}, o_{V1@T2}, o_{V2@S1}, \\ o_{V2@S2}, o_{V3@T1}, o_{V3@T2}, o_{V3@T3}, o_{V3@B2}\},$$

and the specification was given by (7.2). The tree $\mathcal{T}_p$ corresponding to this specification is given in Figure 7-2.

7.3.2 Precedence of Atomic Objectives

Let me introduce the following partial order on the set O of atomic objectives determined by the specification p. This partial order will prove to be useful in the next section, when generating the feasible schedules corresponding to the leaf nodes of the assignment tree.

Definition 7.3.2 (Precedence of Atomic Objectives) *Given a set O of atomic objectives, a specification p defined on O, and two atomic objectives $o_1, o_2 \in O$, o_1 is said to precede o_2 in p, denoted by $o_1 <<_p o_2$, if and only if the following conditions hold:*

- *there exists at least one trace of γ of p such that o_1 and o_2 both appear in γ and $o_1 \prec_\gamma o_2$ holds,*
- *there is no trace γ' of p such that $o_2 \prec_{\gamma'} o_1$ holds.*

Given an atomic objective o, let $\mathtt{Pred}_p(o)$ denote the set of atomic objectives that precede o in specification p. Note that Definition 7.3.2 does not devise an efficient algorithm for computing $\mathtt{Pred}_p(o)$.

In the rest of this section, an algorithm, which computes $\mathtt{Pred}_p(o)$ in time polynomial with respect to the size of the specification, is introduced. This algorithm operates on the parse tree of the specification p. In order to make the presentation of this algorithm easier, let me introduce the following notation. Recall that the root node of the parse tree $\mathcal{B}_p = (\mathcal{M}_p, \mathcal{F}_p)$ of p is not labeled, the leaf nodes

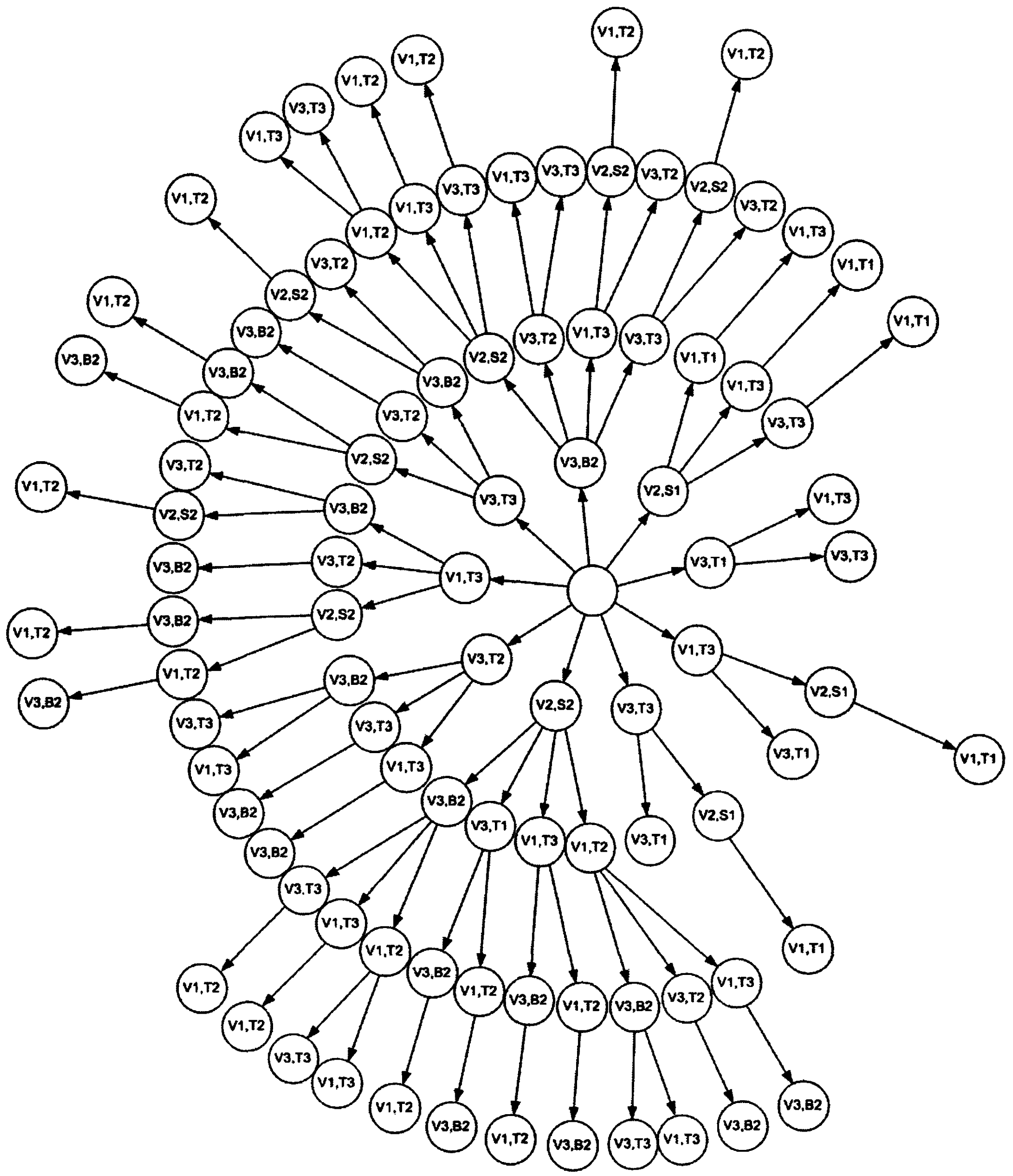


Figure 7-2: Tree Structure for the rescue mission. The labels of edges are shown on the nodes that they are directed to in order to make the visualization more readable.

```
$S := \emptyset$
$m := n$
while $m \neq \texttt{Root}(\mathcal{B}_p)$ do
    $m' := \texttt{Parent}_{\mathcal{B}_p}(m)$
    if $\texttt{Label}_{\mathcal{B}_p}(m) = (\cdot)$ and $\texttt{RightChild}_{\mathcal{B}_p}(m') = m$ then
        $S := S \cup \texttt{ChildrenAO}_{\mathcal{B}_p}(\texttt{LeftChild}_{\mathcal{B}_p}(m'))$
    end
    m := m'
end
return $S$
```

Algorithm 2: Computation of $\texttt{Pred}_p(o)$

are labeled with actions (or equivalently the atomic objectives), and all the other nodes are labeled with operators. To indicate these distinctive cases, given a node $n \in \mathcal{M}_p$, let me define the functions $\texttt{AtomicObjective}_{\mathcal{B}_p}(n)$ and $\texttt{Operator}_{\mathcal{B}_p}(n)$, which both evaluate to $\texttt{Label}_{\mathcal{B}_p}(n)$, but the former function is defined for the leaf nodes whereas the latter one is defined for all the nodes except the root and leaves. Let me also write $\texttt{ChildrenAO}_{\mathcal{B}_p}(n)$, referring to children atomic objectives, instead of $\texttt{ChildrenActions}_{\mathcal{B}_p}(n)$ just for clarification.

The algorithm that efficiently generates $\texttt{Pred}_p(o)$ is based on the following lemma.

Lemma 7.3.3 (A Characterization of Precedence) *Given a set O of atomic objectives, a specification p defined on O, and two atomic objectives $o_1, o_2 \in O$, let $n, m_1, m_2 \in \mathcal{M}_p$ be nodes in the parse tree of p such that (i) $m_1 = \texttt{LeftChild}_{\mathcal{B}_p}(n)$, $o_1 \in \texttt{ChildrenAO}_{\mathcal{B}_p}(m_1)$ and (ii) $m_2 = \texttt{LeftChild}_{\mathcal{B}_p}(n)$, $o_2 \in \texttt{ChildrenAO}_{\mathcal{B}_p}(m_2)$. Then, $o_1 <<_p o_2$ holds if and only if $\texttt{Operator}_{\mathcal{B}_p}(n) = (\cdot)$.*

A procedure that computes $\texttt{Pred}_p(o)$ for any p and $o \in O$ is given in Algorithm 2, The correctness of the algorithm can be proven using Lemma 7.3.3.

7.3.3 Feasible Complete Schedules

The essence of the algorithm that solves Problem 7.2.5 is a search procedure that explores the assignment tree $\mathcal{T}_p = (\mathcal{N}_p, \mathcal{F}_p)$ of p using a best-first branch-and-bound search heuristic. During the exploration process, the algorithm associates each node of the tree with a complete schedule. Denoting the unique path starting at the root and ending at a node $n \in \mathcal{N}_p$ of the tree with $(e_1, e_2, \ldots, e_l)$, the schedule $\mathcal{S}$ associated with n is such that an atomic objective o is assigned in $\mathcal{S}$ if and only if o is the label of one of the edges e_i, $i \in \{1, 2, \ldots, l\}$. The construction of the complete schedule at each node is done such that every observation of the schedule $\mathcal{S}$ corresponding to a node $n \in \mathcal{N}_p$ is a prefix of some trace of the specification p. Moreover, if n is a leaf node, then every observation of $\mathcal{S}$ is indeed a trace of p.

The salient property these schedules is that a schedule corresponding to a given node n can be computed from the schedule corresponding to the unique parent of n in the tree. Let Schedule(n) denote the schedule associated with node n. In what follows, a formal definition of Schedule(n) for all $n \in \mathcal{N}_p$ is presented. This definition can easily be converted to an efficient algorithmic procedure.

- If n is the root node, then

$$\texttt{Schedule}(n) = \{\sigma_{u_1}, \sigma_{u_2}, \ldots, \sigma_{u_K}\},$$

where $\sigma_{u_i} = \Lambda$, for all $i \in \{1, 2, \ldots, K\}$.

- For any node n other than the root, let n' be the unique parent of n and Schedule(n') $= \{\sigma'_{u_1}, \sigma'_{u_2}, \ldots, \sigma'_{u_K}\}$. Let also the edge $(n', n) \in \mathcal{E}_p$ be labeled by the atomic objective o. Then,

$$\texttt{Schedule}(n) = \{\sigma'_{u_1}, \ldots, \sigma'_{u_{i-1}}, \sigma_{u_i}, \sigma'_{u_{i+1}}, \ldots, \sigma'_{u_K}\},$$

such that $\sigma_{u_i} = \sigma'_{u_i} \circ (o, t)$ where

$$t = \max\{\tau^c_{\sigma_{u_i}}, \tau^c_{\texttt{Pred}_p(o)}\}$$

and $\tau^c_{\texttt{Pred}_p(o)}$ is the maximum completion time of the all the atomic objectives that are assigned in Schedule(n') and are predecessors of o in p, i.e., $\tau^c_{\texttt{Pred}_p(o)} =$

$\max\{\bar{t} \mid \bar{o} \in \texttt{Pred}_p(o)$, and there exists $\sigma_{u_i} \in \texttt{Schedule}(n')$ such that $(\bar{o}, \bar{t}) \in \sigma_{u_i}\}$.

Note that if the schedule $\texttt{Schedule}(n)$ is known, then the schedules for all the children of n can be computed.

7.3.4 Searching the Tree

The assignment tree $\mathcal{T}_p$ of p is searched using a best-first branch-and-bound search heuristic. The search algorithm marks each node of tree either as evaluated or not evaluated. The function $\texttt{Evaluated}_{\mathcal{T}_p}(\text{n})$ returns `True` if n is marked as evaluated and `False` otherwise. Initially, $\texttt{Evaluated}_{\mathcal{T}_p}(n) = \texttt{False}$ for all nodes of $\mathcal{T}_p$.

This search procedure is given in Algorithm 3, which gets the root node of the tree as an input and returns the leaf node that has the corresponding minimum cost complete schedule. The algorithm always maintains a current node, n, that it is working on and a node *minNode* which refers to the node with smallest cost complete schedule that was explored so far. The algorithm sets n to the root node (line 1) and *minNode* to the null node Λ (lines 2-3), initially. If node n has at least one child that is not evaluated (line 6), then the algorithm first determines the child of n with minimum cost (lines 7-15). If the cost of this child is less then the cost of *minNode* (line 16) and this child is a leaf node (line 17), then an assignment with a smaller cost is found and *minNode* is modified to be the child node (lines 18,19). In that case, the leaf node is marked as evaluated (line 20). If, on the other hand, the child node has a cost smaller than *minNode* but it is not a leaf node, then the search continues by assigning n to the child node (line 22) and exploring the child node (as in the best-first search methods). Finally, if the child node has a cost that is more than the cost of *minNode*, then the child node is marked as evaluated (line 25) and the branch starting from the child node is cut (as in the branch-and-bound methods). Finally, if all the children of n are evaluated, then n is marked as evaluated and the search continues with the parent of n (lines 28,29) in order to explore the rest of the tree. The algorithm ends when there is no more nodes to explore (line 4).

Algorithm 3 employs the functions $\texttt{Children}_{\mathcal{T}_p}(n)$ and $\texttt{Parent}_{\mathcal{T}_p}(n)$. One straightforward implementation of these functions for the tree $\mathcal{T}_p$ could be to use the definition of $\mathcal{T}_p$ given in Section 7.3.1 and construct $\mathcal{T}_p$ before starting Algorithm 3. However, then the spirit of best-first and branch-and-bound methods are lost. Instead, the tree has to be created on demand, i.e., on-the-fly, as the algorithm proceeds.

First, notice that Algorithm 3 runs the function $\texttt{Parent}_{\mathcal{T}_p}(n)$ only if it had run $\texttt{Children}_{\mathcal{B}_p}(n')$ before, where n' is the parent of n in $\mathcal{T}_p$. Hence, the on-the-fly tree construction reduces to determining the set $\texttt{Children}_{\mathcal{T}_p}(n)$ for any given n, since $\texttt{Parent}_{\mathcal{T}_p}(n)$ can be evaluated with some extra bookkeeping.

Given a process algebra term p, the procedure given in Algorithm 4 returns the set of all atomic objective and process algebra term pairs (o', p'), for which $p \xrightarrow{o} p'$ holds. The correctness of this algorithm is follows directly from the semantics given in Definition 6.3.4.

The existence of an efficient procedure to compute the children nodes is crucial, since it allows implementation of the best-first and branch-and-bound heuristics.

7.3.5 Feasibility and Optimality

In this section, let me first prove that each schedule that corresponds to a leaf node of the assignment tree is feasible. Later in this section, it will be shown that at least one of those schedules achieves the minimum cost.

As far as feasibility is concerned, the following theorem formalizes the claim.

Theorem 7.3.4 *Given a set O of atomic objectives, a specification p defined on O, let $\mathcal{T}_p = (\mathcal{N}_p, \mathcal{E}_p)$ be the assignment tree of p, and n be any leaf node in the assignment tree. Then, any observation of* $\texttt{Schedule}_p(n)$ *is a trace of p.*

The proof of this theorem is rather technical and is provided in the appendix.

Regarding optimality of the algorithm, the following lemma plays a key role.

```
$n$ := Root($\mathcal{T}_p$)
$minCost$ := $\infty$
$minNode$ := $\Lambda$
while $n \neq \Lambda$ do
    $S$ := Children($n$)
    if Evaluated($S$) = False then
        $minChildCost$ := $\infty$
        for all $m \in S$ do
            if Evaluated($m$) = False then
                if Cost($m$) < $minChildCost$ then
                    $minChildCost$ := Cost($m$)
                    $minChild$ := $m$
                end
            end
        end
        if $minChildCost < minCost$ then
            if Leaf($m$) = True then
                $minCost$ := $minChildCost$
                $minNode$ := $m$
                Evaluated($m$) := True
            else
                $n$ := $m$
            end
        else
            Evaluated($m$) := True
        end
    else
        Evaluated($n$) := True
        $n$ := Parent($n$)
    end
end
return $minNode$
```
Algorithm 3: Best-First Branch-and-Bound Tree Search

```
switch $p$ do
    case $p_1 + p_2$
        return $Next(p_1) \cup Next(p_2)$
    case $p_1 \cdot p_2$
        return $\{(p_1' \cdot p_2, t') \mid (p_1', t') = Next(p_2)\}$
    case $p_1 \| p_2$
        return $\{(p_1' \| p_2, t') \mid (p_1', t') = Next(p_1)\} \cup$
        $\{(p_1 \| p_2', t') \mid (p_2', t') = Next(p_2)\}$
    otherwise
        return $\{(\surd, y)\}$
    end
end
```

Algorithm 4: Computation of *Next(p)*

Assumption 7.3.5 *Let $\mathcal{S} = \{\sigma_{u_1}, \sigma_{u_2}, \dots, \sigma_{u_n}\}$ and $\mathcal{S}' = \{\sigma'_{u_1}, \sigma'_{u_2}, \dots, \sigma'_{u_n}\}$ be two feasible complete schedules such that $\tau_{\sigma_{u_k}} \leq \tau_{\sigma'_{u_k}}$ and $\tau_{\sigma_{\bar{u}}} = \tau_{\sigma'_{\bar{u}}}$ for all $\bar{u} \in \mathcal{U} \setminus \{u_k\}$, where τ_{σ_u} is the completion time of the single UAV schedule σ_u. Then, the cost function is such that the cost of $\mathcal{S}$ is less than or equal to the cost of $\mathcal{S}'$.*

Assumption 7.3.5 basically suggests that increasing the completion time of a single UAV schedule can not decrease the cost of its complete schedule. Let me note that several cost functions of practical interest satisfy this assumption, e.g., cost functions 7.3. Indeed, it is unintuitive and impractical to have the cost of the mission decrease by increasing the completion time of one of the UAVs while fixing the completion time of all the others. This assumption, however, is presented here to view the applicability of the results more concretely with the following lemma.

Lemma 7.3.6 *Let the cost function in Problem 7.2.5 satisfy Assumption 7.3.5. Given a specification p, and a complete schedule $\mathcal{S}$ that satisfies p, let $\pi = (o_1, o_2, \dots, o_k)$ be an observation of $\mathcal{S}$. Furthermore, let n be the node in the assignment tree $\mathcal{T}_p$ of p such that the unique path $(e_1, e_2, \dots, e_k)$ from the root node in $\mathcal{T}_p$ leads to n, where $\texttt{Label}(e_i) = o_i$ for $i = 1, 2, \dots, k$. Then, we have that $\texttt{Cost}(\texttt{Schedule}_p(n)) \leq \texttt{Cost}(\mathcal{S})$.*

This lemma is fundamental to the results of this chapter in the sense that it partitions the (uncountably infinite) set of all complete schedules that satisfy p into a finite set of classes, where any schedule with an observation π as in the above theorem is in the class of schedules that contains $\texttt{Schedule}_p(n)$ [3]. Notice that each class contains uncountably many schedules and one computable schedule in each class has the the minimum cost given a very mild assumption on the cost function holds. Since there is a path for each and every trace of p on the assignment tree of p, it is enough compute the cost of only those schedules that are in the tree to obtain a minimum cost complete schedule for Problem 7.2.5.

Theorem 7.3.7 *Given a specification p, let $\mathcal{T}_p$ denote its parse tree. Then, for at least one of the leaf nodes n in $\mathcal{T}_p$, its corresponding complete schedule* $\texttt{Schedule}_p(n)$ *minimizes the cost function of Problem 7.2.5, given Assumption 7.3.5 is satisfied.*

The proof of this theorem follows directly from Lemma 7.3.6.

7.4 Example Scenarios and Simulations

Consider a simple illustrative scenario where two UAVs are required to first attack six targets and then search for possible new targets in the same area. Let o_d and o_s denote objectives for the engagement of six targets and the search mission respectively. Then, the specification of the mission is $p = o_d \cdot o_s$. The objective o_d is equal to $o_{d1} \,||\, o_{d2} \,||\, o_{d3} \,||\, o_{d4} \,||\, o_{d5} \,||\, o_{d6}$, where o_{di} denotes the atomic objective of attacking the target i for $i = 1, 2, \dots, 6$. The objective o_s, on the other hand, requires searching the there regions in parallel, i.e., $o_s = o_{s1} \,||\, o_{s2} \,||\, o_{s3} \,||\, o_{s45}$, where o_{s1}, o_{s2}, o_{s3} denote the atomic objectives for searching the regions $s1$, $s2$ and $s3$, respectively. To make the matters more complicated, let it be the case that searching either area $s4$ or $s5$ is adequate for accomplishing $s3$, but not both, i.e., $o_{s45} = o_{s4} + o_{s5}$, where o_{s4} and o_{s5} are the atomic objectives corresponding to searching the regions $s4$ and $s5$, respectively. A solution of this problem instance is presented in Figure 7-3, which was generated in less than a second of computation time, after exploring 1000 nodes of the tree.

[3] Note that a complete schedule can be in more than one class, since it may have more than one observation.

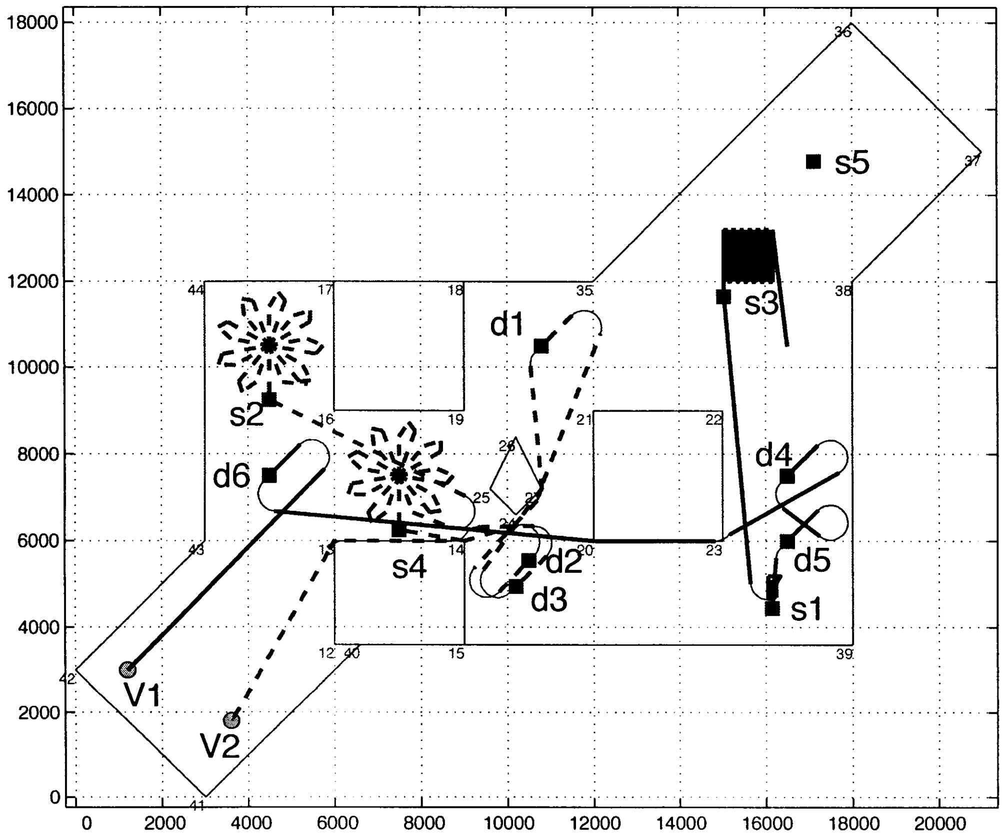


Figure 7-3: Scheduling that satisfies $p = o_d \cdot o_s$.

Chapter 8

Conclusions

This thesis studied novel extensions of the standard vehicle routing problem, where complex constraints of the problem can be specified naturally. Applications in multiple-UAV mission planning were highlighted and several motivating example scenarios of interest were presented, in which formal methods allow the precise formulation of complex mission specifications given in natural language.

The thesis was structured in two main parts. In the first part, linear temporal logics were considered as candidates for specification of constraints in Vehicle Routing Problems (VRPs), whereas in the second part, process algebras were considered for similar purposes. The temporal logics discussed in the first part are known to be more expressive than the process algebra of the second part. However, process algebra specifications lead to computationally more attractive algorithms. In particular, in this thesis, we presented an anytime algorithm, which returns a best-first feasible solution in polynomial time (with respect to the size of the specification) and improves this solution to achieve an optimal solution in finite amount of time.

Two temporal logics were studied as candidates for natural specification of complex constraints in Vehicle Routing Problems (VRPs). The temporal logic was the linear temporal logic LTL_{-X}. A new variant of the VRP, called the Vehicle Routing Problem with Linear Temporal Logic Specifications (VRPLTL), was proposed. In the problem formulation, the complex constraints of the problem were modeled using a high-level formal language called LTL_{-X}. A systematic procedure that outputs a set of linear constraints in mixed binary and continuous variables for any LTL_{-X} was given; it was shown that, under some technical assumptions, an LTL_{-X} formula is satisfied if and only if the corresponding set of inequalities hold. Using this procedure a MILP-based computational method was proposed which solves VRPTL to optimality. Throughout the examples, it was shown that interesting multi-UAV mission planning scenarios can be modeled naturally using LTL_{-X}, while optimal plans for problems of practical sizes can be computed in reasonable amount of time. Even though only multiple-UAV mission planning applications have been considered, it should be noted that several other logistics problems with complex timing constraints can be modeled using the VRPTL.

Two different MILP formulations of VRP were studied and adapted for solving VRPLTL. Based on the experiments presented in the thesis, the network flow based formulation had a smaller number of variables and tended to give solutions more quickly. Moreover, it did not require a long pre-solve phase. The set covering based formulation, on the other hand, had exponential number of variables in the size of customers; and it had a computationally intense pre-solve phase. But the advantage of set covering based formulation was that more general cost functions could be used. The LP relaxation of the set-covering formulation was known to be very tight for the VRP; however, for the VRPTL, similar observation did not follow in the experiments presented in this thesis because of the LTL_{-X} constraints. The exact CPU times for the examples were not presented, but none of the examples in this thesis require more than a few seconds of computation time.

Next, the real-time temporal logic, metric temporal logic (MTL) was considered. MTL differs from LTL_{-X} in the sense that it allows specification of constraints that involve qualitative properties of time; examples include deadlines, time windows, and qualitative relative timing constraints. Another novel variant of Vehicle Routing Problem, which we have called Vehicle Routing Problem with Metric Temporal Logic Specifications (VRPMTL), was introduced. A MILP-based algorithm that solves the problem to optimality was proposed. Applications to multi-UAV mission planning were considered, where high level complex mission tasks were specified naturally via Metric Temporal Logic.

Then, we considered process algebra as a specification language in vehicle routing problems. This part of the thesis was presented more towards the multi-UAV mission planning application domain. The task assignment problem with process algebra specifications was introduced and some problem instances of interest were pointed out. The problem was solved using a computationally effective algorithm, which returns a best-first feasible solution to the problem in polynomial time and terminates with the optimal solution in finite time. These algorithmic properties of the solution method were formalized and proven.

A challenge for future work is a unifying study of the languages for their suitability to optimal planning. In particular, it is of interest to determine the class of languages for which a feasible plan can be found in polynomial time and an optimum one can be found in finite time. The development of algorithms with nice properties such as constant factor approximations for such problems would be very valuable for applications. A second future direction is the consideration of optimal planning with dynamics. In this case, the key objective is to identify the class of dynamics, for which effective optimal planning algorithms exist.

Appendix A

Proofs

A.1 Proof of Lemma 4.2.3

Proof Let $\mathcal{TS} = (Q, Q_0, \to, \Pi, L)$. Given that Assumption 4.2.2 holds, $\sigma \models \phi$ implies $\sigma \models \bigwedge_{k \in \{1,\dots,K\}} \left(\Diamond\Box p_0^k\right)$ and $\sigma \models \psi$ according to (3.6). Using the former consequence with the semantics definitions (3.7) and (3.8), there exists a finite $j \geq 1$ such that $\forall j' \geq j$ we have $(\sigma, j') \models p_0^k$ for all $k \in \{1, \dots, K\}$. Noting that $p_0^k \in \Pi$, there holds $p_0^k \in L(q_{j'})$, which implies that $q_{j'}$ must satisfy $q_{j'} = \left(\left(\theta^1, v_0^1\right), \dots, \left(\theta^K, v_0^K\right)\right)$ for some $\theta^1, \dots, \theta^K \in \mathbb{R}$. This is true for j. But for all other $j' > j$, we have $(q_{j'-1}, q_{j'}) \notin \to$ by the definition of the vehicle-routing transition system. Hence, $\sigma = (q_1, q_2, \dots, q_j)$, which implies the lemma. ▮

A.2 Proof of Theorem 4.3.2

Proof (Sketch) The proof is an induction on the height of the LTL$_{-X}$ formula ψ, which is a common proof technique in mathematical logic [29]. The base case of formulae with height one holds by Lemma 4.3.1. Assume that the theorem holds for formulae of height n, then given a formula ψ of height $n+1$, note that ψ is composed of at most two height n formulae bound with an operator. Let us denote these formulae by φ_1 and φ_2. By the hypothesis we have $\boldsymbol{P}_{\varphi_i} \in \mathcal{G}_{\varphi_i}(\boldsymbol{P}_{p_1}, \dots, \boldsymbol{P}_{p_m})$ for $i = 1, 2$. Moreover, with algorithm 1 we also have $\boldsymbol{P}_\psi \in G_\psi(\boldsymbol{P}_{\varphi_1}, \boldsymbol{P}_{\varphi_2})$. These together mean, by set inclusion, $\boldsymbol{P}_{\psi} \in G_{\psi}(\boldsymbol{P}_{p_1}, \dots, \boldsymbol{P}_{p_m})$. This implies the result by induction, since the formula ψ has a finite height. ▮

A.3 Proof of Lemma 7.3.3

Before proving Lemma 7.3.3, let us note the following lemma, which states that if a specification p includes an atomic objective o in its parse tree, then there exists at one trace γ of p such that o appears in γ. More formally,

Lemma A.3.1 *Given a specification p, let $\mathcal{B}_p$ denote its parse tree and $n = \mathtt{Root}(\mathcal{B}_p)$. For any atomic objective o, if $o \in ChildrenAO(n)$, then there exists a trace $\gamma \in \Gamma_p$ such that o appears in γ.*

Proof The proof of this lemma is by induction on the depth of the parse tree[1]. If the depth of the parse tree is one, then the specification p is of the form $p = o$, where o is an atomic objective. Trivially, o is the only atomic objective in $\mathtt{ChildrenAO}_{\mathcal{B}_p}(n)$ and o appears in the only trace $\gamma = (o)$ of p.

Assume that the hypothesis holds for specifications with parse trees of depth $k' < k$. Let p be a specification with a depth k parse tree $\mathcal{B}_p$. Recall that $n = \mathtt{Root}(\mathcal{B}_p)$. Let m_1 and m_2 be the left and right children of n in $\mathcal{B}_p$. Moreover, let p_1 and p_2 be the specifications that have parse trees $\mathcal{B}_{p_1}$ and $\mathcal{B}_{p_2}$ rooted at nodes m_1 and m_2, respectively. Since $\mathcal{B}_p$ has depth k, the parse trees $\mathcal{B}_{p_1}$ and $\mathcal{B}_{p_2}$ both have depth strictly less than k.

Let o be an atomic objective with $o \in \mathtt{ChildrenAO}_{\mathcal{B}_p}(n)$. Then, we have that either $o \in \mathtt{ChildrenAO}_{\mathcal{B}_p}(m_1)$ or $o \in \mathtt{ChildrenAO}_{\mathcal{B}_p}(m_2)$ hold. Without loss of any generality, let us assume that the former one holds. Since $\mathcal{B}_{p_1}$ has depth less than k, by the hypothesis, there exists a trace γ_1 of p_1 such that o appears in γ_1.

To show that there exists a trace γ' of p, with $o \in \gamma$, let us consider the three cases: $\mathtt{Operator}_{\mathcal{B}_p}(n) = (+), (\cdot), (\|)$. If $\mathtt{Operator}_{\mathcal{B}_p}(n) = (+)$, then $\gamma' = \gamma_1$ is a trace of

[1]The depth of the parse tree is defined as the length of the longest path from the root to a leaf node.

p with $o \in \gamma$. If, on the other hand, $\texttt{Operator}_{\mathcal{B}_p} \in \{(\cdot), (\|)\}$, then $\gamma' = \gamma_1 \bullet \gamma_2$ is a trace of p with $o \in \gamma$, where γ_2 is any trace of p_2. Hence, the lemma follows by induction on k. ▮

Proof of Lemma 7.3.3 First, let us show the sufficiency. Let $\texttt{Operator}_{\mathcal{B}_p}(n) = (\cdot)$ hold; we would like to show that $o_1 << o_2$ holds, i.e., (i) there exists a trace of p in which o_1 precedes o_2, (ii) there is no trace of p in which o_2 precedes o_1. To show (i), let p_1 and p_2 be the specifications that have the parse trees rooted at nodes m_1 and m_2, respectively. Let also γ_i be a trace of p_i such that o_i appears in γ_i, for $i = 1, 2$. Note that γ_i exist by Lemma A.3.1. Notice that $\gamma = \gamma_1 \bullet \gamma_2$ is a trace of p such that o_1 and o_2 both appear in γ and $o_1 \prec_\gamma o_2$ holds. To show (ii) by contradiction, assume that there exists a trace γ' of p such that $o_1 \prec_{\gamma'} o_2$. Notice, however, that for any trace γ' of p can be written in the form $\gamma' = \gamma'_1 \bullet \gamma'_2$, where γ'_1 and γ'_2 are traces of p_1 and p_2, respectively. Since o_1 appears in γ', it must appear in γ'_1 (not in γ_2). Moreover, since o_2 precedes o_1, o_2 must also appear in γ'_1, which contradicts the fact that o_2 is not in the tree rooted at m_1.

To prove necessity by contradiction, assume that $o_1 << o_2$ holds, but $\texttt{Operator}_{\mathcal{B}_p} \neq (\cdot)$. Consider the two cases: $\texttt{Operator}_{\mathcal{B}_p}$ is $(+)$, $(\|)$. In the former case, there is no trace of p, in which o_1 and o_2 appear both. Hence, $o_1 << o_2$ does not hold, since the first condition in Definition 7.3.2 is not satisfied. In the latter case on the other hand, one can construct a trace γ of p, in which o_2 precedes o_1. More precisely, let γ_i be a trace of p_i such that o_i appears in γ_i, for $i = 1, 2$. Note that the existence of γ_i is guaranteed by Lemma A.3.1. Then, $\gamma = \gamma_2 \bullet \gamma_1$ is a trace of p, where $o_1, o_2 \in \gamma$ and $o_2 \prec_\gamma o_1$, which contradicts the second condition in Definition 7.3.2. ▮

A.4 Proof of Theorem 7.3.4

Before presenting the proofs, let us note three lemmas. The first lemma yields a characterization of the following definition.

Definition A.4.1 *Alternative Atomic Objectives Given a set O of atomic objectives, a specification p defined on O, and two atomic objectives $o_1, o_2 \in O$, o_1 and o_2 are said to be alternatives if the following condition holds:*

- *for all traces γ of p, either $o_1 \in \gamma$ or $o_2 \in \gamma$, but not both.*

The following lemma is a characterization of the alternative atomic objectives.

Lemma A.4.2 *Characterization of Alternatives Given a set O of atomic objectives, a specification p defined on O, and two atomic objectives $o_1, o_2 \in O$, let $n, m_1, m_2 \in \mathcal{M}_p$ be the nodes in the parse tree of p such that (i) $m_1 \in \texttt{LeftChild}_{\mathcal{B}_p}(n)$, $o_1 \in \texttt{ChildrenAO}_{\mathcal{B}_p}(m_1)$ and (ii) $m_2 = \texttt{LeftChild}_{\mathcal{B}_p}(n)$, $o_2 \in \texttt{ChildrenAO}_{\mathcal{B}_p}(m_2)$. Then, o_1 and o_2 are alternatives if and only if $\texttt{Operator}_{\mathcal{B}_p}(n) = (+)$.*

The proof of is this lemma is similar to that of Lemma 7.3.3 and is omitted here.

The second lemma is the following key lemma that simplifies the proof of Theorem 7.3.4.

Lemma A.4.3 *Given a set O atomic objectives, two specifications (PA terms) p, p' both defined on O, a sequence γ defined on O, and two atomic objectives $o_1, o_2 \in O$, if $p \xrightarrow{\gamma \bullet (o_1, o_2)} p'$ and $o_1 \notin \texttt{Pred}_p(o_2)$ both hold, then $p \xrightarrow{\gamma \bullet (o_2, o_1)} p'$ also holds.*

Proof Consider the nodes $n, m_1, m_2 \in \mathcal{M}_p$ in the parse tree of p, for which the following hold: (i) $m_1 \in \texttt{LeftChild}_{\mathcal{B}_p}(n)$, $o_1 \in \texttt{ChildrenAO}_{\mathcal{B}_p}(m_1)$ and (ii) $m_2 = \texttt{LeftChild}_{\mathcal{B}_p}(n)$, $o_2 \in \texttt{ChildrenAO}_{\mathcal{B}_p}(m_2)$. It is given that $o_1 \notin \texttt{Pred}_p(o_2)$; hence, $\texttt{Operator}_{\mathcal{B}_p}(n) \neq (\cdot)$ by Lemma 7.3.3. Moreover, notice that there exists a trace of p, in which o_1 and o_2 appear together, since there exists a γ', for which $p' \xrightarrow{\gamma'} \surd$ holds and, consequently, $p \xrightarrow{\gamma \bullet (o_1, o_2)} p' \xrightarrow{\gamma'} \surd$; hence, $\gamma \bullet (o_1, o_2) \bullet \gamma'$ is a trace of p. This, however, implies that o_1 and o_2 are not alternatives, which in turn implies that $\texttt{Operator}_{\mathcal{B}_p}(n) \neq (+)$.

Given $\texttt{Operator}_{\mathcal{B}_p}(n) \neq (\cdot), (+)$, there holds $\texttt{Operator}_{\mathcal{B}_p}(n) = (||)$. Let, p_1 and p_2 denote the processes with parse trees rooted at nodes m_1 and m_2, respectively. Let $\bar{p}$ be such that $p \xrightarrow{\gamma} \bar{p} \xrightarrow{(o_1, o_2)} p'$. Note that the term p contains $p_1 || p_2$ as a subterm, where as p' contains $p'_1 \| p'_2$ as a subterm, where p'_1 and p'_2 are such that $p_1 \xrightarrow{o_1} p'_1$ and $p_2 \xrightarrow{o_2} p'_2$. Other than these subterms, p and p' are the same. Notice that $\bar{p} \xrightarrow{o_1} p'' \xrightarrow{o_2} p'$ also holds, where p'' is the same as $\bar{p}$ except that instead of $p_1 \| p_2$, it contains $p'_1 \| p_2$ as a subterm. Let p''' be a term which is the same as $\bar{p}$, except, it contains $p_1 \| p'_2$ as a subterm instead of $p_1 \| p_2$. In this case, $\bar{p} \xrightarrow{o_2} p''' \xrightarrow{o_1} p'$ also holds, which implies that $p \xrightarrow{\gamma \bullet (o_2, o_1)} p'$ holds. ∎

Lemma A.4.3 has particular importance, since it suggests that two adjacent atomic objectives in a trace can be exchanged if the one that appears before does not precede the other one in p. An important corollary of Lemma A.4.3 is the following.

Corollary A.4.4 *Given a set O atomic objectives, two specifications (PA terms) p, p' both defined on O, two sequences γ_1, γ_2 defined on O, and an atomic objective $\bar{o} \in O$, if $p \xrightarrow{\gamma_1 \bullet \gamma_2 \bullet (\bar{o})} p'$ and for all $o \in \gamma_2$we have $o \notin \mathtt{Pred}_p(\bar{o})$ hold, then $p \xrightarrow{\gamma_1 \bullet (\bar{o}) \bullet \gamma_2} p'$ also holds.*

This corollary follows by successive application of Lemma A.4.3.

The third lemma is the following that follows from Corollary A.4.4.

Lemma A.4.5 *Given a set O of atomic objectives, a specification p defined on O, let $\mathcal{T}_p = (\mathcal{N}_p, \mathcal{E}_p)$ be the assignment tree of p. For any node $n' \in \mathcal{N}_p$ and for any observation γ of $\mathtt{Schedule}_p(n')$, we have that $p \xrightarrow{\gamma} p'$, where p' is the specification that node n' is labeled with.*

Proof The proof is an induction on the length of the unique path from the root node in $\mathcal{T}_p$ to n'.

For the base case, i.e., when the path length is equal to one, we have that n' is one of the children of p in $\mathcal{T}_p$. That is, there is an edge $(n, n') \in \mathcal{E}_p$ in the assignment tree of p, which implies, by the definition of $\mathcal{T}_p$, that $p \xrightarrow{(\bar{o})} p'$, where $p' = \mathtt{Label}_{\mathcal{T}_p}(n')$ and $\bar{o} = \mathtt{Label}_{\mathcal{T}_p}((n, n'))$ in the assignment tree $\mathcal{T}_p$. By definition of $\mathtt{Schedule}(n')$, only the atomic objective $\bar{o}$ is assigned, the only observation of this schedule is $(\bar{o})$, for which $p \xrightarrow{\bar{o}} p'$ holds.

Let $\bar{n}$ be a node in $\mathcal{T}_p$, to which the unique path from the root node of $\mathcal{T}_p$ has length $k - 1$. Let $\bar{p} = \mathtt{Label}_{\mathcal{T}_p}$; by definition the induction hypothesis, we have that

$p \xrightarrow{\gamma} \bar{p}$ holds for all observations γ of $\texttt{Schedule}(\bar{n})$. Consider any children n' of $\bar{n}$; note that the unique path from the root node to n' has length k. Let $p' = \texttt{Label}_{\mathcal{T}_p}(n')$; note that, by definition of $\mathcal{T}_p$, we have that $\bar{p} \xrightarrow{\bar{o}} p'$ holds, where $\bar{o} = \texttt{Label}_{\mathcal{T}_p}$. Note also that, by its recursive definition, ant observation of $\texttt{Schedule}(n')$ is of the form $\gamma_1 \bullet (\bar{o}) \bullet \gamma_2$, where $\gamma_1 \bullet \gamma_2$ is an observation of $\texttt{Schedule}(\bar{n})$ and for all $o \in \gamma_2$, there holds $o \notin \texttt{Pred}(\bar{o})$. Recall that $p \xrightarrow{\gamma_1 \bullet \gamma_2} \bar{p} \xrightarrow{p}{}'$; that is, $p \xrightarrow{\gamma_1 \bullet \gamma_2 \bullet (\bar{o})} p'$. Then, by Corollary A.4.4, we have that $p \xrightarrow{\gamma_1 \bullet (\bar{o}) \bullet \gamma_2} p'$ holds. Hence, the lemma follows. ∎

Lemma A.4.5 simply suggests that the observations of a schedule associated with a node n' in the assignment tree complies with the label p' of the same node, in the sense that by executing every such observation, p evolves to p'.

Proof of Theorem 7.3.4 Noting that each leaf node is labeled with the terminated process $\surd$ and that every sequence γ with $p \xrightarrow{\gamma} \surd$ is a trace of p, Theorem 7.3.4 directly follows from Lemma A.4.5 as a special case. ∎

A.5 Proof of Lemma 7.3.6

Let me state and prove a slightly stronger lemma, from which the proof of Lemma 7.3.6 easily follows. Let me fix a given set O of atomic objectives; any specification that will be referred to is defined on O.

Lemma A.5.1 *Given a specification p, and a complete schedule $\mathcal{S}$, let p' be a process algebra term such that there exists a sequence γ with $p \xrightarrow{\gamma} p'$, and for all observations π of $\mathcal{S}$, there holds $p \xrightarrow{\pi} p'$. Let $\pi = (o_1, o_2, \ldots, o_k)$ be an observation of $\mathcal{S}$. Furthermore, let n be the node in the assignment tree $\mathcal{T}_p$ of p such that the unique path $(e_1, e_2, \ldots, e_k)$ from the root node in $\mathcal{T}_p$ leads to n, where $\texttt{Label}(e_i) = o_i$ for $i = 1, 2, \ldots, k$. Then, for all $o_i \in \pi$ and $t_i, \bar{t}_i \in \mathbb{R}$, where t_i and $\bar{t}_i$ are such that $(o_i, t_i) \in \mathcal{S}$, $(o_i, \bar{t}_i) \in \texttt{Schedule}_p(n)$, there holds $t_i \geq \bar{t}_i$.*

Proof This lemma is shown by induction on the size of π. If π is of size one, i.e., $\pi = (o)$ for some $o \in O$, then the hypothesis trivially holds, since $\texttt{Schedule}_p(n)$

will schedule the only atomic objective o at the earliest time the UAV it is assigned to arrives at the initial coordinates of o. The schedule $\mathcal{S}$ can not assign the same atomic objective earlier to the same UAV, since the UAV will not be able to arrive at the initial coordinates of o on time. Moreover, $\mathcal{S}$ can not assign o to another UAV, since, by definition, there is only one UAV that is capable of executing o.

Let the hypothesis hold for π of size $k-1$; let us show that it also holds for size k. Let $\pi = (o_1, o_2, \ldots, o_k)$ be an observation of size k. Consider the schedule $\overline{\mathcal{S}}$, which is the same as $\mathcal{S}$, but o_k is not assigned. Let $p_1, p_2, \ldots, p_{k-1}$ be such that $p \xrightarrow{o_1} p_1 \xrightarrow{o_2} p_2 \ldots p_{k-1} \xrightarrow{o_k} p'$. Notice that $\overline{\pi} = (o_1, o_2, \ldots, o_{k-1})$ is a size $k-1$ observation of $\overline{\mathcal{S}}$. Let $\bar{n}$ be the parent of n in the parse tree of p. Hence, for all o_i with $i \in \{1, 2, \ldots, k-1\}$, there holds $t_i \geq \tilde{t}_i$, where $(o_i, t_i) \in \overline{\mathcal{S}}$ and $(o_i, \tilde{t}_i) \in \texttt{Schedule}_{\mathcal{S}}(\bar{n})$. Hence, for all o_i with $i \in \{1, 2, \ldots, k-1\}$, there holds $t_i \geq \tilde{t}_i$, where $(o_i, t_i) \in \mathcal{S}$ and $(o_i, \tilde{t}_i) \in \texttt{Schedule}_p(n)$, since $\mathcal{S}'$ and $\mathcal{S}$ differ only by (o_i, t_i) and $\texttt{Schedule}_p(\bar{n})$ and $\texttt{Schedule}_p(n)$ differ only by $(o_i, \tilde{t}_i)$. Now, it remains to show that same holds for o_k, i.e., $t_k \geq \tilde{t}_k$. To prove by contradiction, assume it does not. Then, o_k must be scheduled before one of its predecessors in p, i.e., there exists $o_i \in \pi$ such that $o_i \in \mathit{Pred}_p(o_k)$ and $t_k < t_i$. In this case, notice that there exists an observation π' of $\mathcal{S}$, in which o_k appears before o_i. However, there is no such trace of p, by the definition of precedence; hence, $p \xrightarrow{\pi'} p'$ does not hold and we reach a contradiction. ∎

Lemma A.5.1 is stronger than Lemma 7.3.6 in two ways. Firstly, the node n is a leaf node in the latter one, whereas in the former one it is any one of the nodes in the tree. Secondly, in the latter one Assumption 7.3.5 is assumed, whereas in the former one there is no such assumption and the result is stated in terms of the execution times of the atomic objectives. Given these facts, it is easy to show that Lemma 7.3.6 is a special case of Lemma A.5.1.

Proof of Lemma 7.3.6 Letting $p' = \surd$ in Lemma A.5.1 proves the claim, given that the cost function in Problem 7.2.5 satisfies Assumption 7.3.5.

Bibliography

[1] P. Adao and P. Mateus. A process algebra for reasoning about quantum security. *Electronic Notes in Theoretical Computer Science*, 170:3–21, 2007.

[2] Y. Agarwal, K. Mathur, and H.M. Salkin. A set-partitioning-based exact algorithm for the vehicle routing problem. *Networks*, 19(7):731–749, 1989.

[3] M. Alighanbari, Y. Kuwata, and J.P. How. Coordination and control of multiple uavs with timing contraints and loitering. In *American Control Conference*. IEEE, 2003.

[4] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. *Journal of the Association for Computing Machinery*, 43(1):116–146, 1996.

[5] E. Angelelli and M.G. Speranza. The applications of a vehicle routing model to a waste-collection problem: two case studies. *Journal of the Operational Research Society*, 53:944–952, 2002.

[6] J.M.C. Baeten. *Process Algebra*. Cambridge University Press, 1990.

[7] J.M.C. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335:131–146, 2005.

[8] M.L. Balinski and R.E. Quandt. On an integer program for a delivery problem. *Operations Research*, 12(2):300–304, March-April 1964.

[9] J. Bellingham, M. Tillerson, A. Richards, and J.P. How. *Coopertive Control: Models, Applications and Algorithms*, chapter Multi-Task Allocation and Path Planning for Cooperating UAVs. Kluwer Academic Publishers, 2001.

[10] A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35:407–427, 1999.

[11] J.A. Bergstra and C.A. Middelburg. Process algebra for hybrid systems. *Theoretical Computer Science*, 335(2-3):215–280, 2003.

[12] M. Bowden. *Black Hawk Down: A Story of Modern War*. Atlantic Monthly Press, Berkeley, CA, 1999.

[13] J. Bramel and D. Simchi-Levi. On the effectiveness of set covering formulations for the vehicle routing problem. *Operations Research*, 45(2):295–301, March-April 1997.

[14] O. Braysy and M. Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithmsh time windows, part i: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, February 2005.

[15] O. Braysy and M. Gendreau. Vehicle routing problem with time windows, part ii: Metaheuristics. *Transportation Science*, 39(1):119–139, February 2005.

[16] J. E. Büchi. On a decision method in restricted second order arithmetic. *Z. Math. Logik Grundlag.*, 6:66–92, 1960.

[17] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, Cambridge, MA, 1999.

[18] G. Clarke and J.W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.

[19] G.B. Dantzig and J.H. Ramser. The truck dispatching problem. *Management Science*, 6:80–91, 1959.

[20] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354, March-April 1992.

[21] K. Erol, J. Hendler, and D. Nau. Semantics of hierarchical task network planning. Technical Report CS-TR-3239, University of Maryland, 1995.

[22] G.E. Fainekos, H. Kress-Gazit, and G.J. Pappas. Hybrid controllers for path planning: A temporal logic approach. In *American Control Conference*. IEEE, 2005.

[23] W. Fokkink. *Introduction to Process Algebra*. Springer, 2000.

[24] A. Fusaoka, H. Seki, and K. Takahashi. A description and reasoning of plant controllers in temporal logic. In *Proceedings of the 8th International Joint Conference in Artificial Intelligence*, pages 405–408, August 1983.

[25] D.M. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects*, volume 2 of *Oxford Logic Guides*, chapter Intervals and Planning. Oxford Science Publications, 1994.

[26] M.R. Garey and D.S. Johnson. *Computers and Intractibility: a Guide to Theory of NP-completeness*. W.H. Freeman Co, New York, 1979.

[27] E. Hadjiconstantinou, C. Lucas, G. Mitra, and S. Moody. Tools for reformulating logical forms into zero-one mixed integer programs: Software tools for mathematical programming. *European Journal of Operational Research*, 72(2):262–276, 1994.

[28] K.L. Hoffman and M. Padberg. Solving airline crew scheduling by branch-and-cut. *Management Science*, 39(6):657–682, June 1993.

[29] M. Huth and M. Ryan. *Logic in Computer Science: Modeling and Reasoning about Systems*. Cambridge University Press, 2nd edition, 2004.

[30] R.G. Jeroslow. *Logic Based Decision Support: Mixed Integer Model Formulation*. Elsevier Science, 1989.

[31] S. Karaman and E. Frazzoli. Complex mission optimization for multiple-UAVs using linear temporal logic. In *American Control Conference*, 2008.

[32] S. Karaman and E. Frazzoli. Optimal vehicle routing with metric temporal logic specifications. In *IEEE Conference on Decision and Control*, 2008.

[33] S. Karaman and E. Frazzoli. Vehicle routing with linear temporal logic specifications: Applications to multi-uav mission planning. In *AIAA Guidance, Navigation, and Control Conference*, 2008.

[34] S. Karaman and G. Inalhan. Large-scale task/target assignment for uav fleets using a distributed branch and price optimization scheme. In *IFAC World Congress*, 2008.

[35] S. Karaman, S. Rasmussen, D. Kingston, and E. Frazzoli. Planning complex UAV operations: A process algebra and tree search approach. In *American Control Conference*, 2009.

[36] S. Karaman, R.G. Sanfelice, and E. Frazzoli. Optimal control of mixed logical dynamical systems with linear temporal logic specifications. In *IEEE Conference on Decision and Control*, 2008.

[37] D.B. Kingston and C.J. Schumacher. Time-dependent cooperative assignment. In *American Control Conference*. IEEE, 2005.

[38] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, 53:287–297, 2007.

[39] M. Kloetzer and C. Belta. Temporal logic planning and control of robotic swarms by hierachical abstractions. *IEEE Transactions on Automatic Control*, 23(2):320–331, 2007.

[40] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2:255–299, 1990.

[41] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. Where's waldo? sensor-based temporal logic motion planning. In *IEEE Conf. on Robotics and Automation*, 2007.

[42] G. Laporte and Y. Robert. Exact algorithms for the vehicle routing problem. *Annals of Discrete Mathematics*, 31:147–184, 1987.

[43] J.K. Lenstra and A.R. Kan. Complexity of the vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

[44] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.

[45] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[46] H. Onal, B.M. Jaramillo, and M.A. Mazzoco. Two formulations of the vehicle routing problem: An emprical application and computational experience. *Logistics and Transportation Review*, 32:177–190, 1996.

[47] A. Pnueli. The temporal logic of programs. In *18th annual IEEE-CS Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[48] S.J. Rasmussen and D. Kingston. Assignment of heterogeneous tasks to a set of heterogenous unmanned aerial vehicles. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2008.

[49] S.J. Rasmussen and T. Shima. Tree search algorithm for assigning cooperating UAVs to multiple tasks. *International Journal of Robust and Nonlinear Control*, 18:135–153, 2008.

[50] A. Richards, Y. Kuwata, and J.P. How. Experimental demonstrations of real-time MILP control. In *Guidance, Navigation, and Control Conference*. AIAA, 2003.

[51] R. Ruiz, C. Maroto, and J. Alcaraz. A decision support system for a real vehicle routing problem. *European Journal of Operational Research*, 153:593–606, 2004.

[52] G. Salaun, L. Bordeaux, and M. Schaerf. Describing and reasoning on web services using process algebra. *International Journal of Business Process Integration and Management*, 1(2):116–128, 2006.

[53] P. Schnoebelen. The complexity of temporal logic model checking. In *4th Workshop on Advences in Modal Logics*, 2002.

[54] C. Schumacher, P. Chandler, M. Pachter, and L. Pachter. UAV task assignment with timing constraints via mixed-integer linear programming. In *AIAA 3rd Unmanned Unlimited Technical Conference, Workshop and Exhibit*. AIAA, 2004.

[55] C. Schumacher, P.R. Chandler, M. Pachter, and L.S. Patcher. Optimization of air vehicles operations using mixed-integer linear programming. *Journal of the Operational Research Society*, 58:516–527, 2007.

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

[57] J.G. Thistle and W.M. Wonham. Control problems in a temporal logic framework. *International Journal of Control*, 44(4):943–976, 1986.

[58] P. Toth and D. Vigo. Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, 123:478–512, 2002.

[59] P. Toth and D. Vigo. *The Vehicle Routing Problem*. SIAM, 2002.

[60] A.L. Weinstein and C. Schumacher. UAV scheduling via the vehicle routing problem with time windows. Technical Report AFRL-VA-WP-TP-2007-306, Air Force Research Laboratory, January 2007.

[61] H.P. Williams. Logic applied to integer programming and integer programming applied to logic. *European Journal of Operational Research*, 81:605–616, 1995.

[62] H.P. Williams and S.C. Brailsford. *Advances in Linear and Integer Programming*, chapter Computational Logic and Integer Programming. Oxford Science Publications, 1996.

[63] P. Wolper. Constructing automata from temporal logic formulas: A tutorial. *Lecture Notes in Computer Science*, 2090:261–277, 2001.