Cyclic Routing of Unmanned Aerial Vehicles

Nir Drucker

Cyclic Routing of Unmanned Aerial Vehicles

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Operations Research
and Systems Analysis (with thesis)

Nir Drucker

Submitted to the Senate
of the Technion — Israel Institute of Technology
Av 5774   Haifa   August 2014

This research was carried out under the supervision of Prof. Penn Michal and Prof. Strichman Ofer, in the Faculty of Industrial Engineering.

Publications

N. Drucker , M. Penn and O. Strichman, "Cyclic Routing of Unmanned Air Vehicles". ISMP (International Symposium on Mathematical Programming) 2012, Berlin, Germany, August 2012

N. Drucker , M. Penn and O. Strichman, "Cyclic Routing of Unmanned Air Vehicles". ORSIS (Operations Research Society - Israel) 2012, Israel, June 2012

N. Drucker , M. Penn and O. Strichman, "Cyclic Routing of Unmanned Air Vehicles". AUVSI (Association for Unmanned Vehicle Systems International) 2010, BGU, Israel, August 2010, http://ie.technion.ac.il/tech_reports/1393234936_AUVSI-Abstract-31Aug2010-submitted.pdf

Contents

List of Figures

Abstract

Many defense and civilian-related tasks targeted by Unmanned Aerial Vehicles (UAVs) are concerned with monitoring of a predefined set of ground targets under various timing constraints. In particular we are concerned with tasks in which each target is associated with a *relative deadline*, which means that there is an upper bound on the time between two consecutive scans of that target. Such constraints may be related to the nature of the target and the speed in which the client needs to react to a particular scenario. One may imagine a long border patrolled by UAVs, where certain sensitive locations are associated with a relative deadline that is defined by the speed in which ground forces can react to an event detected by the UAV operator; or a situation in which a military monitors enemy gatherings, attempting to detect various changes when they occur. Civilian applications may include monitoring of facilities and monitoring of forests for fire. In each such application the relative deadline is calculated according to the relative value of shortening the time to react versus the cost of additional UAVs.

The tasks discussed above are (seemingly endless) routines that can be solved with a cyclic plan. Only rarely it is necessary to deviate from such a plan. This stands in stark contrast to the common practice today of manually guiding the UAVs from the ground. Loading preplanned flight routes are supported by modern UAV systems, but no one as far as we know used this capability for planning optimal cyclic routes of fleets of UAVs. Automation of UAVs in various levels is an urgent need since the market, both the defense and civilian-related, is growing rapidly given the major progress in their capabilities and proven success in the last decade.

In this work we formally define the CR-UAV problem and prove a lower-bound on the number of required UAVs. This bound is useful for saving computation time, as it is easy to compute and avoids costly search that is bound to fail. We study several venues for solving this (NP-hard) problem. Specifically, we propose a model based on disjunctive MILP. We identify the set of constraints as belonging to the first-order theory of *difference constraints*, namely a Boolean combination of Boolean variables and constraints of the form $x - y \leq c$ where $x, y \in \mathbb{R}$ and c is a constant, and explain how they can be solved not only with MILP tools, but also with SMT (Satisfiability Modulo Theory) solvers. We present a DFS-based search algorithm that explores bounded cyclic paths; We show a DFS-based algorithm that explores a discretized version of the (otherwise continuous) state-space. Finally, we present the results of our extensive

empirical evaluation of these methods.

Abbreviations and Notations

AUV	:	Autonomous Underwater Vehicle
BDD	:	Binary Decision Diagram
CR-UAV	:	Cycle Routing of Unmanned Aerial Vehicle
CTL	:	Computation Tree Logic
DFS	:	Depth First Search
DPLL	:	Davis Putnam Logemann Loveland Algorithm
DTSP	:	Deadline Travelling salesman problem
FS	:	Feasible Solution
FT	:	Flight Time
GAV	:	Ground Aerial Vehicle
MILP	:	Mixed Integer Linear Problem
MTSP	:	Multiple Travelling salesman problem
PN	:	Polygon Number
RD	:	Relative Deadline
SAT	:	Short for Satisfiability
SMT	:	Satisfiability Modulo Theories
SMV	:	Symbolic Model Verifier
SN	:	Slot Number
ST	:	Scanning Time
TSP	:	Travelling Salesman Problem
TSP-TW	:	Travelling Salesman Problem with Time Window
UAV	:	Unmanned Aerial Vehicle
UN	:	UAVs Number
VRP	:	Vehicle Routing Problem
VRP-TW	:	Vehicle Routing Problem with Time Window

Chapter 1

Introduction

Many defense and civilian-related tasks targeted by Unmanned Aerial Vehicles (UAVs) are concerned with monitoring of a predefined set of ground targets under various timing constraints. In particular we are concerned with tasks in which each target is associated with a *relative deadline*, which means that there is an upper bound on the time between two consecutive scans of that target. Such constraints may be related to the nature of the target and the speed in which the client needs to react to a particular scenario. One may imagine a long border patrolled by UAVs, where certain sensitive locations are associated with a relative deadline that is defined by the speed in which ground forces can react to an event detected by the UAV operator; or a situation in which a military monitors enemy gatherings, attempting to detect various changes when they occur. Civilian applications may include monitoring of facilities and monitoring of forests for fire. In each such application the relative deadline is calculated according to the relative value of shortening the time to react versus the cost of additional UAVs.

A closely related problem is that of planning a cyclic *agent patrol* [BGA09]. It tackles the problem of finding a route for a robot patrolling an enclosed area. The relative deadlines are related to the time it takes an adversary to break in, in specific vulnerable locations along the cyclic path. The goal defined there is to find whether there exists a cyclic route for the patrolling agent such that no break can go undetected. The problem we define here — Cyclic Routing of Unmanned Aerial Vehicle (CR-UAV) — is very much related to that problem, but has slightly different constraints and different goals. The constraints are different because whereas the possible paths of the agent in [BGA09] are restricted because of physical constraints (modeling a scenario in which the patrolling agent is restricted to a rail), we model UAVs which have no such constraints. Finally, we have to account for the time it takes to scan each target. The goal is also different. Whereas the goal in [BGA09] is to check feasibility for a single agent, our goal is to find the minimal number of UAVs that are required in order to satisfy the constraints. It is obvious that dedicating a UAV for each target is sufficient, but we aspire to use less if possible. We require that a solution is accompanied by corresponding cyclic routes for the UAVs, each such route is not a simple route, i.e. a UAV can visit the same vertex several times.

The tasks discussed above are (seemingly endless) routines that can be solved with a cyclic plan. Only rarely it is necessary to deviate from such a plan. This stands in stark contrast to the common practice today of manually guiding the UAVs from the ground. Loading preplanned flight routes are supported by modern UAV systems, but no one as far as we know used this capability for planning optimal cyclic routes of fleets of UAVs. Automation of UAVs in various levels is an urgent need since the market, both the defense and civilian-related, is growing rapidly given the major progress in their capabilities and proven success in the last decade. As indicated in [Ome12]: "*The field of air-vehicle autonomy is a recently emerging field, whose economics is largely driven by the military to develop battle-ready technology. Compared to the manufacturing of UAV flight hardware, the market for autonomy technology is fairly immature and undeveloped. Because of this, autonomy has been and may continue to be the bottleneck for future UAV developments, and the overall value and rate of expansion of the future UAV market could be largely driven by advances to be made in the field of autonomy.*" Later in the same article it is pointed out that one of the categories of automation is "*determining an optimal path for vehicle to go while meeting certain objectives and mission constraints, such as obstacles or fuel requirements*". Somewhat related, concerning a review of the Pentagon for the 2011 budget it was noted in CNN that: "*The review also stresses learning better and more efficient ways to use the drones by improving operating effectiveness and using new technologies*" [CNN10].

In the next section we formally define the CR-UAV problem and prove a lower-bound on the number of required UAVs. This bound is useful for saving computation time, as it is easy to compute and avoids costly search that is bound to fail. In Sect. 5– 7.1 we study several venues for solving this (NP-hard) problem. Specifically, in Sect. 5 we propose a model based on disjunctive MILP. We identify the set of constraints as belonging to the first-order theory of *difference constraints* [KS08], namely a Boolean combination of Boolean variables and constraints of the form $x - y \leq c$ where $x, y \in \mathbb{R}$ and c is a constant, and explain how they can be solved not only with MILP tools, but also with SMT (Satisfiability Modulo Theory) solvers [KS08], which use propositional SAT engines to deal with the Boolean structure of such constraints. We will describe how SMT engines work in Sect. 8.1. In Sect. B we present a DFS-based search algorithm that explores bounded cyclic paths; the bound is based on a result by [BGA09] that shows that a solution exists[1] if and only if it exists up to a easily-computed bound. In Sect. 7.1 we show a DFS-based algorithm that explores a discretized version of the (otherwise continuous) state-space. Finally, we present the results of our extensive empirical evaluation of these methods in Sect. 8. Extended literature review can be found in Sect. 10.

[1]In their case for the agent-patrol problem, but it is relevant also to our problem.

Chapter 2

A formal definition of the CR-UAV problem

Let P be the set of target areas (we denote it by P because the area is typically a polygon, but later will be referred as a single point).

2.1 Assumptions

We make several assumptions:

1. When the solution includes more than one UAV, each UAV flies in a different altitude. This allows us to ignore the issue of intersecting routes that may otherwise lead to collisions.

2. Scanning an area $p \in P$ can be done from any point in p.

3. For each pair of targets $p, p' \in P$, the flight time between p and p' is constant. Whereas in reality this is not precisely true because of wind etc., we expect the input figures to include a certain slack to accommodate for such fluctuations. Hence, we can assume that the flight time between areas is given to us as a matrix of constants.[1]

4. For each $p \in P$, the scanning time is large enough to allow any route within p, including turns. This simplifies the problem in two ways:

    - Since this assumption permits us to enter and leave the target area from any location, we can require the flight time figures to refer to the shortest routes between the source and target areas;
    - We can represent each target area p as a point.

[1] This matrix is typically symmetric, but we do not pose this as an assumption since our suggested solutions do not rely on this fact.

5. The input data (e.g., the relative deadlines and the flight times) contains only integers or, equivalently, rationals. Clearly irrational flight times or relative deadlines are irrelevant in practice.

Since each target can be represented as a point, it is clear that we can view the CR-UAV problem as a graph problem. More specifically, it is a weighted, directed graph, with annotations at the vertices. The vertices are the targets of P, the weights on the arcs are the flight times and the annotations on the vertices are the relative deadlines. This view ignores the scanning time, but as we will show later (Sect. 2.3), these can be integrated in the flight times and ignored from thereon.

2.2 Examples

Some example problems appear in Fig. 2.1. The numbers near the vertices are the relative deadlines, and the numbers near the edges are flight times. Assume that in these problems the scanning time is 0[2]. In these examples the flight time in both directions is assumed to be identical, which explains why the graphs are undirected. Additional information about the solutions appear in the caption of the figure. Note that:

- in (a), there is no solution with one UAV following a simple cycle.
- in (b), there is no solution with two UAVs starting each at a point.
- in (c), there is no solution with the two UAVs having non-intersecting routes.

2.3 Problem inputs

In the rest of the article we refer to the elements of P not only as targets, but also as unique indices. Formally this duality can be avoided by defining a 1-to-1 function from a target area to an index, but we avoid it in order to keep the notation simple. We can now define the input to the CR-UAV problem:

1. **Scanning time**: An array ST of size $|P|$, such that for every $p \in P$, $ST[p]$ is the scanning time of p.
2. **Flight time**: A $|P| \times |P|$ matrix FT, such that for every pair $p, p' \in P$, $FT[p, p']$ is the Flying Time between p and p' (recall that by our assumption in Sect. 2.1, the flight time refers to the closest points in p, p').
3. **Relative deadline**: An array RD of size $|P|$, such that for every $p \in P$, $RD[p]$ is the maximum time allowed between consecutive scans of p, where consecutive scans defined from the time of finishing last scan to the time of finishing the next scan.

[2]As mentioned above, Sect. 2.3.1 shows that the problem can be reduced to one in which the scanning time is 0.

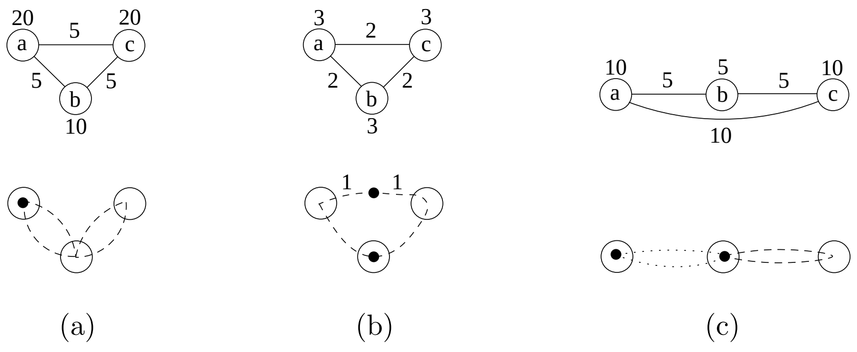


Figure 2.1: Three examples of the CR-UAV problem, and possible solutions for them at the bottom. The numbers above the vertices are the relative deadlines, and the numbers near the edges are flight times. Black circles at the bottom drawings denote the location of a UAV. In (a) the single UAV's route repeats a-b-c-b-... . In (b) both UAVs take the same route, flying in the same direction (e.g., clockwise), where one of them starts in the middle of the distance between areas a and c. In (c) the two UAVs have different routes (denoted by dotted and dashed lines, respectively) which intersect at point b.

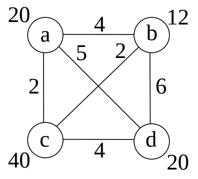


Figure 2.2: As before numbers adjacent to vertices represent relative deadlines, and numbers on edges represent flight-times.

We assume that for each target p, $FT[p,p] = 1$. In the realm of our assumption that the input data is integral (see assumption #5), this does not impose any constraint on the solutions, but simplifies the modeling.

2.3.1 Preprocessing of the input.

As a preprocessing step, we add the scanning-time to the flight time as follows. For each entry $FT[p,p']$ such that $p \neq p'$, we assign $FT[p,p'] + 0.5ST[p] + 0.5ST[p']$. Moving the 'cost' from the vertices to the edges simplifies the modeling later on and allows us to discard ST altogether. The following example demonstrates this transformation.

Example 2.3.1. Consider the following input, which is also depicted graphically in Fig. 2.2.

$$FT = \begin{pmatrix} 1 & 4 & 2 & 5 \\ 4 & 1 & 2 & 6 \\ 2 & 2 & 1 & 4 \\ 5 & 6 & 4 & 1 \end{pmatrix} \qquad ST = [2,4,6,8] \qquad RD = [20,12,40,20]$$

After the transformation, the FT matrix is:

$$FT = \begin{pmatrix} 1 & 7 & 6 & 10 \\ 7 & 1 & 7 & 12 \\ 6 & 7 & 1 & 11 \\ 10 & 12 & 11 & 1 \end{pmatrix}$$

For example, we added 3 to $FT[1, 2]$ because this is half of $(2 + 4)$, the accumulated scanning time of vertices a and b.

The time it takes to complete a cyclic route is equivalent before and after the transformation. For example, in Example 2.3.1 the cyclic route a,b,c takes (beginning from a) $4+4+2+6+2+2 = 20$ time units (note that this includes scanning time of all three target areas). Using the new matrix, the overall time is the same: $7 + 7 + 6 = 20$.

2.4 Objective

The objective is to find the minimal number n of UAVs and respective cyclic routes for each UAV, that satisfy the constraints.

Proposition 2.4.1. *The Primary-CR-UAV problem is NP-hard.*

Proof. Consider the Primary-CR-UAV problem on a complete graph $G = (P, E)$, with P the set of points and E the set of edges with an edge between any two points. In addition, let $l : E \to Q+$, be the length function and the RDs being identical, that is, $RD(v) = RD(\hat{v}) = k$, $\forall p \in P$. Let T be a TSP tour in G, that is T is a tour that visits all points of G, then its length is the sum of the lengths of the edges in the tour. Then, T is of length $\leq k$, iff there is a feasible solution for the Primary-CR-UAV problem with one UAV, that is $|U| = 1$ is an optimal solution. This is since if $|U| = 1$ then, for each $p \in P$, the time between any two consecutive visits to p is at most k, and thus the length of the tour is at most k. The other direction, if the length of the tour is at most k, then this tour is a feasible solution of the Primary-CR-UAV problem with $|U| = 1$. Thus, since the TSP is NP-hard, we obtain that the Primary-CR-UAV problem is NP-hard as well □

Chapter 3

A lower-bound on the number of UAVs

Let U denote the set of UAVs required for a solution. We now show a lower bound on the size of U, which is denoted by $|U|$, .

We define the following notation. For a target $v \in V$, let

$$FT_{min}(v) = min_{\substack{\hat{v}\in V \\ \hat{v}\neq v}}\{FT[v, \hat{v}]\} \; . \tag{3.1}$$

In words, $FT_{min}(v)$ denotes the minimal weight on any outgoing edge of v. We use this notation to define:

Definition 3.0.2 (Isolated vertex)**.** A vertex $v \in V$ is *isolated* if $RD[v] \leq FT_{min}(v)$.

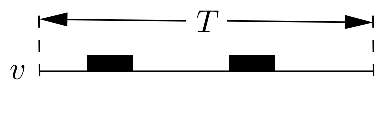
Intuitively, an isolated vertex is one that leaving it takes more time than its relative dead-line. Let $I \subseteq V$ denote the subset of isolated vertices.

We claim that:

Proposition 3.0.3. *A lower bound on* $|U|$ *is given by*

$$|I| + \left\lceil \sum_{v\in(V\setminus I)} \frac{FT_{min}(v)}{RD[v]} \right\rceil \leq |U| \; . \tag{3.2}$$

Proof. Let $T > 0$ be the time interval corresponding to a solution. Let $T_{sl}(v) \leq T$ be the total time spent at vertex v on self-loops. The figure below depicts such a time interval, where the boxes symbolize the time in which some UAV (not necessarily the same one) looped at v. The accumulated length of the boxes is $T_{sl}(v)$.



The number of UAV entries to v during T must be at least

$$\left\lceil \frac{T - T_{sl}(v)}{RD[v]} \right\rceil \; , \tag{3.3}$$

and hence the total flight time dedicated to v must be at least

$$\left\lceil \frac{T - T_{sl}(v)}{RD[v]} \right\rceil \cdot FT_{min}(v) + T_{sl}(v) \; . \tag{3.4}$$

The overall flight time is given by aggregating (3.4) over V:

$$\sum_{v \in V} \left(\left\lceil \frac{T - T_{sl}(v)}{RD[v]} \right\rceil \cdot FT_{min}(v) + T_{sl}(v) \right) \; . \tag{3.5}$$

This term must be lower that or equal to the total flight time of all UAVs during T, which is given by $T \cdot |U|$:

$$\sum_{v \in V} \left(\left\lceil \frac{T - T_{sl}(v)}{RD[v]} \right\rceil \cdot FT_{min}(v) + T_{sl}(v) \right) \leq T \cdot |U| \; . \tag{3.6}$$

We now separate the elements in the sum on the left according to whether $v \in I$:

$$\begin{aligned} &\textstyle\sum_{v \in I} \left(\left\lceil \frac{T - T_{sl}(v)}{RD[v]} \right\rceil \cdot FT_{min}(v) + T_{sl}(v) \right) + \\ &\textstyle\sum_{v \in (V \setminus I)} \left(\left\lceil \frac{T - T_{sl}(v)}{RD[v]} \right\rceil \cdot FT_{min}(v) + T_{sl}(v) \right) \quad \leq T \cdot |U| \; . \end{aligned} \tag{3.7}$$

Let us focus on the first summation: since this expression is monotone in $T_{sl}(v)$ and $0 \leq T_{sl}(v) \leq T$ whereas the other variables are fixed, it is not hard to see that its value is in the range

$$\sum_{v \in I} T \ldots \sum_{v \in I} \left\lceil \frac{T}{RD[v]} \right\rceil \cdot FT_{min}(v) \; . \tag{3.8}$$

Hence the first sum in (3.7) can be lowered to $T \cdot |I|$, which gives us

$$T \cdot |I| + \sum_{v \in (V \setminus I)} \left(\left\lceil \frac{T - T_{sl}(v)}{RD[v]} \right\rceil \cdot FT_{min}(v) + T_{sl}(v) \right) \leq T \cdot |U| \; . \tag{3.9}$$

Furthermore, the second summation is larger than

$$\sum_{v \in (V \setminus I)} \frac{T - T_{sl}(v)}{RD[v]} \cdot FT_{min}(v) + T_{sl}(v) \; , \tag{3.10}$$

(note that we removed the ceiling operator), which can be rewritten into

$$\sum_{v \in (V \setminus I)} \left(\frac{T \cdot FT_{min}(v)}{RD[v]} + T_{sl}(v) \cdot \left(1 - \frac{FT_{min}(v)}{RD[v]} \right) \right) \; . \tag{3.11}$$

Note that by Definition 3.0.2, for every $v \in (V \setminus I)$ it holds that $\frac{FT_{min}(v)}{RD[v]} \leq 1$, which implies that the right operand is positive and consequently (3.11) is larger than

$$\sum_{v \in (V \setminus I)} \frac{T \cdot FT_{min}(v)}{RD[v]} \ . \tag{3.12}$$

Hence, based on(3.9) we have that

$$T \cdot |I| + \sum_{v \in (V \setminus I)} \frac{T \cdot FT_{min}(v)}{RD[v]} \leq T \cdot |U| \ . \tag{3.13}$$

Dividing by T and rounding up gives us the lower bound on $|U|$ as promised in the proposition:

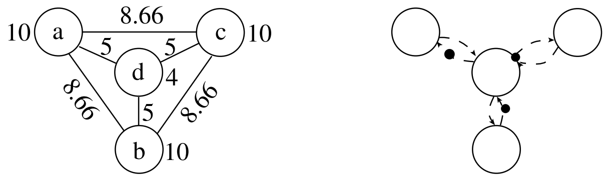
$$|I| + \left\lceil \sum_{v \in (V \setminus I)} \frac{FT_{min}(v)}{RD[v]} \right\rceil \leq |U| \ . \tag{3.14}$$

The bound is tight.

Each of the three examples in Fig. 2.1 requires as many UAVs as specified by (). The examples are small enough to see that they cannot be solved with a smaller number of UAVs. Specifically for the right-most example, the center vertex is the only isolated vertex, and the lower bound is given by $1 + \lceil (\frac{5}{10} + \frac{5}{10}) \rceil = 2$.

Covering isolated vertices

Definition 3.0.2 may tempt the reader to think that in an optimal solution a UAV should be dedicated to each isolated vertex. But the following example proves that this is not the case (the distances on the arcs approximately correspond to a metric). The center vertex (d), which has a relative deadline of 4, is isolated. Dedicating a UAV to it would also force us to dedicate a UAV for each of the other three vertices, hence requiring four UAVs all together. The suggested solution on the right, on the other hand, is based on three UAVS. Each of them cycles between a vertex on the perimeter and d, and they arrive to d at equal gaps of $\frac{10}{3}$ time-units.



Chapter 4

Considering integral solutions is sufficient

We now prove that it is sufficient to consider solutions in which the initial state positions of the UAVs are at integral flight-times from their first destination. The proof is based on assumption#5 in Sect. 2.1, that flight times and relative deadlines are integers.

A state of the system is a vector of size $|P| + 2|U|$, with $|P| = n$ and $|U| = m$, of the form

$$(t_{RD}(p_1), \ldots, t_{RD}(p_n), p(u_1), \ldots p(u_m), t_D(u_1), \ldots t_D(u_m))$$

where $t_{RD}, p(u)$ and t_D are defined as follows:

1 **Residual Relative Deadline** The residual relative deadline $t_{RD}(p)$, is the time left to satisfy the deadline constraint of vertex p at the current state of the system. The residual relative deadlines change dynamically.

2 **Arrival Time to Destination** $t_D(u)$ indicates the residual time required for u to reach its current destination $p(u)$ at the current state (note that u can be at any point along the edge $(p, p(u))$).

3 **Destination** $p(u)$ indicates the current destination of UAV $u \in [1..|U|]$, implying that currently u is using the edge $(p, p(u))$ for some $p \in P$.

For any $x \in R^+$, let $fr(x) = x - \lfloor x \rfloor$ denote the fraction of x. The following theorem states that if there is a solution to the CR-UAV problem, then there is also a solution in which $fr(t_D(u))$ is equal for all $u \in [1..|U|]$, with the same number of UAVs:

Theorem 4.1. *Assume that there exists a feasible solution FS to the CR-UAV problem. Then there also exists a feasible solution $\widehat{FS}$ with the same number of UAVs, and with starting state $\widehat{FS}_{start}$ such that $fr(\widehat{t_D}(u))$ is equal for all $u \in [1..|U|]$ in $\widehat{FS}_{start}$.*

Proof. let $uu = arg\max_{u \in 1..|U|} \{fr(t_D(u))\}$ Given a solution FS with starting state FS_{start}, we construct $\widehat{FS}_{start}$ from FS_{start} by setting $\forall u \in 1..|U|$: $\widehat{t_D}(u) = \lfloor t_D(u) \rfloor +$

$fr(t_D(uu))$. In other words, we increase the fractional part of $t_D(u)$ to the maximal fraction of $t_D(w)$ for $w \in [1..|U|]$. We will now prove by negation that $\widehat{FS}$ is a feasible solution, i.e., $\forall p.\ \widehat{t_{RD}}(p) \geq 0$ in every state of $\widehat{FS}$. If $\widehat{FS}$ is not feasible then there is at least one vertex p such that the (nonuniform) change in the values of $t_D(u)$ made a UAV miss the deadline at p. Let u_2 be such a UAV, and let u_1 denote the last UAV to visit p on time before u_2, the following visit, visited it in an untimely manner. The following claim implies that this is impossible, because the extra time left for u_2 upon reaching p is always greater or equal to the possible increase in the gap between u_2 and u_1.

In the following let $\Delta fr(u_1, u_2)$ denote the change in the gap between $t_D(u_1)$ and $t_D(u_2)$ owing to the changes in the fractions of their respective t_D values, i.e.,

$$\forall u_1, u_2 \in 1..|U| : \Delta fr(u_1, u_2) = \begin{cases} fr(t_D(u_1)) - fr(t_D(u_2)) & fr(t_D(u_2)) \leq fr(t_D(u_1)) \\ 1 + fr(t_D(u_1)) - fr(t_D(u_2)) & Otherwise\ . \end{cases}$$

Note that only the first case (when $fr(t_D(u_2)) \leq fr(t_D(u_1))$) is potentially problematic, because only in that case the gap between $t_D(u_1)$ and $t_D(u_2)$ is potentially larger in $\widehat{FS}$ then it is in FS.

Claim 4.0.4. *Let FS_s be the state where u_2 is at p and u_1 is the previous UAV that visited p. Then in FS_s*

$$t_{RD}(p) \geq \Delta fr(u_1, u_2)\ .$$

Proof. By definition of FS_s, vertex p was last visited by u_1 before $CL(p) = RD_p - t_{RD}(p)$ units of time, where $CL(p)$ stands for the clock at p at the current state (we consider $t_{RD}(p)$ upon arrival at p, before it is reset. At that point $t_{RD}(p) \geq 0$ since FS is a feasible solution). By definition

$$fr(CL(p)) = ((RD_p - t_{RD}(p)) - \lfloor RD_p - t_{RD}(p) \rfloor)\ ,$$

which, because RD_p is assumed to be an integer and assuming $t_{RD}(p)$ is not an integer, is equal to

$$\begin{aligned} &= (RD_p - t_{RD}(p)) - (RD_p - \lfloor t_{RD}(p)) \rfloor - 1) \\ &= \lfloor t_{RD}(p) \rfloor - t_{RD}(p) + 1\ . \end{aligned}$$

We know that $fr(t_D(u_2)) = 0$ because u_2 is at p. in addition, since the flight time is also an integer, then

$$\begin{aligned} fr(t_D(u_1)) &= 1 - fr(CL(p)) = \\ &= 1 - \lfloor t_{RD}(p) \rfloor + t_{RD}(p) - 1 = \\ &= t_{RD}(p) - \lfloor t_{RD}(p) \rfloor \ . \end{aligned}$$

If $t_{RD}(p)$ is an integer, then $fr(CL(p)) = 0$ and $fr(t_D(u_1)) = 0$. Therefore, since $fr(t_D(u_1)) \geq 0$ and $fr(t_D(u_2)) = 0$, we have that

$$\Delta fr(u_1, u_2) = fr(t_D(u_1)) - fr(t_D(u_2)) = t_{RD}(p) - \lfloor t_{RD}(p) \rfloor \ .$$

If the claim is false, then

$$\Delta fr(u_1, u_2) = t_{RD}(p) - \lfloor t_{RD}(p) \rfloor > t_{RD}(p) \ ,$$

implying

$$- \lfloor t_{RD}(p) \rfloor > 0 \ ,$$

which is a contradiction. Hence the claim must hold. (Proof of Claim 4.0.4)

From Claim 4.0.4 $t_{RD}(p) \geq \Delta fr(u_1, u_2)$ at FS_s, and therefore $t_{RD}(p) \geq \Delta fr(u_1, u_2)$ at each state between the visits of u_1 and u_2 at p (because $t_{RD}(p)$ is constantly decreasing). Hence setting $\widehat{t_D}(u_2) = t_D(u_2) + \Delta fr(u_1, u_2)$ will never result in a state in which $\widehat{t_{RD}}(p) < 0$, implying that $\widehat{FS}$ is feasible. (Proof of Theorem 4.1) □

Independently of the previous result, we now claim that adding to $t_D(u_1)$ and to $t_D(u_2)$ the same number does not change the feasibility of the solution:

Theorem 4.2. *Assume that there exists a feasible solution FS for which $fr(t_D(u))$ is equal for all u in FS_{start}. Then there exists a solution $\widehat{FS}$ with starting state $\widehat{FS}_{start}$ such that $\forall u \in 1..|U|.\ t_D(u) = \lceil t_D(u) \rceil$.*

Proof. Since there exists a value $0 < x \leq 1$ such that $fr(t_D(u)) = x$ for all u, then clearly the solution obtained by increasing each $t_D(u)$ by $1 - x$ remains feasible. (Proof of Theorem 4.2) □

Theorems 4.1 and 4.2 imply:

Corollary 4.3. *Given a feasible solution FS, a solution $\widehat{FS}$ with a starting state $\widehat{FS}_{start}$ such that $\forall u \in 1..|U|.\ \widehat{t_D}(u) = \lceil t_D(u) \rceil$ is also feasible.*

Chapter 5

A constraints model

Our modeling of the CR-UAV problem can be depicted with an array of size SN, where each entry is called a *slot*. Each such slot represents a visit to a vertex. The value of SN represents the length of the route to be repeated indefinitely. Since we do not know this length in advance, solution strategies based on this model must search for a route starting with $SN = |P|$ and then increase it if a solution is not found. Since we do not have an upper-bound for SN, this method is *incomplete*, i.e., it is not guaranteed to terminate. Practically, in our experiments, we decide on some bound a-priory but if there is no solution up to that bound then we cannot know if it is because there is no solution or because the bound is not high enough. In contrast, in Sect. 6 we will introduce a *complete* method, which is not based on mathematical programming.

In the next section we show how the slots model can be used to solve the related satisfiability problem for a single UAV, i.e., a solution implies that a single UAV satisfies the input problem. In Sect. 5.2 we will extend it to multiple UAVs.

5.0.1 Symbols definition

Two symbols which are widely used in model checking is defined below:

- $\bigwedge_{i\in 1..4}(X_i) = X_1 \wedge X_2 \wedge X_3 \wedge X_4$
- $\bigvee_{i\in 1..4}(X_i) = X_1 \vee X_2 \vee X_3 \vee X_4$

5.1 A model for a single UAV

The decision variables are:

- $O_{i,j}$: Boolean – for $i \in [1..SN]$, $j \in [1..P]$, $O_{i,j}$ is true if and only if in slot i the UAV entered vertex j.
- S_i: Real – for $i \in [1..n]$ denotes the entry time to slot i.

The constraints are:

- Exactly one vertex is associated with each slot:

$$\forall i \in [1..SN], p \in P.\ O_{i,p} \implies \bigwedge_{\substack{\hat{p} \in P \\ \hat{p} \neq p}} \neg O_{i,\hat{p}}\ . \tag{5.1}$$

$$\forall i \in [1..SN].\ \bigvee_{p \in P} O_{i,p}\ . \tag{5.2}$$

- Defining the accumulated time:

$$\forall i \in [1..SN], p_1 \in P, p_2 \in P.\ O_{i,p_1} \wedge O_{i+1,p_2} \implies S_{i+1} = S_i + FT[p_1, p_2]\ . \tag{5.3}$$

- Defining S_1:

$$\forall p_1 \in P, p_2 \in P.\ O_{SN,p_1} \wedge O_{1,p_2} \implies S_1 = FT[p_1, p_2]\ . \tag{5.4}$$

- Time between visits to the same vertex:

$$\begin{array}{lr} \forall p \in P, i \in [1..SN]. & \\ \big(\bigvee_{l=1}^{i-1} O_{l,p} \wedge (S_i - S_l \leq RD[p])\big)\vee & \textit{visited p in an earlier slot} \\ \big(\bigvee_{l=i+1}^{SN} O_{l,p} \wedge (S_i + S_{SN} - S_l \leq RD[p])\big)\vee & \textit{visited p in a later slot} \\ \big(O_{i,p} \wedge \bigwedge_{l=1,l\neq i}^{SN} \neg O_{l,p} \wedge S_{SN} \leq RD[p]\big) & \textit{visited p only in slot i} \end{array} \tag{5.5}$$

5.2 Multiple UAVs

A generalization of the solution given in Sect. 5.1 to multiple UAVs solves indirectly the primary objective as stated in Sect. 2.4, because one only needs to gradually increase the number of UAVs until a solution is found. Recall that there is always a solution with $|P|$ UAVs, which means that this process is guaranteed to terminate. However, since the solution for a given number of UAVs is incomplete, as explained in Sect. 5.1, then it is possible that our solution is not optimal since the search with a lower number of UAVs was stopped prematurely.

In order to generalize the model to multiple UAVs, we require that at each slot at least one UAV is reaching a new vertex, whereas other UAVs can be between vertices. For that we define a new variable $A_{u,i}$ that holds the time to destination i of UAV u. In contrast to the single UAV model, here a UAV u can have a route which contains only one vertex where $\forall i \in [1..SN] : A_{u,i} = 0$.

Additional variables for the multiple UAVs model:

- $\forall u \in U, i \in [1..SN], p \in P.\ O_{u,i,p}$ Boolean: $O_{u,i,p} = 1 \iff$ in slot i UAV u enters vertex p.
- $\forall u \in U, i \in [1..SN].\ A_{u,i}$: Time left for UAV u to reach its new destination, when at slot i.

The constraints are:

- At least one UAV should enter a vertex in each slot:

$$\forall i \in [1..SN].\ \bigvee_{\substack{u \in U \\ p \in P}} O_{u,i,p}\ . \tag{5.6}$$

- Each UAV can visit only one vertex at each slot:

$$\forall u \in U, i \in [1..SN], p \in P.\ O_{u,i,p} \implies \bigwedge_{\hat{p} \in \{P \setminus \{p\}\}} \neg O_{u,i,\hat{p}}\ . \tag{5.7}$$

- Consistency of the O variables when a UAV stays at a target:

$$\forall u \in U, i \in [1..SN], p \in P.\ A_{u,i} = 0 \implies (O_{u,i,p} \iff O_{u,i+1,p})\ . \tag{5.8}$$

- If a UAV u visits vertex p at time slot i then there is no other UAV $\hat{u}$ that reaches its vertex $\hat{p}$ before u visits p. An exception is when $\hat{u}$ stays at its current location:

$$\begin{aligned} &\forall u \in U, i \in [2..SN], p \in P, \hat{u} \in U, \hat{u} \neq u. \\ &(O_{u,i,p} \land (A_{\hat{u},i-1} \neq 0)) \implies A_{u,i-1} \leq A_{\hat{u},i-1}\ . \end{aligned} \tag{5.9}$$

- Same as above, for the first slot:

$$\forall u \in U, p \in P, \hat{u} \in U, \hat{u} \neq u.\ (O_{u,1,p} \land (A_{\hat{u},SN} \neq 0)) \implies A_{u,SN} \leq A_{\hat{u},SN}\ . \tag{5.10}$$

- At least one UAV progresses to a different target, from slot i to $i+1$:

$$\forall i \in [1..SN-1], u \in U.(A_{u,i} > 0) \implies \textstyle\bigvee_{p \in P} O_{u,i+1,p}\ . \tag{5.11}$$

- Same as above, for the first slot:

$$\forall u \in U.(A_{u,SN} > 0) \implies \textstyle\bigvee_{p \in P} O_{u,1,p}\ . \tag{5.12}$$

- S_1 is non-negative (the values of other S_i variables will be larger owing to the constraints that follow):

$$S_1 \geq 0\ . \tag{5.13}$$

- S_i progresses according to a UAV that does not stay at a target:

$$\begin{aligned} &\forall u \in U, i \in [2..SN], p \in P. \\ &O_{u,i,p} \land (A_{u,i-1} > 0) \implies S_i = S_{i-1} + A_{u,i-1}\ . \end{aligned} \tag{5.14}$$

- Same, for the first slot:

$$\begin{gathered}\forall u \in U, p \in P. \\ O_{u,1,p} \wedge A_{u,SN} > 0 \implies S_1 = S_{SN} + A_{u,SN} \ .\end{gathered} \tag{5.15}$$

- If a UAV visits p_1 at slot i and p_2 at slot j and does not visit any other vertex in between, then the time to arrive at the destination should be set to the flying time between p_1 and p_2:

$$\begin{gathered}\forall u \in U, i \in [1..SN], p \in P, \hat{i} \in [i+1..SN], \hat{p} \in P. \\ (O_{u,i,p} \wedge O_{u,\hat{i},\hat{p}} \wedge (\neg \bigvee_{\substack{p_2 \in P \\ mid \in [i+1..\hat{i}-1]}} O_{u,mid,p_2})) \implies A_{u,i} = FT_{p,\hat{p}} \ .\end{gathered} \tag{5.16}$$

- If a UAV visits p at slot i and p_2 at slot j and does not visit any other vertex after slot time j and before slot time i, then the arrival time should be set to the flying time between p_2 and p:

$$\begin{gathered}\forall u \in U, i \in [1..SN], p \in P, \hat{i} \in [i+1..SN], \hat{p} \in P. \\ (O_{u,i,p} \wedge O_{u,\hat{i},\hat{p}} \wedge (\neg \bigvee_{\substack{p' \in P \\ mid \in [1..i-1] \cup [\hat{i}+1..SN]}} O_{u,mid,p'})) \implies A_{u,\hat{i}} = FT_{\hat{p},p} \ .\end{gathered} \tag{5.17}$$

- If a UAV visits p at slot i not via a self edge, then for each UAV $\hat{u}$, $A_{\hat{u},i}$ is equal to the difference between $A_{\hat{u},i-1}$ and $A_{u,i-1}$:

$$\begin{gathered}\forall u, \hat{u} \in U, \hat{u} \neq u, i \in [2..SN], p \in P. \\ (O_{u,i,p} \wedge ((A_{\hat{u},i-1} \neq 0)) \wedge (A_{u,i-1} \neq 0)) \wedge (A_{u,i-1} \neq A_{\hat{u},i-1})) \\ \implies A_{\hat{u},i} = A_{\hat{u},i-1} - A_{u,i-1} \ .\end{gathered} \tag{5.18}$$

- Same, for the first slot:

$$\begin{gathered}\forall p \in P, u, \hat{u} \in U, \hat{u} \neq u. \\ (O_{u,1,p} \wedge (\neg(A_{\hat{u},SN} = 0)) \wedge (\neg A_{u,SN} = 0)) \wedge (A_{u,SN} \neq A_{\hat{u},SN})) \\ \implies A_{\hat{u},1} = A_{\hat{u},SN} - A_{u,SN} \ .\end{gathered} \tag{5.19}$$

- Time between visits to the same vertex:

$$\begin{aligned}&\forall p \in P, i \in [1..SN]. \\ &(\textstyle\bigvee_{l=1}^{i-1} (\bigvee_{u \in U} O_{u,l,p} \wedge S_i - S_l \leq RD[p]))\vee && \textit{visited p in an earlier slot} \\ &(\textstyle\bigvee_{l=i+1}^{SN} (\bigvee_{u \in U} O_{u,l,p} \wedge S_i + S_{SN} - S_l \leq RD[p]))\vee && \textit{visited p in a later slot} \\ &(\textstyle\bigvee_{u \in U} O_{u,i,p} \wedge \bigwedge_{\hat{u} \in U} \bigwedge_{l=1, l \neq i}^{SN} \neg O_{\hat{u},l,p} \wedge S_{SN} \leq RD[p]) && \textit{visited p only in slot i}\end{aligned} \tag{5.20}$$

Chapter 6

Modeling CR-UAV as a finite state system

Whereas the CR-UAV problem is defined via continuous variables, which inherently define an infinite state-space, we show that it can be solved by searching a *finite* number of states, under the assumption that the input data (i.e., the relative deadlines RD and the flight times FT) contains only integers, or, equivalently, rationals. This gap is bridged by making the following two observations:

1. The value of the clock at each target is bounded from both sides. Specifically, for each $p \in P$, $t_{RD}(p) \leq RD[p]$, where, recall, $t_{RD}(p)$ is the time left at target p, and $RD(p)$ is the relative deadline at p. For our purpose, $t_{RD}(p)$ is also bounded from below by 0.

2. The search space can be restricted to the states in which at least one UAV is arriving or departing at a point: these are the only states in which a routing decision has to be made. In other words, states in which all UAVs are between targets or doing self arc (self loop), can be ignored.

Recall (chapter 4) that a state s is defined as a tuple with $|P| + 2|U|$ elements as follows:

1. For $p \in P$, $t_{RD}(p)$ is the time left to meet p's relative deadline.

2. For $u \in U$, $p(u) \in P$ is the destination of u.

3. For $u \in U$, $t_D(u)$ is the time left for u to reach its next target $p(u)$.

The initial state is such that $t_{RD}(p)$ and $p(u)$ are clearly integers. As for $t_D(u)$, if u is initially at a point then $t_D(u)$ must be an integer because all the flight times are integers; otherwise it can be restricted to begin on a path between two vertices in an integral distance from its destination, because no solutions are lost this way (see Corollary 4.3 in Sect. 4). A transition between states only happens at an arrival of one or more UAVs to their target(s) or departure of one UAV from a target(s). But Since the flight-times are assumed to be integers, all consecutive states are defined by integers as well. Hence all the variables are discrete and bounded, which implies that our problem can be modeled with a finite state system.

We explored two methods to find cyclic routes in such a system:

1. A DFS-style search for a cyclic route that satisfies the constraints. This is a simple DFS in the finite search space as defined above. The algorithm, as well as several optimization can be found in appendix B.

2. Reduction to a finite-state *symbolic model checking* problem. Described in details in a future article.

Chapter 7

Other models

This chapter presents two additional models. The results of those models will not be compared to any other model due to the fact that any of them did not succeed (or constructed) to solve problems with more than one UAV.

1. Timed automata and its model are described in more details in Appendix A.

2. A search in a discretized space - will be described below.

7.1 A search in a discretized space

In different from modeling the CR-UAV as a finite state system we present below a search algorithm in a discretized space. Algorithm 7.1 is based on the same *slot* model defined in chapter 5 and is also incomplete. Moreover, it constructed to support only one UAV. We present it here because in some cases, it was faster than the other methods. In chapter 8 we will present a comparison between it and the SMT (Z3) method. The algorithm is based on the *DFS* algorithm and when it finds a valid solution it stops.

Algorithm 7.1 DFS based Slot model for the CR-UAV problem

```
1: for all $p \in P$ do
2:     $path[0] \leftarrow p$
3:     $acumTime[0] \leftarrow 0$
4:     DFS($acumTime[]$, $path[]$, 0)

1: function DFS($acumTime[]$, $path[]$, $curSlotIdx$)
2:     if not checkConstraints($accumTime[]$, $path[]$) then
3:         return false
4:     if $curSlotIdx = SN$ then
5:         return true
6:     for all $p \in P$ do
7:         $flightTime \leftarrow FT[path[curSlotIdx]][p]$
8:         $acumTime[curSlotIdx + 1] \leftarrow acumTime[curSlotIdx] + flightTime$
9:         $path[curSlotIdx + 1] \leftarrow p$
10:        if dfs($acumTime[], curSlotIdx + 1$) is true then
11:            return true
12:    return false

1: function CHECKCONSTRAINTS($acumTime[], path[], curSlotIdx$)
2:     for all $p \in P$ do                              ▷ Check that v was visited within the RD(p)
3:         $last \leftarrow (-1)$
4:         $first \leftarrow (-1)$
5:         for $i = 1$ to $curSlotIdx$ do
6:             if $path[i] \neq v$ then                      ▷ Checking only cycles of p
7:                 continue
8:             if $last \neq (-1)$ and $acumTime[i] - acumTime[last] \geq RD(v)$ then
9:                 return false  ▷ The route time between two consecutive visits is too
   large.
10:            if $last = (-1)$ then
11:                $first \leftarrow i$
12:            $last \leftarrow i$
13:        if $curSlotIdx \neq SN$ then
14:            return true                               ▷ no deadline problem till now.
15:        if $last = (-1)$ then
16:            return false                  ▷ Point $p$ does not appear in the solution.
17:        if $last = first$ and $acumTime[SN - 1] > RD(p)$ then
18:            return false     ▷ Point $p$ visited only once and its deadline is violated.
19:        if $last \neq first$ and
20:            $acumTime[SN-1]-acumTime[last]+acumTime[first] > RD(p)$ then
21:            return false  ▷ Cyclic returning to point p does not holds the deadline.
22:    return true
```

Chapter 8

Experimental results

In describing the engines that we evaluated for solving the CR-UAV problem, we distinguish between incomplete and complete methods.

8.1 Incomplete methods

We experimented with several engines for solving the constraints model of Sect. 5.

- *MILP.* The MILP solver MOSEK (Tool with API's in MATLAB).
- *Satisfiability Modulo Theories (SMT) solvers.* To solve the constraint programming model described in Sect. 5, we must bound the number of slots a-priory. For the experiments we chose the bound given by the longest relative deadline divided by the shortest flight time, rounded up. We used a Satisfiability Modulo Theories (SMT) solver to solve the model. Satisfiability Modulo Theories (SMT) [KS08] is an extension of the classical propositional satisfiability problem to other decidable first-order theories, i.e., in addition to propositional variables the formula can contain predicates of some decidable theory T. For example, if T is linear arithmetic, then a formula such as $2x + 3y > 5 \vee \neg(3y - 5z \geq 6) \wedge (x - y < z)$ is a T formula. A standard framework to solve such formulas is called DPLL(T). It combines a propositional SAT solver (hence the name DPLL[1]), and a solver for a conjunction of T predicates, e.g., in the case of T being linear arithmetic that solver can be based on Simplex. This combination is far better than 'case splitting' (transforming the formula to disjunction normal form), because it enjoys SAT's capabilities to prune large parts of the search space by applying *learning* (adding constraints during the solution process, that block search paths that are known not to contain a solution) and other techniques that are known to be very effective in dealing with propositional formulas. There are several dozen SMT solvers and an annual competition between them called SMT-COMP. We experimented with two such solvers, YICES [DM06] and Z3 [dMB08].

[1]DPLL stands for the name of the authors in [DP60, DLL62].

We only report on the results of Z3, since the other two (Yices and Mosek) were completely dominated by the results of Z3 in terms of run-time.

8.2 Complete methods

We tried two complete methods, one symbolic and one explicit.

- *Symbolic model checker.* To solve the model described in Sect. 6 we used a symbolic model checker called Cadence-SMV. This tool has seven different engines, and each has its own set of parameters. Unfortunately most of these engines are only available in the commercial version to which we do not have access. We therefore only experimented with the default engine in the academic release of the tool, which is based on Binary Decision Diagrams (BDDs), and did not change the default parameters of this engine.

- *Clocked DFS.* The Clocked DFS is a simple DFS in the finite search space as defined in chapter 6. The algorithm, as well as several optimization can be found in Appendix B.

8.3 Over- and under- approximations

It is obvious that given a CR-UAV problem one can multiply all the relative deadlines and all the flight time by any constant fraction γ, and as long as the resulting figures are integers the new problem is isomorphic to the original one. We wanted to test, however, what happens if multiplying by γ results in fractions, and then we round the result in a way that guarantees either an over or under approximation (but not both). Keeping the approximation single-sided enables us to know when the answer can be trusted: in an overapproximating model we convert the problem to be easier and then we can only trust UNSAT answers (if a problem is UNSAT in an easier version of it, it is defiantly UNSAT in the harder version), and in an underapproximating model we convert the problem to be harder and then we can only trust SAT answers. To produce an overapproximating model we round up the relative deadlines, and round down the flight time. To produce an underapproximating model we do the opposite. The question is what is the price we pay in terms of correctness, and what is the benefit in run time. The results below include answers to these two questions.

8.4 Results

We generated 600 random input problems, with varying topologies, flight times, relative deadlines and number of UAVs.

8.4.1 input problem parameters

- Number of points was varying between four and seven.
- Number of UAVs was varying between one and three.
- Five different methods were defined in order to calculate the RD for each point:
  - EQUAL_BY_MAX_PATH - $\forall p \in P : RD_p$ defined to be equal to the maximum simple path available in the current graph.
  - EQUAL_BY_MIN_PATH - $\forall p \in P : RD_p$ defined to be equal to the minimum simple path available in the current graph.
  - EQUAL_BY_AVG_PATH - $\forall p \in P : RD_p$ defined to be equal to the average simple path available in the current graph.
  - EQUAL_TO_MAX_TO_ARC - $\forall p \in P : RD_p$ defined to be equal to the maximum arc coming ut of p.
  - EQUAL_TO_MIN_TO_ARC - $\forall p \in P : RD_p$ defined to be equal to the minimum arc coming ut of p
- Six different topologies defined:
  - Line - All points are ordered in one linear line.
  - Two Groups - All points are ordered in two groups where the groups are far but not a lot from one another.
  - Three Groups - All points are ordered in three groups where the groups are far but not a lot from one another.
  - Isolated location - All points are grouped together except for one polygon which is isolated.
  - Mess - one big group of points all very closed but with no specific order.
  - Cycle - All points are ordered in a cycle shape.

Total of 300 input problems. We conducted each test case twice to prevent measurements errors. We also generated over- and under-approximated versions of these problems as explained above, with $\gamma = 0.1$. Fig. 8.1 presents the number of problems solved (the x-axes) within a given amount of time (the y-axes). For example, a point $(x, y) = (50, 100)$ means that 50 problems are solved in 100 seconds or less each. Hence the more the graph is to the right, the better the results are. We used a time-limit of 600 seconds per instance. Instances that are not solved within this time limit are excluded from the graph, which explains why different solving engines end up solving different number of instances. Only Z3 is able to solve the entire set. Since this is also the incomplete engine of the three, this is somewhat of unfair comparison, however, since the bound we used on the number of slots as described above (the maximum

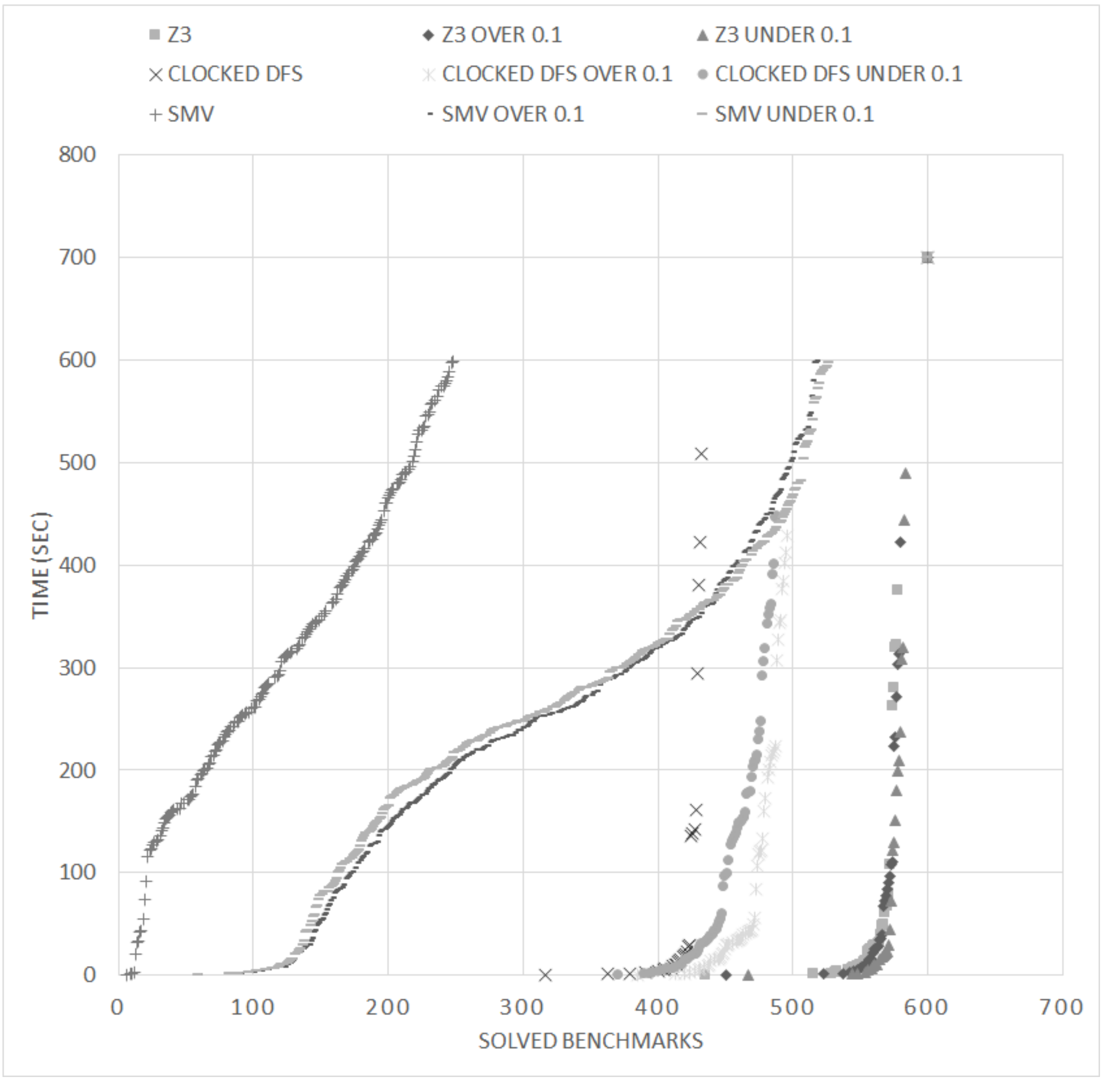


Figure 8.1: A comparison of Z3, SMV and Clocked-DFS.

relative deadline divided by the shortest flight time) is not necessarily sufficient: recall that if the bound is too small then it may turn a satisfiable problem into an unsatisfiable one. Yet, we know that for this set of benchmarks this bound happens to be sufficient, because we validated the UNSAT results with the complete engines.

In addition we found out that a simple DFS algorithm might also compete the Z3 algorithm on some test cases when only one UAV involved. Figure 8.2, shows a comparison between Z3 and algorithm 7.1

8.4.2 The effect of over- and under-approximation

. Our results show the following statistics:

- Over-approximation, with $\gamma = 0.1$: 0.3% of SAT results are incorrect.

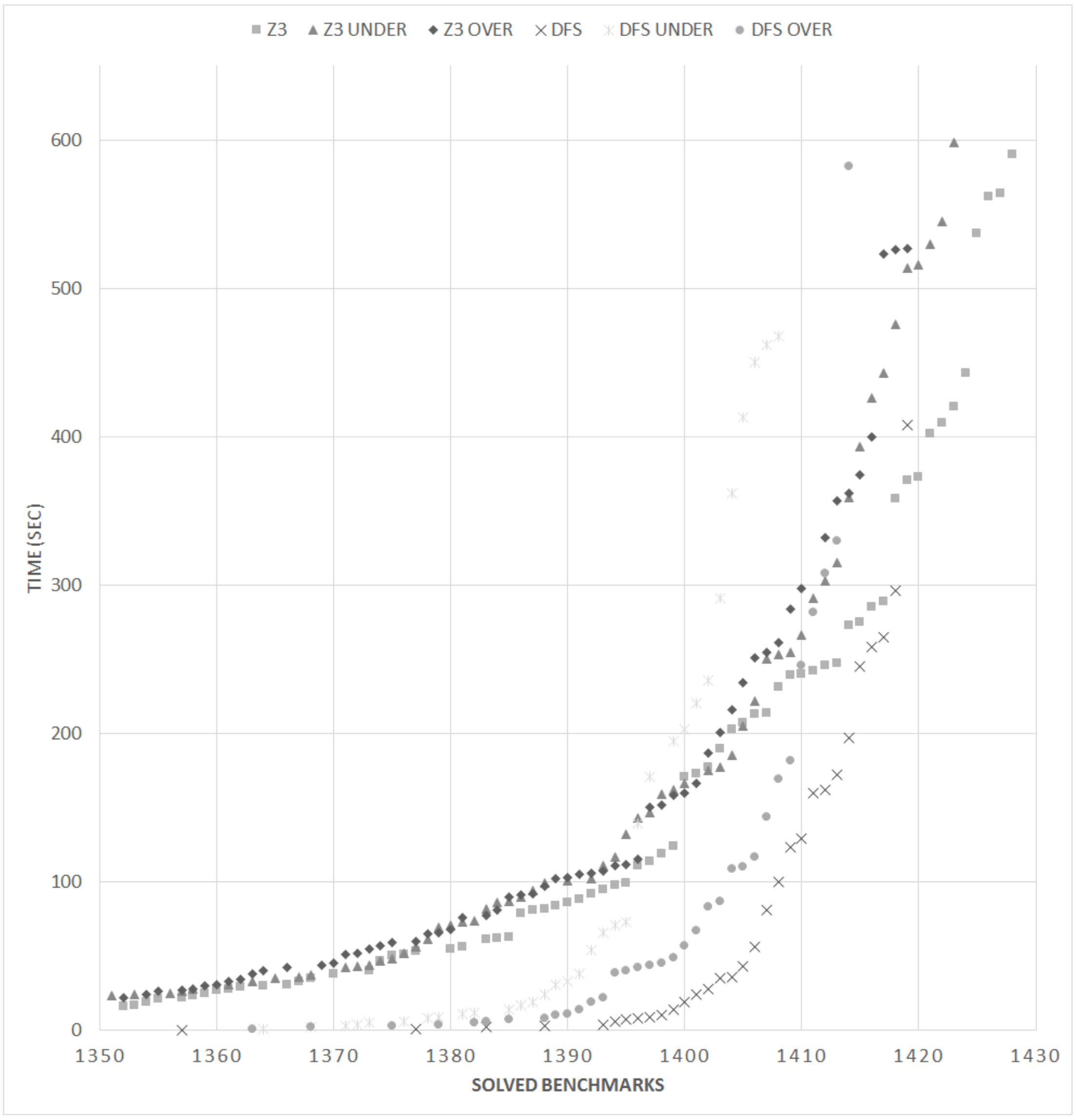


Figure 8.2: A comparison of Z3, DFS.

Method	γ	Z3	Clocked-DFS	SMV
Precise		232	157	43
under	0.1	232	200	147
over	0.1	232	177	153

Table 8.1: Comparing the number of solved instances within a time limit of 10 minutes.

- Under-approximation, with $\gamma = 0.1$: 6.5% of UNSAT results are incorrect.

(note that these statistics represent a property of the problem at hand with respect to a given γ, and not of the solving algorithm). The effect of approximation on run time depends on the engine. Table 8.1 summarizes our results. Our conclusion from these experiments is that with Z3 there is not much gain, in terms of run-time, in approximation. However the SMV solver and the Clocked-DFS method solved many more problems by using approximation.

Chapter 9

Conclusion

The CR-UAV problem is an interesting and challenging problem which appears difficult (At least NP-Hard). We believe that a solution to this problem can significantly contribute to the process of creating autonomous systems - UAV, AUV, GAV etc... In this work we explored several models (complete and incomplete) and we compared them. We found a lower bound that can be done as a pre-process stage to the models suggested and by that save calculation time.

The results revealed that the best current known method for multiple UAVs is Microsoft Z3. We found that using OVER/UNDER approximation might not affect the Z3 solver, but it affects the discrete models. Therefore we suggest to run in parallel the Clocked DFS algorithm with Over/Under approximation and the Z3 solver.

Chapter 10

Literature review

This chapter will review the literature relevant for the CR-UAV problem. It will start from the well known combinatorial problems (TSP, VRP etc..) and will continue by reviewing the specific literature related to autonomous systems like (UAV, AUV, GAV etc..).

Searching for relevant literature, we failed to find any work made explicitly on the CR-UAV problem discussed herein. However, there are several well-studied related problems. In particular, Deadline TSP and the Vehicle Routing Problem with Time Windows (abbreviated VRP-TW) [BG05] and Periodic Scheduling [PG06] [SU89] seem to be relevant, Also the agent patrolling problem solved in [BGA09] is related to the CR-UAV problem and will be reviewed later on.

10.1 Deadline TSP and VRP problems

The Travelling Salesman Problem (TSP) is one of the famous problems in the operation research literature and therefore it has many derived problems such as the DTSP. The Deadline-TSP problem defined at [BBCM04] - Given a metric space G on n nodes, with a start node r and deadlines $D(p)$ for each point p, consider the Deadline-TSP problem find a path starting at r that visits as many nodes as possible by their deadlines. It is solved in [BBCM04] using an $O(\log n)$ approximation algorithm. The Deadline TSP can be extended into the TSP-TW problem in which each node p also has a release time $R(p)$ and the goal is to visit as many nodes as possible within their "time-windows" $[R(v), D(v)]$. The algorithm described in [BBCM04] can be extended to an $O(\log_2 n)$ approximation for the Time-Window problem.

The VRP-TW problem can be define as: there are n customers at n different points, to be served within a specified time window by several capacitate vehicles from one depot. The goal is to minimize the number of vehicles needed such that each customer is reached within its time window while obeying the capacity constraints. A secondary objective is to minimize the total distance travelled. The VRP-TW is known to be NP-hard, but due to its importance and applicability, there is a vast literature on this problem, mainly concerning various heuristic methods [BG05]. There are several differences between the two discussed problems and the CR-UAV. For instance, there are capacities limitations in the VRP-TW but no such limitations in the CR-UAV problem; also in VRP-TW each vehicle should serve the customer within a time window, while in CR-UAV there are upper time limits on the time between two consecutive visits to the same target. In CR-UAV each target should be revisited according to its relative deadline, while in the VRP-TW and Deadline TSP problems they should be visited at most once.

The periodic (cyclic) scheduling problem can be defined in various ways. Serafini and Ukkovich, 1989 [PG06] [SU89], [SU89] consider events and activities to be identically repeated at a constant rate. The periodic activities within a given common period can be considered as "time window", reflecting the relative position of pairs of activities within the period. Note that in the CR-UAV problem, the "time window" stands for the upper bound on the length of time between two consecutive visits, and not an exact strict length of time. Also, in cyclic scheduling each activity appears once in the cycle, while in the CR-UAV problem, as mentioned previously, a target can appear several times in a cycle. [PG06] consider periodic schedules where each client i requests to be served for b_i consecutive time slots every no more than t_i time slots. The aim is to construct a schedule that will satisfy the requirements as good as possible, with the distances (ratio) between the required periods and the actual scheduled ones are minimized. The authors present two algorithms for solving the periodic scheduling and compare the results against the optimal solution, the simulations they did shows no much difference. This problem is similar to CR-UAV, with the difference that the latter needs to consider also the flight time between the targets. This difference cannot be overcome by simply adding the flight time to the service time, because, recall, the flight time depends on the ordering of the targets in the route. The periodic scheduling problem is known to be NP-hard, there are various heuristic methods for solving it but non of them can be applicable to the CR-UAV problem.

10.2 Literature related to UAVs problems

We will present here a survey of related algorithms made especially for UAV, also we will explore some other algorithms related to other machines such as AUV (Autonomic Underwater Vehicle) which have the same moving capabilities as the UAV.

Monitoring assignments can be divided into two categories:

(i) Monitoring static object.

(ii) Monitoring dynamic objects.

In [LPR$^+$10] a UAV launch at the starting point and should observe all targets using EO/IR camera and finally return to the starting point. It Assume that targets to be observed are previously specified in the mission planning phase in consideration of the limitations of the camera such as observation range, resolution, and type of image. In addition, some of the targets have a time constraint (deadlines), They compared a rule of thumb that was based on the TSP algorithm and another algorithm that used Monte Carlo simulation, the results reviled that the more targets (polygons) there is - the monte carlo algorithm wins the TSP based algorithm.

[Cho09] is dealing with the problem of allocating m targets to n vehicles (in this case, AUV - Autonomous Underwater Vehicle). Given a set of vehicles $V_1, V_2, .., V_n$ and targets $D = d_1, d_2, .., d_m$ the problem is to assign a sequence of targets S_i to each vehicle to visit and a path through the sequence S_i with a minimum cost. For that the MTSP (Multiple Travling Salesman Problem) algorithm was investigated. The MTSP solution uses the clustering and auctioning method as described in Chapter 4 in [Cho09]. The difference between the MTSP solution and that of the proposed algorithm in the article is that the MTSP solution does not consider the curvature constraints of the vehicle and assumes the vehicle can turn on the spot. The curvature constraints of the vehicle is negligible if the distance between targets is big enough and important when the targets are more close to each other like in the CR-UAV case. This results in straight line segments between task points with the vehicle changing orientations at each task point. The MTSP solution also does not consider the effect of ocean currents (affecting the AUVs) when creating the sequence for each vehicle, in [Cho09] they tested two algorithms one is based on TSP and alternative one, they tested it on upto 20 locations and 5 AUVs (Autonomous Underwater Vehicle).

In both articles mentioned above there is only one visiting occurrence necessary for each location, while in the CR-UAV multiple visiting occurrences required. In the above two problems as in [Obe10] the monitoring assignment was static, exactly like in the CR-UAV problem. In [Obe10] they took in account that a UAV can monitor a static object from an area above and all it needs is to reach the specified area.

On the other hand in [RDE10] monitoring UGV (Unmanned ground vehicle) is required. Successful convoy protection is achieved, when the centroid of the UGVs is visible to at least one of the UAVs at any time, assuming UGVs travel relatively close to each other.

Previous UAV routing works have focused primarily on static, pre-planned situations; however, scheduling military operations, which are often ad-hoc, drives the need for a dynamic route solver that can respond to rapidly evolving problem constraints. With these considerations in mind, [O'R99] examined the use of a Java-encoded metaheuristic to solve these dynamic routing problems, explore its operation with several general problem classes [O'R99]. Monitoring dynamic objects are often called in the literature Surveillance problem, a solution for this kind of problems can be found in [Jon09] [The09] and [CKBM06] for one or multiple UAVs.

Another research category is the coverage section, where one or multiple UAVs should keep an area covered (monitored) all the time (without stops). Different interesting

UAV continuous, i.e., , uninterrupted, coverage models were introduced in [Ha10] thesis. There is a vast amount of research done on UAVs but very little is directly related to the problem studied in this thesis. Related, but indirect, studies appeared in the area of UAV decision and control [SRR09], UAV swarms [Cor04], [Flo99] and [Now08], and UAV simulation [Wal99].

The problem solved by [Ha10] is calculating the minimal number of UAVs needed to monitor only 1 location, where each UAV should return to the base camp after time of length L was elapsed. Short gaps are allowed while monitoring this location. While in the CR-UAV problem, studied in this paper, each UAV can scan more than one location, T. Ha allows scanning of one location only.

T. Ha demonstrated successfully his solution methods on small problems with a small number of UAVs. These problems were solved in a reasonable time. However, enlarging his model to allow each UAV to scan several locations and not necessarily one location, is not trivial. Therefore, we will not generalize his method in a straightforward way, but rather look for different methods.

More research on target coverage problems can be found in [ABC$^+$06] and [SBF07]. The target coverage problems is different from the CR-UAV problem, by that in those problems a UAV or number of UAVs should cover the largest possible area, and don't leave any monitored gaps. However in the CR-UAV problem the areas which should be monitor are well defined and have tight deadlines.

Last interesting research which is very similar to our work in the CR-UAV is done by [BGA09]. There is a patrolling agent that has to prevent intrusions in some locations, with access points along a perimeter or areas of interest. The patrolling agent can be a mobile robot.

Deterministic patrolling strategies can guarantee to visit a given point within an exact time bound, while nondeterministic patrolling strategies can leave a point unvisited for long time. Moreover, a deterministic strategy that makes inconvenient for the intruder to enter (namely, that guarantees that the intruder will be always captured if it attempts to enter a location) is better than any non-deterministic strategy, which can only provide a probability for capturing the intruder. Hence, deterministic strategies, when they can be found, can be preferred to non-deterministic strategies, which, on the other hand, can be always found. Notwithstanding their important role in some cases, development of deterministic strategies for patrolling agents has not received much attention, except for some results in very special ring-like environments.

Chapter 11

Further research

Bellow are some of our suggestions for further research.

- Finding a bound to the CR-UAV slot number used in Chapter 5 for one or more UAVs. Finding a bound will make those models complete and can reduce the amount of calculations done.

- We found in Chapter 3 a lower bound for the minimum UAVs required in order to solve the CR-UAV problem, finding an upper bound better than the obvious upper bound of $|P|$ is of interest.

- Like any other NP-hard problem, scalability is a non-ending issue in solving real problems. The models suggested above can solve small size problem up to 8 targets. Finding new models or enhancing the models presented above is a very important and interesting direction.

- When testing the Timed automata formulation, a symmetry breaking was not yet implemented for our type of queries, in the tool we used (UPPAAL). Symmetry breaking can accelerate the processing time for three or above UAVs. We believe that in the future when UPPAAL will support symmetry breaking with all types of queries our formulation of Timed Automata would be much faster.

- Adding stochastic data into the model is of interest. In our CR-UAV we assumed flight times had enough slack to cover wind/UAV speed and other parameters affecting the UAV flight time. Adding slack time can make the problem unsatisfied where it can be satisfied with the real world parameters.

- The solution of the CR-UAV problem is a pre-mission route for several UAVs. In real life this solution might changed during a mission, the mission's commander might add or remove a target according to the field need. Determine if after adding/removing or moving a target, the pre-computed route can be adapted in a short time, is also an interesting direction.

- In methods such as the DFS (see Chapter 7.1), Clocked DFS (see Appendix B) or Timed automata (see Fig 6.) that are based on a search tree, trying different heuristics for exploring the search tree - direct search reverse search and even random search might yields different results.

- In the Clocked DFS (see Appendix B) there are many redundant states. For example assume state s_1 is equal to s_2 except that $s_1.t(1) >= s_2.t(1)$ then if there is no path starting at s_1 there won't be any path starting at s_2 and all the sub-tree of s_2 can be ignored. Finding such set of rules for bounding the search space might yields a faster and more scalable algorithm.

Appendix A

Timed automata

Appendix A presents a solution to the CR-UAV based on Timed Automata [AD94]. Timed automata are automata augmented with continuous clock variables whose values grow uniformly at every state. Clocks can be reset to zero at certain transitions and tests on their values can be used as conditions for enabling transitions. Hence they are ideal for describing concurrent time-dependent behaviours. Recently Timed automata were used for solving scheduling problem [AAM$^+$06].

For the CR-UAV purposes this method is complete. In addition, it is supported by a tool which enables easy graphical modelling and an easy way to query the model for different kind of solution traces. The model-checker Uppaal is based on the theory of timed automata and its modeling language offers additional features such as bounded integer variables and urgency. The query language of Uppaal, used to specify properties to be checked, is a subset of CTL (computation tree logic). Next we will present the modeling and the query language of Uppaal and we will give an intuitive explanation of time in timed automata.

A.1 Modeling using UPPAAL

In order to model with UPPAAL the user must be familiar with its terms:

- **Templates:** automata are defined with a set of parameters that can be of any type (e.g., int, chan). These parameters are substituted for a given argument in the process declaration.
- **Constants:** are declared as const name value. Constants by definition cannot be modified and must have an integer value.
- **Bounded integer variables:** are declared as int[min,max] name, where min and max are the lower and upper bound, respectively. Guards,
- **invariants, and assignments:** may contain expressions ranging over bounded integer variables. The bounds are checked upon verification and violating a bound

leads to an invalid state that is discarded (at run-time). If the bounds are omitted, the default range of -32768 to 32768 is used.

- **Binary synchronisation channels:** are declared as chan c. An edge labelled with c! synchronizes with another labeled c?. A synchronization pair is chosen non-deterministically if several combinations are enabled [BDL04].

Each edge in a model based on UPPAAL can have the following attributes:

- **Guard:** A guard is a particular expression satisfying the following conditions: it is side-effect free; it evaluates to a Boolean; only clocks, integer variables, and constants are referenced (or arrays of these types); clocks and clock deference are only compared to integer expressions; guards over clocks are essentially conjunctions (disjunctions are allowed over integer conditions).
- **Synchronisation:** A synchronisation label is either on the form Expression! or Expression? or is an empty label. The expression must be side-effect free, evaluate to a channel, and only refer to integers, constants and channels.
- **Assignment:** An assignment label is a comma separated list of expressions with a side-effect; expressions must only refer to clocks, integer variables, and constants and only assign integer values to clocks.
- **Invariant:** An invariant is an expression that satisfies the following conditions: it is side-effect free; only clock, integer variables, and constants are referenced; it is a conjunction of conditions of the form $x < e$ or $x <= e$ where x is a clock reference and e evaluates to an integer [BDL04].

A.2 The model

We modelled the CR-UAV using Timed Automata for a constant number of UAVs UN.

A.2.1 Global declaration

Global declaration's scope is the whole model i.e., all the templates below.

- **const int UN:** exact number of UAVs in the model.
- **const int PN:** number of points in the graph
- **typedef int[0,UN-1] uav_id_t:** data type
- **typedef int[0,PN-1] point_id_t:** data type
- **chan scanning[PN]:** a channel to pass info between the *Locations* template and the different *UAV*s template. It will alert the *Location* template whenever a UAV reached its destination and ready to fly to its next point.

A.2.2 UAV template

The UAV template represents a single UAV route, independent on other UAVs.

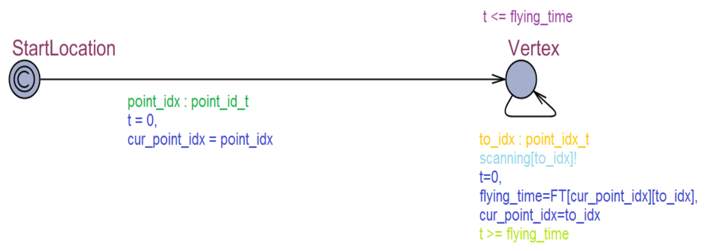


Figure A.1: Timed Automata - UAV template.

Local declaration

Declarations affecting only the UAV template:

- **clock t:** For each UAV we define a clock, counting the elapsed time from the last point visited.
- **int[0,PN-1] cur_point_idx = 0:** This parameter holds the current point index from which the UAV is currently flying.
- **int flying_time = 0:** This parameter holds the flying time from the current point to the next one ($FT(e)$).
- **const int FT[PN][PN]:** distance array.

Template states

The UAV template contains two states:

- **Start Location:** Initializing state. The *StartLocation* state contains only one edge to the *Point* state. On time zero, each UAV should pass through that edge and initialize the following parameters.
    - t = 0 - A UAV is ready to start a new mission.
    - point_idx : point_id_t, cur_point_idx = point_idx - Assigning the UAV its first destination point.
- **Points:** From the *Point* state there is only one edge, which contains the following attributes:

- Guard on the point state $t \leq flying\_time$ - A UAV cannot continue to another point before it reaches it current destination, Note that the flight to the first point will be ignored because the flying_time initialized to zero.
- Selection to_idx : point_id_t, cur_point_idx = to_idx - Whenever a UAV finishes its scanning mission, it takes the loop edge form the Point state to itself. While taking this edge the model assign it with a new destination (point).
- Assignment t = 0: A UAV is ready to start a new mission.
- Assignment flying_time = FT[cur_point_idx][to_idx] - Assign the flying time to the new destination.
- Invariant t $\geq$ flying_time: The UAV can continue to a new mission only if the flying time of its current mission was elapsed.
- Synchronization scanning[to_idx]! - Taking the edge indicate that the UAV is ready to start a new mission the model uses the scanning channel in order to alert the mission controller (Location template) that the next mission of the UAV is to scan to_idx point.

A.2.3 Location template

The Location template will validates that at each time a UAV will take an edge that all the deadline constraints holds.

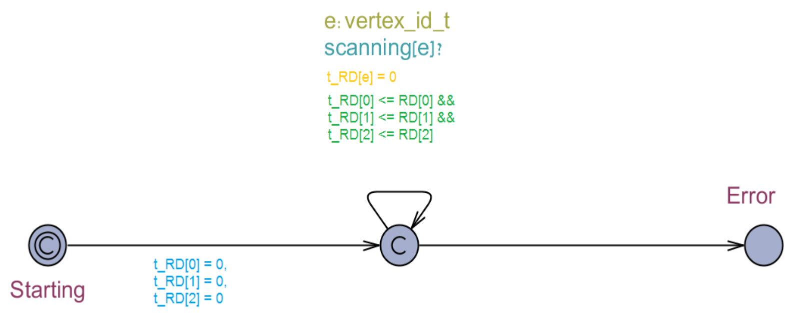


Figure A.2: Timed Automata - Location template.

Local declaration

The Location template will use the following local declarations:

- clock t_RD[PN]: a clock will be defined for each point representing the deadline constraints, It will be reset at each visit of a UAV to the relevant point.
- const int D[VN]: An array contains the deadline constraints for each point.

Template states

The Location template contains three states:

Starting state: Initializing state. The *Starting* state contains only one edge to the *Loc* state. On time zero, each UAV should pass through that edge and initialize its parameters.

Loc state: Whenever a UAV is sending a sync request for moving into another point, it is being verified by taking one of the following edges: an edge to the err state means the deadline was violated, or a self edge to *Loc* state means all clocks deadline are still holds. while taking the self edge the following action will take place:

- Synchronization e - point_id_t, Scanning[e]? A sync request from any UAV should be accepted.
- Invariant t_RD[i] $\leq$ D[i] - Validation of all the deadline constraints should be done.
- Assignment t_RD[e] = 0 - reset the clock for location e - the location which the UAV should fly to.

Error state: an absorbing state. If the model reaches the *Error* state then a solution could not be found.

A.2.4 Query

The main purpose of a model checker is to verify the model with respect to a requirement specification. Like the model, the requirement specification must be expressed in a formally well-defined and machine readable language. Several such logics exist in the scientific literature, and Uppaal uses a simplified version of CTL. Like in CTL, the query language consists of path formulae and state formulae. State formulae describe individual states, whereas path formulae quantify over paths or traces of the model. Path formulae can be classified into reachability, safety and liveness.

The only query included in the CR-UAV model is - “A <> Location.Error ”- For all possible scenarios which the time automata could have, eventually the model will reach the Location.Error state.

If the condition is not valid then there exists at least one scenario, in which the Timed automata will not get eventually into the Location.Error state, and this could occur only if there is a cyclic route.

A.3 Timed Automata results

Testing the Timed Automata model done by a running a random test case generator. 2,493 test cases were generated out of which 1,076 were satisfiable and 1,417 were not.

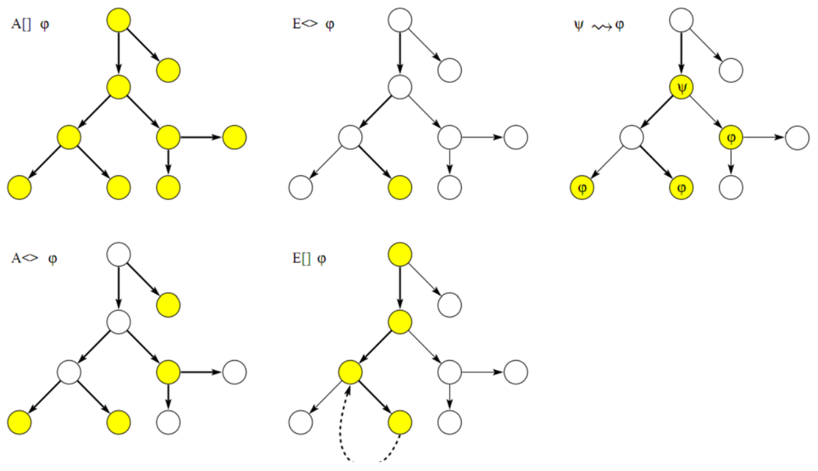


Figure A.3: Illustrates the different path formulae supported by UPPAAL.

We run the same model using UPPAAL feature of over estimating the solution. The results were:

- **In 2312 cases:** UPPAAL return with MAYBE.
- **In 181 cases:** the problem was feasible.
- **In 0 cases:** the problem was not feasible.

The tables below summarize the time(seconds) it took to solve the cases generated above.

Time by number of points (Seconds)				
$\|P\|$	Avg.	MAX.	Min.	Number of tests
3	4.4955	1530	0	882
4	57.3469	6569	0	882
5	73.2517	10584	0	445
6	159.5599	6986	0	284

Time by number of UAVs (Seconds)				
UN	Avg.	MAX.	Min.	Number of tests
1	39.3671	6986	0	1166
2	44.5304	10584	0	739
3	91.2364	6569	0	5888

Note. No tests were done with the combinations of 5 points and 3 UAVs, or 6 points and 2 or 3 UAVs due to the long time it took.

The results found to be as expected, testing a simple test with only six points, worked perfectly with small number of UAVS (less than three), however when the number of UAVs increased it become less efficient and the memory consumption increased exponentially (+2G).

Appendix B

An explicit search algorithm

This appendix presents an explicit search algorithm for the CR-UAV problem, called Clocked_DFS. It is a DFS-based algorithm, and it stops whenever it finds a solution in the shape of a lasso in the *Path* parameter or when it doesn't find any solution. The fact that the CR-UAV can be modelled as a finite state system is explained in details in Chapter 6

As defined in Chapter 6 a state s of the system is a vector of size $|P| + |U| + |U|$ assembled by:

1. For $p \in P$, $t_{RD}(p)$ is the time left to meet p's relative deadline.
2. For $u \in U$, $p(u) \in P$ is the destination of u.
3. For $u \in U$, $t_D(u)$ is the time left for u to reach its next target $p(u)$.

A clarification regarding Algorithm B.1, checking that $s \in S$ return true in one of the following cases:

1. If $s \in S$.
2. There exists a ruling state s' of s in S - assume state s is equal to s' except that there exists $v \in V : s'.t_D(v) >= s.t_D(v)$ then if there is no solution starting at s' there won't be any solution starting at s and all the sub-tree of s can be ignored. the same happen if $s'.t_{RD}(v) <= s.t_{RD}(v)$.
3. Symmetry breaking - Two states s_1, s_2 are equivalent if $\forall p \in P : s_1.t_D(p) = s_2.t_D(p)$ and $\forall u_1 \in U : s_1.t_{RD}(u_1) = a$ and $s_1.p(u_1) = b$ there exists $u_2 \in U : s_2.t_{RD}(u_2) = a$ and $s_2.p(u_2) = b$

Algorithm B.1 Clocked DFS

```
1: Initialization($S$)
2: while there exists a state s where $s \in S_{multiple}, s \notin S$ do
3:     if clockedDfs($S$, $s$, $Path$) = true then
4:         return true
5: return false

1: function clockedDfs($S, CurState, Path$)
2:     for all $p \in P$ do
3:         $s \leftarrow$ CreateState($CurState$ ,$p$ )
4:         if not ValidateDeadlines(s) then
5:             return false
6:         if $s \in Path$ then
7:             return true                                    ▷ Found a cycle.
8:         if $s \in S$ then
9:             continue                    ▷ State s has been already visited.
10:        $Path \leftarrow Path \cup \{s\}$                        ▷ Add s to path.
11:        $S \leftarrow S \cup \{s\}$                     ▷ Mark s as being visited.
12:        if clockedDfs($S, s, Path$) = true then
13:            return true
14:        $Path \leftarrow Path - \{s\}$  ▷ State s is not a good direction remove it from Path.
15:    return false

1: function CreateState($oldState, dest$)
2:     $u_{min} \leftarrow \arg\min_{[u \in U]} oldState.t_D(u)$ s.t. $oldState.t_D(u) \neq 0$
                          ▷ Find the UAV that will reach its destination first.
3:     $\Delta \leftarrow oldState.t_D(u_{min})$              ▷ Time to reach destination.
4:     for all $p \in P$ do
5:         if $p = dest$ then
6:             $newState.t_{RD}(p) \leftarrow oldState.RD(p)$         ▷ Reset the target clock.
7:         else
8:             $newState.t_{RD}(p) \leftarrow oldState.t_{RD}(p) - \Delta$   ▷ Decrease the target clock.
9:     $e = (oldState.p(u_{min}), dest)$
10:    for $u = 1$ to $|U|$ do
11:        if $u = u_{min}$ then
12:            $newState.t_D(u) \leftarrow FT_e$
13:            $newState.p(u) \leftarrow dest$
14:        else
15:            $newState.t_D(u) \leftarrow oldState.t_D(u) - \Delta$
16:            $newState.p(u) \leftarrow oldState.p(u)$
17:    return $s$

1: function ValidateDeadlines($s$)
2:     for all $v \in V$ do
3:         if $s.t_{RD}(v) < 0$ then
4:             return false
5:     return true
```

Bibliography

[AAM+06] Y. Abdeddaïm, E. Asarin, O. Maler, et al. Scheduling with timed automata. *Theoretical Computer Science*, 354(2):272–300, 2006.

[ABC+06] A. Ahmadzadeh, G. Buchman, P. Cheng, A. Jadbabaie, J. Keller, V. Kumar, and G. Pappas. Cooperative control of uavs for search and coverage. In *Proceedings of the AUVSI conference on unmanned systems*, 2006.

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[BBCM04] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. Approximation algorithms for deadline-tsp and vehicle routing with time-windows. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 166–174. ACM, 2004.

[BDL04] G. Behrmann, A. David, and K. Larsen. A tutorial on uppaal. *Formal methods for the design of real-time systems*, pages 33–35, 2004.

[BG05] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation science*, 39(1):104–118, 2005.

[BGA09] N. Basilico, N. Gatti, and F. Amigoni. Developing a deterministic patrolling strategy for security agents. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 02*, pages 565–572. IEEE Computer Society, 2009.

[Cho09] B. Chow. *Assigning closely spaced targets to multiple autonomous underwater vehicles*. University of Waterloo Ontario, Canada, 2009.

[CKBM06] D.W. Casbeer, D.B. Kingston, R.W. Beard, and T.W. McLain. Cooperative forest fire surveillance using a team of small unmanned air vehicles. *International Journal of Systems Science*, 37(6):351–360, 2006.

[CNN10] CNN. Review shows dramatic shift in Pentagon's thinking, Feb 2010. http://edition.cnn.com/2010/POLITICS/02/01/us.pentagon.review/index.html.

[Cor04] J.J. Corner. Swarming reconnaissance using unmanned aerial vehicles in a parallel discrete event simulation. Technical report, DTIC Document, 2004.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[DM06] Bruno Dutertre and Leonardo De Moura. The Yices SMT solver. Technical report, SRI, 2006.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[Flo99] R.A. Flood. A java based human computer interface for a uav decision support tool using conformal mapping, 1999.

[Ha10] T. Ha. The uav continuous coverage problem. Technical report, DTIC Document, 2010.

[Jon09] P.J. Jones. *Cooperative area surveillance strategies using multiple unmanned systems*. ProQuest, 2009.

[KS08] Daniel Kroening and Ofer Strichman. *Decision procedures – an algorithmic point of view*. Theoretical computer science. Springer-Verlag, May 2008.

[LPR$^+$10] C.W. Lim, S. Park, C.K. Ryoo, K. Choi, and J.H. Cho. A path planning algorithm for surveillance uavs with timing mission constrains. In *Control Automation and Systems (ICCAS), 2010 International Conference on*, pages 2371–2375. IEEE, 2010.

[Now08] D.J. Nowak. Exploitation of self organization in uav swarms for optimization in combat environments. Technical report, DTIC Document, 2008.

[Obe10] K.J. Obermeyer. *Visibility problems for sensor networks and unmanned air vehicles*. PhD thesis, UNIVERSITY of CALIFORNIA, 1 2010.

[Ome12] Omega, May 2012. http://www.theuav.com/.

[O'R99] K.P. O'Rourke. Dynamic unmanned aerial vehicle (uav) routing with a java-encoded reactive tabu search metaheuristic. Technical report, Faculty of the Graduate School of Engineering Air Force Institute of Technology Air University, 3 1999.

[PG06] S. Patil and V.K. Garg. Adaptive general perfectly periodic scheduling. *Information processing letters*, 98(3):107–114, 2006.

[RDE10] A. Rahmani, X.C. Ding, and M. Egerstedt. Optimal motion primitives for multi-uav convoy protection. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 4469–4474. IEEE, 2010.

[SBF07] K. Savla, F. Bullo, and E. Frazzoli. The coverage problem for loitering dubins vehicles. In *Decision and Control, 2007 46th IEEE Conference on*, pages 1398–1403. IEEE, 2007.

[SRR09] T. Shima, S. Rasmussen, and S.J. Rasmussen. *UAV cooperative decision and control: challenges and practical approaches*, volume 18. Society for Industrial Mathematics, 2009.

[SU89] P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2:550–581, 1989.

[The09] P. Theodorakopoulos. *On autonomous target tracking for UAVs*. PhD thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier, 05 2009.

[Wal99] J.G. Walston. Unmanned aerial vehicle mission level simulation. Technical report, DTIC Document, 1999.