UNIVERSITÀ DEGLI STUDI DI TORINO

SCUOLA DI SCIENZE DELLA NATURA

Corso di Laurea Magistrale in Informatica

Master's Degree Thesis

Numerical and temporal planning for a multi-agent team acting in the real world

Relatore:

Prof. Pietro Torasso

Contro-relatore:

Prof. Gian Luca Pozzato

Candidato:

Davide Dell'Anna

Academic year 2015/2016

Abstract

This thesis addresses the problem of automatic planning in real world domains where temporal and consumable resource constraints have to be taken into account. The classes of problems modelled are based on real world domains (such as UAV, robotic agents, etc.) and require a complex encoding process in powerful planning languages such as the latest PDDL extensions. As test bed of the different modelling solutions described we used a multi-UAV domain with multiple target observation requests. We show how to encode the problems both in numerical planning and in temporal planning and how the state-of-art planners are able to deal with the complexity of the class of problems. Planners can efficiently handle the baseline problems but they proved to have difficulties in solving more complex problems where numerical and temporal constraints are introduced. In particular, temporal constraints on targets observation introduce a complexity that many state-of-art planners cannot easily face. For some of the modelled classes of problems we developed appropriate heuristics and solutions in order to facilitate the planning and obtain acceptable results in reasonable computational time. The thesis reports extensive experimental results concerning multi-UAV multi-target planning problems encountered in the industrial research project SMAT-F2.

Acknowledgements

Desidero ringraziare innanzitutto il Prof. Pietro Torasso per la straordinaria disponibilità dimostratami durante questi anni e per i preziosi insegnamenti che mi ha fornito. Lavorare insieme a lui è stato un onore e un'enorme fonte di crescita ed ispirazione.

Un ringraziamento speciale va ai miei genitori, che mi hanno dato l'opportunità di intraprendere questo percorso di studi e di concluderlo nel migliore dei modi.

Infine ringrazio tutte le persone che ho incrociato lungo il mio percorso e che mi hanno insegnato e trasmesso qualcosa condividendo con me una parte della loro vita.

Contents

Chapter 1

Introduction

Planning is a deliberative process that searches for a sequence of actions that results in a state of the world which satisfies a set of goal conditions. Starting from a particular description of the initial state, planning evaluates the applicability of a set of possible actions and simulates their execution by anticipating the expected outcomes. The deliberation produces a set (or a sequence) of actions that must be executed in order to achieve as best as possible the set of goals.

Automated planning is a central area of Artificial Intelligence that aims to study and develop efficient automated solutions for this deliberative process. The main goal of automated planning is the design of a powerful deliberation layer for autonomous intelligent agents which requires a small effort by human operators. This purpose is due to the growing presence of intelligent systems acting in real world and to the need to make them more autonomous also in high-level decision making. While sometimes autonomy of intelligent systems is only an option, in some contexts it is almost essential. In some domains in which environment is only partially known a priori or where it is not possible to perform an on-line monitoring by a human operator, autonomy of intelligent systems is compulsory. Some intelligent systems (e.g. a rover in a extra-planetary mission), in fact, cannot always be fully remotely controlled because of operational constraints (e.g. delay in communications). These systems need a complex deliberation level which allows them to be enough autonomous w.r.t. to their tasks. It is also worth noting that autonomy of intelligent systems doesn't concern only planning and deliberation but also acting. These two aspects are not completely disjoint: actors may deliberate or plan both before and during acting in order to perform intelligent executions, and deliberation may be strictly influenced by acting details ([1]).

Automated planning initially focused on propositional planning (which nowadays is called classical planning [2]) and it was supported by the introduction of STRIPS ([3]). Classical planning problems are composed of three elements: an initial state (the description of the initial state of the world), a final state (the description of a set of goal conditions on world's elements to be achieved) and a set of actions that, when applied, produce a state transition. States are symbolic high-level descriptions of the world in terms of a set of propositions. Actions are instead characterized by a set of preconditions and a set of effects which respectively specifies which propositions must belong to the state in order for the action to be applicable and which propositions the action will add or remove when applied (we talk of ADD- and DEL- lists). Solving a classical planning problem means searching for an ordinate sequence of actions which leads the world from the initial state to a final one where goal conditions are satisfied. This definition of problems led to address planning by means state space search algorithms (e.g. GraphPlan) and to develop many planners implementing these algorithms and exploiting languages such as STRIPS and the first development of PDDL ([4]).

By reasoning only with symbolic entities, classical planning has to make some assumptions and simplifications on the world, such as considering actions as instantaneous or ignoring numerical aspects of domains. However in dealing with complex real-world problems (e.g. planning an expedition of a rover in space) it is necessary to consider elements such as time or resource consumption, because obviously actions in real world occur over a time span, consume a certain quantity of resources and can be scheduled concurrently. Because of the limitations due to the abstraction level of symbolic planning, the gap between the produced plans and the real actions' execution is too wide and it doesn't allow an intelligent monitoring of execution. In order to reduce this gap between planning and acting, it is thus necessary to refine the domain's model adopted by deliberation and introduce structures allowing to express significant acting aspects also at planning time.

In recent years automatic planning languages have been extended with primitives allowing to express also numerical and temporal aspects of problems (e.g. PDDL 2.1, the PDDL extension to numerical aspects and durative actions ([5])) and some assumptions of symbolic planning, such as the absence of actions' duration or temporal aspects, have been relaxed. This allowed to take into account at planning time of more acting aspects and to redce the gap between deliberation and control layers, leading to the generation of more realistic and robust plans and facilitating the on-line intelligent monitoring.

Numerical and temporal planning allow in fact to deal with more realistic planning problems involving complex autonomous systems such as aircrafts or robotic agents acting in real world. Solving a numerical planning problem requires to consider, in addition to propositional constraints, also numerical constraints concerning consumable resources. Solving a temporal planning problem requires instead to consider, beside all the constraints of both classical and numerical planning, also the temporal characteristics of problems. In particular it must consider the possibility of several temporally overlapping (concurrent) actions, the actions' temporal location and their duration. At the same way it must consider during the planning the current absolute time and how far the execution of each active action has proceeded as well as constraints on total plan duration.

It is worth noting that, despite the need to reduce the gap between planning and acting, the capabilities of agents to model abstracts away from their low-level sensor or motor commands. The detailed complexities of agents' movements, engines power cycle, sensors pointing, data acquisition, etc., which are obviously important in a completely accurate representation of a real-world problem are abstracted away by the agent capabilities formulation of a planning model, and assumed to be dealt by the acting low-level control processes. However the choice of the right abstraction level of the planning model is a challenging task: it should both allow to perform deliberation in a reasonable time and also produce plans which aren't too abstract w.r.t. the real acting needs, allowing intelligent monitoring and execution.

Another important aspect of automated planning is the distinction between off-line and on-line planning. While on-line planning has to be performed during the agents' execution after that the running plan for some reason at a given time is no more adequate (we talk of re-planning or continual planning, [6]), off-line planning is a procedure that has to be performed before the real execution. The main distinction between the two approaches concerns the constraints at which is subjected the planning process. On-line planning is mainly constrained by requirements on computational resources and on time response, due to the need to obtain a new plan as soon as possible, off-line planning instead does not require particularly strict constraints on the computational or temporal resources needed for synthesizing a plan and the interest is usually more focused on the quality of the solution provided.

In this thesis we addressed a class of real-world problems based on real world domains involving teams of agents (such as UAV[1] or robotic agents) in which temporal and consumable resource constraints are mandatory. We adopted an abstraction level that, as we'll see in the following chapters, allows to define actions for planning purposes oriented to a centralized off-line planning system of managing and coordination of teams of (possibly heterogeneous) agents. We tried to reduce as much as possible the gap between deliberation and control levels while both maintaining a reasonable planning time and producing plans that allow intelligent monitoring during on-line agents' execution.

We developed a complex software architecture which includes planning process and permits to encode the domain in a way as simply as possible in order to solve complex problems in reasonable times while also maintaining the level of abstraction close enough to the real acting needs. We demanded the low-level controls (e.g. sensor pointing, agents' movements, etc.) to single agents which have specific on-line acting and deliberative capabilities. The system also performs a series of abstraction during encoding of planning problems which allow to simplify the problems without losing significant information. These simplification reduce the problems' complexity by hiding to the planners a series of information and allow to easier and faster find (if possible) a valid solution to the problems. However by exploiting an internal database to store the hidden data (see chapter 4) there is no significant lose of information. In fact, during encoding phase, the system stores into an internal database all information hidden to the planner, in order to restore them during solution decoding (see chapters 5 and 6 for more details about the abstractions performed and the decoding process).

We studied how to deal with real world planning problems for teams of agents by adopting a centralized approach to multi-agent planning. Multi-agent planning is the problem of planning for a group of agents in order to achieve goals that single agents couldn't be able to achieve by themselves, or at least to achieve them better. A solution to a multi-agent planning problem is a partially ordered plan that as well as in classical planning results in a set of achieved goals for some of the agents. The main difference is that a plan is not necessary centralized

[1]Unmanned aerial vehicles, also called RPA, Remotely piloted aircrafts, or simply drones. They are aircrafts without a human pilot aboard.

but it can be created and distributed on each agent which performs a private planning (or re-planning) supported by its private and available actions and then communicates with other agents ([7]). However, when problems require many interactions among agents the coordination activities are very expensive and a centralized approach to multi-agent planning may be preferable. We choice to adopt a centralized approach because of the strict coordination necessary between different agents in order to satisfy temporal constraints on the observations of targets expressed as requirements of the problems. This choice is also due to the off-line nature of our purposes. Our solutions have been in fact designed to be employed in a central station of control of a team of heterogeneous agents[2], which aims to off-line build mission plans for teams of agents, therefore a centralized approach is more appropriated.

In this thesis we modeled the class of problems both in numerical and temporal planning and we analyzed how the main state-of-art planners are able to deal with the complexity of the problems. The modeling of domain and problems in temporal and numerical formalisms is not trivial and the challenges are many. Chapters 5 and 6 identify in detail the challenges to face when adopting the two approaches and describe the solutions we found. A challenge common to both approaches concerns planners efficiency, which is a relevant problem in dealing with real world problems. In fact, even if off-line planning doesn't have strict time limits to obtain a solution, real world problems can require a quantity of time which is unacceptable, especially when planning is integrated in a system that must interact with human operators. For this reason for some of the modeled classes of problems we also developed appropriate heuristics and solutions in order to facilitate the planning and obtain acceptable results in reasonable computational time.

[2]Part of our solutions have actually been employed in the large industrial research project SMAT-F2, whose primary objective was to develop an Advanced Environment Monitoring System, based on Unmanned Air Systems. Our planning model has been developed in SMAT-F2 together with a complex software architecture adopted by the SSC (Station of Supervision and Coordination, the central node of the architecture, allowing the operators to coordinate and support specific multi-UAV missions (Altec SMAT)) in order to single out an hypothesis of mission plan possibly involving multiple UAVs. (http://beta.di.unito.it/index.php/italiano/ricerca/gruppi/intelligent-semantic-technologies/progetti/smat).

1.1 Outline of the thesis

The thesis is organized as follows:

- **Chapter 2** first describes the main features of PDDL (which is considered the state of art in automatic planning languages) and its extensions. In particular it describes the concepts of numerical fluents, durative actions and timed initial literals, introduced in PDDL 2.1 and PDDL 2.2 and essential in our modeling. The second part of the chapter provides a formal definition of the above mentioned planning types (classical, numerical, temporal and multi-agent planning) and reports a brief description of the main state-of-art planners able to deal with these modeling.
- **Chapter 3** defines the class of problems we addressed. It is subdivided in five sections: section 3.1 describes a baseline class of problems, the following sections increasingly extend the baseline class of problems by introducing new levels of complexity in terms of additional constraints on the problems.
- **Chapter 4** describes the adopted architecture which allows the encoding and the planning of the class of problems according to the choice of using a PDDL formalism.
- **Chapters 5 and 6** describe in detail how to encode the main features of the addressed class of problems and its extensions in both temporal and numerical planning models. They also show the difficulties to be faced during the phase of decoding of the planners solutions in order to report to the end user a consistent and easy-to-read solution.
- **Chapter 7** reports a description of a series of real-world multi-UAV multi-target planning scenarios and the results that can be obtained with both numerical and temporal planning. The chapter also analyzes the capabilities of planners to handle our class of problems. Planners, in fact, can efficiently handle the baseline problems but they proved to have difficulties in solving more complex problems where numerical and temporal constraints are introduced. In particular, temporal constraints on targets observation introduce a complexity that many state-of-art planners cannot easily face.

Chapter 2

PDDL and planning

2.1 PDDL

In order to represent planning problems, in 1998 has been introduced by Drew McDermott et al. the Planning Domain Definition Language (PDDL). PDDL is a language inspired by the STRIPS formulations of planning problems, with a Lisp-like syntax. A planning problem is defined in PDDL by two components: a domain description and a problem description. The former provides the definition of parameterized actions that characterize domain behaviors as well as the definition of types of objects, predicates and fluents (functions) used in the problems. The latter provides instead the definition of specific objects, initial conditions, goals of a problem and a plan's metric. Such type of separation allows the use of variables to parameterize actions and allows the reuse of a single domain with different problem instances.

Although the core of PDDL follows the STRIPS formalism, the language extends beyond that. PDDL allows negative preconditions and conditional effects in action definition and the use of quantification in expressing both pre- and post- conditions. Furthermore PDDL allows to express a type structure for the objects in a domain, typing the parameters that appear in actions and constraining the types of arguments to predicates. During the planning process, and in particular during the grounding phase of planning, the parametrized actions are then replaced by a set of grounded actions where formal parameters are replaced by actual parameters (the specific predicates in the problem).

Another important feature provided by PDDL is the possibility of representing functions whose value changes in different situations. These functions are numerical variables called fluents and allow to represent numeric quantities and to assign or update them. Fluents are obviously useful in planning with realistic domains in which it is common to have to deal with non-binary resources that can change (e.g. the fuel or the energy level of a vehicle, the distance or the time). Numerical variables, in fact, can be exploited to define action preconditions (e.g. a vehicle cannot go from one point to another if it has not fuel enough) or numeric constraints to be satisfied by the final plan (e.g. the total time of the plan must be less than a certain value).

2.1.1 PDDL 2.1

In 2003 has been developed PDDL 2.1, an extension of PDDL which allows to express both temporal and numeric properties introducing numeric expressions, plan metrics and durative actions.

2.1.1.1 Numeric expressions

Numeric expressions are constructed, using arithmetic operators, from primitive numeric expressions (values associated with tuples of domain objects by domain functions). Numeric expression can be non-linear expression and they can be used both in action preconditions, defining numeric constraints as a comparison between pairs of numeric expressions (e.g. `(>= (fuel_level ?v) 10))`), and in action effects, updating a numeric fluent value assigning it a new value (e.g. `(assign (fuel_level ?v) 7))`) or increasing/decreasing it(e.g. `(increase (fuel_level ?v) 3))` or `(decrease (fuel_level ?v) 4))`). Numeric expressions allow to increase the expressiveness of PDDL and permit a better representation of domains. They also extend the possibilities of planners which can keep track of numerical information and exploit them during the planning process.

2.1.1.2 Plan metrics

Plan metrics allow to specify the criteria on which a plan has to be evaluated in terms of numeric expressions (e.g. `(:metric minimize (powerC))`). Metrics can be defined in the problem description with any arithmetic expression with no requirement of linearity and they can require to minimize or maximize the given expression.

It is worth noting that different plan metrics might yield entirely different optimal plans also with the same initial and goal states. It is also worth noting that a planner might choose to not use the metric to guide its development of a solution but just to evaluate a solution post hoc, and unfortunatley it is an approach adopted by many planners. This approach might lead to sub-optimal, and possibly even poor quality plans. No guarantee of optimality is provided by the first (and only) solution found (if possible). In such cases it might be necessary to develop a framework around the planner which permit to try to improve the plan quality.

2.1.1.3 Durative actions

Another important extension of the representaion power of PDDL 2.1 concern the notion of durative anctios. This kind of formalism allows to define temporally annotated actions. A durative action is in fact defined by duration constraints (lower and/or upper bounds on duration of the action), and temporally annotated conditions and effects (conditions and effects that must be satisfied or applied at specific points in time).

```
1  (:durative-action a
2      ...
3      :condition(and
4          (at start (p))
5          (over all (p))
6          (at end (p))
7      )
8      ...
9  )
```

Listing 2.1: Temporally annotated conditions of a durative action

Conditions can be instantaneous or continuous (and both propositional and numerical). The formers are conditions that must hold either at the start or end of the action's execution (respectively the point in time at which the action is applied and the point at which the final effects of the action are asserted) (see rows 4 and 6 in Listing 2.1). The latter are conditions that must hold continuously *over all* the action's execution (in the interval between the start and the end, extremes excluded) (see row 5 in Listing 2.1). Notice that in Listing 2.1 the reported conditions concern only a propositional fluent `p` but conditions can relate to more than a single propositional fluent and can also relate to numerical fluents.

Effects can be instantaneous or continuous too. The formers are propositional and numeric effects that occur at the start and/or end the action's execution. In case of numerical fluents they are instantaneous updates of their value (see Listing 2.2) and they are applied at the specified point in time. The latter are exclusively numeric effects (logical changes are always considered as instantaneous) that occur continuously throughout its execution. In this case the value of a numeric fluent can be increased or decreased according to a specific rate of change over time. It is used the language keyword `#t` that represents a local clock of the action and refers to the continuous changing time from the start of the action itself (in Listing 2.3 is shown an effect that, during the execution of the action, continuously decrease the value of the numeric fluent `fuel-level`). Continuous effects are not temporally annotated, in this way they can be evaluated by the planner at any time during the interval of the action.

```
(:durative-action a
    ...
    :effect(and
        (at start (p))
        (at end (p))
        (at end (decrease (fuel-level) 10))
    )
    ...
)
```

Listing 2.2: Instantaneous effects of a durative action

```
(:durative-action move
    ...
    :effect(and
        (decrease (fuel-level) (* #t (consumption-rate)))
    )
    ...
)
```

Listing 2.3: Continuous effects of a durative action

As well as for plan metrics it is worth noting that planners following PDDL 2.1 syntax not necessarily supports continuous numerical effects.

An important conseguence of the introduction of durative actions is related to concurrency. This kind of action in fact implies the presence of time in planning problems. Before the introduction of time, final plans were implicitly considered as list of sequential actions. With the introduction of durative actions, which are scheduled at a specific point in the time and have a duration, plans are not anymore simply lists of sequential actions. Every action, instead, has its own starting point in time and potentially every action can be concurrently scheduled in overlap with all others in plan. Of course the introduction of this kind of features enormously extends the possibility of temporal reasoning but at the same time introduces additional complexity in modeling and solving planning problems. In fact, on one hand it allows the definition of more realistic domains in which actions can be concurrently executed, as well as the possibility of modeling multiple cuncurrent agents problems. One the other hand it has a consistent impact on both planners complexity, in terms of search alghoritms, plan validation and heuristics, and domains and problems definition, in terms of modeling conditions and effects of cuncurrent actions and defining sets of actions which can be cuncurrently scheduled or which are mutually exclusive.

It is worth considering the conseguences of durative actions in terms of final plan structure. As already mentioned, durative actions have not to be sequential. On one hand it allows the planner to schedule overlapping actions (where possible). However on the other hand it also implies that in a final valid plan an action can start after a certain amount of time after the end of the execution of the previous one, leaving in plan "holes" between actions. In fact there is no constraint that forces the sequentiality of actions and some problem goals may require the execution of an action at a specific time point. Listing 2.4 reports an example of plan with a "hole" between actions (see also Fig. 2.1 which report a graphical representation of the plan).

```
1 0.000: (drive r1) [5.000]
2 9.000: (tp r1 target1) [2.000]
3 11.001: (comm r1) [3.000]
```

Listing 2.4: A simple temporal plan in which there is a hole of four time units between the first action and the following `tp` (take_picture) action (see rows 1 and 2).

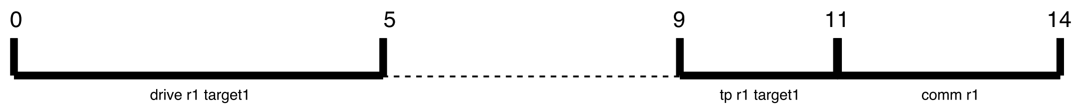


FIGURE 2.1: A graphic representation of the plan reported in Listing 2.4. The dotted line represents the hole between the actions.

The plan is composed by three actions of a simple rover domain and solves a problem which requires the rover `r1` take a picture of target `target1` and communicate the picture to the base. Listing 2.5 reports a partial version of the problem. Notice that the target `target1` is only observble between time 9 and time 15. Since there is no constraint of sequentiality between durative actions in PDDL, the plan reported in Listing 2.4 is a valid plan for the planner.

```
...
(:init
    (at 9 (observable target1))
    (at 15 (not (observable target1)))
    ...
)
(:goal (and
    (observed target1)
    (communicated_picture r1)
))
```

LISTING 2.5: A partial description of the problem solved by the plan in Listing 2.4.

While in some domains this is the desired or at least an acceptable result (e.g a rover after a scheduled action can simply do nothing for a certain amount of time, even if not necessarily optimal), in other domains doing nothing is not a valid option. Let's consider for example a domain of UAVs: a drone, after a monitoring task can not simply do nothing. In fact, normally a drone (e.g a MALE, Medium-Altitude Long-Endurance, drone) is not able to hover. If hover is not possible the UAV has to immediately performs some other task, even just a loiter, which requires a certain amount of time and fuel and which also has effects in terms of change of UAV position in space. However even if hover is possible, perform it requires a certain amount of time and also a significant amount of fuel. Therefore simply left "holes" in a plan in a domain UAV-like is not a viable solution.

2.1.2 PDDL 2.2

Another significant extension of PDDL concern the introduction in PDDL 2.2 ([5]) of a formalism called timed initial literals. Timed initial literals are a simple way to express facts that become true or false at specific time points indipendently of the action scheduled by the planner.

This formalism has been introduced in order to easily represent events that occur at specific time points and exclusively depend on environment. This kind of event is quite common in realistic domain especially in terms of time windows. In fact it is easy to model with timed initial literals time windows in which a certain fact holds. For example they can be used to model time windows within which a target is observable by an UAV, a shop is open, there is daylight, etc.

Listing 2.6 reports an example of usage of timed initial literals. In problem file they define a time window (from `3` to `20`) in which the target `target1` is observable, that is in which the propositional fluent `observable target1` holds. In domain file the durative action `monitor` requires the observability of the target `?t` taken as parameter, during the entire duration of the action. Therefore in order to observe target `target1` the `monitor` action must be performed within the time windows defined by the timed initial literals.

```
;;problem file
(:init
    (at 3 (observable target1))
    (at 20 (not (observable target1)))
    ...
)

;;domain file
(:durative-action monitor
    :parameters (?t - Target ...)
    ...
    :condition(and
        (at start (observable ?t))
        (overall (observable ?t))
        (at end (observable ?t))
        ...
    )
    ...
)
```

Listing 2.6: Example of usage of timed initial literals.

2.1.3 PDDL 3.1 and multi-agent extensions

In the most recent International Planning competitions (from 2008 IPC) a new version of PDDL has been adopted. PDDL 3.1 [8] is the latest version of the language. W.r.t. the previous version it mainly introduced object-fluents which are functions whose range could be any object-type and not only numerical. Listing 2.7 reports an example of definition of object-fluents definition. Notice at lines 7, 8 and 9 the definition of three object-fluents whose types are objects.

```
1  ;;domain file
2  (:types
3      uav - agent
4      target waypoint flevel airport - object)
5  ...
6  (:functions
7    (at ?u - uav) - (either waypoint airport)
8    (in ?t - target) - waypoint
9    (fuel-level ?u - uav) - flevel
```

Listing 2.7: Example of definition of object-fluents in PDDL 3.1.

Some PDDL 3.1 extensions (such as MA-PDDL [9] and FMAP's extension of PDDL 3.1 [10]) are used in the most recent planning competitions in which multi-agent planning is increasingly becoming a point of interest. The key points of these languages are:

- The possibility to distinguish capabilities of agents defining different actions for different agents. Listings 2.8 reports an example of definition of two different `take_off` actions respectively applicable only by `uav1` and `uav2` and each one with different effects.

```
(:action take_off
     :agent uav1
     :parameters ()
     :precondition (= (at uav1) airport1)
     :effect (and (assign (at uav1) wp1)
                  (decrease (fuel-level uav1) 2)
     )
)
(:action take_off
     :agent uav2
     :parameters ()
```

```
    :precondition (= (at uav2) airport1)
    :effect (and (assign (at uav1) wp2)
                 (decrease (fuel-level uav2) 4)
        )
)
```

Listing 2.8: Example of action definition in MA-PDDL.

- The possibility to define different goals or metrics for different agents. The following Listing reports an example of definition of a metric that every agent must maximize. It is also possible define different metrics (or goals) according to agents.

```
(:metric :agent ?u - uav
        :utility maximize (fuel-level ?u))
```

Listing 2.9: Example of multi-agent metric definition in MA-PDDL.

- Extensions to partially observable environments, introducing private objects in domain definition. In the following Listing for example the object-fluent `fuel-level` is defined as private.

```
(:predicate
    ...
    (:private  ?u - uav
        (fuel-level ?u) - flevel
        ...
    )
)
```

Listing 2.10: Example of private predicates definition in MA-PDDL.

- Probabilistic and conditional effects. Listing 2.11 reports an example of action that may fail according to a certain probability.

```
;;domain file
...
(:action sensor_power_on
    :parameters (?s - camera ?u - uav)
    ...
    :effect (and ...
                 (when (not (fail_sensor_power_on))
                       (active_sensor ?s ?u))
            )
)

;;problem file
...
```

```
(:init
    (probabilistic 0.1 (fail_sensor_power_on))
    ...
)
```

Listing 2.11: Example of action with probabilistic effects.

2.2 Planning

Development of planning languages has occurred in parallel to the evolution of planning research in which over the years have been developed several refinements of search alghoritms and new heuristics. Practically, in fact, classical planning can be treated as a problem of search in states space. Task of a planner is to create a plan (classically a sequence) of actions that permits to achieve, starting from an initial state, a final state in which goal conditions are satisfied.

2.2.1 Classical Planning

Let F be a set of propositional variables(fluents). In classical planning a state $s \subseteq F$ is a subset of fluents F' that are true, while $F \backslash s$ is a subset of fluents that, according to the Closed Word Assumption, are assumed to be false.

Let A be a set of actions, $I \subseteq F$ an initial state (a set of fluents that hold at the beginning of planning), and $G \subseteq F$ a set of goal conditions, we can define a classical planning instance as a tuple $P = \langle F, A, I, G \rangle$. Each action $a \in A$ is defined by a set of preconditions $\text{pre}(a) \subseteq F$, and a set of effects divided in an ADD-LIST ($\text{add}(a) \subseteq F$) and a DEL-LIST ($\text{del}(a) \subseteq F$). We say that an action is applicable in a state s if and only if $\text{pre}(a)$ holds in s. To apply a in s implies a state transition that results in a new state $s' = (s \backslash \text{ del}(a)) \cup \text{add}(a)$.

Finally a plan for the problem P in classical planning is a sequence of actions $\Pi = a_1, ..., a_n$ such that for each $1 \leq i \leq n$, a_i is applicable in s_{i-1}, with $s_0 = I$ and s_j (for each $j > 0$) is the state obtained applying the sequence of actions $a_1, ..., a_j$. The plan Π solves the problem P if G holds in s_n.

As shown, classical planning doesn't take into account of numerical fluents. Classical planning is in fact restricted to symbolic reasoning and there isn't the possibility of expressing numerical or temporal information. Languages used in classical planning are STRIPS and the first version of PDDL. A storical and efficient planner able to solve traditional problems is FF [11]. It performs a very efficient forsward state-space best-first search exploiting a relaxed version of GraphPlan algorithm as heuristic. In particular it searches for a relaxed plan (in polynomial time using a planning graph) and takes the relaxed-plan length as heuristic value. This efficient approach popularized FF and currently it is the basis of many state-of-art planners (even not symbolic). Many other symbolic planners have been developed in the past years and they can now manage symbolic problems in a very performant way.

2.2.2 Numerical Planning

Classical planning has been treated in depth in the past years and it can be considered a well-understood problem. However it is evident that many aspects of real world (in particular those involving numbers) are not fully represtable with such type of abstraction. The necessity of solve more complex and closer to real world problems have led to development of PDDL 2.1 and, as shown above, to the introduction of numeric fluents (e.g. not every planner supports continuous numeric effects). It is worth noting that despite numerical fluents have been introduced in PDDL 2.1 together with durative actions, they are not considered in numerical planning. Talking of durative actions we refer to temporal planning.

Numerical planning is an interesting task and have been developed many planners that can handle numerical fluents. However planning with numeric fluents is an undecidable problem [12], so the performances of planners are not always high and not every planner supports all languages features.

A numerical planning instance is analogous to a classical planning instance: it is a tuple $P = \langle F, A, I, G \rangle$, where F, I, and G are defined as for classical planning but $F = \langle S, N \rangle$ is a set composed by a set of propositional variables (S) and a set of numerical variables (N) (propositional and numerical fluents) and G is a set of goal conditions that can be both propositional and numerical. A numerical condition is a condition on the value of one or more numerical fluents, expressed as a numeric expression. It is satisfied in a state s if the numeric expression is satisfied by the values of numerical fluents involved.

A state $s \subseteq F$ is a couple $\langle S', N \rangle$ with $S' \subseteq S$, a set of propositional fluent that are true, and N is the set of numerical variables with an associated numerical value.

Let A be a set of actions, each action $a \in A$ is defined by two types of preconditions and two types of effects: propositional and numerical precondition (pre(a)) and propositional and numerical effects (eff(a)). A propositional precondition is a on the truth value of a propositional fluents $p \in S'$ that must be satisfied by the state to make the action applicable. A numerical precondition is a numerical condition on the value of one or more numerical fluents, expressed as a numeric expression using relational operators $\{=, \neq, <, \leq, \geq, >\}$. It is satisfied in a state s if the numeric expression is satisfied by the values of numerical fluents involved. A propositional effect is an assignment of a truth value to a propositional variable. As well as in classical planning, propositional effects are divided in an ADD-LIST (add(a)) and a DEL-LIST (del(a)). Finally a numerical effect is an update of the value of a numeric fluent by assigning it a specific value or increasing or decreasing it by a certain quantity.

An action $a \in A$ is applicable in a state s if and only if all propositional conditions in pre(a) holds and all numerical conditions in pre(a) are satisfied by the fluent values in s. To apply a in s implies a state transition that results in a new state in which for all propositional effects in eff(a) the relative propositional fluent's truth value is updated according to the ADD- and DEL- lists, while the truth value of the propositional fluents not involved in eff(a) remains unchanged. Similarly for all numerical effects in eff(a) the numerical fluent's value is updated according to the effect and the values of the numerical fluents not involved in eff(a) remains unchanged.

Finally a plan Π for a problem P is almost identical to a classical planning plan. Π solves P if G holds in s_n where s_n is the final state and G is the set of propositional and numerical goals, that is a set of propositional fluents that must hold and a set conditions on numerical variables that must be satisfied in the final state.

Numerical planning is a typology of planning which is commonly adopted in order to solve many planning problems of real-world domains. There are many numeric planners which in IPCs competed with each other. One of the earliest but still performing is Metric-FF [13], a 2003 planner which extends FF to numerical variables and that has proved to be very competitive in the numerical domains of the 3rd International Planning Competition [14], also allowing to optimize the final plan according to a specified metric. One of the most efficient numerical planner is COLIN [15] which is capable of reasoning with COntinuous LINear numeric changes, using a combination of forward-search and linear programming. Its forward-chaining search heuristic is similar to the one used by FF (i.e. it is based on the same relaxed planning graph used in FF), while the pruning techniques are based on temporal constraints. Even if it has been released in 2012, Colin still is a state-of-art planner and because of its versatility in handling PDDL 2.1 and 2.2 features it is the planner we preferred and the one we used more in our study, also in temporal planning (because of its support to continuous numeric effects, which, as we'll see in 5, are important in modeling temporal problems in UAV domain). A disadvantage in using Colin is that it does not perform any effective optimization. It supports plan metrics but it uses them as guidelines and it doesn't provide any optimization feature. More recently other numeric planners emerged in planning community (see LPRPG-P [16] or one of the best performer in 2014 IPC [17], SymBA* [18]) which also proved to be very performing, even if less versatile and appropriate than Colin in our domain.

Numerical planning is a very important task and it allows to solve many real-world problems. Despite this, as reported in detail in 6, it presents limitations in many situations like multi-agent domains or domains in which time and temporal constraints are fundamental. Numerical planning requires in such cases heavy pre- and post- processing phases, as well as a simulation of time passing and concurrency.

2.2.3 Temporal Planning

In order to represent and solve more easily domains and problems strictly based on time and concurrency, temporal planning is a viable alternative to numerical planning. Temporal planning has been made possible in PDDL by the introduction of durative actions which, as already shown, have a duration and can be scheduled at specific time points. The use of this kind of formalism allows to explicitly consider time in action scheduling, to permit concurrency between them and to express temporal features such as events occurring at arbitrary time points..

As well as numerical planning a temporal planning instance is analogous to a classical planning one. Notice that temporal planning doesn't necessary implies numerical planning. In fact numerical values can be only considered in terms of action durations and scheduling time but a problem may contain only propositional fluents. However temporal planning is a natural extension of numerical planning and its purpose is to make planning capabilities closer to real world needs, so we consider temporal planning starting from numerical planning.

A temporal planning instance is a tuple $P = \langle F, A, I, G \rangle$, where F, I, and G are defined as for numerical planning. The difference is that each action $a \in A$ is a temporal durative action defined as in 2.1.1.3.

A plan for a problem P is not anymore a sequence of actions but rather a set of time labeled actions $\Pi = \{(a_1, t_1), ..., (a_n, t_n)\}$ where t_i (with $1 \leq i \leq n$) is the time label associated with action a_i and represents its scheduled start time. As mentioned in 2.1.1.3 every durative action a_i has two associated events: $start_{a_i}$ and end_{a_i}, respectively at time points t_i and $t_i + d(a_i)$ (with $d(a_i)$ the duration of action a_i) at which preconditions must be satisfied and effects are applied. A plan Π in temporal planning can also be seen as an event sequence $e(\Pi)$ that includes $start_{a_i}$ and end_{a_i} for each $1 \leq i \leq n$ and is ordered by the associated times of events. Notice that because of the concurrency of actions the events associated to different actions may alternate in the ordered sequence $e(\Pi)$. If we apply the effects of events in the order reported in $e(\Pi)$ starting from the initial state I, we obtain a sequence of states $s_1, ..., s_{2n}$. Each state s_i is described by the value of domain fluents (both propositional and numerical) obtained by the application of the effect of the subsequence of events in $e(\Pi)$ from 1 to $i - 1$. Since each event in $e(\Pi)$ is associated to a specific time point, at any time we know the value of each fluent. Notice that since durative actions also have continuous effects that are applied in the segment between the *start* and the *end* of the action, we also know at any time the value of the continuously affected fluents.

A plan Π solves a problem P if G holds in s_{2n}, where G is the set of propositional and numerical goals (as defined for numerical planning) and s_{2n} is the final state obtained applying the full ordered sequence of action effects in $e(\Pi)$ starting from I. A plan is valid if each event e_i in $e(\Pi)$ is applicable in s_{i-1} (with s_j the state obtained applying the ordered sequence of action effects in $e(\Pi)$ until e_j) and continuous preconditions and effects are satisfied and applicable in the respective time segments.

Temporal planning is required in order to solve problems in which action concurrency is a key point. Because of this it is also a possible approach to multi-agent planning. In fact, allowing concurrency, it allows a centralized planning for multi-agent problems with shared and private resources. The UAV domain is an example of such domains requiring a cooperation between UAVs in order to observe a set of targets in specified temporal windows, possibly minimizing the global mission duration. For these reasons we also considered this kind of modeling in our study.

Temporal planning is a relatively recent introduction in planning competitions. Despite this many planners have been developed and many approaches to this complex task have been attempted. As already mentioned, Colin planner, supporting PDDL 2.2, is able to handle temporal planning problems and for this reason, and because of its versatility, we mainly used it in our tests. Another planner, based on Colin core, is POPF2 [19] which is a forwards-chaining temporal planner which incorporates ideas from partial-order planning during the search. It retains the Colin's ability to handle domains with linear continuous numeric effects but it doesn't fully support ADL: it cannot handle negative preconditions, disjunctive preconditions, conditional effects etc. Even if POPF2 allows plan optimization (while COLIN doesn't), we preferred to use COLIN which resulted to have a more stable behavior and supports more PDDL features like ADL and timed-initial-literals (POPF2 only supports a portion of PDDL 2.1 level 5).

In the latests IPC some other temporal planners emerged, in particular YASHP3 [20] (winner at IPC-2011 and IPC-2014) and TFD [21]. They performed very well in competitions but they also present some limitation that have led us to prefer COLIN even if less performing. In particular YASHP3 doesn't support ADL conditions, numerical state variables and timed-initial literals, as well as continuous numeric effects, while TFD doesn't support timed initial literals and continuous numeric effects. As we'll see in 5 continuous numeric effects is a necessary requirement in order to obtain feasible plans in UAV domain in presence of temporal constraints on target observation.

2.2.4 Multi-Agent Planning

Multi-agent planning is one of the most recent fronts in automated planning. Because of the computational cost of the task it has not been explored yet like instead single-agent planning has been. However in the last few years some approaches and strategies emerged.

Multi-agent planning is the problem of planning for a group of agents in order to achieve goals that single agents couldn't be able to achieve by themselves, or at least to achieve them better. A solution to a multiagent planning problem is a partially ordered plan that as well as in classical planning results in a set of achieved goals for some of the agents. The main difference is that a plan is not necessary centralized but it can be created and distributed on each agent which performs a private planning supported by its private and available actions.

Two approaches are in fact available in order to solve multi-agent problems: a centralized planning approach and a distributed one. In general we can say distributed approaches in planning creation are preferable for many reason (e.g. the maintaining of agents privacy and autonomy, the cost of a centralized systems, the easiness in re-planning in case of contingencies, etc. [22]). However, despite individual planning and post-planning coordination phase are a good approach, in case of problems that require many interactions among agents the coordinations activities are too expensive.

Some recent study tried to investigate how multi-agent centralized problems can be automatically encoded to PDDL and solved via temporal planning [23]. This approach allows to express constraints between concurrent actions through numerical fluents. The work shown that this kind of problem, even if still simplified in the study, are also representable (as we also did in this thesis) as temporal planning problems.

An alternative adopted approach in literature is an integrated planning and coordination approach. MAP-POP [24] is an integrated multi-agent planner which combines planning and coordination through a multi-agent POP-based refinement planning procedure. It proved to efficiently solves loosely-coupled problems and also to be competitive when solving problems with a higher coupling level and cooperation among agents. More recently it has also been presented FMAP [10], a new planner based of MAP-POP which uses a language based on PDDL 3.1 and presents a novel approach to integrated MAP in which each agent implements a forward-chaining POP performing a multi-agent weighted A^* with a distributed heuristic based on DTG (Domain Transition Graph). We tested the capabilities of FMAP on our purposes and we analyzed the limits and the possibilities it provides. This kind of systems present some practical limitations. It is in fact hard to represent temporal or numeric information (despite PDDL 3.1 fully supports them, planner doesn't) so many natural constraints or features of UAV domain are difficult or impossible to define.

It is also worth mentioning some different but related areas of work in multi-agent planning. A first alternative approach is modeling multi-agent problems directly in terms of a centralized mixed-integer linear programs. Some MIT works ([25], 26) shown that in multi-UAV domains this approach behave relatively well considering heterogeneous vehicles, timing precedence constraints between observations, no-fly zones and vehicle dynamics, but without considering time windows of observability of targets. This approach has proved to be a valid alternative in problems with no particular temporal requirement on target observation (all targets are observable during the entire mission duration). In this thesis we show that similar results are obtainable with state-of-art planners and a right modeling of problems, also including temporal constraints.

Another important and in evolution kind of approach in multi-agent temporal planning involves CSP (Constraint-satisfaction problems). CSP are used in a great variety of application areas ([27]) including planning. Certain classes of temporal planning problems can in fact be addressed recurring to CSP solving techniques, supported by the SMT (Satisfiability Modulo Theories) which generalizes boolean satisfiability (SAT) by adding richest theories such as arithmetic, quantifiers or first-order theories. SMT solvers (such as Z3 [28]) are able to efficiently check formulas with hundreds of thousand of variables and have proven to be a valid alternative to solve planning problems. There are also in literature examples of SAT-based planners (such as TM-LPSAT [29]) which demonstrated that this kind of approach can be successfully extended to temporal planning problems including real-valued fluents, exogenous events, atomic and durative actions with numeric parameters and numerical continuous changes.

Chapter 3

The class of problems

In this chapter we describe the class of problems we addressed in this thesis. We adopt an increasing complexity approach in the description. We first describe the general properties of the class of problems and then we introduce new features which increases the complexity of the problems.

3.1 The baseline

Let us consider a class of problems characterized by a set of robotic agents $R = R_1, R_2, ..., R_k$ which have to explore and monitor a given environment in order to get information about specified targets. Agents can get observation on a portion of the environment depending both on the their location and on the characteristics of the devices of which are equipped. The robotic agents have perceptive abilities (mainly via observation devices) and can move in the environment but they do not have manipulation abilities (i.e. they do not alter the environment since their task concern monitoring/exploration). The robotic agents can perform activities in concurrency (limited by some constraints).

A set of devices $D^i = D^i_1, , D^i_{n^i}$ is put on board of the robotic agent R_i. The devices allow the agent to get information from the environment and each device has a number of capabilities. In principle the devices can work in parallel, even if there are some configurations requiring mutual exclusion. At least one device has to be on board of R_i.

A robotic agent R_i has an internal status S_{R_i} which models both qualitative aspects of R_i and the numerical ones (i.e. the numerical value of the resources of R_i). Resources of an agent R_i are a set Res_{R_i} of consumable resources (e.g. fuel level or battery charge level) which can be modeled as numerical fluents.

As well as robotic agents each device D_i has its own status S_{D_i} which models both qualitative aspects and the numerical ones (i.e. the numerical value of the resources of D_i). Resources of a device D_i are a set Res_{D_i} of consumable resources (e.g. memory status or battery charge level) which can be modeled as numerical fluents.

We assume that resources of both agent and devices are partitioned among the elements of the system, so that there is no shared resource. We also consider time as a resource and even if it is conceptually a shared resource it doesn't impose constraints on shared usage. It is explicitly represented in metric terms and it has an absolute scale and it is the same for all agents in R and all devices.

The overall task of the robotic agents is to acquire information about the environment. A robotic agent R_i is able to acquire information just when it is sufficiently close to the portion of environment it has to observe. For this reason an important notion is the one of target which specifies the entity to be observed and some basic properties of such an entity. A problem therefore specifies a set of targets $T = T_1, ..., T_t$ to be observed by the robotic agents in order to satisfy a goal G.

Two types of targets are considered: Point Targets (POINT) and Line Targets (LOC). In the former case the target is characterized by one vertex described by a tuple of coordinates in space. In particular we restricted our space at bi-dimensional space so a position can be defined as a couple of coordinates. In the latter case (LOC target) it is characterized by a sequence of vertices defining a polygonal chain.

The information about the targets to be observed is just part of the request that the system has to satisfy. In fact, the target specifies just what has to be observed, but it is also very important to specify how it has to be observed and how long.

For this reasons we define the concept of observation requests $ObsReq = \langle tr, md, dv \rangle$ where $tr \in T$ specifies the target to be observed, $md \in \mathbb{R}$ specifies the minimum duration for the actual observation and $dv \in D$ is a parameter specifying the constraint on the device to be used for the observation. It is quite clear that tr

represents what has to be observed, dv how t has to be observed and md how long t has to be observed.

We define $OR = (ObsReq_1, ..., ObsReq_m) \subseteq G$ as the set of observation requests to be performed in order to solve a problem and a problem is considered solved just in case all $ObsReq_i \in OR$ are achieved.

3.1.1 Agent's capabilities

In order to perform this complex task, agents have at disposal a set of actions. It is worth notice that the modelled capabilities of agents abstracts away from their low-level sensor or motor commands. The complexities of agents movement, engines power cycle, sensors pointing, data acquisition, etc., witch are obviously important in a completely accurate representation of a real-world problem are completely abstracted away by the agent capabilities formulation of our model, and assumed to be dealt with by low-level control processes. However the adopted abstraction level allows to define actions for planning purposes oriented to a centralized planning system of managing and coordination of teams of (possibly heterogeneous) agents and the low-level controls are demanded to single agents managing.

3.1.1.1 Transfer actions

The main qualitative aspect of an agent R_i concerns its position. For this reason there are *Transfer* actions which allow changing the position of R_i. A position is defined by a couple of coordinate in a bi-dimensional space. A *Transfer* action is an action $A^T = \langle r, p_s, p_e, t_s, d \rangle$, where $r \in R$ is the robotic agent which performs the action, $p_s \in \mathbb{R} \times \mathbb{R}$ is the initial position of the agent r when it start executing A^T, $p_e \in \mathbb{R} \times \mathbb{R}$ is the final position of r after the execution of A^T, $t_s \in \mathbb{R}$ is the time point at which A^T is scheduled and $d \in \mathbb{R}$ is the action duration. As all other actions, a A^T action is located in the time line; in particular the actions starts at time t_s and ends at time $t_e = t_s + d$.

It is worth noting that in a specific domain we could have many variants of transfer depending on the nature of the robotic agent r. For example if r is a UAV we could have actions such as *TakeOff*, *Landing*, *Loiter*, etc..

3.1.1.2 GetInfo actions

Each device D_i is characterized by an action *GetInfo* which allows the agent on which the device is on board to acquire information on a target. A *GetInfo* is an action $A^M = \langle dv, tr, t_s, d\rangle$ where $dv \in D$ is the device which performs the action, $tr \in T$ is the target observed, $t_s \in \mathbb{R}$ is the time point at which A^M is scheduled and $d \in \mathbb{R}$ is the action duration. This action has an impact on Res_r (with r the agent on which the device is on board). The action A^M starts at time t_s and ends at time $t_e = t_s + d$. One typical precondition of such an action concerns the "proximity" of r to target tr. In order to observe a target, in fact, the agent has to be within a certain distance from the target itself. This distance reflects the radius of action of the device, that is the radius within the device is able to successfully observe the target. It is also worth noting that this condition should be satisfied for the whole duration of the action. This is a tricky situation because there is concurrency between agent r and its devices. This implies that satisfaction of the "proximity" condition depends on the actions concurrently performed by r (because dv is on board r) so that suitable predicates have to be defined by the action performed by r and should persist during the action performed by dv.

In order to be scheduled by a planner all action preconditions have to be satisfied. Each agent's action specifies not only preconditions on the qualitative aspects of p_s, but also on the Res_r (the agent's resources) and on time, as well as each device's action specifies precondition both on qualitative and quantitative aspects.

It is worth noting that in our modeling each agent, as well as each device, is able to only perform one action at a time. This simplification is allowed by the chosen level of abstraction which hides logistics and allows to only consider one high level action at a time for each agent or device. Despite this an agent and its devices can concurrently execute their actions because they are considered as different entities, even if highly correlated. This allow to a team of robotic agents to concurrently act in the environment and, for each robotic agent R_i, it allows to the suite of devices on board to R_i to concurrently perform *GetInfo* actions while R_i is moving in the environment.

3.1.2 The planning problem

In order to define the planning problem to be solved we have to specify the goal and the initial state.

3.1.2.1 Goal

As already mentioned in the specification of the problem we have a set of observation requests OR (which express temporal constraints and devices for the observation of a target) that must be satisfied.

We have also other temporal constraints that the entire plan (mission) must satisfy. In particular we can define these constraints as the tuple $MTC = \langle t_e, t_l, d_{min}, d_{max} \rangle$ where MTC means *Mission Temporal Constraints* and t_e specifies the earliest time when the mission can start, t_l specifies the latest time when the mission can end, d_{min} represents the minimum duration of the actual mission and d_{max} represents the max duration of the actual mission.

Furthermore there is also a set of resource constraints $RC = (RC^R, RC^D)$ where $RC^R = (RC^R_1, ...RC^R_k)$ are the robotic agents resource constraints (with RC^R_i the set of resource constraints relative to consumable resources (e.g. fuel in the tank, battery power, etc.) of agent R_i and k the number of agents) and $RC^D = (RC^D_1, ...RC^D_d)$ are the devices resource constraints (with RC^D_i the set of resource constraints relative to consumable resources (e.g. available memory, battery power, etc.) of device D_i. All resources must be managed in a way that its amount results greater than a certain threshold at any given time. In fact each action performed consumes an amount of these resources, possibly proportional to its duration, e.g. a transfer consumes fuel in proportion both to the distance and the speed of the agent.

We can finally define the goal as the set $G = (OR, MTC, RC)$ where OR, MTC and RC are defined as above.

3.1.2.2 Initial state

The initial global status I for this class of problems details the time windows during which each property of the domain's objects holds as well as the initial value of the numerical variables.

Let S and N respectively be the set of predicates (propositional variables) and the set of functions (numerical variables) defined in the domain. Let also be $H = (H_1, ..., H_h)$ a set of tuples, where H_i is a tuple $\langle p, t_s, t_e \rangle$ with $p \in S$ a propositional variable and $[t_s, t_e]$ (with $t_s \in \mathbb{R}^+$ and $t_e \in \mathbb{R}^+ \cup \{+\infty\}$) a definition of an interval where p holds. Notice that $t_e = +\infty$ means that the interval is open.

We can define the initial state as the couple $I = (H, N_I)$ where H is the set defined above and N_I is an initial assignment of numerical values to the numerical variables in N.

Notice that as well as the values of numerical variables, the truth value of a propositional fluents may change over time as a result of the application of an action. Notice also that the value of some numerical variables will never change because they represents (as well as the validity intervals of propositional variables) exogenous characteristics of the environment (e.g. the consumption rate of an agent).

Some of the variables that could be used for describing the domain are

- Robotic agents information. They concern agents qualitative and numerical characteristics like initial position, cruise speed, consumption rate, etc.
- Devices information. They concern devices qualitative and numerical characteristics like device status or modality, maximum available memory, battery level, etc.
- Inter-agent relations information. They concern information of devices loaded on board the agents.
- Targets and observation requests information. They concern information about the target geometries, the position in the environment, temporal information (required minimum duration of observation) and information on devices required for the observation.

- Environment information. They concern information like the size of the environment, the distance between objects, etc.

Notice that these pieces of information describe the baseline class of problems. More information will be added in the following sections and a detailed description of the specific variables used in our modeling will be reported in chapters 6 and 5 according to the planning strategy adopted and the encoding phase of the problem.

3.1.2.3 Solution

A solution for the class of problems above defined is a plan Π that ends in a final state s_n in which all goals in G are satisfied. A plan is a set of temporally located actions $\Pi = (A_1, ..., A_n)$ and we consider it a valid plan if:

- All constraints among observation requests $\in OR$ are satisfied. In particular there is exactly one *GetInfo* action for each observation request, that is for each $ObsReq_i = \langle tr_i, md_i, dv_i \rangle$ there is an action $A_k^M = \langle dv_k, tr_k, t_k^s, md_k \rangle$ such that dv_k matches dv_i, tr_k matches tr_i and md_k is $>= md_i$.
- The plan satisfies all temporal constraints in MTC (as defined above). In particular for each action $A_i \in \Pi$, A_i starts after t_e and it ends before t_l. Furthermore the total plan duration is between d_{min} and d_{max}.
- The plan satisfies all resources constraints in RC (as defined above). In particular for each set of constraints $RC_i \in RC$ the amount of each of them results greater than a certain threshold at any given time of the mission.
- The preconditions of the actions are satisfied by the global state of the system in which they are applied. With an opportune modeling of actions it also implies that the plan satisfies all resources constraints in RC.
- The mutual exclusion constraints among actions are satisfied.

3.2 UAV extension

A first extension of the baseline class of problems is the instantiation of robotic agents with UAV agents.

The use of UAVs instead of simple robotic agents leads to an increase of the problem complexity. In fact it both implies additional agents capabilities and a series of additional constraints that a solution must respect. Therefore we have to extend the above defined agents capabilities as well as the planning problem definition.

A UAV is obviously a more difficult to manage entity than a generic robotic agent. Omitting again the specific logistics (demanded to a lower level control system) there is a series of conditions we have to take into account in order to build a consistent plan for a team (fleet) of UAVs.

As well as a generic robotic agent, each UAV R_i is equipped with a set of devices $D^i = (D^i_1, ..., D^i_n)$ which allows it to get information from the environment. In particular the devices are sensors that can be of various types (Electro Optical, Radar, Hyperspectral, Lidar, etc.) and the information from the environment typically concern specific targets for surveillance purposes (e.g. areas prone to natural disaster, etc.). A UAV is able to get information of a target by flying at a certain altitude and within a certain distance from the target depending on the sensor to be used. Each UAV is able to load on board a limited number of sensors: typically two or three on MALE (Medium Altitude Long Endurance) or MAME (Medium Altitude Medium Endurance) UAVs.

Notice that talking about UAV's class of problems we'll refer to *GetInfo* actions as monitoring or observation actions. So a *monitoring* action o_i is a *GetInfo* action performed in order to accomplish an observation request $OR_i \in OR$.

3.2.1 Agent's capabilities extension

As well as a generic robotic agent a UAV requires *Transfer* actions to move around in the environment. As already mentioned a UAV requires different actions of this type in order to handle different moving situations.

In particular, beside the already defined *Transfer* action, a *TakeOff* action allows the UAV to take off from the airport where it is initially located and to reach its cruising altitude.

A *Landing* action allows the UAV to land at the airport. Typically the landing airport correspond with the departure one for logistics reasons.

Some *Loiter* actions allow the UAV to execute a particular flight pattern in order to maintain its position within a certain area. This kind of action is useful both in case the UAV needs to wait a certain amount of time before to proceed to the next task and in case the UAV must stay on a target for a certain amount of time (e.g. the minimum duration of observation) in order to accomplish a *ObsReq*.

Each of these action is a specialization of the *Transfer* action and it is similarly defined. Notice that for the *TakeOff* and *Landing* actions, p_s and p_e are not variables but they are defined a priori. Also notice that p_s of the *TakeOff* action and p_e of *Landing* action represents points on ground while p_e and p_s (respectively of *TakeOff* and *Landing*), as well as all the points between these two actions, are up in the air.

It is important notice that each transfer action has not only an impact in terms of changing the position of the UAV but also in terms of resources consumption and time spending. This is one of the reason why it is important to model them.

An important aspect of agents capabilities concerns the necessity to continuously perform some task. Unlike a generic robotic agent which can simply do nothing for a certain amount of time (possibly without consuming resources), it is not possible for a UAV agent. In fact also "doing nothing" requires a certain amount of consumable resources (e.g. fuel or battery charge) ad therefore has a potentially significant impact on the final plan. Furthermore "doing nothing", meant as hover, as already mentioned is not a kind of action that all UAVs are able to perform. In particular UAVs that aren't VTOL (Vertical Take-off and Landing), such as MAME or MALE UAVs, are not able to perform it. As already mentioned, this introduces a significant complexity level in some practical PDDL problem types of modeling, especially in temporal planning where it can only explicitly modeled using requirements not supported by all planners.

3.2.2 The planning problem extension

The planning problem definition for the UAV class of problems extends the baseline definition. In particular it requires some additional goals as well as new constraints and variables.

3.2.2.1 Goal

We can mainly introduce a new set of goals that must be satisfied by the final plan, concerning the UAVs positions. We can define a set $L = (L_1, ..., L_k)$ (with k the number of UAVs in the problem) where L_i is a goal which requires the UAV R_i is landed in the final state.

We can then define the goal as the set $G = (OR, MTC, RC, L)$ where OR, MTC and RC are defined as in baseline class of problem definition and L is the set of goals here specified.

3.2.2.2 Initial state

The initial state can be defined in the same way of the baseline class of problems. The main difference is that some additional variables are necessary for describing the domain.

In particular additional pieces of information about airports have to be introduced, concerning the initial and final positions of the UAVs. [1]

3.2.2.3 Solution

The solution, as well as the initial state, can be defined in the same way of the baseline class of problems as a plan $\Pi = (A_1, ..., A_n)$ where each A_i is a temporally located action.

In order to consider Π a valid plan it has to satisfies some more constraints than a baseline plan. In particular:

- In the final state s_n all UAVs must be landed.
- A UAV executes exactly one *TakeOff* and one *Landing* actions.

[1]Notice that in order to carefully model the domain it is also useful to provide information about the take off and landing of a UAV. In particular it is useful to define the position of the UAV after its take off and the position from which it can start landing at a certain airport. These information depend both on the airport's structure and on the UAV's type (e.g. a MAME UAV requires a different type of landing than a Micro-Uav). For this reason also more agents information are required, such as the time needed by a UAV to take off or land, temporal and spatial information on loiter operations, etc.

- A UAV that took off covers at least one target observation.
- All the actions of a UAV must meet each others. That is no holes between the actions of a UAV are allowed.

3.3 Temporal constraints between targets observations

A further extension of the class of problems concerns the introduction of a new set of temporal constraints among pairs of *ObsReq*. Because of the temporal dimension, the *ObsReq* may not be completely independent on each other. In particular we introduce the capability of specifying a set of temporal constraints among pairs of observation requests, based on Allen's interval algebra predicates ([30]).

Notice that for simplicity in the following we refer to *observations*. The specified constraints concern the specific *GetInfo* action o_i that have to be performed in order to satisfy the observation request $OR_i \in OR$. These temporal constraints have the form:

- $After(o_1, o_2, t_1, t_2)$. I.e. the observation $o_1 \in \Pi$ has to start after the conclusion of the observation $o_2 \in \Pi$ and the temporal gap should be included in $[t_1, t_2]$ with $t_1 \leq t_2$ and $t_1 \in \mathbb{R}^+$ and $t_2 \in \mathbb{R}^+ \cup \{+\infty\}$.
- $Before(o_1, o_2, t_1, t_2)$. I.e. the observation $o_1 \in \Pi$ has to finish before the start of the observation $o_2 \in \Pi$ and the temporal gap should be included in $[t_1, t_2]$ with $t_1 \leq t_2$ and $t_1 \in \mathbb{R}^+$ and $t_2 \in \mathbb{R}^+ \cup \{+\infty\}$.
- $Overlap(o_1, o_2, d)$. I.e. the observations $o_1 \in \Pi$ and $o_2 \in \Pi$ have to be performed in the same temporal interval and the duration of this joint activity must last at minimum d with $d \in \mathbb{R}^+$.
- $Equal(o_1, o_2, d)$. I.e. the observations $o_1 \in \Pi$ and $o_2 \in \Pi$ have to be performed in the same temporal interval and the durations of the two action must not differ by more than d with $d \in \mathbb{R}^+$. Notice that d is a flexibility parameter which allows to modeling realistic small time intervals of synchronization between observation performed by different agents.

It is worth noting that this extension leads to a significant increase of the problem complexity. In particular a temporal constraint between two observations that must be performed by different agents requires a strong coordination between them. In order to satisfy such type of constraints agents must both cooperate with each other and opportunely schedule their action to perform the constrained observation in the right time interval. It is clear that it is not a trivial problem. Because we adopted a centralized planning approach it is the centralized planner that must face this difficulty. For this reason it has no impact on agents capabilities in our modeling. Conversely it affects the planning problem definition.

3.3.1 The planning problem extension

The introduction of temporal constraints between targets observations leads on goal and consequently on the solution of the problem.

3.3.1.1 Goal

We have to introduce a new set of goals that must be satisfied by the final plan, concerning the described temporal constraints. We can define a set $M = (M_1, ..., M_m)$ where M_i is a goal of the form above specified. It temporally constraints two targets observations.

We can then define the goal $G = (OR, MTC, RC, [L,]M)$ where OR, MTC, RC are defined as in the baseline class of problems, L is the set of goals added in UAVs class of problems extension and it is present only if that is the class of problems considered, and M is the set of goals here specified.

3.3.1.2 Solution

The solution can be defined in the same way of the baseline class of problems as a plan $\Pi = (A_1, ..., A_n)$ where each A_i is a temporally located action.

In order to consider Π a valid plan it has to satisfy some more constraints than a baseline plan (or a plan for the UAVs extension). In particular all the goals specified in M have to be satisfied by the final plan, that is for each $M_i \in M$ constraining two target observations o_1^i and o_2^i, Π must contain two observation actions A_j^M and A_k^M such that their temporal location satisfies the constraint expressed by M_i (e.g. for a $Before(o_1, o_2, t_1, t_2)$ constraint, Π must contain two actions A_j^M and A_k^M respectively related to o_1 and o_2 such that $end_{A_j^M} < start_{A_k^M}$ and $t_1 \leq (start_{A_k^M} - end_{A_j^M}) \leq t_2$).

3.4 Time windows on targets observation

Another important extension of the class of problems concerns temporal constraints on targets observation in terms of time windows of observability.

Until now we considered the targets as observables during the entire time of the mission. Now we aim to reduce the observability of targets at limited time windows.

At this purpose we introduce a set of temporal requirements that specifies the time windows within targets are observable. In particular we define a concept of target observability window $W_i = (W_i^1, ..., W_i^w)$ as a set of disjunct observability windows W_i^j that define time intervals $[e_i^j, l_i^j]$ in which target T_i is observable. $e_i^j \in \mathbb{R}$ specifies the earliest time when T_i is observable and $l_i^j \in \mathbb{R}^+ \cup \{+\infty\}$ specifies the latest time.

If $|W_i| = 1$ there is only one time windows in which the observations on target T_i can be acquired. But it is also possible to define more than one observability windows. Notice that they must be disjunct so if an interval IT_k has no latest time constraint (latest time with $+\infty$ value) there must not be any other observability window starting after IT_k.

As well as the introduction of temporal constraints between targets observations, the introduction of time windows on targets observation leads to a significant increase of problem complexity. In fact it requires the planner to schedule agents observation actions within the observability windows. This has an impact both on the scheduling of actions which are required by the observation (e.g. a *Transfer* action which takes the agent near the target) and on the following actions whose scheduling is affected by the previous actions.

3.4.1 The planning problem extension

The introduction of such temporal constraints has an impact on goals, initial state and solution definition. Therefore also in this case we need to extend the definition of the planning problem.

3.4.1.1 Goal

We have to introduce a new set of goals that must be satisfied by the final plan, concerning the specified temporal constraints. We can define a set $W = (W_1, ..., W_{|T|})$ where W_i is a goal of the form above described which specifies within what time intervals a target can be observed and $|T|$ is the number of targets.

We can then define the goal $G = (OR, MTC, RC, [L,][M,]W)$ where OR, MTC, RC are defined as in the baseline class of problems, L is the set of goals added in UAVs class of problems extension and it is present only if that is the class of problems considered, M is the set of goals which must be introduced in order to extend the class of problems with temporal constraints between targets observations and W is the set of goals here specified.

3.4.1.2 Initial state

We also need to extend the baseline definition of the initial state of the problem. It is necessary to introduce the new information on targets observability windows. In particular for each target $T_i \in T$ we have to specify all time intervals $W_i^j \in W_i$ in which it is observable.

In order to represent this constraints in a PDDL problem we can use timed initial literals which allow to easily model this kind of information. Notice that if an interval has no limit on latest time of observation it is sufficient to specify the earliest time of the interval.

This information can be then used as additional precondition in *GetInfo* observation actions in order to allow a target observation only in the specified observability windows and consequently satify the goals.

3.4.1.3 Solution

The solution can be defined in the same way of the baseline class of problems as a plan $\Pi = (A_1, ..., A_n)$ where each A_i is a temporally located action.

In order to consider Π a valid plan it must satisfy some more constraints than a baseline plan (or a plan for one of the described extensions). In particular all the required observation requests in OR must be satisfied during a time window in which the target to be observed is observable.

3.5 Target observations assignment

Parallel to the reported extensions of the baseline class of problems there is another important aspect to take into account.

Each described extension, in fact, can be specialized in two different ways concerning the assignment of the observation requests to the agents.

A first definition concerns a priori assignment of the targets to the agents involved in the planning problem. For each $ObsReq_i \in OR$ it is introduced an additional constraint which forces the assignment of $ObsReq_i$ to a specific agent (chosen from those in the problem). This model requires additional information concerning the assignments both in the initial state definition and in the goals and obviously it has an impact also on the validity of a plan (similarly to the already described extensions).

A second definition, instead doesn't specify any assignment of observations to agents, leaving complete autonomy to the planner in order to satisfy the problem. Obviously it entails an increase of problem complexity because of the wider set of agents usable to satisfy the observation requests.

It is worth noting that in the former case it is required a human intervention in order to assign the observations to the agents. This implies that the set of valid plans for a problem with assigned target observations is smaller than the set of valid plans for the same problem without assignments. This also implies that in some cases there may not be a solution for the first kind of problem because of bad assignments but there may exist a solution with a different assignment.

A further intermediate extension of this type may lead to agents capabilities in terms of subsets of OR that each agent can satisfy. This kind of extension gives only a partial autonomy to the planner in assigning observations to agents and requires only a intermediate human intervention in order to specify which agents are able (or allowed) to satisfy the observation requests.

Chapter 4

The architecture

In the previous chapter we defined the class of problems we aim to face in this thesis. Because of its complexity, in order to model it and take into account all its characteristics it is not possible a direct encoding of the problem requirements into a PDDL implementation (e.g. temporal constraints cannot be directly specified in a goal state). It is necessary a translation process starting from the general problem specification to the PDDL implementation (encoding process), as well as a translation from the planner solution to a human easy to read representation (decoding process). Notice that, in order to solve the class of problems, we decided to exploit state-of-art temporal and numerical planners instead of building a new planner or extending the existing ones. This choice is mainly due to the fact that state-of-art planners implementation of algorithms is complex and often already optimized. Therefore building a new temporal planner which is competitive with the state-of-art planners would be a task that cannot be done in the time required for a thesis of master degree.

In this chapter we present the adopted architecture which allows the translations and planning for the class of problems according to the choice of using a PDDL 2.2 formalism (see 2.1.2 for more details).

Fig. 4.1 reports a flowchart representing the entire translation and planning process. Starting from a general definition of the problem and the requirements (which is the input of the entire process specified by the user), the encoding process translates them in the PDDL formalism. The process generates two files which provide a problem description and a domain model in the PDDL syntax. The files are then feed to a general purpose planner (able to deal with PDDL 2.1 and PDDL 2.2) in

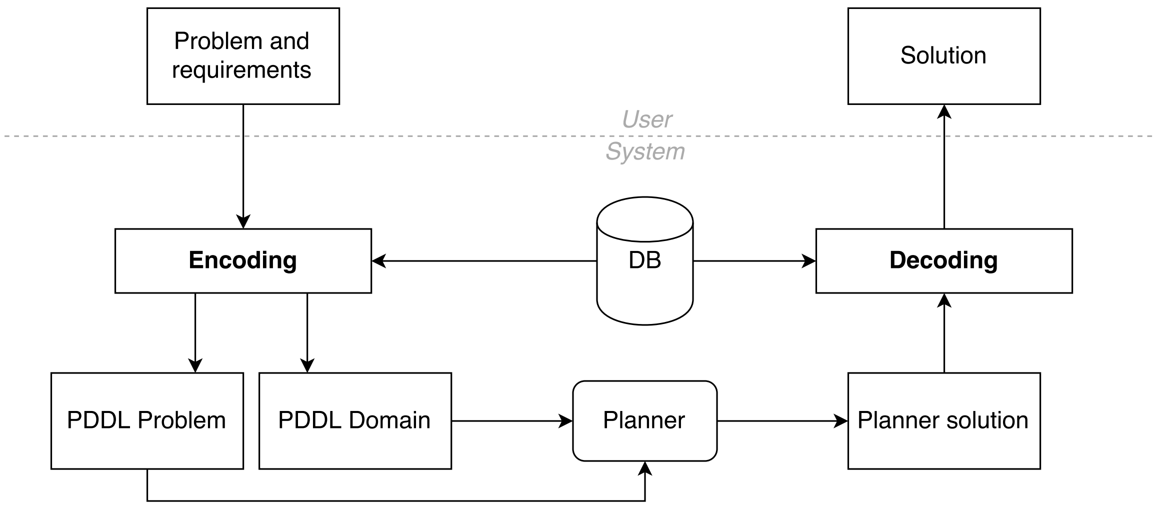


FIGURE 4.1: A graphical representation of the translation process.

order to find a solution for the problem. The decoding process finally translates the planner solution in a readable output to the user.

4.1 User input

The user provides in input a formal high level description of the problem. The description defines the elements involved in the problem and the constraints that must be met by the final plan. In particular it specifies a representation of a subset of information reported in the previous chapter in goal and initial state sections, according to the adopted extension of the class of problems. Therefore some pieces of the information it defines are:

- The set of robotic agents that must be involved in the problem (UAVs in case of UAV extension of the class of problems).
- The set of targets that must be observed and the requirements on their observation (i.e. the required minimum durations of observations and the type of devices to be used for the observations). In case one of the extensions of the class of problems is adopted it may also define time windows within the observations has to be done and/or temporal constraints between pairs of target observations.

- Temporal constraints for the entire plan: the time window within the plan has to be performed (i.e. earliest and latest time for the mission) and the maximum duration.
- The assignment of the observation requests to the agents. Notice that this information, if provided, is the result of some choices of a human operator. However we are assuming that the assignments are consistent with the actual ability of the agents to observe the target assigned with the devices loaded on board in the time frame of the mission. Consistency controls are demanded to a higher level structure which is beyond the purpose of this thesis.

4.2 Encoding

The encoding process concerns the definition of the initial state, the actions schema, and the goal state, in the chosen PDDL formalism.

The following chapters will describe in detail the encoding phase according to the planning model (i.e. temporal planning or numerical planning). This process, in fact, highly depends on the chosen model. The representation of some concepts or requirements differs from one model to the other because of the different formalisms and the limits of the models. A simple example of what said concerns time representation and management: while in temporal planning planners directly handle time and actions are automatically located in time, in numerical planning it is necessary to explicitly represent time. This has a significant impact both on actions schema definition (because it requires more preconditions and effects concerning the time management) and on the definition of fluents in domain and their initialization in initial state, as well as a different modeling of some of the requirements (such as temporal constraints between targets or time windows on targets observations). For these reasons we postpone to the following chapters the specific encoding process. In particular in chapter 5 we show how to encode the problems in temporal planning, while in 6 we show how to do it in numerical planning.

It is worth noting that beside the mere translation of user information and requirements in the planning model formalism, the encoding phase also introduce all the information that are necessary to correctly define the domain and the problem and that are beyond the user interest and knowledge. In particular with the support of an internal knowledge base (DB in Fig. 4.1) the system also provides to the planner (through the domain and the problem files):

- The required agents logistic information, such as their initial position, cruise speed, consumption rate, etc., as well as the suite of devices loaded on board.
- The target geometries and their position in the environment.
- Information on the environment, such as its size, the distance between objects, etc.
- Predicates and functions of support to the planning (e.g. status variables for agents or boolean variables which enable/disable actions)

This kind of information is necessary in order to carefully and correctly define the domain and the problem but they are hidden from the user because they only depend on domain and planning model.

Encoding, according with the chosen planning model, also performs a series of simplifications on objects representation in order to simplify the planning.

Notice that typically the domain is not influenced by the user input and for a specific class of problems it is unique for each planning model. Therefore the domain requires an encoding (and that's the reason in fig. 4.1 it is between encoding phase and the planner as well as the problem) but because it depends only on the class of problems and on the planning model in real usage it is encoded only once and directly fed to the planner when required. On the other hand it is also worth noting that consider the PDDL domain as independent from the problems is not realistic. Actions schema, objects, predicates and functions definition, as we'll see in the following chapters, strictly depend on the type of problems. Problem constraints and requirements in fact deeply influence the set of necessary fluents and the preconditions and effects of actions.

4.3 Planning

The encoding process generates two PDDL files representing the domain and the problem. These files are given as input to a domain independent planner to search for a solution.

The choice of planner strictly depends on the planning model used in encoding phase. As mentioned in chapter 2 not all planners support the totality of features of the PDDL extensions. Therefore it is necessary to employ a specific planner according to the choices made during the encoding phase.

Planning phase aims at finding, if possible, a solution to the specified problem. A solution may be, depending on the adopted planner, optimal or sub-optimal. It is reported according to a formalism which is characteristic of the planner and it is not particularly user-friendly (especially in case of multi-agent problems because actions are mixed together in the plan).

However we can distinguish two kinds of formalisms adopted by almost all planners and depending on the planning model.

In case of temporal planning a plan is presented by the planner as a sequence of actions in the following format:

$$t:\ (action\ p_1\ ...\ p_n)\ [d]$$

where t is the starting time of the action (with respect to the zero time of the problem), *action* is the name of the action, p_1 ... p_n are the action's parameters and d is the duration of the action, expressed in the unit of time implicitly chosen while initializing fluents or timed initial literals in the initial state. An example of action may be the following (in a UAV domain):

$$10.000:\ (take\_off\ uav1)\ [240]$$

It states that the `uav1` is scheduled to *take off* at time point (second) `10` and the duration of the `take_off` action is `240` seconds. Notice that the actions in the plan are ordered by their starting time.

In case of numerical planning, instead, a plan is presented as a sequence of actions in the following format:

$$n:\ (action\ p_1\ ...\ p_n)$$

where n is a numerical value representing the order of the action in the total ordered plan (it gives an order between actions in the sequence but this information is not particularly useful. The order in fact corresponds to the order of the actions in the sequence), *action* is the name of the action, p_1 ... p_n are the action's parameters. An example of action may be the following (in a UAV domain):

$$1:\ (take\_off\ uav1)$$

It states that the `uav1` is planned to perform as first action a `take_off`. Notice that in this case the solution provided by the planner is less clear than the temporal planning one in terms of temporal location of the actions. In fact no information is provided about duration or starting time of the actions. This implies that, especially in case of multi-agent plans, it is not easy to understand where concurrent actions are temporally located. As we'll see in chapter 6 it has significant consequences in terms of both the architecture and domain's elements and actions required by the numerical model. Concurrency, in fact, is a key aspect in a multi-agent system and it is present both between different robotic agents and between a robotic agent and the on-board devices and also between the devices themselves. Without information in the final plan about the temporal location of the actions it is necessary to develop strategies to successfully handle concurrency aspects. Some of these aspects (such as concurrency between robotic agents) must be simulated, while other concurrencies can be simplified by an abstraction on actions (e.g. by creating a joint action which encloses a transfer action of a robotic agent and a monitoring action performed by a device mounted on board). In chapter 6 we will describe in detail the modeling solutions needed to address these problems.

4.4 Decoding and output

The decoding of the solution can be performed if the planning ends up successfully. This procedure depends on the output of the planner and in particular it depends, as above reported, on the chosen planning model, because it affects the solution format.

This process allows to translate the planner solution in a representation easy to read by a human. The output of this translation can be of various types (e.g. text format, Gantt diagrams representing the scheduling, route of the agents on a map) according to the needs. For some examples of decoding see 7.3.

It is worth noting that actions (both in temporal and in numerical planning) are characterized (among other things) by a duration. However, as already mentioned in 4.3, while temporal planning solutions easily allow to temporally locate the scheduled actions because they are temporally annotated in the plan, decoding in numerical planning instead is much more complex. Because of the lack of temporal information in numerical plans, decoding requires a complex underlying structure which preserves information from the encoding phase and exploits them during decoding. We'll describe in detail the process in the following chapters according to the planning model.

Chapter 5

Temporal planning

In this chapter we present how to deal with the encoding of the class of problems (and its extensions) described in chapter 3 using a temporal planning model. We describe how to encode the high level description of problems and requirements in PDDL 2.2 formalism. The PDDL 2.2 extension is a requirement to exploit both durative actions and, when needed, timed initial literals features.

Notice that encoding phase doesn't only performs a translation of the high level problems' requirements and of the information required by the planners into a PDDL formalism. This process also executes a series of important abstractions and simplifications which facilitate the planning phase. For instance we encode a LOC target which is described by a sequence of waypoints with only two points: the initial and final ones of the polygonal chain which describes the target. In order to maintain the consistency of LOC target information, the time required by the robotic agents to observe a LOC target (as well as the other resources required or the distance between the two points representing the LOC) is calculated on the basis of the actual LOC shape which, however, is hidden to the planner. Modeling only the relevant points for the planning allows us to abstract from the effective representation of points in space (through coordinates) and to simplify the entire planning process reducing the problems' complexity.

It is worth noting that this kind of abstraction is mainly adopted for reducing the problems' complexity but the information hidden to the planner are not lost w.r.t. both what is considered during planning (targets' shapes information are all implicitly expressed through the initialization of some numerical fluents, such as `time-required` which expresses the time required to observe a target and, in case of LOC targets, is initialized by taking into account of the actual LOC shape) and also what can be expressed by the final plan (during decoding phase it is possible to retrieve the information stored into the database in order to consistently enrich the plans produced by planners).

The work of knowledge engineering involving the reported abstractions and simplifications implies a significant effort in encoding and decoding phases. It sometimes implies a modeling that is not immediate to be read in all of its aspects because of the pre-calculated information. However a representation of this type is necessary in order to be able to successfully employ general purpose state-of-art planners. In fact complexity of the problems grows proportionally to the number of features we represent and the majority of state-of-art planners is not able to handle a quantity of elements that also includes a detailed space representation. In 5.3 we'll show that a multitude of fluents (and in particular numerical fluents) are required even in the simple running example and with the described abstractions.

In describing the modeling, as well as we have done in the definition of the class of problems, we initially specify how to model the baseline features and then we introduce the additional constraints.

5.1 Example of problem and requirements

In order to more easily describe the modeling, in the rest of the chapter we'll take advantage of a simple example of multi-agent problem which requires all the features of the class of problems. We'll also use the same example in chapter 6 to describe how to encode it with a numerical planning model.

Let us suppose we have a set of two robotic agents $R = (R_1, R_2)$ and a set of two devices $D = (D_1, D_2)$. The agent R_1 is equipped with device D_1 and agent R_2 with device D_2. The problem we consider concerns four observation requests about a set of three point targets $T = (T_1, T_2, T_3)$. The observation requests are the following:

- $ObsReq_1$ requires that target T_1 must be observed with device D_1 for a minimum duration of 30 seconds.
- $ObsReq_2$ requires that target T_2 must be observed with device D_1 for a minimum duration of 60 seconds.

- *ObsReq*$_3$ requires that target T_1 must be observed with device D_2 for a minimum duration of 15 seconds.
- *ObsReq*$_4$ requires that target T_3 must be observed with device D_2 for a minimum duration of 180 seconds.

Notice that *ObsReq*$_1$ and *ObsReq*$_3$ concern the same target T_1 but they require to observe it with different sensors and for a different duration. The observations have to be performed in a wide temporal window of three hours (10800 seconds).

The problem also specifies a temporal constraint $Before(ObsReq_1, ObsReq_3, 1, +\infty)$ which specifies that the *GetInfo* action covering the observation request *ObsReq*$_3$ must start at least one second after the end of the *GetInfo* action covering the observation request *ObsReq*$_1$.

Finally there is also a set of temporal constraints on target observations. In particular:

- T_1 is observable only in the interval $[0, 3000]$ (values are expressed in seconds).
- T_2 is always observable during the mission temporal window (interval $[0, +\infty]$).
- T_3 is observable both in the interval $[0, 1500]$ and in the interval $[4500, 8000]$.

5.2 Encoding the baseline problem features

Listing 5.1 defines the four types of objects describing a taxonomy of the elements involved in the class of problems we aim to encode.

```
;;domain file
(:types
        Agent Device Target Site2D  - object
        Point - Target)
```

Listing 5.1: The definition of PDDL types for the baseline class of problems in temporal planning.

Notice that the specific objects of a problem are instead defined in the problem file (see Listing5.2).

```
;;problem file
(:objects
      R_1 R_2 - Agent
      D_1 D_2 - Device
      T_1_1 T_2_2 T_3_1 T_4_3 - Point
      L1 L2 L3 L4 - Site2D)
```

LISTING 5.2: The definition of PDDL objects for the instance of problem described in section 5.1.

Site2D type represents the locations involved in the problem.

Before continuing the description of the types of objects involved in the domain, it is worth noting that, as we mentioned in this chapter's introduction, in modeling the class of problems we need a series of assumptions on the domain. First of all we are assuming agents only move by using straight line path. This is an approximation which is operable and adopted especially in UAV domain ([31]), however for planning purpose it can be applied also to generic robotic agents. In fact for each couple of points (Site2D) we calculate in the current encoding phase their distance and we give this information to the planner in the initial state through the initialization of a constant numeric fluent

```
(= (Distance2D ?l1 ?l2) ?m)
```

where ?l1 and ?l2 are two Site2D and ?m $\in \mathbb{R}^+$ is the distance between the two points. Pre-calculating the distances allows to hide to the planner the details of the path between the points and it can be simply considered as a straight line during the planning.

Notice that this also allows to simplify the space representation only to the points which represent target vertices or points related to other objects in domain (such as the starting position of the robotic agents), ignoring the rest of points in space. This representation is also supported by the fact that in order to observe a target we consider sufficient to reach its location and "stay on it" for a certain amount of time. We are in fact supposing that once a robotic agents arrives within a certain radius from the target (radius depending both on robotic agent and on the target), the agent is able to observe it. This assumption is also appropriate in UAV domain assuming the UAV is able to perform a lower level loiter action which maintains it over the target. Modeling only the relevant points for the planning allows us to abstract from the effective representation of points in space (through coordinates) and to simply mapping points to PDDL objects (see row 6 in Listing 5.2). We can apply this abstraction also to the representation of LOC targets: as reported in the introduction of this chapter, we can describe them in PDDL with only two points representing the initial and final points of the polygonal chain which describes the LOC.

Despite a modeling without abstractions and works of knowledge engineering such as those reported would be clearer and it would require less effort in encoding and decoding phases, as we already mentioned they are necessary in order to be able to successfully employ the general purpose state-of-art planners otherwise too less efficient.

`Agent` and `Device` types allow to represent respectively robotic agents and devices. The main qualitative aspects of a robotic agent concern its initial position and the set of devices loaded on board. These features can be modeled via two PDDL predicates defined in the initial state of the problem:

```
(current_site ?r ?l)

(loaded_on_board ?r ?d)
```

where `?r` is a variable representing a robotic agent (type `Agent`), `?l` a variable of type `Site2D` representing the initial location of `?r`, and `?d` a device (type `Device`). The `current_site` predicate allows to define the initial position of a robotic agent in the space (and it is updated during planning in order to keep the current position), while the `loaded_on_board` predicate is a constant fluent defining the association between a robotic agent and a device loaded on board.

For each numerical resources Res_{R_i} of the agent R_i we can then define a numerical fluent and initialize it in initial state of the problem. For instance we can model the fuel level of a robotic agent or the time it requires to take a unit of space, with the functions

```
(= (Fuel-Level ?r) ?n)

(= (Time-unit-distance ?r) ?m)
```

where `?r` is a variable representing a robotic agent (type `Agent`), `?n` $\in \mathbb{R}^+$ is the initial fuel level and `?m` $\in \mathbb{R}^+$ is the time required by `?r` to take a unit of space (it is calculated as the inverse of the cruise speed of the agent: $\frac{1}{cruise\_speed(r)}$).

`Target` is a type that represents a combination of a *ObsReq* and the related *Target*. Because of the close connection between the two concepts and in order to simplify the definition of actions schema we decided to merge them. Specifically we have as many objects of this type as observation requests in the problem. In fact for each target T_i in the problem there must be at least one *ObsReq* and all observation requests concern exactly one *Target*. For instance notice that in our example both $ObsReq_1$ and $ObsReq_3$ concern the same target T_1. Merging observation requests and targets in a unique object we obtain two `Target` objects corresponding to the pairs $\langle ObsReq_1, T_1\rangle$ and $\langle ObsReq_3, T_1\rangle$ and this representation simplify the modeling of both actions schema and goals. In fact we can easily represent both targets and observation requests properties exploiting the same object.

Notice also that the above described representation of `Target` objects as pairs $\langle ObsReq, Target\rangle$ allows an immediate definition of new temporal constraints. In particular the concept of temporal windows of observation of a target (described in 3.4) can be directly extended to express also temporal windows in which the observation is required. For instance it could be required in a mission an observation of a target T_i in a specific temporal window (e.g. between the 8 am and 8.30 am) and another observation of T_i in a different temporal window (e.g. between 9.30 am and 10 am) while T_i is observable within a third set of time windows. The observability windows defined for a `Target` can then be easily used to express the intersection between the time windows in which the observation task is required and the time windows in which the related target is observable.

Some `Target` properties can be modeled as follow:

```
;;target properties
(site_target ?l ?t)

;;observation request properties
(= (Time-min-obs ?t) ?m)
(to_observe ?t ?d)
```

where `?l` is a location (object of type `Site2D`), `t` is a pair $\langle ObsReq, Target\rangle$ of type `Target`, `?m` $\in \mathbb{R}^+$ is the requested minimum duration of observation and

?d is a Device. The predicate site_target allows to define the point in space at which a target is located. Time-min-obs is a function which allows to define the requirement on the minimum duration of observation of a target. Finally the predicate to_observe specifies the device to use in target observation.

Listing 5.1 also reports another type Point that is a subtype of the type Target. As already mentioned, while Point targets are described only by one Site2D, LOC targets are described by two points which are the initial and final points of the polygonal chain.

In this section we described the elements that allow us to encode the information concerning all the objects in our problems. More predicates and functions of support for the planning and for the modeling of the problems features are defined below.

5.2.1 Encoding the action model

As already mentioned, temporal planning exploits the durative actions formalism of PDDL 2.1 to devise a particular schema of action that can be temporally constrained both in its scheduling times and duration. In this section we describe how to model the actions schema of the two main concurrent agent types involved in our class of problems: robotic agents and devices. In particular robotic agents need a transfer action which allows them to move in the environment to reach a target and a stay_on_trg action to stay along a LOC target during the observation and get to its ending point. Devices need a monitor action which allows them to observe a target.

5.2.1.1 Transfer action

```
(:durative-action transfer
:parameters (?r - Agent ?l_start ?l_end - Site2D ?t - Target)
:duration (= ?duration
            (* (Distance2D ?l_start ?l_end) (Time-unit-distance ?r)))
:condition (and
                (at start (current_site ?r ?l_start))
                (at start (>= (Fuel-Level ?r)
                              (* (Distance2D ?l_start ?l_end)
                              (Rate-Consumption ?r))))
                (at start (> (Distance2D ?l_start ?l_end) 0))

12                  (at start (site_target ?l_end ?t))
13              )
14 :effect (and
15                  (at start (not (current_site ?r ?l_start)))
16                  (at end (current_site ?r ?l_end))
17                  (at end (decrease (Fuel-Level ?r)
18                              (* (Distance2D ?l_start ?l_end)
19                              (Rate-Consumption ?r))))
20
21                  (at start (chosen_start_site_target ?r ?l_end ?t))
22          )
23 )
```

Listing 5.3: The definition of the PDDL durative action `transfer` for the baseline class of problems.

Listing 5.3 reports the `transfer` durative action schema. The parameters of the action schema (row 2) are the agent `?r` that must execute the action, two locations `?l_start` and `?l_end` respectively representing the current location of the agent and the final point after the transfer, and a target `?t`. The duration of this kind of action (see rows 3-4) is expressed as the time `?r` needs to get in `?l_end` from `?l_start`. Notice that in order to speed up the planner execution by simplifying its operations it is possible to replace the numerical expression of the duration of the action with a pre-calculated numerical fluent

```
(= (time_required ?a ?b ?r) ?n)
```

where `?a` and `?b` are two `Site2D`, `?r` is the `Agent` and `?n` $\in \mathbb{R}^+$ is the time required by `?r` to go from `?a` to `?b` and it can be calculated as $\frac{dist(a,b)}{cruise\_speed(r)}$. In the following we will use this numeric fluent instead of the numerical expression. We will also take advantage of other pre-calculated numerical fluents such as

```
(= (fuel_required_t ?a ?b ?r) ?ft)
(= (fuel_required_m ?t ?r) ?fm)
```

where `?a`, `?b` and `?r` are defined as above, `?t` is a `Target`, `?ft` is the quantity of fuel required by the agent `?r` to go from `?a` to `?b` (calculated as $dist(a, b) * rate\_consumption(r)$) and `?fm` is the quantity of fuel required by the agent `?r` to stay on a target for the time required by the observation request. As well as a speed up of the planning, the usage of these numerical fluents instead of the numerical expressions implies also a longer (but not significantly) pre-processing phase.

Rows `6-12` report the preconditions of the `transfer` action. They require that the robotic agent is located at the beginning of the action in `?l_start` and that the agent has enough fuel to go from `?l_start` to `?l_end` (notice that this numerical expression can be replaced by the pre-calculated numerical fluent `fuel_required_t` as reported above). Precondition at row `10` requires that the distance between the two points is greater than zero, otherwise it is not necessary a transfer action (the agent is already in the desired position). Finally the last precondition requires that in the final point `?l_end` is a target location. This precondition is consistent because of the assumptions we have done on spatial modeling. In fact because we don't model all the points in space but only the significant ones (and in particular only points characterizing targets or the initial location of the agents) and because we assume that agents only move following straight paths, the shortest path between two different points cannot involve other locations but the two points. Therefore there is no need in the baseline class of problem that a robotic agent moves from a point to another that is not a target location. This precondition has no impact on action consistency but it facilities the planning by reducing the number of applicable actions.

Rows `15-21` report the effects of the action. They negate at the beginning the current location of the agent and states at the end of the action the new position. They also decrease the level of fuel of the agent. The last effect concerns the choice of the starting point for the observation of a target for a specific observation request. It exploit a propositional fluent `chosen_start_site_target` which asserts that for `?t` has been chosen the location `?l_start` as starting point for the observation. If the related target is a point target the location is the unique position where it is located and the information is not particularly useful, but if it is a LOC target then the predicate allows to correctly update the position of `?r` and to decide at what end to start the observation of the target (this may have a significant impact of the quality of the resulting plan).

Robotic agents also require a `stay_on_target` action which allows them to stay on (or near) a target enough to perform an observation task. The action schema is similar to the `transfer` action (we omit the implementation details to avoid verbosity). The duration of this action must be greater than the minimum time of observation required by the related *ObsReq*. In case of LOC target the action updates the current position of the agent at the end of the execution exploiting the `chosen_start_site_target` fluent. It also asserts at the beginning of its execution a propositional fluent `(on_trg ?r ?t)` (with `?r` the `Agent` and `?t` the `Target`) and denies it at the end. This fluent is required as precondition by the monitoring action and expresses the fact that the robotic agent is located in a position from which the target is observable.

5.2.1.2 monitor action

```
1  (:durative-action monitor
2  :parameters (?d - Device ?r - Agent ?t - Target)
3  :duration (>= ?duration (Time-min-obs ?t))
4  :condition (and
5                   (at start (to_observe ?t ?d))
6                   (at start (loaded_on_board ?r ?d))
7                   (at start (not (covered ?t)))
8
9                   (at start (on_trg ?r ?t))
10                  (over all (on_trg ?r ?t))
11                  (at end (on_trg ?r ?t))
12              )
13 :effect (and
14                  (at start (covered ?t))
15         )
16 )
```

Listing 5.4: The definition of the PDDL durative action `monitor` for the baseline class of problems.

Listing 5.4 reports the `monitor` durative action schema. The parameters of the action schema (row 2) are the device `?d` that must execute the action, the robotic agent `?r` on which must be loaded `?d` and a target `?t`. The duration of this action (row 3) must be greater or equal to the minimum duration of observation required by `?t` (specified by the numerical fluent `Time-min-obs`).

Rows `4-16` report the preconditions of the action. They require that the target related to `?t` must be observed with the device `?d` which must be loaded on board the robotic agent `?r`. They also require that the predicate `(on_trg ?r ?t)` (which, as reported above, is stated at the beginning of the `stay_on_trg` action and denied at the end) holds for the entire duration of the observation (see rows `9-11`). These conditions guarantee the correct concurrency and synchronization between the robotic agent and the device during the observation task.

Precondition at row 7 requires that the observation request `?t` has not been already covered at the beginning of the action scheduling. On one hand it prevents that the same observation task is performed more than once, and on the other hand it prevents that many devices start performing the same observation task concurrently.

For simplicity we don't model numerical resource consumption in the `monitor` action performed by the device. However their modeling would be equivalent to the reported modeling of fuel level consumption in `transfer` action.

The effects (rows `13-15`) mainly concern the completion of the observation task. At the beginning of the action is asserted a propositional fluent `(covered ?t)` which is one of the predicates that must hold in the goal state. As above reported, asserting this predicate at the beginning of the action, together with the precondition at row 7, allows to avoid that many agents concurrently perform the same observation task.

5.2.2 Encoding the goal state

As the previous sections demonstrate, the majority of the reasoning over the constraints and requirements of the problems is performed by declaring a series of predicates and functions while checking their consistency through the preconditions of the actions.

Given these premises, the goal state can concern just the satisfaction of a `covered` predicates, one for every *ObsReq* in problem (see Listing 5.5 for the goal definition for the running example).

```
(:goal
    (and
        (covered T_1_1)
        (covered T_2_2)
        (covered T_3_1)
        (covered T_4_3)
    )
)
```

Listing 5.5: The definition of PDDL goals for the instance of problem described in section 5.1.

5.3 An example of encoding in UAV domain

In the previous section we saw how to encode the main features of the baseline class of problems. In the following of this chapter we describe how to adapt and extend the encoding to introduce the extensions of the class of problems. In order to avoid boring the reader but still give her a clear idea of what must be done we describe the main elements of the two PDDL files (problem and domain files) generated by the encoding for the running example and we comment them.

Listing 5.6 reports the definition of the main elements of the PDDL problem file for the running example.

```
(define (problem Problem_2airp_2uav_4trgob_1mobs)
    (:domain SmatF2)
    (:objects
        UAV_0 UAV_1 - Uav
        S_0 S_1 - Sensor
        AIRPORT_0 AIRPORT_1 - Airport
        T_1_1 T_2_2 T_3_1 T_4_3 - Point
        L0 L1 L2 L3 L4 L5 L6 L7 L8 L9 L10 L11 - Site2D)

    ;;initial state definition
    (:init
        ;;Airports information
            (site_for_begin_takeOff AIRPORT_0 L0)
            (site_for_end_takeOff AIRPORT_0 L1)
            (site_for_landing AIRPORT_0 L2)
            (site_after_landing AIRPORT_0 L3)

            (site_for_begin_takeOff AIRPORT_1 L4)
            (site_for_end_takeOff AIRPORT_1 L5)
            (site_for_landing AIRPORT_1 L6)
            (site_after_landing AIRPORT_1 L7)
        ;;UAVs information
            (current_site UAV_0 L0)
            (= (Time-take-off UAV_0 AIRPORT_0) 180)
            (= (Time-landing UAV_0 AIRPORT_0) 360)
            (loaded_on_board UAV_0 S_0)

            (current_site UAV_1 L4)
            (= (Time-take-off UAV_1 AIRPORT_1) 120)
            (= (Time-landing UAV_1 AIRPORT_1) 240)
            (loaded_on_board UAV_1 S_1)

        ;;targets and observation requests information
            (= (Time-min-obs T_1_1) 30)
            (site_target L8 T_1_1)
            (to_observe T_1_1 S_0)
            (at 0 (observable T_1_1))
            (at 3000 (not (observable T_1_1)))
            (before T_1_1 T_3_1)
            (observable_p T_1_1)

            (= (Time-min-obs T_2_2) 60)
            (site_target L9 T_2_2)
            (to_observe T_2_2 S_0)
            (at 0 (observable T_2_2))
            (observable_p T_2_2)

            (= (Time-min-obs T_3_1) 15)
            (site_target L10 T_3_1)
            (to_observe T_3_1 S_1)
            (at 0 (observable T_3_1))
            (at 3000 (not (observable T_3_1)))

            (= (Time-min-obs T_4_3) 180)
            (site_target L11 T_4_3)
            (to_observe T_4_3 S_1)
            (at 0 (observable T_4_3))
            (at 1500 (not (observable T_4_3)))
            (at 4500 (observable T_4_3))
            (at 8000 (not (observable T_4_3)))
            (observable_p T_4_3)

        ;;temporal and spatial information
            (= (time_required L0 L0 UAV_0) 0)
            (= (time_required L0 L0 UAV_1) 0)
            (= (time_required L0 L1 UAV_0) 281.8)
            (= (time_required L0 L1 UAV_1) 281.8)
            ...
            (= (time_required L11 L11 UAV_0) 0)
            (= (time_required L11 L11 UAV_1) 0)
            ...

        ;;action contiguity
            (at 0 (= (start_next UAV_0) 0.01))
            (at 0 (= (start_next UAV_1) 0.01))

    )
    ;;goal and metric definition
    (:goal
        (and
            (covered T_1_1)
            (covered T_2_2)
            (covered T_3_1)
            (covered T_4_3)
```

```
85                (landed UAV_0 AIRPORT_0)
86                (landed UAV_1 AIRPORT_1)
87            )
88        )
89        (:metric minimize (total-time))
90    )
```

LISTING 5.6: The definition of the main elements of the PDDL problem for the instance of problem described in section 5.1.

Rows `3-8` defines the objects involved in the problem. Notice that we replaced the types `Agent` and `Device` respectively with the (equivalent) types `Uav` and `Sensor`. It is a simple renaming to be more coherent to the domain but the new types have the same features of the previous ones. Notice also that we introduced a new type of objects `Airport` which represents the airports where the UAVs are initially located and where they have to land.

Rows `10-77` contain the definition of the initial state of the problem. The first information we need concern the airports. Rows `13-21` report, for each airport, information on start and end positions of take off and landing. For simplicity we assumed that all airports have only one runway dedicated to UAVs and that all aircrafts respect the defined locations during their operation. In the running example there are two UAVs (`UAV_0` and `UAV_1`) initially located respectively in `AIRPORT_0` and `AIRPORT_1`. Rows `23-31` report these information as well as information on the sensors loaded on board the UAVs (`loaded_on_board` fluents) and information on time required by the UAVs to take-off (`Time-take-off` fluents) and land (`Time-landing` fluents).

The information above reported are required by the `take_off` and `landing` durative actions expressed in the domain file. Their schema is similar to the `transfer` action schema (see Listing 5.3). The `take_off` action has the additional task to assert a propositional fluent `(in_flight ?u)` (with `?u` a `Uav`) which is a precondition of all the UAVs operation in flight. This precondition constraints the planner to schedule a `take_off` action before all the others. The `landing` action instead, has the additional task to assert one of the predicates that are required to hold in the goal state (see rows `85-86`). In particular at the end of its execution is asserted a predicate `(landed ?u ?a)` (with `?a` the `Airport` at which the UAV `?u` landed).

Rows `33-61` report information on targets and observation requests. They express the minimum duration required by the observations (via the numerical fluent `Time-min-obs`, e.g. rows `34` and `42`), the points where the targets are located (predicate `site_target`, e.g. rows `35` and `43`) and the sensors that must be used to satisfy the observation requests (predicate `to_observe`, e.g. rows `36` and `44`). These information are used by the `monitor` and `transfer` actions as described in the previous section. Rows `33-61` also report the encoding, for the running example, of the features which characterize the extensions of the class of problems: temporal constraints between targets observation and time windows on targets observation. See the following 5.3.1 and 5.3.2 sections for a detailed description of their encoding.

Rows `64-71` report temporal and spatial information. In particular for each pair of location involved in the problem it is reported the time required by the UAVs to get from one location to the other. For simplicity Listing 5.6 reports the initialization of only few of the `time_required` numerical fluents. Similarly it is necessary to define, as already mentioned, all the numerical fluents expressing information on the fuel quantity required by each UAV to get from one location to the other. The fluents that provide temporal information are then used in actions schema to define the durations of the actions. Fluents that provide information on fuel level (or on any other agents resource) are used in actions preconditions and effects.

It is worth noting that despite the running example is very simple and involves only two UAVs and four target observations, it is necessary to define in the initial state 288[1] numerical fluents only for the `time_required` functions. Therefore increasing the number of targets (that implies increasing the number of locations) as well as the number of UAVs leads to a significant increase in problem complexity.

Another important aspect to encode in a PDDL problem in UAV domain in temporal planning concerns the contiguity of the actions. In a generic robotic agents domain the contiguity of actions is not a critical aspect. Let us suppose robotic agents that move on ground: it can be a good (or at least acceptable) approximation to assume that the internal status of the agents only change as effect of the execution of an action. Therefore holes between actions in a plan in a generic robotic agents domain can be considered acceptable. Conversely in UAV domain

[1]For each type of temporal and spatial information of such type it is necessary to initialize a numerical fluent for each pair of locations and for each UAV involved in the problem ($12 * 12 * 2$ fluents in the running example involving twelve locations)

they are not. As already explained, in every time instant UAVs have to perform a flight action which implies both effects in terms of aircraft position up in the air and numerical effects due to resources consumption. Therefore it is not a reasonable choice to ignore holes between actions. PDDL 2.1 and 2.2 extensions don't provide any internal features to express this kind of constraint. However with the help of the planners in terms of supporting continuous effects it is possible to encode a set of preconditions and effects which allow the actions to be almost contiguous.

For each UAV it is necessary to initialize a numerical fluent `start_next` to `0.01` (e.g. see rows `74-75`). Furthermore for each action that can be performed by a UAV agent it is necessary to introduce a new set of preconditions

```
(at start (> (start_next ?uav) 0))
(at start (< (start_next ?uav) 0.1))
```

and a new set of effects

```
(at start (assign (start_next ?uav) <action_duration>))
(decrease (start_next ?uav) #t)
```

where `?uav` is the `Uav` that has to perform the action, `<action_duration>` is the duration of the action which varies according to the action, and `#t` is the PDDL keyword that represents a local clock of the action and refers to the continuous changing time from the start of the action itself.

These elements allows to implement a mechanism in which at the beginning of the actions the numerical fluent `start_next` is initialized to the duration of the action and for the entire duration of the action the value of the fluent is constantly decreased until it reaches the `0` value. All the actions can only be scheduled by the planner when the value of `start_next` is between `0` and `0.1`, that is when the previous scheduled action is terminating.

In other words we force all actions to overlap for a defined small time interval which we consider negligible at our level of abstraction and we obtain for each UAV a sequence of almost contiguous actions, solving the problem of contiguity.

It is worth noting that this mechanism has a significant impact on definition of actions schema. In fact because of the small overlap between actions it is necessary to revise time events at which some preconditions must be satisfied and some effects must be applied. For instance it is necessary to anticipate at the beginning of the actions the effects (before applied at the end of the action) that are preconditions for the following actions. With the introduction of these elements, in fact, actions have to start before the actual ending of the previous ones.

5.3.1 Temporal constraints between targets observation

In the running example (see 5.1) there is only one temporal constraint between the targets observation: the problem requires that the $ObsReq_3$ is satisfied at least 1 second after the end of the monitor action which satisfies the $ObsReq_1$. This constraint is encoded in the PDDL problem file via a propositional fluent `(before T_1_1 T_3_1)` and with the introduction of a propositional fluent `(observable_p ?t)` (with `?t` a `Target`) for each `Target` which is observable from the beginning of the mission (see rows `40`, `46` and `61`).

The introduction of such type of constraints also requires a series of changes in actions schema definition in the domain. It requires the introduction of a new action `monitor_before` with an action schema similar to the `monitor` schema defined in Listing 5.4. It takes two `Target` parameters `?t1` and `?t2` and it is only applicable if exist a predicate `(before ?t1 ?t2)` and a predicate `(observable_p ?t1)` holds. If applied, this action asserts at the end of its execution a predicate `(observable_p ?t2)` which enables the observation of `?t2`. It is also necessary to add as a precondition of the original `monitor` action the constraint that the predicate `(observable_p ?t)` (with `?t` the `Target` object taken as parameter) holds at the beginning of the action.

It is worth noting that in order to encode the temporal gap between the constrained observations it is necessary to introduce a set of numerical fluents which keep track of the passage of time during the planning (as well as we have to do in numerical planning, see chapter 6). Exploiting such type of variables it is possible to define new temporal preconditions on monitoring actions and encode the temporal gap between constrained monitoring actions.

The running example requires only one *Before* constraint. The encoding of the other types of constraints is analogous.

5.3.2 Time windows on targets observation

The time windows within which the targets are observable in the running example are modeled in PDDL problem using timed initial literals formalism (e.g. see rows 57-60). Notice that because in problem description target T_2 is defined as always observable during the mission temporal window and because there are no further temporal constraints on the observation $ObsReq_2$ it is sufficient to specify in the PDDL problem that T_2_2 is observable from the beginning of the mission (time 0) without constraints on latest time of observation (see row 45 in Listing 5.6).

It is also necessary to introduce a new set of preconditions (reported below) in monitoring actions which require that the target is observable for the entire duration of the action.

```
(at start (observable ?t))
(over all (observable ?t))
(at end (observable ?t))
```

?t is the Target object taken as parameter by the monitoring action.

5.3.3 Target observations assignment

Listing 5.4 reports in the goal definition the four predicates covered which require that in the goal state the Target T_1_1, T_2_2, T_3_1, T_4_3 have been covered. This representation implies that there is no assignment of the target observations to the UAVs and the planner has to automatically choice which UAV to use to satisfy an observation request.

In order to specify the assignments of the observation requests to the UAVs it is sufficient to add a parameter to the covered propositional fluent, specifying the UAV ?uav that must satisfy the request: (covered ?t ?uav).

Notice that in case the problem specifies the assignments it is also possible to modify some of the actions and define a set of new fluents in order to both help the planner by reducing the set of actions applicable in each state and allowing it to find better solutions. For instance Listing 5.7 reports the action schema of a new type of transfer action transfer_to_trg which allows a Uav to transfer to a location at which is located a target only if the UAV must observe it. This action require a new propositional fluent as precondition: `(to_cover ?t ?uav)` where `?t` is a `Target` and `?uav` is the `Uav` that must perform the action (see row `13`).

```
1  (:durative-action transfer_to_trg
2  :parameters (?r - Agent ?l_start ?l_end - Site2D ?t - Target)
3  :duration (= ?duration
4             (* (Distance2D ?l_start ?l_end) (Time-unit-distance ?r)))
5  :condition (and
6                  (at start (current_site ?r ?l_start))
7                  (at start (>= (Fuel-Level ?r)
8                                (* (Distance2D ?l_start ?l_end)
9                                (Rate-Consumption ?r))))
10                 (at start (> (Distance2D ?l_start ?l_end) 0))
11
12                 (at start (site_target ?l_end ?t))
13                 (at start (to_cover ?t ?uav))
14             )
15 :effect (and
16                 (at start (not (current_site ?r ?l_start)))
17                 (at end (current_site ?r ?l_end))
18                 (at end (decrease (Fuel-Level ?r)
19                                   (* (Distance2D ?l_start ?l_end)
20                                   (Rate-Consumption ?r))))
21
22                 (at start (chosen_start_site_target ?r ?l_end ?t))
23         )
24 )
```

Listing 5.7: The definition of the PDDL durative action `transfer_to_trg` for the baseline class of problems.

It is also possible to define a `transfer_to_land` action which allows the `Uav` to perform a transfer to the `site_for_landing` only if it has covered all its target observations. In order to define this type of action it is necessary to initialize for each `Uav` two new numerical fluents in the initial state

```
(= (NumberTrgCovered ?uav) 0)
(= (NumberTrgAssigned ?uav) ?n)
```

where `?uav` is a `Uav` and `?n` is the number of target observations assigned to `?uav`. It is also necessary to increase the value of the `NumberTrgCovered` after every `monitor` action.

5.4 Solution decoding

Listing 5.8 reports a valid plan[2] solving the problem modeled in section 5.3.

```
1  0.001: (take_off uav_1 l4 l5 airport_1)  [120.000]
2  0.002: (take_off uav_0 l0 l1 airport_0)  [180.000]
3  119.901: (transfer_to_trg uav_1 l4 l11 t_4_3)  [435.200]
4  179.902: (transfer_to_trg uav_0 l0 l8 t_1_1)  [1456.000]
5  554.901: (stay_on_trg uav_1 l11 t_4_3)  [190.000]
6  564.603: (monitor s_1 uav_1 t_4_3)  [180.000]
7  744.602: (transfer_to_trg uav_1 l11 l10 t_3_1)  [1609.100]
8  1635.702: (stay_on_trg uav_0 l8 t_1_1)  [40.000]
9  1635.903: (monitor_before s_0 uav_0 uav_1 t_1_1 t_3_1)  [30.000]
10 1675.403: (transfer_to_trg uav_0 l8 l9 t_2_2)  [1181.100]
11 2353.302: (stay_on_trg uav_1 l10 t_3_1)  [25.000]
12 2353.703: (monitor s_1 uav_1 t_3_1)  [15.000]
13 2377.802: (transfer_to_land uav_1 l10 l6 airport_1)  [1899.400]
14 2856.103: (stay_on_trg uav_0 l9 t_2_2)  [70.000]
15 2856.504: (monitor s_0 uav_0 t_2_2)  [60.000]
16 2925.604: (transfer_to_land uav_0 l9 l2 airport_0)  [498.300]
17 3423.304: (landing uav_0 l2 l3 airport_0)  [360.000]
18 4276.603: (landing uav_1 l6 l7 airport_1)  [240.000]
```

LISTING 5.8: A valid temporal plan solving the problem described in section 5.1.

The plan contains 18 actions: 14 concerns the UAVs and are equally divided between `UAV_0` and `UAV_1`, the remaining 4 are the monitoring actions (see rows `6, 9, 12` and `15`) scheduled for the devices mounted on board the UAVs.

The first aspect it is worth noting is that the actions of every UAV are nearly contiguous. For instance the `take_off` of `UAV_1` (see row `1`) is scheduled at time point `0` (we are ignoring the last significant decimal digit of the starting time of the actions) and it lasts `120` seconds. The following action of `UAV_1` is a `transfer_to_trg` (see row `3`) and it is scheduled at `119.9`, that is 0.1 seconds before the ending of the previous action. By considering not significant this small time interval[3] we obtain a sequential plan for each UAV without temporal holes between actions.

Another important aspect to notice concerns concurrency between UAVs and devices. Every time a device has to perform a monitoring action it is required that the UAVs on which it is loaded is near enough to the target to observe. In order to obtain it the planner always schedules a device's monitoring action together

[2]The plan was obtained by running the planner COLIN.

[3]Notice that if 0.1 seconds are significant in a certain domain, it is sufficient to initialize the duration of the actions to a value which takes into account of this time interval.

with a UAV's `stay_on_trg` action. An observation task is therefore always represented by two concurrent actions `stay_on_trg` and `monitor` (or `monitor_before`) respectively performed by a UAV and a device (see for instance rows `5` and `6`: the monitoring action is scheduled 10 seconds after the start of the execution of a `stay_on_trg` action by the UAV and they end together).

In order to decode and to make easier to read the solution returned by the planner it is necessary to perform a series of tasks. First of all actions must be divided according to the UAV or the device that must execute them.

Two different kinds of plan representations may be submitted to an end user. A temporal representation can be obtained by a translation of the plan into Gantt diagrams which allow the end user to easily understand where actions are temporally located and to verify if the plan is satisfying in terms of temporal displacement of the observation tasks. A spatial representation instead can be obtained by a translation of the plan into a set of sequences (one for each UAV involved in the problem) of waypoints in space reported on a map. Notice that while the temporal translation can be directly done with pieces of information included in the plan, the spatial translation requires instead a set of information that have to be stored in an internal database during the encoding phase. In particular the locations submitted to the planner (and then reported in the final plan, e.g. `l4` or `l5` at row `1`) are only symbolic representations of the actual geographical points. Therefore it is necessary to retrieve the actual spatial information from the internal database by matching them with the symbolic locations in the plan. The translation into waypoints takes into account of the type of action. If the action is a point monitoring action, it involves only one location that is translated into a unique waypoint. Otherwise if the action is a LOC monitoring action, it involves two locations in PDDL problem and it is translated into a sequence of two or more waypoints according to the description of the related target. Therefore, at the end of the translation, all the waypoints entirely describe the route of the UAV.

Both temporal and spatial translations require also to report the names of actions and entities involved in the problem in a format easy to read by the end user (e.g. "Transfer to target" instead of `transfer_to_trg`, "Radar Sensor" instead of `S_0`, but more complex translations are also required). This can be done by storing, at the same way of the locations, the matching between objects and symbols used by the planner.

Chapter 6

Numerical planning

In this chapter we present how to encode the class of problems (and its extensions) described in chapter 3 using a numerical planning model. This type of modeling doesn't use neither durative actions or timed initial literals. However in order to exploit numerical expressions it is necessary to adopt the PDDL 2.1 formalism.

The main difference with the temporal model concerns, as already mentioned, the time representation. Numerical planners doesn't provide any internal mechanism which explicitly (in a way visible by a user) represents the time passage, the actions' duration or their temporal location. The numerical model requires to explicitly manually model the time passage by using PDDL objects. For this reason it is necessary to define some functions (see section 6.1) to represent the current time of each agents after every action. It is also necessary to keep track, in order to satisfy the temporal constraints of the problems, of the significant events of the scheduled actions (e.g. the time point at which a UAV performs a take off, the starting time of monitoring a target, etc.) by using numerical fluents that are updated by the effects of the actions and represent the delta from an origin (set to 0) that corresponds to the earliest time of the mission.

The use of non-durative actions also requires to find mechanisms to be able to obtain enough information from the planner solution, that, in this type of modeling, doesn't contain temporal information on the actions scheduling. In fact a numerical plan only shows the sequence of actions with the related parameters. While in case of single robotic agent mission this is quite enough to easily decode the solution (despite the concurrency between robotic agents and devices that, as we'll see in 6.1.2, can be simplified through the introduction of joint actions), the lack of information in planner's solution is a particularly relevant problem instead when the problems involve multiple robotic agents. The planner builds in fact a total-ordered-plan, treating actions as all sequential, while we desire a set of concurrent total ordered plans. As we'll see in 6.3 this problem can be faced by a complex mechanism of decoding which retrieves all the required information from an internal database in which they are stored.

It is worth noting that a numerical approach, despite a less clear modeling, also has some advantages. For instance the problem of contiguity between actions is implicitly solved by the planner, which builds sequential plans. Therefore it is not necessary the mechanism described at the end of 5.3 to force actions' contiguity. Numerical planning is also a less recent and more in depth studied approach in automated planning, therefore planners are typically more efficient in solving such type of problems.

6.1 Encoding the action model

In order to make the description of the actions schema lighter we briefly describe the two most significant types of actions of a numerical model. The rest of them is in fact quite similar.

6.1.1 Initial waiting action

Because of the lack of an explicit representation of time the first action we need to model concerns the passage of time. As mentioned, numerical planners build sequential plans without temporally locate the actions. This implies that in order to obtain plans which satisfy temporal constraints on target observations it is necessary to introduce actions that simulate the passage of time.

Listing 6.1 reports the action schema of a initial waiting action for a generic robotic agents domain in numerical planning. This action is applicable only when the robotic agent `?r` (taken as parameter) has not started yet its mission (see the precondition at row `3`). The effects of the actions impact on two different numerical fluents which have to be introduced in a numerical model. `(Delta-Time ?r)` (where `?r` denotes an `Agent`) is a function initialized in initial state to a value (typically 0[1]) that represents the first useful moment (referring to the earliest time of the global mission) when `?r` starts its actions. `(Time-mission ?r)` (with `?r` an `Agent`) is a function initialized in initial state to the same value of `(Delta-Time ?r)` and represents the time clock of `?r` during the mission. The `initial-waiting` action affects these fluents by increasing their values of a value `waiting-time` defined in the initial state of the problem. The use of a fixed value such as `waiting-time` is required by the lack of the possibility to specify a duration of the actions.

```
(:action initial-waiting
  :parameters (?r - Agent)
  :precondition (and  (ready_to_go ?r))
  :effect (and (increase (Delta-Time ?r) (waiting-time))
                (increase (Time-mission ?r) (waiting-time))
          )
)
```

LISTING 6.1: The definition of the PDDL action `initial_waiting` for a generic robotic agents domain in numerical planning.

The `initial-waiting` action, therefore, represents the time passing at the beginning of the mission of `?r`. It doesn't impact any agent's resource or environment variable. It is a type of action which only allows to model the time passage which is instead explicit in temporal planning.

Notice that in case of baseline class of problems we don't impose any constraint on contiguity of actions. Since numerical planners build sequential plans, if it can be useful in a specific domain to model time passing between subsequent actions of the same robotic agent, it is necessary to model a further waiting action. The `waiting` action only increases the `(Time-mission ?r)` fluent and can be scheduled by the planner after all other actions, except for the `initial-waiting` one. The action has no effects on fluent `(Delta-Time ?r)` (unlike the `initial-waiting` action) because it describes the time point at which the robotic agent `?r` start its mission. In UAV class of problems, because of the need to have contiguous actions it is sufficient to represent the time passing before the UAV takes off, therefore it is sufficient the `initial-waiting` action.

[1]The initial value of `Delta-Time` can be also initialized to a value greater than 0 and calculated by using some heuristics.

6.1.2 Monitor action

The second main type of action concerns monitoring actions. Because of the difficulty of modeling temporal aspects and the concurrency between actions, it is possible to simplify the modeling of monitoring tasks.

While in temporal planning it is possible to easily model concurrency between a robotic agent and its devices (monitoring action are automatically scheduled during the robotic agent's `stay_on_trg` action, see 5.2.1.2), in numerical planning it is more complex because of the need to simulate the time passing. In order to obtain concurrency between `stay_on_trg` actions and `monitor` actions the planner should schedule, for each device that must perform an observation, a set of waiting actions which move the monitoring actions at the right time (at the time point when the agent on which the device is loaded is performing the related `stay_on_trg` action). This would imply plans with many waiting actions for each device (as we mentioned in 6.1.1, duration of the actions is fixed in numerical planning, therefore it is not possible to schedule only one action of the required duration). It would also imply an increase of the problem complexity because of the higher number of actions.

A easier and cleaner solution consists in defining a joint action of monitoring which includes the `monitor` and the `stay_on_trg` actions previously performed by devices and robotic agents. It implies a simplification of the model by interpreting devices only as parameters of actions performed by robotic agents and it also allows to both reduce the problem complexity and to obtain more understandable final plans.

Listing 6.2 reports the action schema of the joint action of monitoring a point target for UAV class of problems with the extension of observability windows of targets.

```
(:action monitor_point_target
:parameters (?uav - Uav ?l_start - Site2D ?t - Point ?s - Sensor)
:precondition (and
    ;agent preconditions
    (on_fly ?uav)
    (current_site ?uav ?l_start)
    (loaded_on_board ?uav ?s)
    (>= (Fuel-Level ?uav)
              (* (* (Time-min-obs ?t) (Cruise-speed ?uav))
                    (Rate-Consumption ?uav)))

    ;target preconditions
    (to_observe ?t ?s)
    (not (covered ?t)))

    ;temporal preconditions
    (>= (Time-mission ?uav) (Earliest-time-start-obs-target ?t))
    (<= (+ (Time-mission ?uav) (Time-min-obs ?t))
                            (Latest-time-end-obs-target ?t))
)
:effect (and
    ;target effects
    (covered ?t)

    ;time effects
    (assign (Time-mission-start-obs-target ?t)
            (Time-mission ?uav))
    (assign (Time-mission-end-obs-target ?t)
            (+ (Time-mission ?uav) (Time-min-obs ?t)))
    (increase (Time-mission ?uav)  (Time-min-obs ?t))
    (increase (total-fly-time)  (Time-min-obs ?t))

    ;agent effects
    (increase (NumberTrgCovered ?uav) 1)
    (decrease (Fuel-Level ?uav)
                (* (* (Time-min-obs ?t) (Cruise-speed ?uav))
                (Rate-Consumption ?uav)))
))
```

LISTING 6.2: The definition of the PDDL joint action `monitor_point_target` for the UAV class of problems with observability windows extension in numerical planning.

This type of action takes as parameters a `Uav ?uav`, a `Point ?t`, a `Sensor ?s` and the initial location of the UAV `?l_start`.

The agent's and target's preconditions and effects are analogous to the ones described in chapter 5 for the `stay_on_trg` and `monitor` actions.

The main difference of this type of action concerns the modeling of the time and temporal windows on targets observation.

Temporal windows on targets observation are modeled by using two numerical fluents `(Earliest-time-start-obs-target ?t)` and `(Latest-time-end-obs-target ?t)` which represent the extremes of the temporal window in which the target related to `?t` is observable. Notice that this modeling is less flexible than timed initial literals (e.g. it is not possible to define more than one window of observability) but it doesn't require durative actions. The two preconditions at rows `17-18` constraint the scheduling of the monitoring action within the related temporal window. They check if the `Time-mission` of `?uav` (which represents the time point at which `?uav` is situated when the action has to be performed) is greater than the earliest time of observation of the target and if the time at which `?uav` would be situated at the end of the action's execution (`(+ (Time-mission ?uav) (Time-min-obs ?t))`) is less than the latest time of observation of the target.

Time is modeled via a series of numerical fluents. `Time-mission`, as we already mentioned, represents the current time clock of the UAV. As effect of the monitoring action, this fluent is updated by increasing it with the duration of the action itself. The same effect is applied to `total-fly-time` fluent which represents the total time the entire team of UAVs has been in flight during the mission. Rows `26-29` furthermore assign a value to the numerical fluents
`(Time-mission-start-obs-target ?t ?uav)` and
`(Time-mission-end-obs-target ?t ?uav)`
which respectively represent the time point at which the observation of `?t` started and the time point at which it ended. These fluents are relevant in order to model temporal constraints between targets observations because they maintain the temporal information (otherwise lost during the planning) on previous monitoring actions that must be used for the following ones.

We reported the schema of the two actions which enclose the main characteristics of a numerical model. The other actions are similarly defined. In the following of this chapter we first briefly illustrates how to encode in numerical planning the running example and then we analyze the difficulties of the decoding phase to be faced when the numerical approach is chosen.

6.2 Encoding the running example in UAV domain

Listing 6.3 reports the definition of the main elements of the PDDL problem file for the running example in numerical planning.

```
(define (problem Problem_2airp_2uav_4trgob_1mobs)
    (:domain SmatF2)
    (:objects
        UAV_0 UAV_1 - Uav
        S_0 S_1 - Sensor
        AIRPORT_0 AIRPORT_1 - Airport
        T_1_1 T_2_2 T_3_1 T_4_3 - Point
        L0 L1 L2 L3 L4 L5 L6 L7 L8 L9 L10 L11 - Site2D)

    ;;initial state definition
    (:init
        ;;Airports information
            (site_for_begin_takeOff AIRPORT_0 L0)
            (site_for_end_takeOff AIRPORT_0 L1)
            (site_for_landing AIRPORT_0 L2)
            (site_after_landing AIRPORT_0 L3)

            (site_for_begin_takeOff AIRPORT_1 L4)
            (site_for_end_takeOff AIRPORT_1 L5)
            (site_for_landing AIRPORT_1 L6)
            (site_after_landing AIRPORT_1 L7)
        ;;UAVs information
            (current_site UAV_0 L0)
            (= (Time-take-off UAV_0 AIRPORT_0) 180)
            (= (Time-landing UAV_0 AIRPORT_0) 360)
            (loaded_on_board UAV_0 S_0)
            (ready_to_takeoff UAV_0 AIRPORT_0)
            (= (Delta-Time UAV_0) 0)
            (= (Time-mission UAV_0) 0)

            (current_site UAV_1 L4)
            (= (Time-take-off UAV_1 AIRPORT_1) 120)
            (= (Time-landing UAV_1 AIRPORT_1) 240)
            (loaded_on_board UAV_1 S_1)
            (ready_to_takeoff UAV_1 AIRPORT_1)
            (= (Delta-Time UAV_1) 0)
            (= (Time-mission UAV_1) 0)

        ;;targets and observation requests information
            (= (Time-min-obs T_1_1) 30)
            (site_target L8 T_1_1)
            (to_observe T_1_1 S_0)
            (= (Earliest-time-start-obs-target T_1_1) 0)
            (= (Latest-time-end-obs-target T_1_1) 3000)
            (before T_1_1 T_3_1)
            (= (observable T_1_1) 0)
            (observable_p T_1_1)
            (= (Time-mission-start-obs-target T_1_1) 0)
            (= (Time-mission-end-obs-target T_1_1) 0)

            (= (Time-min-obs T_2_2) 60)
            (site_target L9 T_2_2)
            (to_observe T_2_2 S_0)
            (= (Earliest-time-start-obs-target T_2_2) 0)
            (= (Latest-time-end-obs-target T_2_2) 999999995904)
            (= (observable T_2_2) 0)
            (observable_p T_2_2)
            (= (Time-mission-start-obs-target T_2_2) 0)
            (= (Time-mission-end-obs-target T_2_2) 0)

            (= (Time-min-obs T_3_1) 15)
            (site_target L10 T_3_1)
            (to_observe T_3_1 S_1)
            (= (Earliest-time-start-obs-target T_3_1) 0)
            (= (Latest-time-end-obs-target T_3_1) 3000)
            (= (observable T_3_1) 999999995904)
            (= (Time-mission-start-obs-target T_3_1) 0)
            (= (Time-mission-end-obs-target T_3_1) 0)

            (= (Time-min-obs T_4_3) 180)
            (site_target L11 T_4_3)
            (to_observe T_4_3 S_1)
            (= (Earliest-time-start-obs-target T_4_3) 0)
            (= (Latest-time-end-obs-target T_4_3) 1500)
            (= (observable T_4_3) 0)
            (observable_p T_4_3)
            (= (Time-mission-start-obs-target T_4_3) 0)
            (= (Time-mission-end-obs-target T_4_3) 0)

        ;;temporal and spatial information
            (= (time_required L0 L0 UAV_0) 0)
            (= (time_required L0 L0 UAV_1) 0)
            (= (time_required L0 L1 UAV_0) 281.8)
            (= (time_required L0 L1 UAV_1) 281.8)
            ...
            (= (time_required L11 L11 UAV_0) 0)
            (= (time_required L11 L11 UAV_1) 0)
            ...

            (= (waiting-time) 100)
            (= (total-fly-time) 0)
            (= (min_delta_time) 999999995904)
            (= (max_time_mission) 0)

    )
    ;;goal and metric definition
    (:goal
        (and
            (covered T_1_1)
            (covered T_2_2)
```

```
101                 (covered T_3_1)
102                 (covered T_4_3)
103                 (landed UAV_0 AIRPORT_0)
104                 (landed UAV_1 AIRPORT_1)
105                 (<= (- (max_time_mission) (min_delta_time)) 10800)
106             )
107         )
108         (:metric minimize (total-fly-time))
109 )
```

Listing 6.3: The definition of the main elements of the PDDL problem in numerical planning for the instance of problem described in section 5.1.

Most of the reported information are the same we have described in chapter 5.

Notice in the *UAVs information* (rows `22-37`) the assertion, for each UAV in the problem, of the propositional fluent `ready_to_takeoff` which is a precondition of the action `initial-waiting` and it is then denied by the `take_off` action. Also notice the initialization to 0 of the numerical fluents `Delta-Time` and `Time-mission` related to each UAV.

We already defined how to model temporal windows of targets observation, see rows `43-44` or `54-55` for some example in the running example. In section 6.2.1 instead we describe how to model temporal constraints between targets observation.

Rows `92-93` reports two numerical fluent exploited by the planner to choose between some actions to apply. In particular `(min_delta_time)` represents the moment (delay from time 0) of the first take-off. It is initialized to an approximation of an infinite value in the initial state and it is updated by the *TakeOff* actions. In particular, every time the planner schedules a take-off which is temporally located before another take-off previously scheduled, this fluent is updated to the earliest time. Notice that in order to implement it you must define a further *TakeOff* actions (e.g. called `take_off_prev`) which is applicable only in case it has to be scheduled before the current `min_delta_time`. `(max_time_mission)` is a similar fluent which represents the moment of latest landing. It is initialized to 0 in the initial state and it is updated by the landing actions. Also this update requires a further *Landing* action applicable only in case it has to be scheduled after the current `max_time_mission`. The meaning of this additional fluents and actions is related to the goal of the max duration of the mission that can be calculated as reported in row `105` and that has not to consider the initial waiting actions.

Finally notice that the numerical fluent `(total-fly-time)` described in the previous section, which represents the total time in which all UAVs have been in flight, can be used as metric to minimize in problem evaluation and optimization (see row `108`).

6.2.1 Temporal constraints between targets observation

Fluents `before, observable` and `observable_p` are used, as well as in temporal planning, to model temporal constraints between targets observation. The predicate `before` (as well as the other temporal predicates described in 3.3) is used to specify the temporal constraint between targets observations. The numerical fluent `observable` expresses the first time point at which a target's observation can be performed. If the target's observation is not constrained by another one, this fluent is initialized to the same value of the `(Earliest-time-start-obs-target ?t - Target)` fluent (e.g. see row `75`). Otherwise if the target's observation is constrained by another one (e.g. in the running example `T_3_1` can be observed only after `T_1_1`) this fluent is initialized to a positive infinite value (or an approximation, see row `66`) and the related predicate `observable_p` is not asserted in the initial state (e.g. in the initial state reported in Listing 6.3 is not asserted the predicate `(observable_p T_3_1)`). These predicates are then used as preconditions in monitoring action as follows:

```
(osservabile ?t)
(<= (observable ?t) (Time-mission ?uav))
```

with `?t` the `Target` and `?uav` the `Uav` that has to perform the monitoring action. Therefore an observation `?t` constrained by a `before` predicate can be performed only after both the predicate `(observable_p ?t)` has been asserted and the value of the numerical fluent `(observable ?t)` has been updated by some other action to a value that is less than the time at which the agent needs to perform the action. It is worth noting that in order to express these constraints it is not sufficient to use the propositional fluent `observable_p` because, in case of multi-agents problems, there are no guarantee that the planner applies the actions of different agents in the right temporal order. Conversely it would be sufficient to use only the numerical fluent`(observable)` to express the constraint, however we noticed that using both of them helps the planner to find a solution faster.

It is also worth noting that we are assuming that a target's observation cannot be constrained by more than one other observation. For example we allow sets of constraints like
$(Before(OR_1, OR_2, 1, +\infty), Before(OR_3, OR_1, 1, +\infty), Before(OR_1, OR_4, 1, +\infty))$ in which a specific target's observation (e.g. OR_1) may be involved as constraining other observations in many temporal relations but at most one time as constrained observation. Conversely we don't allow sets of constraints like $(Before(OR_1, OR_2, 1, +\infty), Before(OR_3, OR_2, 1, +\infty))$ in which an observation (e.g. OR_2) is involved multiple times as constrained observation. This assumption implies that a certain observation could enable more than one other observation. Because the majority of planner doesn't support universal quantification and conditional effects, which would allow to enable, as effect of a monitoring action, all the other observations constrained by the current one, it is necessary to introduce a new type of action to do it.

```
(:action enable_target_bef
  :parameters (?t1 ?t2 - Target)
  :precondition (and (before ?t1 ?t2) (covered ?t1))
  :effect (and
        (assign (observable ?t2) (Time-mission-end-obs-target ?t1))
        (observable_p ?t2))
)
```

Listing 6.4: The definition of the PDDL action `enable_target_bef` in numerical planning.

Listing 6.4 reports the action schema which enable targets observation in case of *Before* constraint. In action's effects the `observable` function is updated to the value of the ending moment of observation of `?t1`. This means in case of *Before* constraint we permit the observation of `?t2` only after the observation of `?t1` is ended.

Notice that an alternative modeling of this constraints, which is cleaner, requires less fluents and no additional actions, consists in defining only the constraints as problem goals, by exploiting the fluents `Time-mission-start-obs-target` and `Time-mission-end-obs-target`. For instance it is possible to define for the running example the goal
`(< (Time-mission-end-obs-target T_1_1) (Time-mission-start-obs-target T_3_1))`. However this approach has proven to be particularly inefficient and to make unsolvable in reasonable time even very simple problems. Therefore we

adopted the former approach described in this section, even if more tricky to model.

6.3 Solution decoding

Plan

```
0.000: (take_off uav_1 l4 l5 airport_1)  [0.001]
0.001: (transfer_to_trg uav_1 l5 l11 t_4_3)  [0.001]
0.002: (monitor_point_target uav_1 l11 t_4_3 s_1)  [0.001]
0.003: (take_off uav_0 l0 l1 airport_0)  [0.001]
0.004: (transfer_to_trg uav_0 l1 l8 t_1_1)  [0.001]
0.005: (monitor_point_target uav_0 l8 t_1_1 s_0)  [0.001]
0.006: (enable_target_bef t_1_1 t_3_1 uav_0 uav_1)  [0.001]
0.007: (transfer_to_trg uav_0 l8 l9 t_2_2)  [0.001]
0.008: (monitor_point_target uav_0 l9 t_2_2 s_0)  [0.001]
0.009: (transfer_to_land uav_0 l9 l2 airport_0)  [0.001]
0.010: (transfer_to_trg uav_1 l11 l10 t_3_1)  [0.001]
0.011: (landing_succ uav_0 l2 l3 airport_0)  [0.001]
0.012: (monitor_point_target uav_1 l10 t_3_1 s_1)  [0.001]
0.013: (transfer_to_land uav_1 l10 l6 airport_1)  [0.001]
0.014: (landing_succ uav_1 l6 l7 airport_1)  [0.001]
```

LISTING 6.5: A valid temporal plan solving the problem described in section 5.1 in numerical planning. The actions in the plan have been manually annotated with temporal information on their ending time (see the numerical values reported after the squared brackets).

UAV_0	UAV_1
	120
	265
	445
180	
1054	
1084	
-	-
2265	
2325	
2823	
	2054
3183	
	2069
	3968
	4208

LISTING 6.6: The ending time of the actions of the plan in Listing 6.5.

Listing 6.5 reports a valid plan solving the problem modeled in section 6.2. The plan contains 15 actions: 14 concerns the UAVs and are equally divided between `UAV_0` and `UAV_1`, the remaining one (action `0.006`) is the support action `enable_target_bef` which enables the target's observation `T_3_1`.

Notice that the numerical values reported by the planner, which in temporal planning give information about the starting time and the duration of the actions, here don't provide any relevant temporal information. In fact the first numerical value only gives an order between actions in the sequence and the value within brackets is always 0.001 by default. In order to have a clearer view of the plan we manually calculated the ending time of the actions of the plan (which is also the starting time of the next action of the same UAV.) and we reported them in Listing 6.6. The manual operation we have done corresponds to part of the automatic decoding process required in numerical planning.

The decoding phase is essential in numerical planning in order to provide exhaustive and meaningful information to the user. The plan in fact cannot be directly displayed to the user because it is not self-explanatory and it also could be misleading. It is easy to see by looking at our manual temporal annotations that actions in planner's solution are only temporally ordered w.r.t. the same UAV but they aren't ordered among different UAVs. For instance actions `0.002` and `0.003`, which belong to different UAVs, are in a wrong temporal order (the monitoring action performed by `UAV_1` starts at time `265` and it is located in the plan before the `take_off` of `UAV_0` which starts at time `0`). Without temporal information this can be obviously confusing.

In order to represent the plan in a consistent higher level format easily interpretable by a human operator it is thus necessary a complex decoding procedure in which starting from the solution provided by the planner you must perform a kind of simulation of the execution in which each action is replaced by a temporally annotated one according to its type. It is necessary a strict link between encoding and decoding phase because they have to share a set of information which are used both on initial state definition and in solution translation (e.g. the time a UAV needs to land or the fixed duration of waiting actions necessary to simulate the passage of time). Therefore it is necessary to store in an internal database during the encoding phase all the used temporal information, such as the a priori defined durations of the actions (e.g. the duration of the *take off* of a certain UAV from a specific airport), as well as the time required to move between all couples of targets and points in the environment in order to recalculate the duration of the *Transfer* actions, or the minimum time of observation of a target (which is used as duration for the monitoring actions).

For instance the first three actions in Listing 6.5, concerning the `UAV_1`, require to store in a database a series of information in order to correctly decode them:

- The time required by `UAV_1` to take off. This information is encoded in initial state via the numerical fluent `(Time-take-off UAV_1 AIRPORT_1)` and its value is 120 seconds (see row `32` in Listing 6.3).
- The time required by `UAV_1` to move from the location `L5` to the location `L11`. This information is encoded in initial state via the numerical fluent

`(time_required L5 L11 UAV_1)`, its value is 145 seconds and it is used as duration of the transfer action.

- The minimum duration of observation of target T_3 required by the $ObsReq_4$ in the problem requirements (as specified in 5.1). This information is encoded in initial state via the numerical fluent `(Time-min-obs T_4_3)`, its value is 180 seconds and it is used as duration of the monitoring action.

- The mapping between the `Site2D L4`, `L5` and `L11` (which are the locations involved in the first three actions) and their spatial information in terms of coordinates in space. These information must be stored in the database during the encoding but they are not encoded in PDDL files. They are useful in order to decode the route that the UAV must perform. If the encoding process executed any approximation on paths in order to reduce the number of locations given to the planner (e.g. the representation of LOC target with only two points), also those operations must be taken into account in this phase in order to provide a correct representation of the agent's route (e.g. by retrieving the coordinates of the points of the LOC).

- The mapping between the `Sensor S_1` and the actual sensor's information, such as its identifier or its type. In the running example `S_1` corresponds to the device D_2. In a real world UAV problem these information would be, for instance, the identifier of the sensor loaded on board the agent and its type (e.g. *Active Radar 12345*).

This kind of information is used in order to decode the planner solution and obtain a annotated plan which contains all the interesting and useful information to generate a consistent solution representation easy to read by the end user.

As well as in temporal planning the decoded solution, in fact, is then used to build a Gantt Diagram of the mission and a visual representation on a map of the routes of the UAVs involved. Notice that while in temporal planning only spatial representation requires storing information into a database, in numerical planning it is required also by temporal representation. The decoding process consists first of all in a decomposition of the plan in separated plans, one for each UAV involved in the mission. Every action in the plan is then decoded in a task associated to a UAV (and to its devices) and in a sequence of waypoints involved in the action. As well as in temporal planning action's locations (`Site2D` in PDDL) are translated in waypoints of the route of the plan according to the action's type. For some action in the plan there is no corresponding UAV's task and they are not explicitly reported in user's solution. It is the case of the `wait_on_ground` actions which are simply used to retrieve information on the time of takeoff of the UAVs. Finally other actions are simply ignored during decoding because they don't contain relevant information for the user. It is the case of the enabling target observation actions.

Chapter 7

Experimental evaluation

In this chapter we present a series of experiments we performed. We encoded the class of problems and its extensions described in chapter 3 in both numerical and temporal models (as reported in chapters 5 and 6). We analyze how some of the main state-of-art numerical and temporal planners are able to deal with the classes of problems.

In the first part of the chapter we present a series of experiment performed on a dataset of synthetic problems automatically generated. These data allow us to analyze the complexity of the classes of problems and to compare both models and planners in terms of solution's quality, computational cost and planners' coverage.

It is worth noting that because problems are synthetic and their numeric values are randomly generated, some of them may be very constraining for planners. For instance a problem involving only two UAVs and four target observations may specify a set of temporal constraints on observability of targets so strict (even if problems encountered in the real word are often not so hard) that planners may not find a solution in reasonable time. The results of the experiments should then be analyzed w.r.t. these considerations. However the main purpose of our work concerns the capability to solve real-world problems.

In the second part of the chapter therefore we present in detail some real-world multi-UAV multi-target planning scenarios and the results that can be obtained with both numerical and temporal planning. In analyzing these scenarios we performed the experiment by using the planners which offered the best performances according to the results of the synthetic cases.

planner	metric-ff	lpg	lprpg-p	symba	colin	popf2	yashp3	tfd
type	num	num	num	num	n&t	n&t	n&t	tem
optimization	yes	yes	yes	yes	no	yes	no	yes
typed repres.	yes	yes	yes	yes	yes	yes	yes	yes
negative cond.	yes	yes	no	yes	yes	no	no	yes
ADL cond.	yes	yes	no	no	yes	no	no	yes
conditional eff.	yes	yes	no	yes	yes	no	no	yes
num. state var.	yes	yes	yes	yes	yes	yes	no	yes
til	-	-	-	-	yes	yes	no	no
continuous eff.	-	-	-	-	yes	yes	no	no
further req.	-	-	-	-	-	cplex	-	-

TABLE 7.1: A table showing the PDDL features supported by some of the main state-of-art available numerical and temporal planners. The *type* feature indicates the type of planning problems supported by the planner (*num*, if the planner supports only numeric problems, *temp* if it only supports temporal problems, *n&t* if it supports both types). The type *til* indicates timed initial literals, the PDDL feature introduced in PDDL 2.2 and described in 2.1.2. CPLEX is an optimization software package which is required by POPF2.

7.1 Planner presentation

The choice of the planner to use is often a difficult choice because it is necessary to mediate between expressive power and efficiency. Different planners, in fact, usually support different subsets of the language's features. Table 7.1 shows the PDDL features involved in the running class of problems that are supported by some of the main state-of-art available planners. The tables compares eight numerical and temporal planners. While some of them only supports either numerical (e.g. Metric-FF, LPG ([32])) or temporal (e.g. TFD) problems, others can be employed both in numerical and temporal planning (e.g. Colin, POPF2, YASHP3). Timed initial literals and continuous effects are PDDL features which can be employed only in temporal planning by using durative actions, therefore numerical planners don't support them. However it is worth noting that they aren't mandatory to perform temporal planning and some planners (e.g. TFD) doesn't support them.

It is easy to see that the most versatile planner among those presented, in terms of supported PDDL features, is COLIN. Because of its versatility in handling PDDL 2.1 and 2.2 features it is still a state-of-art planner and can be employed (as well as POPF2) both with numeric and temporal models. Planners supporting continuous change are obviously more complex than others which support less features. COLIN and POPF2 are the only two planners supporting continuous numeric effects, which play a critical role in modeling temporal problems in UAV domain (see the discussion in chapter 5). TFD and YASHP3 despite they are more recent temporal planners (they entered the IPC-2011 and IPC-2014 competitions), don't support such feature, therefore they don't guarantee actions' contiguity.

7.2 Synthetic examples

In this section we report the results of an extensive experimental evaluation performed on a dataset of synthetic problems of UAV domain.

We defined a test bed of multi-UAV problems with multiple target observation requests. We defined three main classes of examples based on the dimensionality of problems in terms of number of UAVs and targets involved:

- **Class 2U6T** involves two UAVs and six targets.
- **Class 3U8T** involves three UAVs and eight targets.
- **Class 4U10T** involves four UAVs and ten targets.

The definition of these classes of examples is due to two main reasons. On one hand problems involving less of six targets are easily solved by the most of planners and don't give us useful information. Planners start instead being distressed in solving problems involving more than six targets. By choosing these classes of examples we are then able to perform a more significant analysis on planners capabilities. On the other hand the real-world scenarios in which we are interested usually involve not more than three UAVs because of the logistic and operative difficulties in simultaneously coordinate integrated missions (we consider missions requiring MALE or MAME UAVs). Therefore we limited the number of UAVs to four.

Notice that for simplicity there is only one observation request for each target involved in problems. Furthermore in this synthetic examples we consider only Point target.

We specialized each class according to the classes of problems defined in chapter 3. In particular we defined four subclasses of examples of increasingly complexity, which can be combined between them:

- **Class UAV (U)** involves UAV problems without any temporal constraints between different target observations or specific temporal windows on target observation. Therefore targets can be observed at every time point during the mission and there is no precedence constraint. Moreover target observations are a priori assigned to UAVs.

- **Class MULTIOBS (M)** involves UAV problems specifying temporal constraints between different target observations. This class of problems is specialized in four other subclasses according to the number of temporal constraints: the first subclass involves one constraint, the fourth involves four different constraints.

- **Class WINDOWS (W)** involves UAV problems specifying specific temporal windows on target observations.

- **Class FREE (F)** involves UAV problems in which target observations are not assigned to specific UAVs.

The complexity of problems we analyze is therefore due to three main aspects: the problem dimensionality (i.e. the number of agents and targets involved), the types of constraints (i.e. to which extension of the baseline class **U** the problem belongs) and the planning model adopted. In the following sections we'll see that, despite dimensionality of problems is an important aspect to consider in problems complexity, increasing its value doesn't have a significant impact on problems complexity compared to introducing the temporal constraints. Furthermore we analyze how different types of constraints impact on the two planning models in terms of difficulties they introduce.

We analyze all these aspects by evaluating the ability of planners to solve the problems in terms both quantitative and qualitative.

For each combination of the classes of examples above reported we randomly generated ten different problems in both numerical and temporal formalisms, and we fed them to six of the planners above reported. In particular we used two numeric planners (Metric-FF and LPG) and three temporal planners (Colin, POPF2 and TFD). The selection of the set of planners was due to the features they support. In particular we discarded YASHP3 because of its lack of support to most of the PDDL features we used in our modeling. It is worth noting that both COLIN and

POPF2 are also able to due with numerical planning, therefore we employed them also in solving such type of problems.

We obtained a dataset of 600 different problems encoded in both numerical and temporal formalism. We fed them to each planner with a timeout of 180 seconds for every problem[1] and we obtained a huge set of 4200 different results (2400 concerning numerical problems and 1800 temporal ones). Notice that the choice of 180 seconds as timeout is due to two reasons: because we are studying an off-line approach to automated planning (therefore we can consider 3 minutes as a reasonable time to search for a solution) and because state-of-art planners are able to find solutions in tens of seconds or few minutes.

In our experiments we adopted two different versions of the domain for each planning model. In particular we defined a domain able to handle problems with assigned target observation and a domain able to handle problems with no assignments. Furthermore, because many planners don't support negative conditions we defined our domains without using that formalism. In order to be able to use the temporal planner TFD, which doesn't support continuous effects, we also defined another different domain, employed only by this planner and not containing the mechanism described in 5.3 which guarantees contiguity of actions. Therefore solutions found by TDF in UAV classes of examples don't provide any guarantee in terms of actions contiguity. However while they still guarantee that actions of the same UAV don't overlap, we decided to test it even without the support to continuous changes, obtaining potentially invalid plans for UAV domain. However because of the TDF's ability to build low makespan plans, the solutions were often valid and competitive with the ones of the other planners.

In the following sections we first analyze how dimensionality of problems impact on their complexity. This allows us to also compare the set of planners in terms of coverage and plans quality. After finding the best competing planners we'll use them to analyze in detail the impact of the introduction of different constraints on problems complexity w.r.t. the two planning models.

	Class 2U6T	Class 3U8T	Class 4U10T	Tot
LPG	99/200	94/200	76/200	269/600
COLIN	86/200	60/200	54/200	200/600
POPF2	65/200	48/200	35/200	148/600
Metric-ff	26/200	4/200	0/200	30/600

Table 7.2: A comparison between the numerical planners' coverage. Every cell reports the number of problems of a certain class solved by a planner, out of a total of 200 problems. Therefore the *Tot* column reports the total number of problems solved by the planners, out of a total of 600.

	Class 2U6T	Class 3U8T	Class 4U10T	Tot
COLIN	159/200	130/200	104/200	393/600
TDF	90/200	89/200	88/200	267/600
POPF2	0/200	0/200	0/200	0/600

Table 7.3: A comparison between the temporal planners' coverage. Every cell reports the number of problems of a certain class solved by a planner, out of a total of 200 problems. Therefore the *Tot* column reports the total number of problems solved by the planners, out of a total of 600.

7.2.1 Planners comparison

Tables 7.2 and 7.3 report a comparison between planners in terms of coverage w.r.t the classes of examples. Tables 7.4 and 7.5 respectively report a comparison between the three top planners of our competition in numerical planning and a comparison between the two top planners in temporal planning. The performances of the systems are measured according to the International Planning Competition metrics. Given a parameter p (plan cost), the score of a case for the system (planner) s in a set of tested systems S is defined by means of $\frac{bestValue(p,S)}{value(p,s)}$. For the time score, let T^* be the minimum time required by any planner, the formula $1/(1 + \log_{10}(T/T^*))$ is used to evaluate the performance of a system which spent T sec to solve the case. Cases solved in less than 1ss take the maximum score 1. Coverage scores 1 for solved case, 0 otherwise. The total score for a domain is the sum of scores obtained for each case in that domain.

In numerical planning LPG outperformed all the other planners both in quantitative and qualitative terms. LPG is a planner based on local search and planning graphs with a search scheme inspired by SAT-problems solvers. However it is worth noting that it was able to find a solution only for problems in which no constraints

[1]All planning was executed on a machine equipped with SO Linux Mint 12 64bit, Intel Core i3-2367M CPU@ 1.40GHz x 4, 4GB di RAM.

	LPG	COLIN	POPF2
Coverage	269	200	148
Plan quality	263,40	174,59	141,20
Time	259,78	182,23	77,56
Total	792,19	556,82	366,76

Table 7.4: A comparison between numerical planners.

	COLIN	TDF
Coverage	393	267
Plan quality	344,10	263,62
Time	340,94	240,45
Total	1078,04	771,08

Table 7.5: A comparison between temporal planners.

on target assignment were specified. The planner COLIN, which ranked second a number of solved problems similar to LPG, was instead able to solve problems belonging to any class. The POPF2 planner, which is built on the implementation of COLIN ranked third and despite its support to cost-optimisation it wasn't able to outperform COLIN in terms of plan's quality.

In temporal planning COLIN outperformed all the other planners in quantitative terms. The strongest temporal planner competitor of COLIN is TFD. Because of the lack of support of timed initial literals it was unable to solve any problem involving targets' observability windows. Despite this it was able to solve a total number of problems quite similar to COLIN. In fact it was the only temporal planner able to solve before the timeout a good rate of the submitted problems that meet its language requirements (the 85%). Furthermore, thanks to its optimization capability its solutions always outperformed the COLIN ones when both planners found one. POPF2 planner instead wasn't able to solve any temporal problem. It was mainly due to an incompatibility between CPLEX and the temporal formalisms we adopted.

In domains which don't require neither timed initial literals or continuous effects, we consider TFD as the best planner of those considered. For instance if the class of problems to solve is the baseline class described in 3.1, TFD is the best choice. However, as we already reported, the UAV domain requires contiguity between actions and it can be forced in temporal planning only through the use of continuous numerical effects. Since TFD doesn't support this PDDL feature it doesn't provide any guarantee on contiguity actions. Without continuous effects

	U	M	M+W	M+F	W	F	W+F	W+F+M
Class 2U6T	66,7%	29,2%	15,0%	46,7%	66,7%	100,0%	100,0%	40,8%
Class 3U8T	66,7%	23,3%	7,5%	37,5%	66,7%	100,0%	90,0%	35,8%
Class 4U10T	66,7%	24,2%	9,2%	33,3%	60,0%	86,7%	66,7%	28,3%
Total	66,7%	25,6%	10,6%	39,2%	64,4%	95,6%	85,6%	35,0%

Table 7.6: A comparison between the classes of examples w.r.t. the set of constraints involved, in numerical planning. Every cell reports the rate of problems of a class with a certain set of constraints solved by the set of planners.

	U	M	M+W	M+F	W	F	W+F	W+F+M
Class 2U6T	100,0%	90,0%	33,8%	93,8%	50,0%	45,0%	40,0%	35,0%
Class 3U8T	100,0%	90,0%	21,3%	96,3%	45,0%	15,0%	10,0%	23,8%
Class 4U10T	100,0%	81,3%	8,8%	93,8%	40,0%	15,0%	5,0%	16,3%
Total	100,0%	87,1%	21,3%	94,6%	45,0%	25,0%	18,3%	25,0%

Table 7.7: A comparison between the classes of examples w.r.t. the set of constraints involved, in numerical planning. Every cell reports the rate of problems of a class with a certain set of constraints solved by the set of planners.

the temporal model can guarantee that a UAV doesn't perform more than one actions concurrently (except for `stay_on_trg` and `monitor` actions), but it cannot provide any guarantee that there aren't "holes" between actions in the final plan. Therefore TFD, even if it has proven to build small makespan plans, is not the best planner for a UAV domain.

Notice that no problem was declared unsolvable by planners. This is not surprising because of the characteristics of the synthetic problems we generated. In fact the constraints are generated so as to always be feasible.

Starting from the previous comparison, despite it lost against LPG in numerical planning, we considered COLIN as the best planner for our purposes, and we'll use it in the following section in order to analyze real world scenarios. In fact in the domain we are studying, the automatic assignment of observations to UAVs is only a possible extension and in real world missions usually it is the human operator which decides the assignments according to his knowledges.

7.2.2 Classes of problems comparison

Tables 7.6 and 7.7 report a comparison between the three main classes of problems in terms of rate of problems solved for each subclass of examples w.r.t. the planning models. The tables are useful in order to evaluate how the extensions of the UAV class of problems impact on problems' complexity. Notice that in these analysis we didn't take into account the worst planner in the previous competition (Metric-FF in numerical planning and POPF2 in temporal planning) in order to have less noisy data.

First of all we can note that in both models incrementing the number of UAVs and observation requests doesn't significantly impact on problems' complexity. In fact the number of solved problems of the UAV class (see for instance columns U, M or W), doesn't significantly change when increasing the number of UAVs and targets.

We can note that while the two models behave similarly in such case, they behave differently when constraints are introduced in problems. Extending the class of problems with temporal constraints between target observations in numerical model, has a strong impact on problems' complexity. In temporal model, instead these constraints don't influence so heavily. Let us compare column M with column U in both tables: while in numerical planning the rate of solved problem is halved, in temporal planning it is only reduced by 10%.

This can be also easily noted in Fig. 7.1 ad 7.2 which report two line charts displaying the trend in the number of solved problems in both models as the number of constraints grows. Notice that while numerical model allows to more easily solve problems with a small number of constraint between different observations, temporal planning has a more stable behavior when the number of constraints grows.

Conversely extending the class of problems with targets observations' windows has a more significant impact on problems' complexity in temporal planning than in numerical planning. While numerical model is almost not influenced by the introduction of such type of constraint, temporal model's rate of success is instead halved.

It is worth noting that when the constraints are combined between them, they lead to a sharp increase in difficulty of solving problems in both models (e.g. see columns M+W+F and M+W) and the number of solved problems become really small). It is also easy to see that in both Fig. 7.3 and Fig. 7.4 which report the rate of solved problems divided according to the set of constraints.

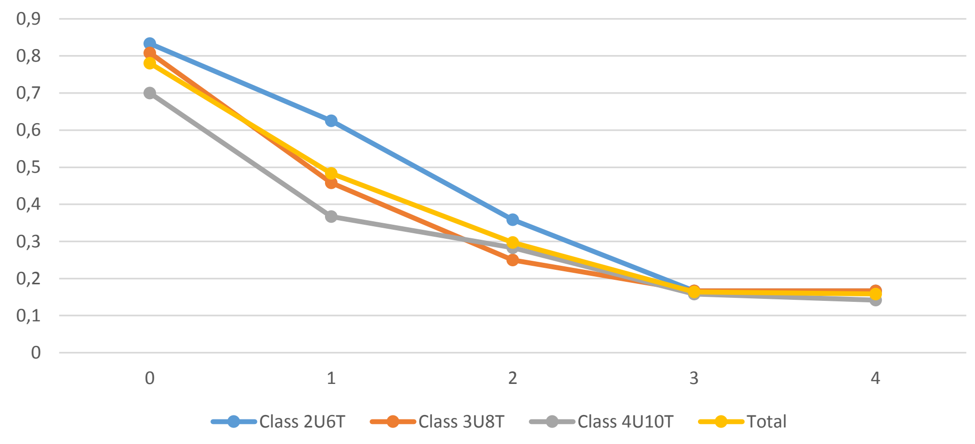


FIGURE 7.1: A line chart displaying the trend in the number of solved problems in numerical planning as the number of temporal constraints between target observation grows.

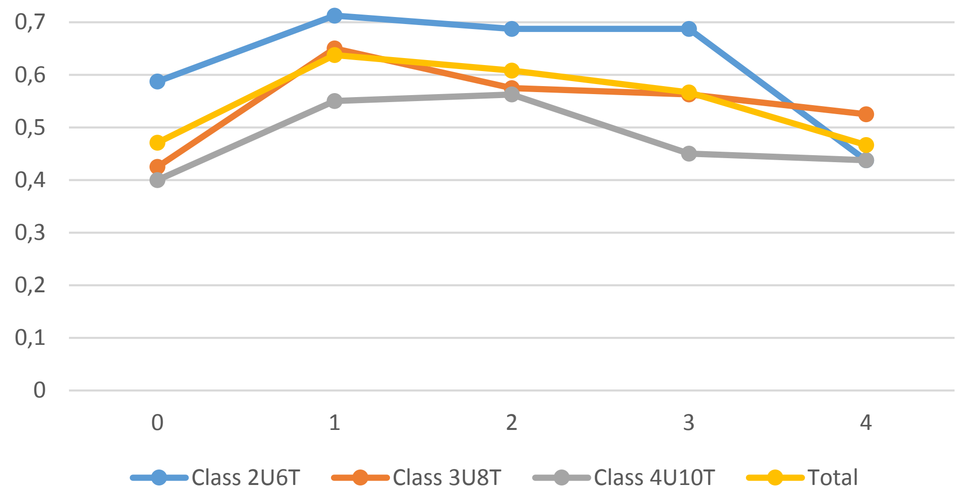


FIGURE 7.2: A line chart displaying the trend in the number of solved problems in temporal planning as the number of temporal constraints between target observation grows.

Finally we can see that, against expectations, in numerical planning requiring the planners to autonomously chose which UAV to use to satisfy an observation request, doesn't have significant consequences on problems' complexity. Conversely it seemed to simplify the problems.

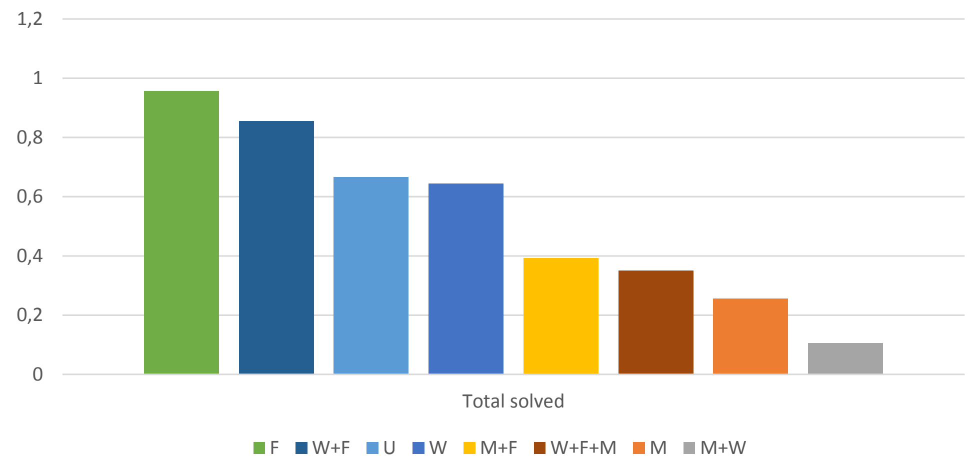


FIGURE 7.3: A histogram displaying the rate of solved problems in numerical planning w.r.t. the set of constraints involved.

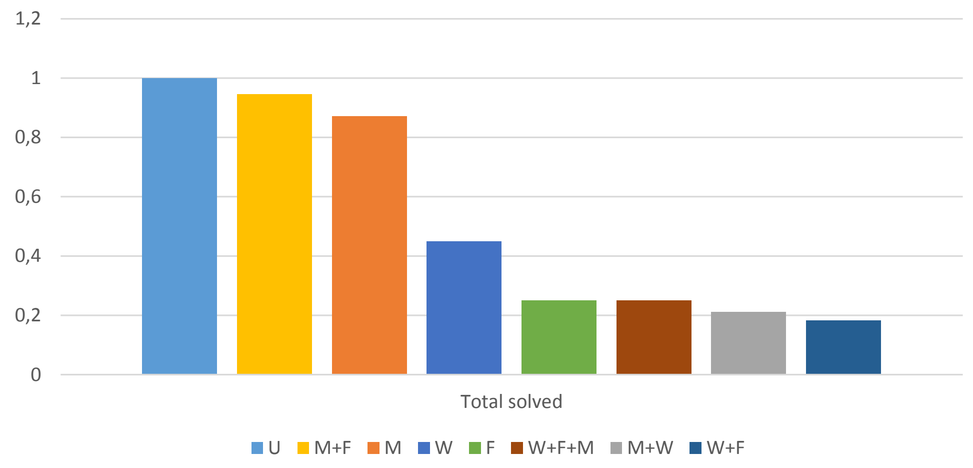


FIGURE 7.4: A histogram displaying the rate of solved problems in temporal w.r.t. the set of constraints involved.

	solved
Numerical	39%
Temporal	55%

TABLE 7.8: A comparison between the two planning models presented in this thesis in terms of the rate of problems solved by the set of numerical and temporal planners (without considering Metric-FF in numerical planning and POPF in temporal planning).

7.2.3 Models comparison

In this section we perform some more detailed analysis on differences between the two models. Table 7.8 reports a comparison between them. It displays the number of problems solved by the two sets of numerical and temporal planners. Notice that temporal problems are more easily solved compared to numerical problems within the same timeout.

As we stated above, the two models differently react to the extensions of the class of problems with different constraints. In particular data showed that in numerical planning it is difficult to solve problems involving temporal constraints between target observations, while in temporal planning it is more difficult to solve problems involving temporal windows for targets observation and the automatic assignment of observations to UAVs. These behaviors are mainly due to the way constraints are modeled in both formalisms. In fact in temporal planning it is easy to express in actions schema a series of requirements on different targets observations by using the PDDL 2.2 syntax which allows to express conditions at specified time points. In numerical planning instead it is necessary to simulate time passing and agents coordination in time is not automatically performed by the planner, therefore introducing constraints which require a strict coordination between agents leads to a significant increase of complexity. Conversely, defining temporal windows of observability of targets requires more work to a temporal planner, which treats these constraints as actual temporal information, than a numerical planner which treats them as numerical constraints ignoring time concept.

In order to take into account in our considerations of the planner efficiency we isolated the results of COLIN (see Table 7.9), which is the only planner among those considered able to handle both numerical and temporal problems. The first column reports the rate of problems solved by COLIN in both models. These values comply with the general observation that temporal problems are easier to solve (COLIN solved nearly twice as many problems). We also compared the two solutions provided by COLIN, when it was able to found a solution for the same problems in both formalisms (column 2). The solution provided by COLIN (it doesn't perform any optimization) are roughly equivalently in temporal and numerical planning in qualitative terms. Furthermore the last column of table shows that only 6% of problems (35 out of 600) were solved in numerical model when COLIN didn't find a solution in temporal one.

	solved	better	find
Numerical	34.5%	51%	6%
Temporal	65.5%	49%	37%

TABLE 7.9: A comparison between the behavior of planner COLIN in solving problems with the two planning models presented in this thesis. *solved* column reports the rate of solved problems in both models. *better* column reports, for each planning model, the rate of times the solution found by COLIN with that formalism was better than the one found with the other model. The last column reports, for each planning model, the rate of times COLIN found a solution with that model while it didn't with the other one.

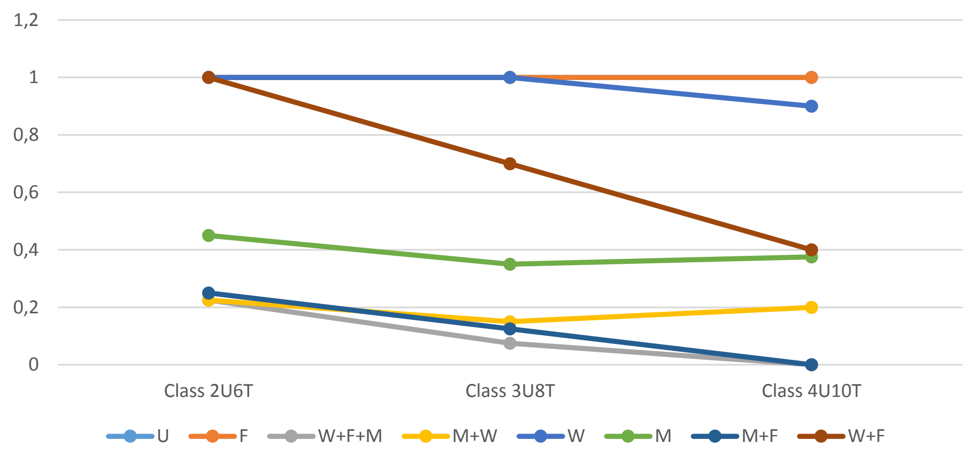


FIGURE 7.5: A line chart displaying the rate of problems solved by COLIN with the numerical model w.r.t. the set of constraints involved.

COLIN performances also confirm the considerations made on the complexity introduced in models by the different extensions of the class of problems. Fig. 7.5 and 7.6 report two line charts displaying, for each model, the rate of problems solved by COLIN w.r.t. the set of constraints involved. It is easy to see that with numerical model COLIN finds more difficult to solve problems involving constraints between different target observations. Conversely with temporal model these problems are more easily solved, while the planner finds more difficult to solve the problems involving targets observability windows.

Finally we can say that, generally speaking, temporal model is more performing than numeric one w.r.t. both problems' complexity and plans' quality. Furthermore as described in the previous chapters it requires a less complex encoding and decoding processes. However its difficulty in solving problems involving the extensions of the targets windows of observability can make it less performing than

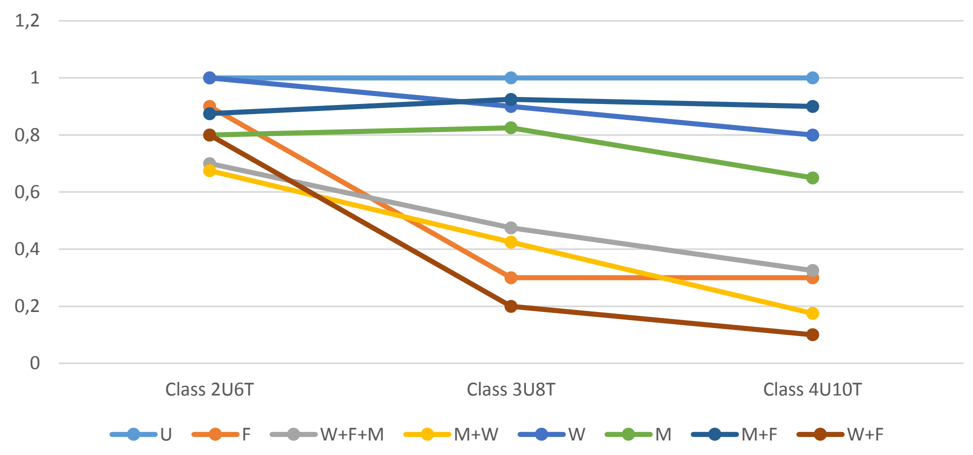


FIGURE 7.6: A line chart displaying the rate of problems solved by COLIN with the temporal model w.r.t. the set of constraints involved.

numerical model and how we'll see in the following section, this type of constraints is quite common in real-world UAVs scenarios.

7.3 Real-world scenarios

In this section we describe some real world scenarios in order to give the reader a more practical view of the capabilities of developed models. The examples of missions we report are based on real scenarios studied during the industrial research project SMAT-F2 and they are considered of possible interest in real world. The previous described synthetic problems involve objects which don't refer to real world entities (e.g. the used UAVs are only described by an identifier and they don't refer to any real type of aircraft). Furthermore numeric fluents (e.g. the distance between targets or the minimum duration of observations) are characterized by randomly generated values. In this section we describe missions in which the involved objects refer to real world entities. The targets considered correspond to real entities defined by specific geographical locations. The distances between locations are calculated according to great-circle distance ([33]). The UAVs features (e.g. cruise speed) refer to real MALE and MAME aircrafts as well as their ability to load on board a certain suite of sensors, and the airports involved in scenarios are real airports in which such types of UAV may likely be located.

UAV	Type	Speed	Sensor Suite
MAME N4	MAME	46 m/s	EO + Radar
MALE N7	MALE	50 m/s	EO + Hyperspectral

TABLE 7.10: A table reporting the available fleet of UAVs and their main features.

In solving the following problems we adopt the planner COLIN which, as shown just above, is the most complete and versatile of the analyzed ones and also allows us to easily compare numerical and temporal models.

7.3.1 The overall scenario

Before introducing and describing the specific tests, let us consider the overall scenario involving the North West part of Italy which will be used along the tests. Let us suppose the logistic base for MALE and MAME UAVs is the airport of Levaldigi and that for our missions we have the simple fleet of UAVs with the associated suite of sensors reported in Table 7.10 [2].

The tests we describe in this section spread over two different days and aim to show a series of difficulties that may came out in real world scenarios.

Table 7.11 reports the main pieces of information, divided by days, of the observation requests we'll use in the scenarios. It is worth noting that many pieces of information are omitted in the table for sake of readability (e.g. geographical information of targets). Some of them, however, are visible in the following maps representations. Furthermore, as mentioned in previous chapters, we perform some approximations on targets representation. In particular Point targets are represented just via their coordinates, while targets of type LOC are approximated via a polygonal chain represented by a sequence of vertices.

It is also worth noting that the couples of observation requests $6-7$ and $11-12$, which involve the same target ($CentraleTrino$) in the same day with the same temporal constraints but with different sensors (EO and Radar), are constrained by the following *Equals* constraints:

- *Equal(6, 7, 10)*

[2]Notice that, for simplicity, we are assuming the fleet is available all the days we need it. However, usually the availability of UAVs in specific dates and airports depends on logistic aspects which are beyond the purposes of this thesis.

OR	Date	Target	Type	Sensor	Obs. Win.	MinDur
1	02/08/16	Confl-Orco-Po	Point	EO	8.00 - 13.00	1m
2	02/08/16	PontePo-Verolengo	Point	EO	8.00 - 13.00	1m
3	02/08/16	Confl-DoraBaltea-Po	Point	EO	8.00 - 13.00	1m
4	02/08/16	PontePo-Crescentino	Point	EO	8.00 - 13.00	1m
5	02/08/16	Confl-Sesia-Po	Point	EO	8.00 - 13.00	90s
6	02/08/16	Centrale Trino	Point	EO	8.00 - 13.00	5m
7	02/08/16	Centrale Trino	Point	Radar	8.00 - 13.00	5m
8	02/08/16	TangTO	LOC	EO	7.40 - 8.30	NA
9	02/08/16	A4(TO-NO)	LOC	EO	7.40 - 10.30	NA
11	03/08/16	Centrale Trino	Point	EO	8.00 - 13.00	5m
12	03/08/16	Centrale Trino	Point	Radar	8.00 - 13.00	5m
13	03/08/16	TangTO	LOC	EO	7.40 - 9.00	NA
14	03/08/16	A4 (TO-NO)	LOC	EO	7.40 - 9.00	NA
15	03/08/16	A21(TO-Tortona)	LOC	EO	7.40 - 9.00	NA

Table 7.11: A table reporting the main pieces of information of observation requests. The column *Sensor* reports the sensors to use to observe the targets: *EO* stands for Electro-Optical. The column *Obs. Win.* reports the time windows in which targets are observable. The *NA* values in *MinDur* column (reporting the minimum durations of observation of targets) for LOC targets is due to the fact that we allow linear targets observation only by following their paths. Therefore the time of observation depends on target's length and UAV's cruise speed.

- *Equal(11, 12, 10)*

which specify the interest in observing the couples of targets with both the sensors in the same time window. Notice that we consider these constraints as also implicitly specifying that the two observations must be performed by the same UAVs, that is data fusion is required.

In the following scenarios we will also introduce some additional constraints between targets observation.

A last remark about observation requests concerns the apparent similarities between some of them. In some cases they are exactly the same but the date (see the two couples of observations of *CentraleTrino*). In other cases there are apparently minor modifications (e.g observation requests 8 and 13 differ just for the time window of observation). We will see that these minor changes have a significant impact on the mission requirements and consequently also on the final plan.

The description of the overall scenario shows that all the following scenarios belong to the class of problems above called **M+W** which involves, besides UAVs, both temporal constraints between different target observations and time windows of target observability. As reported in the previous section this is one of the most tricky class to solve. We describe four scenarios of increasingly complexity:

- **Scenario A** is the simplest scenario we report. It involves only 1 UAV that must perform 4 observations within specific temporal windows. Two observations are also temporally constrained among them. This scenario allows us to make some introductory considerations.
- **Scenario B** involves 2 UAVs and 5 target observations. Observations are constrained to specific temporal windows that, in some cases, are particularly strict. Furthermore two observations are temporally constrained among them. This scenario allows us to make some considerations on targets assignment.
- **Scenario C** is a more complex scenario. It involves, as scenario B, 2 UAVs and 5 target observations constrained within specific temporal windows. In this case, while temporal windows are not particularly strict, the problem specifies 2 constraints between different observations. The difficulty lies in the fact that these observations, which involve three different targets, are assigned to different UAVs.
- **Scenario D** is the hardest scenario we report. It involves only 1 UAV that must perform 9 observations within specific temporal windows. The problem also specifies 3 constraints among 6 different observations. This final scenario, as well as Scenario C, allows us to make some considerations about real-world problems complexity and about the limits of both planners and planning models.

7.3.2 The results in testing Scenario A

The first scenario we report concerns a simple mission with the following requirements:

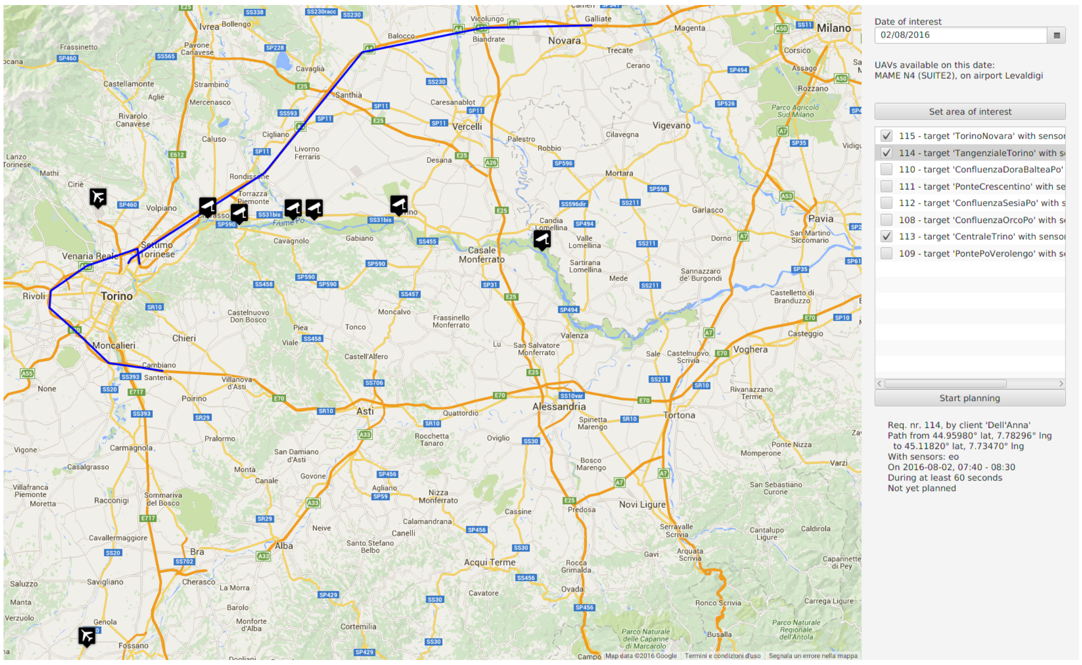
- Observation requests $6, 7, 8$ and 9. They concern the observation of the point target *CentraleTrino*, the LOC target *TangTO* (representing the highway around the city of Torino) and the LOC target *A4(TO-NO)* (representing the stretch of highway between Torino and Novara). The details of the requests, such as the time windows of targets observability and the sensors to be used, are reported in the first section of Table 7.11.

- The temporal constraint *Equal(6, 7, 10)*, which specifies that the monitoring actions covering the observation requests 6 and 7 must be performed by the same UAV in the same temporal interval and the durations of the two actions must not differ by more than 10 seconds.
- All the observations must be performed by the same UAV *MAME N4*, which has loaded on board a sensor suite compatible with the sensors requested.
- The mission must be performed between the 7 a.m. and 12 a.m.
- The maximum duration of the mission is 11000 seconds.

A first consideration we can do is that the planner COLIN doesn't perform any optimization. This means that the first plan it returns is potentially suboptimal w.r.t. the metric defined in problem and it may be considered a bad plan by a human operator. While in the previous section we have not dealt with this aspect, here we do it. The solutions we report in the following scenarios are obtained (when not specified otherwise) with an any-time approach that iteratively invokes COLIN by adding to the problem new constraints based on the previous solutions (e.g. if a solution has a final cost, in terms of `total-fly-time`, of 9000 seconds, we invoke again the planner with a new goal requirement which specify that `total-fly-time` must be < 9000). This mechanism forces the planner to search for a solution better than the previous one, until it reaches the timeout.

Figure 7.7 reports a map displaying the targets involved in the first day we defined. The selected observation requests correspond to those in the running scenario: the observations of the two LOC targets and of *CentraleTrino* (which in the map is the second Point target starting from right).

It is worth noting that if we execute COLIN only once it returns, as expected, a plan which is not optimal w.r.t. the metric `total-fly-time`. Listing 7.1 and Fig. 7.8 respectively report a decoding of the plan provided by COLIN and a graphical representation of it on a map. It has a cost of 8427 seconds but it is obviously an unacceptable plan for a real mission. In fact there is no specific temporal constraint that requires the UAV starts its observation of target *TangTo*



Date of interest
02/08/2016

UAVs available on this date:
MAME N4 (SUITE2), on airport Levaldigi

- [✓] 115 - target 'TorinoNovara' with sensor
- [✓] 114 - target 'TangenzialeTorino' with s
- [ ] 110 - target 'ConfluenzaDoraBalteaPo'
- [ ] 111 - target 'PonteCrescentino' with se
- [ ] 112 - target 'ConfluenzaSesiaPo' with s
- [ ] 108 - target 'ConfluenzaOrcoPo' with s
- [✓] 113 - target 'CentraleTrino' with sensor
- [ ] 109 - target 'PontePoVerolengo' with s

Req. nr. 114, by client 'Dell'Anna'
Path from 44.95980° lat, 7.78296° lng
to 45.11820° lat, 7.73470° lng
With sensors: eo
On 2016-08-02, 07:40 - 08:30
During at least 60 seconds
Not yet planned

FIGURE 7.7: A map displaying the targets related to all the observation requests defined in the first day (02/08/2016). The blue lines represents the LOC targets, tha camera icons the observation requests of point targets.

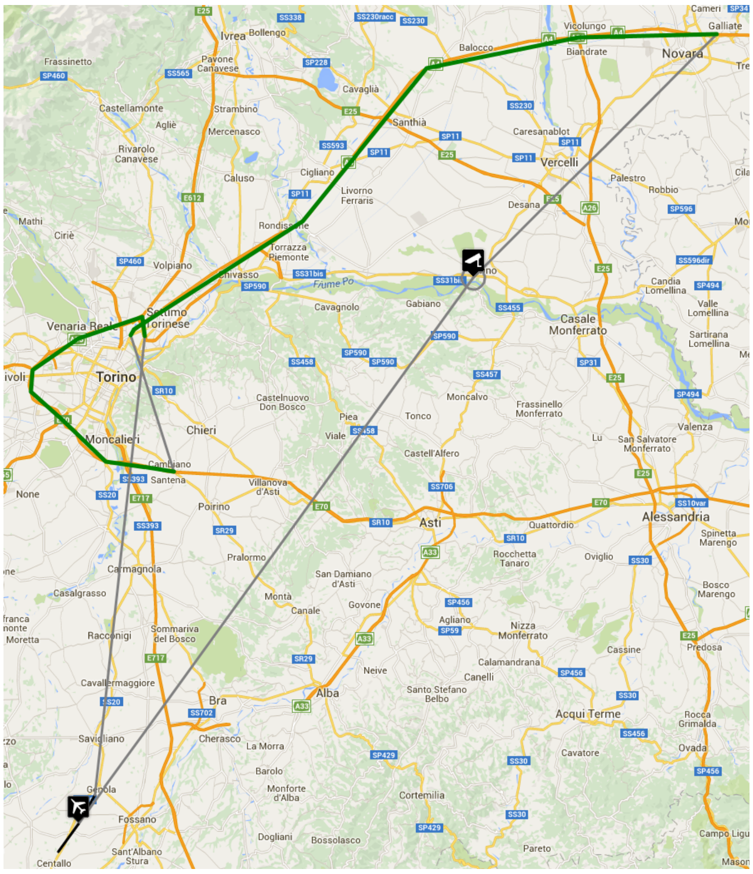


FIGURE 7.8: A map displaying the first suboptimal path (plan cost 8427 sec.) planned by COLIN with numerical formalism for UAV *MAME N4* in Scenario A.

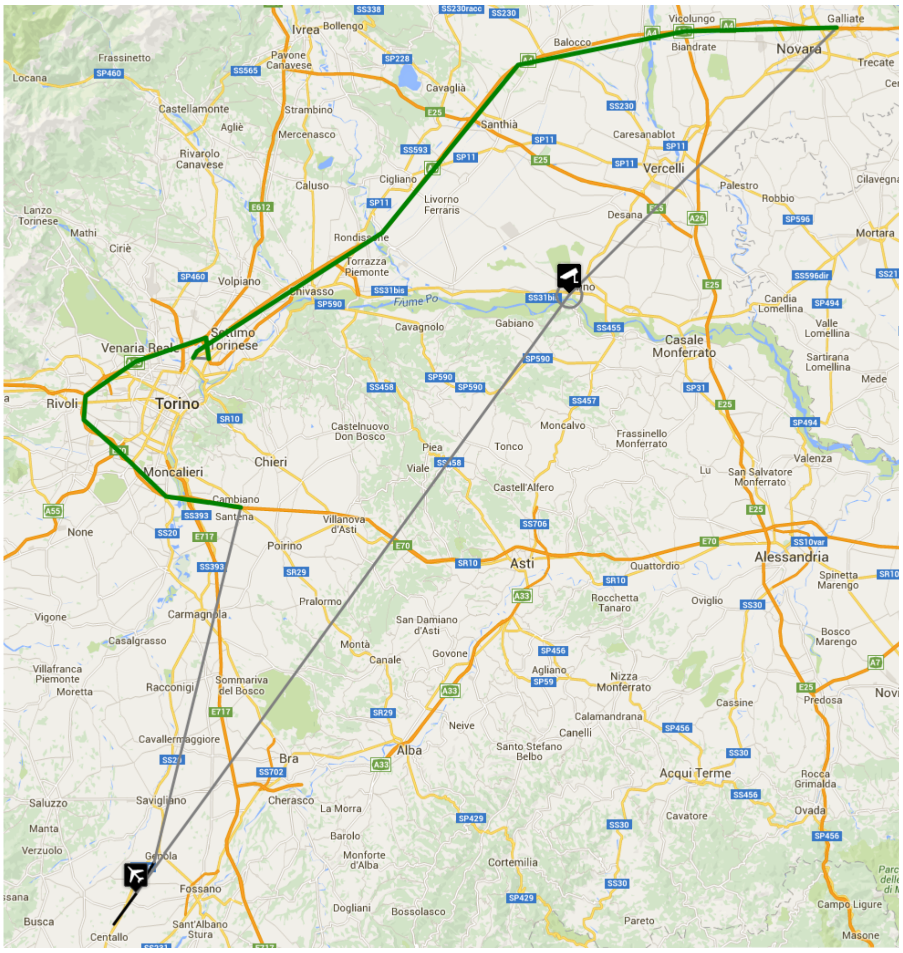


FIGURE 7.9: A map displaying the optimal path (plan cost 7698 sec.) for UAV *MAME N4* in Scenario A, obtained by iteratively feeding the problem to COLIN with numerical formalism while adding more strict constraints on plan cost.

from the northernmost point, therefore there is no reason to fly over the target without monitoring it.

In this phase we adopted as metric in both numerical and temporal planning the numerical fluent `total-fly-time` which represents the total time the entire team of UAVs has been in flight during the mission and it is updated by all the UAVs' move actions. It allows us to take into account of the entire team's flight time, without considering the time spent waiting on ground. Minimizing it means minimizing the time all UAVs stay on flight and therefore also the total fuel consumption, as well as the entire mission makespan.

If we iteratively ask COLIN better plans (as shown above), in few seconds we are able to obtain a much better solution (see Fig. 7.9).

It is worth noting that by using a temporal model, instead of a numerical one, in this case COLIN is able to find as first solution the optimal plan of Fig. 7.9.

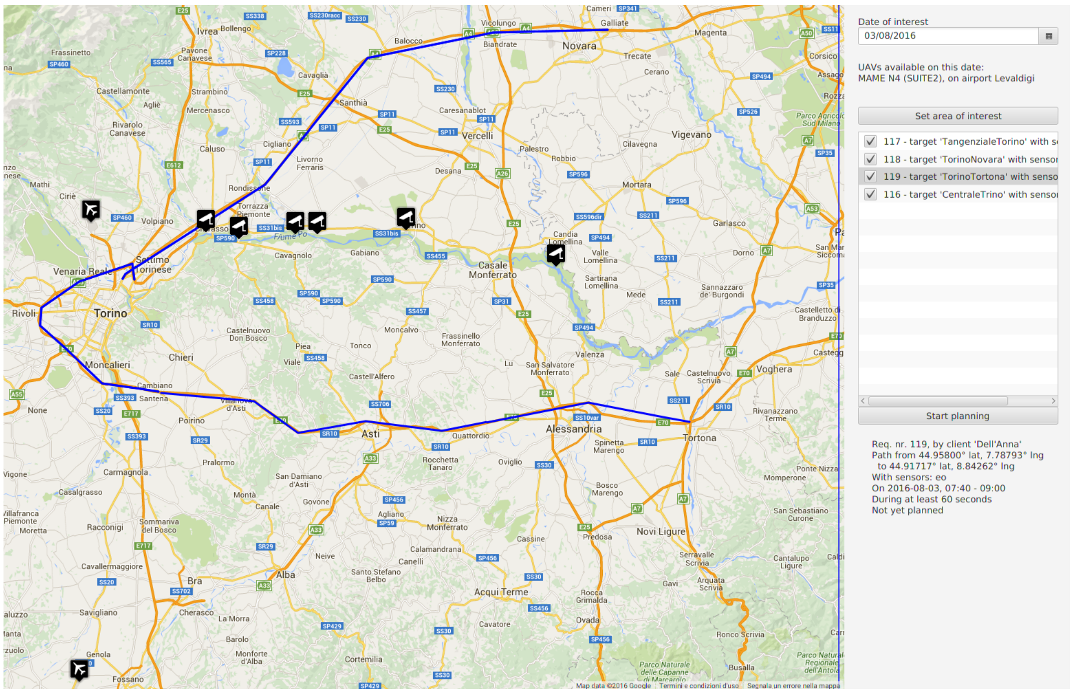
Notice that the output of the decoding can be displayed to a human operator in terms of both temporally annotated actions (also divided by agents) and a route on a map. Notice also that the plan foresees the use of both Radar end EO on *CentraleTrino* target (two simultaneous sensor tasks) as required in the mission requirements (see rows 38-39 Listing. 7.1).

For sake of readability starting from now we omit (unless it is necessary) the textual representation of plans. It contains many specific details useful in real mission planning but often unnecessary for our purpose.

```
;Plan for UAV MAME N4
08:50:52 - 08:54:52 (240 sec) Taking off from airport: Levaldigi

08:54:52 - 09:10:35 (943 sec) Transfering to target: TangenzialeTorino

09:10:35.694 - Flying through target TangenzialeTorino,
    from 45.11820  lat, 7.73470  lng
    to 44.95980  lat, 7.78296  lng (948 secs)

09:26:23 - 09:33:08 (405 sec) Transfering to target: TorinoNovara

09:33:08.694 - Flying through target TorinoNovara,
    from 45.11940  lat, 7.71163  lng
    to 45.46840  lat, 8.67987  lng (1956 secs)

10:05:44 - 10:21:51 (967 sec) Transfering to target: CentraleTrino

10:21:51 Loiter on this place

10:26:51 - 11:00:16 (2005 sec) Transfering to airport: Levaldigi

11:00:16 - 11:05:16 (300 sec) Landing to airport:  Levaldigi
11:05:16.694

;Plan for suite sensor EO+Radar, mounted on MAME N4
09:10:35 - 09:26:23 (948 sec)
    Monitoring target TangenzialeTorino Using sensors [eo]

09:33:08 - 10:05:44 (1956 sec)
    Monitoring target TorinoNovara Using sensors [eo]

10:21:51 - 10:26:51 (300 sec)
```

```
    Monitoring target CentraleTrino Using sensors [radar, eo]
```

LISTING 7.1: A decoding of the first suboptimal numerical plan provided by COLIN in solving the Scenario A.


Date of interest
03/08/2016

UAVs available on this date:
MAME N4 (SUITE2), on airport Levaldigi

- ✓ 117 - target 'TangenzialeTorino' with s
- ✓ 118 - target 'TorinoNovara' with sensor
- ✓ 119 - target 'TorinoTortona' with senso
- ✓ 116 - target 'CentraleTrino' with sensor

Req. nr. 119, by client 'Dell'Anna'
Path from 44.95800° lat, 7.78793° lng
to 44.91717° lat, 8.84262° lng
With sensors: eo
On 2016-08-03, 07:40 - 09:00
During at least 60 seconds
Not yet planned

FIGURE 7.10: A map displaying the targets related to all the observation requests defined in the second day (03/08/2016) and used in Scenario B.

7.3.3 The results in testing Scenario B

Scenario B allows us to make some considerations on targets assignments. It is described by the following requirements:

- All observation requests of the second day (observation requests $11-15$). See the details of all the requests in the second section of Table 7.11. Notice that the observation requests 13 and 14 contain a revised version of the temporal constraints on the target observation w.r.t the requests $8-9$ of the previous scenario. Moreover there is a new observation request involving the LOC target *A21(TO-Tortona)*.

- The temporal constraint *Equal(11, 12, 10)*, which specifies that the monitoring actions covering the observation requests 11 and 12 must be performed by the same UAV in the same temporal interval and the durations of the two actions must not differ by more than 10 seconds.

- The mission must be performed between the 6 a.m. and 12 a.m.

- The maximum duration of the mission is 12000 seconds.

Figure 7.10 reports a map displaying the targets involved in the running scenario.

First of all we can notice that all the observations cannot be performed by only one UAV. In fact the temporal windows specified by observation requests $13 - 15$ are too strict. No UAV is able, according to their cruise speed, to fly over the three specified highways in less than 80 min. Therefore no planner would be able to find a solution for this problem if all targets observations are assigned to one UAV.

In deciding the assignments a human operator may decide to assign the requests to two different UAVs as follows:

- Observation Requests $11, 12$ and 13 (*TangTO* and *CentraleTrino*) to *MALE N7*
- Observation Requests 14 and 15 (*A4(To-No)* and *A21(To-Tortona)*) to *MAME N4*

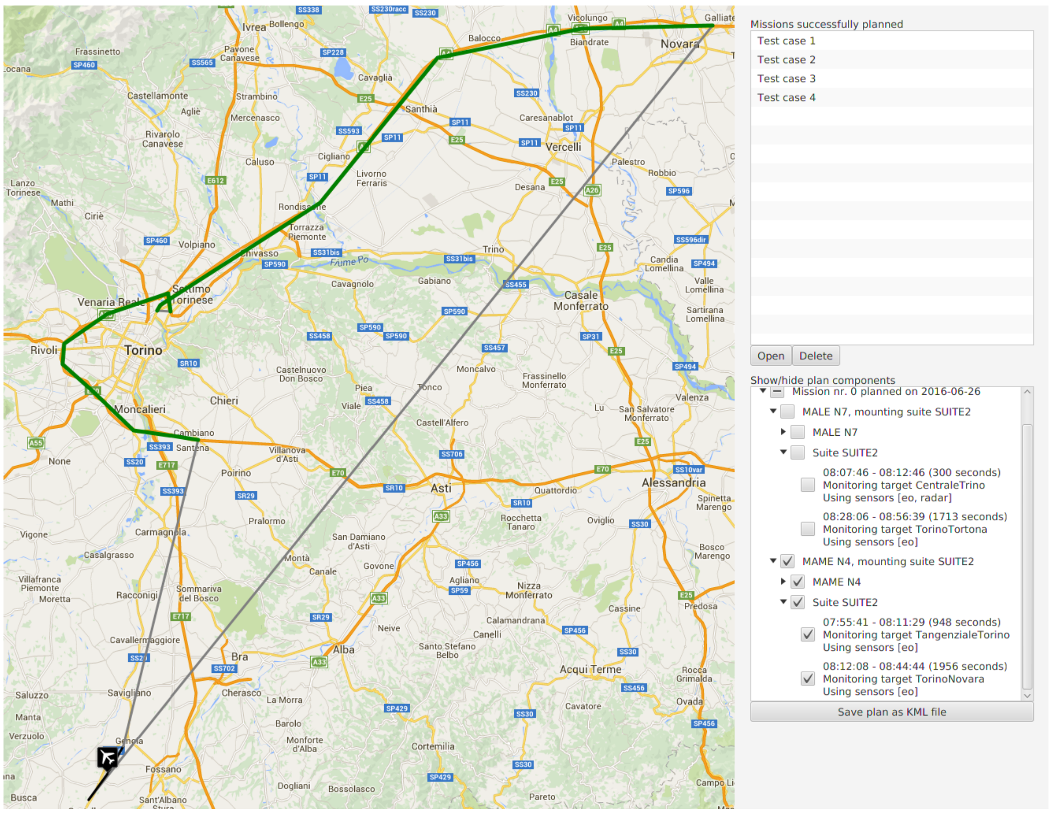
However also these assignments make the problem unsolvable. In fact even if the observations are subdivided between the two different UAVs, *MAME N4* is not able to fly over the two assigned highways within the specified time windows. In fact the airport Levaldigi is located between the two targets. Therefore the UAV, after monitoring one of the highways, must fly for too much time before to be able to observe the remaining target, violating the time constraints.

A feasible set of assignments for this problem is instead the following one:

- Observation Requests 13 and 14 (*TangTO* and *A4(To-No)* to *MAME N4*
- Observation Requests $11, 12$ and 15 (*CentraleTrino*) and *A21(To-Tortona)*) to *MALE N7*

For this last problem definition, the planner is able to easily find with numeric model (by using seconds as unit of measure) the optimal multi-agent plan of cost 13716 sec.. Fig. 7.11 and 7.12 report the paths planned for the team of UAVs, together with some pieces of information on the scheduled monitoring tasks of the sensors loaded on board the aircrafts.

This scenario allows to notice how in real world planning, sometimes the human operator who must perform some operations may find it difficult. For this reason



Missions successfully planned

Test case 1
Test case 2
Test case 3
Test case 4

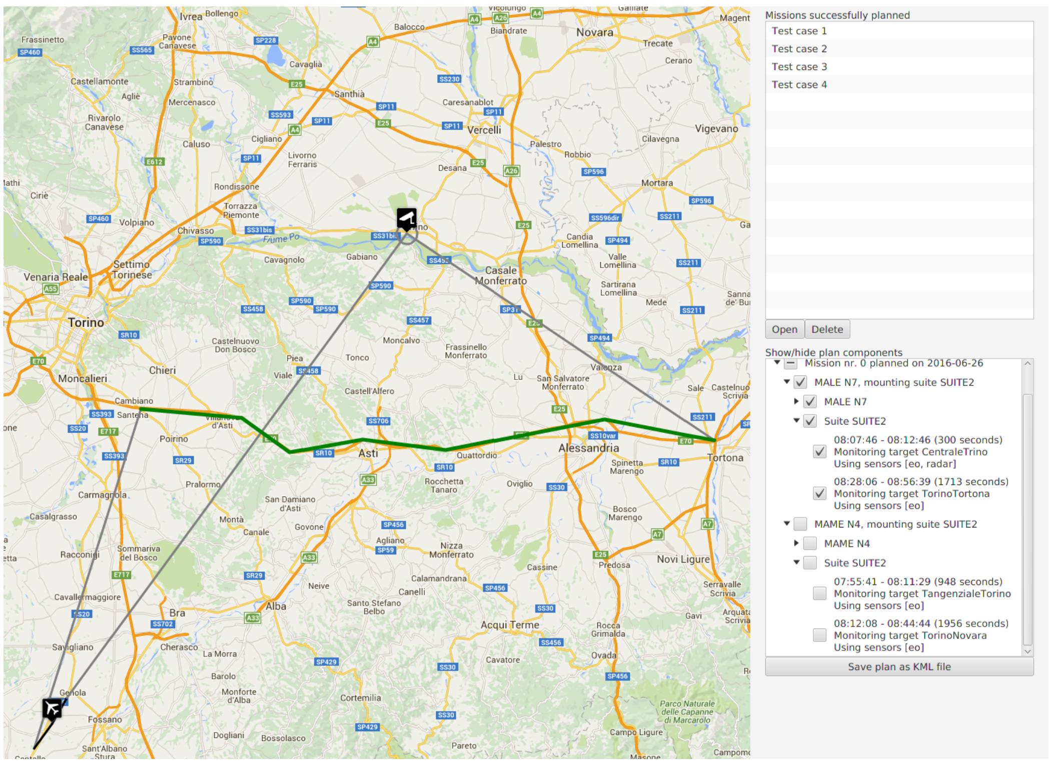
Open Delete

Show/hide plan components

- Mission nr. 0 planned on 2016-06-26
  - MALE N7, mounting suite SUITE2
    - MALE N7
    - Suite SUITE2
      - 08:07:46 - 08:12:46 (300 seconds) Monitoring target CentraleTrino Using sensors [eo, radar]
      - 08:28:06 - 08:56:39 (1713 seconds) Monitoring target TorinoTortona Using sensors [eo]
  - ✓ MAME N4, mounting suite SUITE2
    - ✓ MAME N4
    - ✓ Suite SUITE2
      - ✓ 07:55:41 - 08:11:29 (948 seconds) Monitoring target TangenzialeTorino Using sensors [eo]
      - ✓ 08:12:08 - 08:44:44 (1956 seconds) Monitoring target TorinoNovara Using sensors [eo]

Save plan as KML file

FIGURE 7.11: A map displaying the optimal path for UAV *MAME N4* in Scenario B.



Missions successfully planned

Test case 1
Test case 2
Test case 3
Test case 4

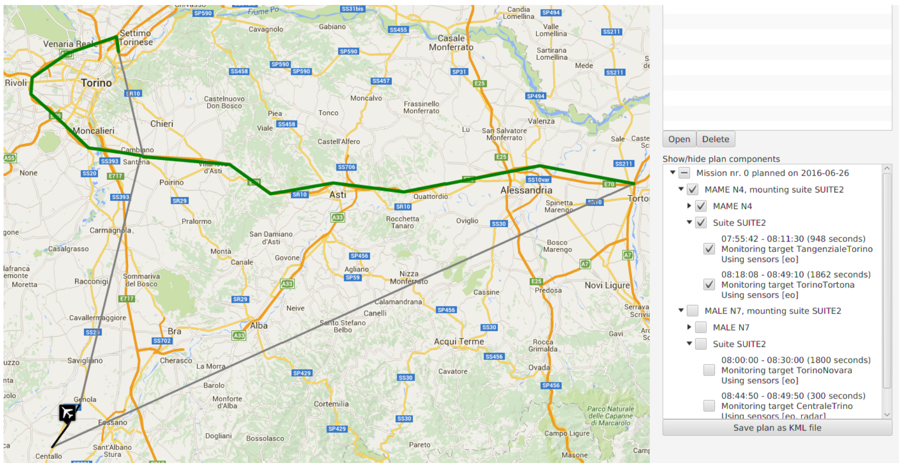
Open Delete

Show/hide plan components

- Mission nr. 0 planned on 2016-06-26
  - ✓ MALE N7, mounting suite SUITE2
    - ✓ MALE N7
    - ✓ Suite SUITE2
      - ✓ 08:07:46 - 08:12:46 (300 seconds) Monitoring target CentraleTrino Using sensors [eo, radar]
      - ✓ 08:28:06 - 08:56:39 (1713 seconds) Monitoring target TorinoTortona Using sensors [eo]
  - MAME N4, mounting suite SUITE2
    - MAME N4
    - Suite SUITE2
      - 07:55:41 - 08:11:29 (948 seconds) Monitoring target TangenzialeTorino Using sensors [eo]
      - 08:12:08 - 08:44:44 (1956 seconds) Monitoring target TorinoNovara Using sensors [eo]

Save plan as KML file

FIGURE 7.12: A map displaying the optimal path for UAV *MALE N7* in Scenario B.



Show/hide plan components

- ☐ Mission nr. 0 planned on 2016-06-26
  - ✓ MAME N4, mounting suite SUITE2
    - ✓ MAME N4
    - ✓ Suite SUITE2
      - ✓ 07:55:42 - 08:11:30 (948 seconds) Monitoring target TangenzialeTorino Using sensors [eo]
      - ✓ 08:18:08 - 08:49:10 (1862 seconds) Monitoring target TorinoTortona Using sensors [eo]
  - ☐ MALE N7, mounting suite SUITE2
    - ☐ MALE N7
    - ☐ Suite SUITE2
      - ☐ 08:00:00 - 08:30:00 (1800 seconds) Monitoring target TorinoNovara Using sensors [eo]
      - ☐ 08:44:50 - 08:49:50 (300 seconds) Monitoring target CentraleTrino Using sensors [eo, radar]

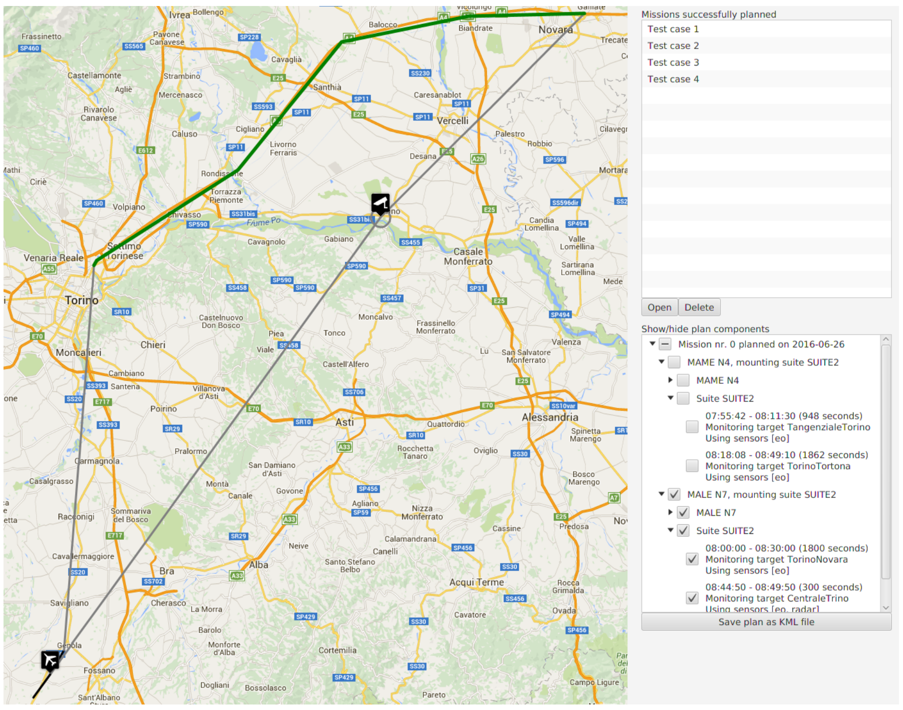
FIGURE 7.13: A map displaying the path for UAV *MAME N4* in Scenario B, obtained by the planner by automatically assigning observation requests to UAVs.

the planning tools must support the operator in making decisions and allowing to revising them when not feasible (we talk of Mixed-Initative Planning, which is a spotlight theme of recent years, see for instance [34], [35] and [36] works). A good Mixed-Initiative tool should be able to provide a solution to the human operator by reducing as much as possible his effort. For instance the operator may require the planner to find a solution by using a specific team of agent but without constraints on the assignments.

In some cases this may also lead to a final plan that is better than the one obtained with the fixed assignments made by the human operator. Fig 7.13 and 7.14 report the paths planned for the team of UAVs in the running scenario when the planner automatically assigns the observation requests. It is a slightly better plan than the previous one (its cost is 13609 sec.). It proves that sometimes may be difficult for a human operator manually evaluate all problem's constraints and make the optimal decisions.

However it is worth considering that sometimes an optimal plan is not necessarily the best choice. For instance in the last multi-agent plan the trajectories of the two UAVs intersect during the first transfer tasks. An expert human operator may prefer a solution slightly less optimized which, however, can be considered more robust.

It is finally worth noting that in this scenario numerical model easily allowed to find a first solution in only 0.08 seconds with a cost of 14731 and finding the final optimal solution required only 3 iterations of the anytime approach above



Test case 1
Test case 2
Test case 3
Test case 4

- [–] Mission nr. 0 planned on 2016-06-26
  - [ ] MAME N4, mounting suite SUITE2
    - [ ] MAME N4
    - [ ] Suite SUITE2
      - [ ] 07:55:42 - 08:11:30 (948 seconds) Monitoring target TangenzialeTorino Using sensors [eo]
      - [ ] 08:18:08 - 08:49:10 (1862 seconds) Monitoring target TorinoTortona Using sensors [eo]
  - [✓] MALE N7, mounting suite SUITE2
    - [✓] MALE N7
    - [✓] Suite SUITE2
      - [✓] 08:00:00 - 08:30:00 (1800 seconds) Monitoring target TorinoNovara Using sensors [eo]
      - [✓] 08:44:50 - 08:49:50 (300 seconds) Monitoring target CentraleTrino Using sensors [eo, radar]

FIGURE 7.14: A map displaying the path for UAV *MALE N7* in Scenario B, obtained by the planner by automatically assigning observation requests to UAVs.

described for a total time of 2.26 seconds (with a final fourth iteration which taken 39.54 seconds to declare the problem unsolvable, because of the constraints too strict on `total-fly-time`). Conversely temporal model was less efficient and it took 35.09 seconds only to find the first solution (the same of the first one found with the numerical model) and it reached a timeout of 5 minutes without finding a new solution.

7.3.4 The results in testing Scenario C

Scenario C allows us to make some further considerations on difficulties that may emerge from targets assignments. The scenario is described by the following requirements:

- The observation requests $1-5$ of the first day. See the details of all the requests in the first section of Table 7.11.
- The following temporal constraints:

  - *Before(1, 3, 1, +∞)*, which specifies that the observation of *Confl-Orco-Po* must end at least 1 second before *Confl_DoraBaltea-Po*.
  - *Before(3, 5, 1, +∞)*, which specifies that the observation of *Confl_DoraBaltea-Po* must end at least 1 second before *Confl-Sesia-Po*.

- The mission must be performed between the 6 a.m. and 12 a.m.
- The maximum duration of the mission is 12000 seconds.

Let us suppose that the operator decides to plan the mission by using two UAVs (this is not strictly necessary since just one of them is able to achieve the goals of the mission). In particular he decides to assigns:

- *Confl-Orco-Po* and *Confl_Sesia-Po* to *MAME N4*
- *Confl_DoraBaltea-P*, *PontePo-Verolengo* and *PontePo-Crescentino* to *MALE N7*

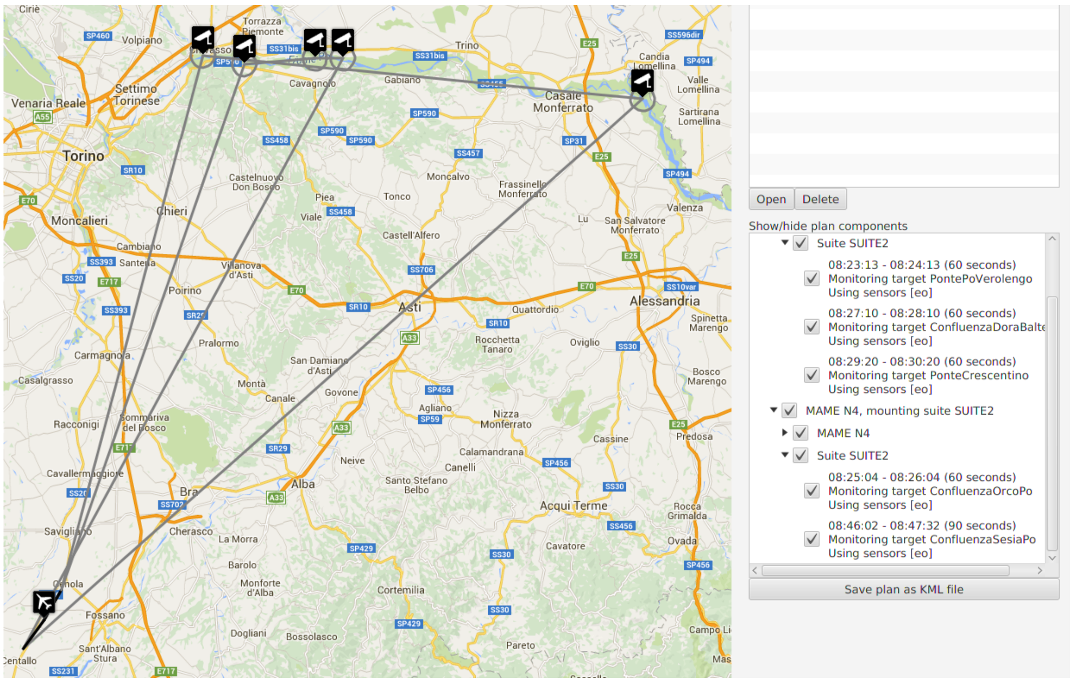
Notice that the two couples of target observations, which are assigned to different UAVs, are related via a temporal constraint.

Such type of constraints is very tricky and they make planning even more difficult than simple constraints between observations assigned to the same UAVs. In fact the plan for *MAME N4* cannot be performed independently on the plan of *MALE N7* since the decision on the observation of *Confl-Orco-Po* and *ConflDora Baltea-Po* influences on the observation of *Confl_Sesia-Po* and viceversa.

Planner COLIN wasn't able to solve the problem with neither numerical or temporal model. This confirms that real world problems, even involving only few UAVs and targets may be very hard to solve. Fig. 7.15 reports a valid planf for this mission. As we'll see in section 7.4 it can be obtained by adopting a different encoding process in numerical formalism which allows to express the values of numerical fluents concerning time aspects by using milliseconds as unit of measure instead of seconds.

7.3.5 The results in testing Scenario D

The scenario D we report is an extension of Scenario A. It is described by the following requirements:



```
▼ ✓ Suite SUITE2
    ✓ 08:23:13 - 08:24:13 (60 seconds)
      Monitoring target PontePoVerolengo
      Using sensors [eo]
    ✓ 08:27:10 - 08:28:10 (60 seconds)
      Monitoring target ConfluenzaDoraBalte
      Using sensors [eo]
    ✓ 08:29:20 - 08:30:20 (60 seconds)
      Monitoring target PonteCrescentino
      Using sensors [eo]
▼ ✓ MAME N4, mounting suite SUITE2
  ▶ ✓ MAME N4
  ▼ ✓ Suite SUITE2
    ✓ 08:25:04 - 08:26:04 (60 seconds)
      Monitoring target ConfluenzaOrcoPo
      Using sensors [eo]
    ✓ 08:46:02 - 08:47:32 (90 seconds)
      Monitoring target ConfluenzaSesiaPo
      Using sensors [eo]
```

FIGURE 7.15: A map displaying the optimal path for the team of UAVs *MAME N4* and *MALE N7* in Scenario C. The figure also reports some pieces of information which confirm the plan satisfies all the temporal requirements of the mission.

- All observation requests of the first day (observation requests $1-9$). See the details of all the requests in the first section of Table 7.11.
- The following temporal constraints:
  - *Equal(6, 7, 10)*, as described in Scenario A.
  - *Before(1, 3, 1, +∞)*, which specifies that the observation of *Confl-Orco-Po* must end at least 1 second before *Confl_DoraBaltea-Po*.
  - *Before(3, 5, 1, +∞)*, which specifies that the observation of *Confl_DoraBaltea-Po* must end at least 1 second before *Confl-Sesia-Po*.
- All the observations must be performed by the same UAV *MAME N4*, which has loaded on board a sensor suite compatible with the sensors requested.
- The mission must be performed between the 7 a.m. and 14 a.m.
- The maximum duration of the mission is 13000 seconds.

The task is very complex since a large number of POINT and LOC targets are involved and there are also many temporal constraints among the target observations

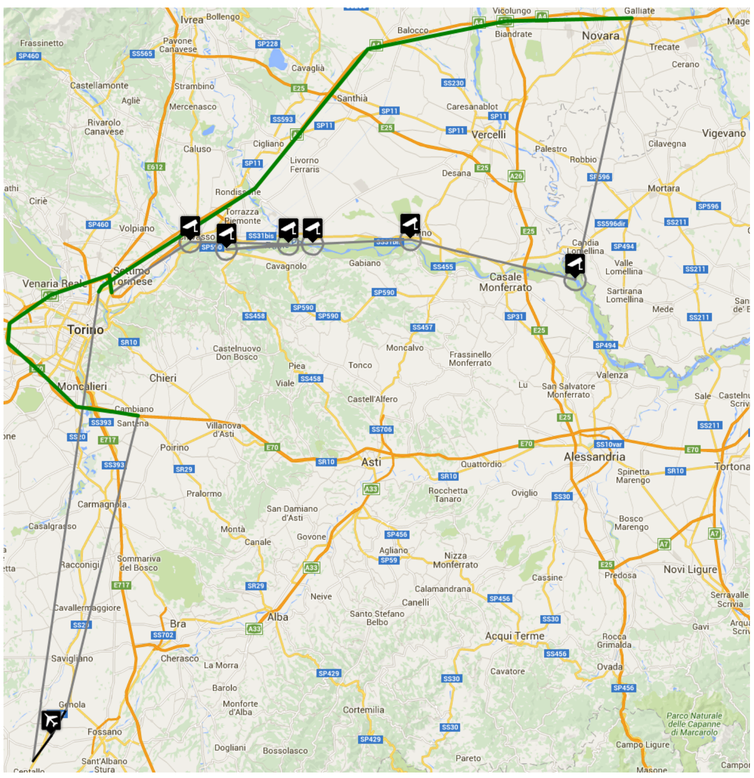


FIGURE 7.16: A map displaying the optimal path (plan cost 8276540 millisec.) for UAV *MAME N4* in Scenario D, obtained by iteratively feeding (as shown above) the problem to COLIN with numerical formalism by using milliseconds as unit of measure of temporal aspects.

that, as we observed in the previous section, significantly increase the problem's complexity. Furthermore the strict time windows of observation required by observation requests $8-9$ make the problem really tricky. COLIN, in fact, also in this case, wasn't able to solve it in reasonable time with none of the two modeling.

However this kind of scenario describes a plausible real world mission and we'd like to be able to solve it. By encoding all temporal aspects of the problems in a way similarly to as we show in section 7.4 for the time windows of observability of *TangTo*, we obtain a new version of the problem which COLIN is able to solve in seconds. Furthermore by applying the optimizing approach we are able to find in less than a minute the optimal solution reported in Fig. 7.16

It is also worth noting that the plan for this scenario is quite different from the one obtained for Scenario A. In particular, the plan for Scenario D is not just an extension of the one of Scenario A: in order to both optimizing the mission duration and satisfying the temporal constraints between the observation requests, the planner is actually able to find a plan where target observations are opportunely performed. (e.g. the LOC target *A4(Torino-Novara)* is observed in the reverse direction (from Novara to Torino)).

This scenario reveals the difficulties that can arise from real world problems even if concerning a single UAV mission. State-of-art planners are still weak in easily solving such type of problems and some intensive work is required in defining domains and problems in order to be able to solve them.

7.4 A different encoding for numerical model

By analyzing the reasons of the problem's complexity in section 7.2 we already noticed that the temporal windows on targets observations lead to a heavy increase in the level of complexity. In particular when they are too strict the planners easily fail in finding a solution. Starting from these considerations we decided to modify the units of measure used in defining time aspects in problems from seconds to milliseconds. This allows to enlarge, from the point of view of the planner, the observability windows of targets. For instance, while time window of observation required by observation request 8 for the target *TangTO* (see Table 7.11) is expressed in seconds by the following two numerical fluents initialization:

```
(= (Earliest-time-start-obs-target TangTO) 2400)
(= (Latest-time-end-obs-target TangTO) 5400)
```

by using instead milliseconds as unit of measure we can express it as:

```
(= (Earliest-time-start-obs-target TangTO) 2400000)
(= (Latest-time-end-obs-target TangTO) 5400000)
```

It is easy to see that this approach, at the expense of a cleaner model, allows to wide, from the point of view of the planner, the temporal windows of targets observations and this leads to a reduction of the problems complexity

We also re-executed, with planner COLIN, the numerical problems of synthetic experimentation and the results confirmed these considerations. Table 7.12 reports a comparison between the planning models in terms of number of problems solved by the planner COLIN. The numerical model with time values expressed in milliseconds is able to solve almost twice the number of problem solved with time

	COLIN numerical msec.	COLIN numerical sec.	COLIN temporal sec
Solved	395/600	207/600	393/600

Table 7.12: A comparison between the planning models in terms of number of problems solved by the planner COLIN.

expressed in seconds. Furthermore it solved about the same number of problems solved by using temporal model.

It is worth noting that while time expressed in milliseconds helps COLIN in solving problems with the numerical formalism, it is not helpful in temporal formalism. Conversely it increases the planner's difficulties in solving the problems because of the internal time management.

7.5 Experiments conclusions

In this chapter we performed a series of experiments which allowed to analyze the classes of problems we defined, the planning models and also the coverage of different planners.

In the former part of the chapter we analyzed a dataset of synthetic problems in order to evaluate the impact of the extensions of the UAV class of problems on problems complexity. The execution of synthetic examples allowed us to also evaluate different state-of-art planners and to choice the best (which resulted to be COLIN) in order to perform a further more accurate set of experiments. In fact, while during synthetic experiments problems were randomly generated and their features weren't specifically based on real-world elements, in the latter part of the chapter we analyzed a set of real-world scenarios. In particular we reported missions that, even if not particularly complex in terms of number of UAVs or targets w.r.t. the previous synthetic problems, are able to undermine planners because of the temporal constraints. These constraints, even if sometimes a bit excessive, can be considered realistic and they show that automated planning is still a difficult task, especially when time and temporal constraints are involved.

Chapter 8

Conclusions

In this thesis we addressed a class of real-world problems based on real world domains involving teams of agents in which temporal and consumable resource constraints are mandatory. We faced a series of increasingly complexity extensions of a baseline class of problems which involves teams of generic robotic agents whose objective is to observe a set of targets without any particular constraint. We initially considered a specialization of the generic type of robotic agent, the UAV, which is the class of agents on which we focused more in our work. Secondly we introduced a type of temporal constraints among different targets observations. Then we introduced a type of temporal constraints which specifies for each target the time windows within it is observable. Finally we introduced a parallel level of complexity requiring an automated assignment of observations to robotic agents.

This class of problems is particularly significant in many real-world applications. It allows to express interesting problems concerning autonomous intelligent machine (such as rovers, satellites, aircrafts or spacecrafts) acting in the real world. These problems are clearly not trivial because of their numerical and temporal nature.

Dealing with this class of problems is therefore a complex task and many challenges must be faced.

8.1 Thesis contributions

First of all it requires a great effort in building models which fully represent characteristics and constraints typical of the domains. In fact a direct translation of missions requirements in a planning model formalism, through the planning languages features, is not possible. An intense work of knowledge engineering, involving abstractions and simplifications on domain's objects is necessary. This implies a significant effort in encoding phase and leads sometimes to modeling that aren't immediate to be understood in all of their aspects. However such type of operations is necessary in order to be able to successfully employ general purpose state-of-art planners.

The intense work required to encode the problems is also due to the limits of both planning languages and planners. Planning languages (such as the latest extensions of PDDL), in fact, don't always provide easy ways to represent real life concepts. Furthermore they often allow to represent some concepts and requirements differently from one extension to another. For instance, while in temporal formalism it is possible to directly handle time and locate actions in time, in numerical planning (in which there is no support to neither durative actions, timed initial literals or continuous changes) it is necessary to explicitly represent time through a series of numerical fluents. As we reported, this has a significant impact both on actions schema definition and on the definition of fluents in the PDDL domain, as well as on a different modeling of temporal constraints. Also the newest PDDL temporal extensions (e.g. PDDL 2.2), however, still don't allow to easily take into account of many temporal aspects during planning and their support to these concepts is still limited (e.g. it is not possible to easily specify in PDDL domain definition any constraint by directly referring within actions model to internal time information such as the current time point at which actions are scheduled, as well as it is not possible to refer to the internal variable `total-time` except in metric definition).

Also planners are important factors which make modeling planning problems difficult. The expressive power provided by planning languages (with the above mentioned limits) is in fact further limited by planners capabilities which may not support all possible features. Some state-of-art planners that proved to be particularly performing don't support many PDDL features (e.g. temporal planner TDF, which was able in our experiments to solve before the timeout the 85% of the submitted problems that meet its language requirements and, thanks to its optimization capability, to provide good quality solutions, doesn't support timed initial literals and continuous effects). Other planners do not support even ADL features such as negative conditions, introduced in planning languages in order to overcome the STRIPS limits. This is limiting because many real life aspects are easily (and sometimes only) representable with specific language features and the lack of their support can lead to further unnecessary encoding difficulties. Planners are also limiting because their final solutions are not fully descriptive of all the features encoded in problems. Sometimes it is not possible to fully understand a planner's solution without information about the initial problem definition. This implies that planning must be supported by a software architecture that, during encoding stores into an internal knowledge base a set of information that are necessary during the phase of decoding of the planners solutions in order to report to the end user a consistent and easy-to-read solution.

Beside the encoding and decoding difficulties, the main challenge to face when dealing with the described class of problems concerns their complexity. In chapter 3 we described in detail the features of the class of problems and the extensions which introduce some not trivial temporal constraints. State-of-art planners are able to efficiently handle the baseline problems but they to have serious difficulties in solving more complex problems which involve the numerical and temporal constraints introduced by these extensions. Furthermore when different types of constraints are combined between them, they lead to a significant increase of problems complexity. In particular the coupling of the two types of temporal constraints described in 3.3 and 3.4 leads to a sharp increase in difficulty of solving problems, allowing planners to solve only about the 15% of the problems which involve them.

In this thesis we aimed to study how to successfully model the described class of problems and how its extensions affect problems complexity. In particular we studied how to encode problems in both numerical and temporal formalisms (chapters 5 and 6) by exploiting the main features of the planning language PDDL and its latest extensions (described in chapter 2). This allowed us to analyze how different types of constraints impact on the two planning models and to compare the results that can be obtained in both modeling by using the main general purpose state-of-art planners.

In order to make these analysis, in chapter 7 we performed a series of experiments on a dataset of automatically generated problems. This allowed us to evaluate the complexity of the class of problems w.r.t. both the set of constraints involved and the planning model adopted. We also evaluated the competence of some state-of-art numerical and temporal planners in terms of both coverage and plans quality.

Experimental analysis has given rise to many considerations.

First of all increasing the number of UAVs and targets involved in problems has shown to have a much less meaningful impact than introducing temporal constraints. For instance, let us consider problems with no temporal constraints among different targets observations (eventually involving time windows on targets observation). While switching from problems with two UAVs and six targets to problems with four UAVs and ten targets leads to a decrease of about 20% in the number of problems solved, adding instead only two temporal constraints to the class of problems with two UAVs and six targets leads to a decrease of more than 30% in the number of problems solved.

Furthermore our experiments have proven that the two planning models differently react to the extensions of the class of problems with different constraints. In particular temporal constraints between target observations have a stronger impact on numerical model, while with a temporal model it is more difficult to solve problems involving temporal windows for targets observation and automatic assignment of observations to UAVs. In our analysis we attributed these behaviors mainly to the way constraints can be modeled in both formalisms. In fact in temporal planning it is easy to express in actions schema a series of requirements on different targets observations by using the PDDL 2.2 syntax which allows to express conditions at specified time points. In numerical planning instead, as already mentioned, it is necessary to simulate time passing and agents coordination in time is not automatically performed by the planner, therefore introducing constraints which require a strict cooperation between agents leads to a significant increase of complexity. Conversely, defining temporal windows of observability of targets requires more work from a temporal planner, which treats these constraints as actual temporal information, than a numerical planner which treats them as numerical constraints ignoring time concept.

These considerations justify our encoding in both planning formalisms. There is no clear winner between the two models: each model is better for a certain type of problems and should be employed in the opportune situations. It is worth noting that the enormous effort required to opportunely build the models and develop encoding and decoding algorithms is an effort that must be done only once during the system development. After that the encoding and decoding processes can be performed in polynomial time when using the system. Furthermore despite the profound differences between the two planning models the input provided by an end user (the mission requirements) and the final output displayed by the system, in our architecture, have the same form, regardless of the underlying model. Therefore it is possible to integrate both planning models within the same architecture and even provide to an end user, in a mixed-initiative perspective, suggestions on which model to use, according to the types of requirements he expressed.

In chapter 7 we also analyzed a series of real-world scenarios which allowed us to make some consideration on real-world problems' complexity. The scenarios confirmed both the actual complexity of real-world problems and the limits of planners in dealing with that. Temporal constraints between two observations, especially if they must be performed by different agents, require a strong coordination between agents which, in order to satisfy them must both cooperate with each other and opportunely schedule their actions to perform the constrained observations in the right time intervals. Furthermore defining time windows for targets observations implies defining complex constraints whose nature is nearest to SMT and MILP problems than classical planning ones. We therefore showed that real-world scenarios of the class of problems we defined imply a series of hard constraints which are not trivial and strongly impact on problems complexity and planners ability to solve them.

In our analysis we also have shown that scalability is one of the major problem in solving these problems and mechanisms of problem's simplification must be often performed in order to be able to find satisfying solutions. Planners have proven to be very sensitive to non obvious factors, such as the unit of measure of numerical fluents, and consequently they don't easily allow to make accurate predictions on problems solvability. However this study have also shown that they are capable of efficiently solving classes of problems with few (and not particularly strict) temporal constraints. This type of class of problems is also relevant in real world. In fact there is a great number of real life problems involving robotic agents that must perform their tasks in a restricted environment (such as domotic agents or mini/micro UAVs) which can be addressed by exploiting the solutions described in this thesis.

8.2 Extensions and future works

Possible extensions of this work may concern analyzing strategies to reduce the problems complexity to help planners in finding a solution. In order to do that it is possible to perform a series of simplifications during problems encoding. In the thesis we mentioned the abstraction we made on a LOC target by only considering the two end points of the polygonal chain which describes it and giving to the related numerical fluents (such as the time required by a UAV to fly over the target) the opportune values based on the real shape of the LOC. However many other strategies and simplifications can be developed.

For instance, when adopting a numerical model in UAV domain, it is possible to employ some heuristics in order to pre-calculate, where possible, the first useful instant when the UAVs can take off, based on the windows of observability of the targets. [1] This is helpful because it means that the planner may not schedule a series of *wait on ground* actions, otherwise necessary in order to obtain an optimal plan, and it has to find a shorter plan.

Another simplification that can be done concerns clustering. Clustering mechanism could be activated when multiple targets are close enough (e.g. the distance between the targets is less than a predefined threshold), and they must be observed with the same sensor and the set of targets could be replaced by a new fictitious target located in the centroid of the cluster of targets. This operation permits to reduce the computational cost of producing an (optimized) plan since it reduces the number of targets (and consequently the number of alternatives the planner has to explore).

A further extension due to reduce problems complexity may be perform some automatic problem decompositions in case of multi-agent problems in which all the agents are independent. In fact if there is no temporal constraint between the observation requests assigned to two different UAVs, the two missions can be considered separately. In this way we can obviously reduce the complexity of the problems the planner has to solve.

Finally in our experiments on synthetic examples we noticed that, against expectations, requiring the planners to autonomously choose which UAV to use to

[1] In particular by taking into account of both the earliest time of targets observability windows and the time required by the UAVs to take off and to reach the first observable target, it is possible to calculate the first useful instant when the aircrafts can take off.

satisfy an observation request, doesn't have significant consequences on problems' complexity w.r.t. the other constraints. A future work may concern modeling a class of problems in which system autonomously decide a subset of UAVs with whom accomplish the tasks and integrate it within a mixed-initiative system of support to mission planning.

Bibliography

[1] Malik Ghallab, Dana S. Nau, and Paolo Traverso. The actor's view of automated planning and acting: A position paper. *Artif. Intell.*, 208:1–17, 2014. doi: 10.1016/j.artint.2013.11.002. URL `http://dx.doi.org/10.1016/j.artint.2013.11.002`.

[2] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.).* Pearson Education, 2010. ISBN 978-0-13-207148-2. URL `http://vig.pearsoned.com/store/product/1,1207,store-12521_isbn-0136042597,00.html`.

[3] Richard Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3/4):189–208, 1971. doi: 10.1016/0004-3702(71)90010-5. URL `http://dx.doi.org/10.1016/0004-3702(71)90010-5`.

[4] Drew V. McDermott. The 1998 AI planning systems competition. *AI Magazine*, 21(2):35–55, 2000. URL `http://www.aaai.org/ojs/index.php/aimagazine/article/view/1506`.

[5] Stefan Edelkamp. Pddl2. 2: The language for the classical part of the 4th international planning competition. 2004.

[6] Michael Brenner and Bernhard Nebel. Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems*, 19(3):297–331, 2009. doi: 10.1007/s10458-009-9081-1. URL `http://dx.doi.org/10.1007/s10458-009-9081-1`.

[7] Roberto Micalizio. Action failure recovery via model-based diagnosis and conformant planning. *Computational Intelligence*, 29(2):233–280, 2013. doi: 10.1111/j.1467-8640.2012.00444.x. URL `http://dx.doi.org/10.1111/j.1467-8640.2012.00444.x`.

[8] Daniel L Kovacs. Bnf definition of pddl 3.1. *Unpublished manuscript from the IPC-2011 website*, 2011.

[9] Daniel L Kovacs. A multi-agent extension of pddl3. *WS-IPC 2012*, page 19, 2012.

[10] Alejandro Torreño, Eva Onaindia, and Oscar Sapena. FMAP: distributed cooperative multi-agent planning. *Appl. Intell.*, 41(2):606–626, 2014. doi: 10.1007/s10489-014-0540-2. URL `http://dx.doi.org/10.1007/s10489-014-0540-2`.

[11] Jörg Hoffmann. FF: the fast-forward planning system. *AI Magazine*, 22 (3):57–62, 2001. URL `http://www.aaai.org/ojs/index.php/aimagazine/article/view/1572`.

[12] Malte Helmert. Decidability and undecidability results for planning with numerical state variables. In *On the Role of Ground Actions in Refinement Planning.*, pages 44–53, 2002. URL `http://www.aaai.org/Library/AIPS/2002/aips02-005.php`.

[13] Jörg Hoffmann. The metric-ff planning system: Translating ”ignoring delete lists” to numeric state variables. *J. Artif. Intell. Res. (JAIR)*, 20:291–341, 2003. doi: 10.1613/jair.1144. URL `http://dx.doi.org/10.1613/jair.1144`.

[14] Derek Long and Maria Fox. The 3rd international planning competition: Results and analysis. *J. Artif. Intell. Res. (JAIR)*, 20:1–59, 2003. doi: 10.1613/jair.1240. URL `http://dx.doi.org/10.1613/jair.1240`.

[15] Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long. COLIN: planning with continuous linear numeric change. *J. Artif. Intell. Res. (JAIR)*, 44:1–96, 2012. doi: 10.1613/jair.3608. URL `http://dx.doi.org/10.1613/jair.3608`.

[16] Amanda Jane Coles and Andrew Coles. LPRPG-P: relaxed plan heuristics for planning with preferences. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011*, 2011. URL `http://aaai.org/ocs/index.php/ICAPS/ICAPS11/paper/view/2668`.

[17] Mauro Vallati, Lukás Chrpa, Marek Grzes, Thomas Leo McCluskey, Mark Roberts, and Scott Sanner. The 2014 international planning competition: Progress and trends. *AI Magazine*, 36(3):90–98, 2015. URL http://www.aaai.org/ojs/index.php/aimagazine/article/view/2571.

[18] Alvaro Torralba, Vidal Alcazar, Daniel Borrajo, Peter Kissmann, and Stefan Edelkamp. Symba: A symbolic bidirectional a planner. In *International Planning Competition*, pages 105–108, 2014.

[19] Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long. Forward-chaining partial-order planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, pages 42–49, 2010. URL http://www.aaai.org/ocs/index.php/ICAPS/ICAPS10/paper/view/1421.

[20] Vincent Vidal. Yahsp3 and yahsp3-mt in the 8th international planning competition. *Proceedings of the 8th International Planning Competition (IPC-2014)*, pages 64–65, 2014.

[21] Luc De Raedt, Christian Bessière, Didier Dubois, Patrick Doherty, Paolo Frasconi, Fredrik Heintz, and Peter J. F. Lucas, editors. *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, 2012. IOS Press. ISBN 978-1-61499-097-0. URL http://www.booksonline.iospress.nl/Content/View.aspx?piid=31572.

[22] Mathijs de Weerdt and Brad Clement. Introduction to planning in multiagent systems. *Multiagent and Grid Systems*, 5(4):345–355, 2009. doi: 10.3233/MGS-2009-0133. URL http://dx.doi.org/10.3233/MGS-2009-0133.

[23] Matthew Crosby and Ronald PA Petrick. Temporal multiagent planning with concurrent action constraints. In *Proceedings of the ICAPS 2014 Workshop on Distributed and Multi-Agent Planning (DMAP)*, pages 16–24, 2014.

[24] Alejandro Torreño, Eva Onaindia, and Oscar Sapena. An approach to multiagent planning with incomplete information. In Raedt et al. [21], pages 762–767. ISBN 978-1-61499-097-0. doi: 10.3233/978-1-61499-098-7-762. URL http://dx.doi.org/10.3233/978-1-61499-098-7-762.

[25] Arthur Richards, John Bellingham, Michael Tillerson, and Jonathan How. Coordination and control of multiple uavs. In *AIAA guidance, navigation, and control conference, Monterey, CA*, 2002.

[26] John S Bellingham, Michael Tillerson, Mehdi Alighanbari, and Jonathan P How. Cooperative path planning for multiple uavs in dynamic and uncertain environments. In *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, volume 3, pages 2816–2822. IEEE, 2002.

[27] Leonardo Mendonça de Moura and Nikolaj Bjørner. Satisfiability modulo theories: introduction and applications. *Commun. ACM*, 54(9):69–77, 2011. doi: 10.1145/1995376.1995394. URL `http://doi.acm.org/10.1145/1995376.1995394`.

[28] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 337–340, 2008. doi: 10.1007/978-3-540-78800-3_24. URL `http://dx.doi.org/10.1007/978-3-540-78800-3_24`.

[29] Ji-Ae Shin and Ernest Davis. Processes and continuous change in a sat-based planner. *Artif. Intell.*, 166(1-2):194–253, 2005. doi: 10.1016/j.artint.2005.04.001. URL `http://dx.doi.org/10.1016/j.artint.2005.04.001`.

[30] James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983. doi: 10.1145/182.358434. URL `http://doi.acm.org/10.1145/182.358434`.

[31] John Bellingham, Arthur Richards, and Jonathan P How. Receding horizon control of autonomous aerial vehicles. In *American Control Conference, 2002. Proceedings of the 2002*, volume 5, pages 3741–3746. IEEE, 2002.

[32] Alfonso Gerevini and Ivan Serina. LPG: A planner based on local search for planning graphs with action costs. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, April 23-27, 2002, Toulouse, France*, pages 13–22, 2002. URL `http://www.aaai.org/Library/AIPS/2002/aips02-002.php`.

[33] Willis Frederick Kern and James R Bland. *Solid Mensuration: With Proofs*. J. Wiley & Sons, Incorporated, 1938.

[34] John L. Bresina and Paul H. Morris. Mixed-initiative planning in space mission operations. *AI Magazine*, 28(2):75–88, 2007. URL `http://www.aaai.org/ojs/index.php/aimagazine/article/view/2041`.

[35] Michael T. Cox and Chen Zhang. Mixed-initiative goal manipulation. *AI Magazine*, 28(2):62–74, 2007. URL `http://www.aaai.org/ojs/index.php/aimagazine/article/view/2040`.

[36] Mary L. Cummings, Jonathan P. How, Andrew Whitten, and Olivier Toupet. The impact of human-automation collaboration in decentralized multiple unmanned vehicle control. *Proceedings of the IEEE*, 100(3):660–671, 2012. doi: 10.1109/JPROC.2011.2174104. URL `http://dx.doi.org/10.1109/JPROC.2011.2174104`.