On-board Planning for Robotic Space Missions using Temporal PDDL

Amanda Coles*, Andrew Coles*, Moises Martinez*, Emre Savas*,
Thomas Keller†, Florian Pommerening† and Malte Helmert†
∗ Department of Informatics, King's College London
† Department of Mathematics and Computer Science, University of Basel

Abstract

The desire for persistent autonomy in ambitious robotic space missions has seen renewed interest in the use of on-board planning. This poses particular challenges in terms of the limited computational power of radiation-hardened hardware, and also the need to integrate with functions external to the planner itself. In this paper, we discuss the development of a forward-chaining temporal planner for use in this setting. We present results illustrating its lightweight resource footprint compared to contemporary planners, and its application to two proposed robotic space scenarios.

1 Introduction

In recent years, a number of ambitious robotic space mission scenarios have been proposed, leading to a renewed interest by national and international agencies in increasing the degree of autonomy within robotic space missions, including planetary exploration and orbital operations. Within this is a desire to develop and consolidate the maturity of technologies for on-board reasoning, including automated planning. Such technologies would support an increased degree of persistent autonomy, with space robots relying on on-board reasoning to support operations over an extended period without intervention from ground-based operations staff.

Autonomous reasoning for space missions is not a new idea (Muscettola et al. 1998; Chien et al. 2010), but reduction in agency budgets has renewed interest in its development. Progress in domain-independent planning over the past decades makes it a viable option to consider for implementing such autonomy for robotic space missions. The opportunity is exciting, but search-based planning and space hardware are not natural bedfellows: modern planning algorithms are often evaluated in terms of the rules and benchmarks of the International Planning Competition (IPC) where a planner is allocated 30 minutes of time on a modern CPU and several gigabytes of memory; but in space, this would exceed the hardware resources available by an order of magnitude or more. Moreover, while portfolio planners have emerged as the forerunners in many IPCs, testing and certifying a single planner as flight-ready would be ambitious; let alone certifying a suite of them.

In this paper, we present a domain-independent temporal planner and its application to two mission scenarios: planetary exploration and orbital operations. These are modeled in a variant of the Planning Domain Definition Language (PDDL), extended only with an interface for external function evaluation, as required for integration with other on-board reasoning components for path planning and robotic arm motion planning. We discuss the engineering and design decisions in the development of the planner, in particular those that support the aim of reducing its computational resource requirements. To evaluate the planner in the planetary exploration mission scenario, we present a summary of our experience of using it on a rover during a field trip to the western Sahara. To evaluate the general low-resource capabilities of the planner, we compare its performance to its closest benchmark, the planner OPTIC (Benton, Coles, and Coles 2012), across a range of standard PDDL planning domains.

2 Background

Reasoning with expressive models that incorporate temporal and numeric constraints is crucial to planning in space based applications. The most popular paradigm for planning in space domains to date and the one that has been most successfully used in practice is timeline-based planning (Chien et al. 2000; Frank and Jonsson 2003; Cesta et al. 2012; Chien et al. 2012). In this setting planning problems comprise a set of timelines, each representing the state of a component of the system. Synchronizations specify constraints between these timelines over transitions between states. Timeline-based planers have many strengths for space applications, in particular their ability to handle externally calculated functions and global temporal constraints, e.g. time windows in which a satellite is visible for communication. The drawback of such systems is that they tend to require the specification of domain-specific heuristics and hence significant engineering to apply the planner to new domains. Scalability also remains a challenge; indeed to this day, much of the planning for space missions is done on the ground, by personnel using these tools, rather than on-board the spacecraft.

Recent decades have seen the development of scalable domain-independent PDDL-based planners capable of handling expressive domains. The advent of PDDL 2.1 (Fox

and Long 2003) for the third IPC saw a flurry of planners supporting numeric (Hoffmann 2003) and temporal (Do and Kambhampati 2003; Gerevini, Saetti, and Serina 2006; Eyerich, Mattmüller, and Röger 2009) models; followed more recently by highly temporally expressive planners including CRIKEY, POPF and OPTIC (Coles et al. 2008; 2010; Benton, Coles, and Coles 2012) and even those capable of handling global temporal interval constraints (Coles et al. 2019). In parallel, these planners have been enhanced with methods that allow to incorporate external functions into the planning process (Dornhege et al. 2009; Hertle et al. 2012; Gregory et al. 2012; Piacentini et al. 2013).

In this work, we show that PDDL-based planning is a viable alternative to timeline-based planning for on-board autonomy that is worth considering. We leverage the expressiveness of temporal PDDL planning alongside the efficiency of the state-of-the-art classical planner Fast Downward (Helmert 2006), which incorporates memory saving features that are crucial for developing a scalable planner that can be deployed on limited on-board hardware. We integrate the resulting planner in a timeline-based system whilst maintaining domain-independence. This has the advantage over a purely timeline-based system that domain-independent heuristics can be re-used and it is easier to apply the system to a new mission scenario. However, the rest of the system maintains the expressiveness of timeline planners. Others have considered the relationship between these two paradigms before, including work bridging the gap between the representation languages (Smith, Frank, and Cushing 2008), comparing the expressiveness of the approaches (Gigante et al. 2016) or using PDDL planners to create timeline plans (Ocon et al. 2017). However, none of these focused on the development of scalable planning systems for on-board use in space missions.

2.1 Problem Definition

Our planner reasons with a restricted class of PDDL 2.2 planning problems. Specifically, we forbid numeric effects which are either continuous over time, non-linear, or depend on the duration of an action. A planning task is a tuple $\langle P, V, A, I, TILs, G\rangle$ where P is a set of propositions and V a set of numeric variables. The *initial state* I maps each propositional variable in P to a truth value and each numeric variable in V to a real number. Each action $a \in A$ is a tuple $\langle pre_{\vdash}, eff_{\vdash}, pre_{\leftrightarrow}, pre_{\dashv}, eff_{\dashv}\rangle$ comprising the sets of *preconditions* and *effects* at the start ($\vdash$) and end ($\dashv$) of the action and a set of *invariant conditions* ($pre_{\leftrightarrow}$) that must hold throughout the execution of the action. An action a is also associated with a *duration constraint* that defines the minimal duration $lb_a \in \mathbb{R}_0^+$ and maximal duration $ub_a \in \mathbb{R}_0^+$, $ub_a \geq lb_a$ it takes to execute the action.

A precondition is a conjunction over propositions $p \in P$ and numeric conditions of the form $w_1v_1 + w_2v_2 + ... + w_nv_n \circ c$, where $v_i \in V$, $\circ \in \{\leq, =, \geq\}$, and $w_i, c \in \mathbb{R}$. A precondition is satisfied in a state s if all contained propositions are true in s and if the inequality of all contained numerical conditions is true when the variables v_i are replaced with their values in s. An action can start at a time t if the state of the world at time t satisfies $pre_{\vdash}$ and it can end at times where $pre_{\dashv}$ is satisfied. It can be executed from t to t' if it can start at t, end at t', and $pre_{\leftrightarrow}$ is satisfied at all times between t and t'.

Effects are either propositions that are added or deleted when an action is applied or numeric expressions that update the values of variables in V of the form: $v_k \circ w_1v_1 + w_2v_2 + ... + w_nv_n + c$ ($v_i \in V, \circ \in \{-=, =, +=\}, w_i, c \in \mathbb{R}$). A propositional effect adding p updates the state by setting p to true, while an effect deleting p sets it to false. A numeric effect decrements, updates, or increments the value of v_k by the value of the expression with the variables v_i replaced by their current value. The effects in $eff_{\vdash}$ update the state at the time the action is started and the effects in $eff_{\dashv}$ update it when the action ends.

The Timed Initial Literals $TILs$ denote time-stamped changes to the world, with each TIL prescribing that a proposition is either added or deleted at a specified time. By referring to TILs in the preconditions of actions, it is possible to constrain the ranges of times during which the respective actions are applicable.

The solution to a planning problem is a *plan*: a time-stamped sequence of actions from A which defines start- and end-time for each action in a way that all actions can be executed, the duration constraints of all actions are satisfied, and the initial state I is transformed into a state that satisfies the set of preconditions G, called the *goal* of the task. We assume that effects take a small time ϵ to be realized in the state, so, for example, an action with a starting precondition added at the end of action a_{pre} can start no earlier than the end time of a_{pre} plus ϵ. Different interpretations are possible and our planner can easily be adapted to them.

3 Project Context and Missions

In this section we summarize the application context in which our planner is being developed and the two mission scenarios that are being targeted. The missions and broader system design were contractually specified, so are outside the scope of our work on the planner itself, but it is helpful to understand the context in which we are operating.

3.1 System Architecture

The architecture in which our planner will be operated is based on T-REX (McGann et al. 2007). T-REX is based on a classic 'sense, deliberate, act' cycle, with a component-based design that distributes functionality between components called *reactors*. Our planner is embedded in what we call the 'task planning' reactor; depending on the mission scenario, there may be other reactors for path planning and navigation, robot arm motion planning, and so on.

The task planning reactor serves as middleware between our planner and the rest of the system. It is responsible for generating the planning problems to be solved, expressed in PDDL; to pass these problems to our planner, along with a domain model for the mission; and to try and execute the generated solution plans. Executing a plan might fail in the real world if the PDDL model is inaccurate, for example if the cost for moving between two waypoints turns out to be higher than estimated. In that case, the task planning reactor will start a re-planning procedure, which we describe in

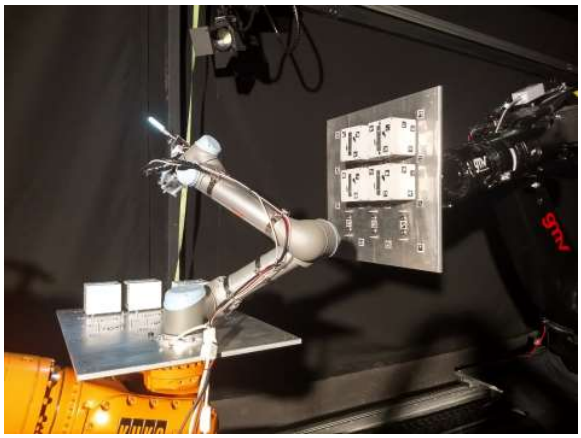

Figure 1: Orbital test case

more detail in Section 5.2. As T-REX is a timeline-based system, the task planning reactor translates solution plans into a timeline-based representation, to support plan dispatch and execution monitoring; this is done following the approach of Ocon et al. (2017).

The system architecture has two important consequences for what is expected of the planner. First, as the PDDL is generated by the middleware, its exact specification is outside of our control, so we have less flexibility than other applications of planning, where one might expect to be able to engineer both the model and the planner itself. This is to ensure that there is a correspondence between the PDDL model and the timeline representation used throughout the rest of T-REX. Moreover, the domain file must remain the same for all problems to be solved, so PDDL modeling techniques that rely on tailoring the domain file on a per-problem basis are not available.

Second, to find a plan, the planner needs estimates of the time and resource needs of the actions in the plan. As these may be determined by the other reactors (for instance, the path planning reactor estimating the time to complete a traversal), the planner needs to be able to use external information. The planning reactor could compile this information into the domain but we instead developed an interface for external functions that gives the planning algorithm control over when other reactors are queried. This makes interesting future extensions possible (see Section 4.5).

3.2 Orbital Mission Scenario

The first mission we consider is an evaluation of the potential for orbital maintenance operations to be performed by autonomous robotic systems. Initially, we consider a robot arm built onto a satellite or other orbital platform that can rearrange modules connected to the platform. In a terrestrial setting, this acts as an alternative to astronauts performing extra-vehicular activities for time-consuming and repetitive tasks, e.g., if the modules are quite heavy and need to be placed with a high degree of precision. In orbit around other celestial bodies, the aim is to increase mission flexibility and service life. The test platform we used is shown in Figure 1.

The initial state in this planning problem is a configuration of a modular satellite. The goal is a desired configuration that either configures the satellite for a new operational mode, or replaces damaged modules. The actions allow the arm to localize, grasp, or place modules. External functions provided by a different reactor estimate the action durations considering the necessary changes of the arm's pose and location. Due to the high degree of precision needed, re-planning based on updated information may be required, though this is handled outside the planner (see Section 5.2).

3.3 Planetary Mission Scenario

The next mission we consider explores the potential for increased autonomy of a planetary rover. We consider a rover equipped with a camera to identify interesting geological targets and a robotic arm with scientific instruments. There are three relevant reactors that handle the following tasks:

- path planning and navigation, using satellite terrain maps supplemented by information gathered from the camera
- motion planning of the robotic arm
- processing images from the camera

The last reactor motivates increased autonomy: identifying an interesting geological target generates an additional science-gathering goal that the planner should consider. The planner reactor uses a simple strategy for new goals: plan execution is paused, and an updated planning problem is passed to the planner; if the problem can be solved, the new plan starts executing, otherwise the old plan is resumed.

The initial state in this scenario is the current status of the rover. Timed Initial Literals (Hoffmann and Edelkamp 2005) constrain the timing of communication windows and science-gathering tasks. The goal describes which science-gathering tasks should be done and what data communicated. Actions model the rover movement, scientific operations, and data transmission. External functions provided by the other reactors estimate their durations.

4 The Stellar Planner

The Stellar planner is a heuristic search temporal planner based on Fast Downward. It has an interface for external functions, handles temporal constraints similar to OPTIC but consumes less memory by not using state annotations.

4.1 Search

The source code of Fast Downward provides Stellar with the basic search procedure. Fast Downward is a tightly implemented codebase, used as the basis for a number of classical planners. *Inter alia*, it minimises the memory used to store each propositional state by using a bit-packed encoding – allocating to each finite-domain state variable the number of bits needed to store it. This makes it a good starting point for our work, the two main limitations being that it supports neither numeric state variables nor temporal planning semantics. Temporal Fast Downward (Eyerich, Mattmüller, and Röger 2009) is a fork of Fast Downward that extended the code to support both of these, but preceded a decade of

intensive development on the Fast Downward code. Hence, we started by developing a hybrid of the two.

First, Fast Downward was extended to reason with snap-actions, where the start- and end-point of each PDDL durative action are considered to be separate instantaneous actions. To respect the temporal semantics, planning with these snap-actions additionally required modifications to the search algorithm such that:

- an 'end' snap action can only be applied if the action has started but not yet ended;
- in the goal state, all actions must have ended; and
- after a snap-action has been applied, the active invariant conditions of all currently executing actions are still true.

We used a technique from OPTIC called *compression safety* (Coles et al. 2009) to reduce the size of the search space. It recognises where it is completeness-preserving to apply the end of an action as soon as it has started.

4.2 Heuristic

The search algorithm is guided by a simple heuristic based on the successful FF heuristic (Hoffmann and Nebel 2001). We relax the task for the heuristic computation by ignoring its temporal and numerical aspects. To relax the temporal aspects, we convert each snap action into a classical action in the following way. For a temporal action $a = \langle pre_{\vdash}, eff_{\vdash}, pre_{\leftrightarrow}, pre_{\dashv}, eff_{\dashv} \rangle$, we introduce two classical actions: the *start action* $a_{\vdash} = \langle pre_{a_{\vdash}}, eff_{a_{\vdash}} \rangle$ has a cost that corresponds to the minimal duration of a, its preconditions $pre_{a_{\vdash}}$ are the propositional conditions in $pre_{\vdash}$ and those propositional conditions in $pre_{\leftrightarrow}$ that are not added by $eff_{\vdash}$, and its effects $eff_{a_{\vdash}}$ are the propositional effects of $eff_{\vdash}$; the *end action* $a_{\dashv} = \langle pre_{a_{\dashv}}, eff_{a_{\dashv}} \rangle$ has a cost of 0 and it preconditions and effects are the propositional preconditions and effects of $pre_{\dashv}$ and $eff_{\dashv}$. We create an additional propositional variable r_a for each temporal action a to represent if a is currently running in a state (i.e., its start snap action has been added to the plan but its end snap action has not). We add r_a to the effect $eff_{a_{\vdash}}$ of the start action of a and to the precondition $pre_{a_{\dashv}}$ of the end action of a. Intuitively, we should also delete the variable in $eff_{a_{\dashv}}$ but this is not necessary as the FF heuristic is based on a delete relaxation that ignores such effects anyway. These propositional start- and end-actions follow a similar idea as the snap actions, but relax the numerical aspects, do not rely on an STN, and use additional propositions to connect the start to the end of an action.

When evaluating the heuristic, we first reconstruct the set of temporal actions that are currently running and add the corresponding facts r_a to the initial state. We then compute the FF heuristic on the projected task. The heuristic computes a relaxed plan for the classical task that consists of start and end actions. We extend this plan by adding the end actions of all started actions or running action where the plan does not contain a corresponding end action. We also use *preferred operators*, a technique that prefers actions from the relaxed plan in the search and had a strong impact on performance in the classical setting.

The heuristic is relatively naïve and ignores central aspects of the temporal task. It could be extended to also consider numerical aspects, either only in the monotonic case (Hoffmann 2003) or more generally with an interval relaxation (Aldinger, Mattmüller, and Göbelbecker 2015). However, we found that this simple heuristic was already sufficient for our mission scenarios and showed good performance on IPC benchmarks (see Section 6).

4.3 Efficient Temporal Constraint Management

In Stellar, we build upon the temporal constraint management approach of POPF and OPTIC, making a distinction between logical and temporal consistency. The search space, as described in Section 4.1, comprises states, each reached by a logically correct plan (i.e., all preconditions are satisfied). At each state, further steps are taken to ensure the plan is temporally correct. The following rules are used to incrementally define temporal constraints on the start- and end-times of actions as states are expanded, i.e. when a snap action is added to the end of a partial plan:

- For each of the snap action's logical or numeric preconditions on a state variable v, it is ordered after the most recent modifier of the value of v in the plan. This ensures the value of v needed for the precondition is supported.
- For each of the snap action's effects on a state variable v, it is ordered after any earlier steps in the plan that have a precondition/effect on v; and it is ordered after what was hitherto the most recent effect on v. This ensures the snap action cannot threaten earlier preconditions on v, and that all effects on v are totally ordered, giving v a known value.
- When an end snap action is added to the plan, the time between its corresponding start and this end must obey the action's duration constraint.

The former two of these yield simple ordering constraints $t_i + \epsilon \leq t_j$. The latter yields a constraint of the form $lb_a \leq t_{end} - t_{start} \leq ub_a$. A sequence of snap actions is called *temporally consistent* if it can be scheduled in a way that satisfies all temporal constraints. Since the constraints depend on the path on which a state is reached, duplicate states reached by different paths cannot be eliminated.

In POPF and OPTIC, the ordering constraints were determined by referring to *annotations* on each state variable v, recording the index of the step to last have an effect on v, and the indices of any steps since then to have a precondition on v. Whilst these annotations support the efficient look-up of which steps the new snap-action needs to be ordered after, they need to be stored in each state, increasing memory usage. Thus, in Stellar, we used a different mechanism: it is possible to recreate these annotations by inspecting the preconditions and effects of the snap-actions by iterating *backwards* through the plan reaching the state being expanded.

Without annotations, the only data that needs to be recorded in each state are the constraints themselves. These take the form of a Simple Temporal Network (Dechter, Meiri, and Pearl 1991). As an implementation detail, we noted that the majority of the temporal constraints were simple ordering constraints (separating pairs of snap-actions by

0 or ϵ) rather than duration constraints: actions tend to have multiple preconditions and effects (hence multiple ordering constraints), but only a single duration constraint. Hence, we used a compact serialised encoding of the temporal constraints, varying the serialised length of each constraint. For each constraint $lb \leq t_j - t_i \leq ub$, 15 bits were reserved for the step index j, and 15 bits for i[1]. The temporal constraint bounds can be recorded in two bits, optionally followed by one or two *continuation bytes*:

`00` : $lb = 0, ub = \infty$; no continuation bytes are needed.

`01` : $lb = \epsilon, ub = \infty$; no continuation bytes are needed.

`10` : $lb = ub = x$; where x is stored in a continuation byte.

`11` : $lb = x, ub = y$; where x then y are stored in continuation bytes.

At each state, temporal consistency of the plan to the state is then checked by unpacking this serialised encoding of the constraints, and running a conventional STN consistency check (Dechter, Meiri, and Pearl 1991).

4.4 Time Window Constraints

To encode time windows and deadlines on which activities can be performed, our domain models use Timed Initial Literals (TIL). A TIL can be thought of as an action effect that happens in any plan at a specified time. A naïve implementation of these is to take the sequence of defined TILs $TIL_0..TILn$ as being dummy snap-actions, with TIL_i being applicable in a state if each of $[TIL_0..TIL_i)$ has been applied in the plan to reach that state; and if plan step a denotes TIL_i, its time t_a must be exactly the time at which TIL_i occurs.

For TILs that denote a one-off time window, or deadlines, static analysis can recognise specific patterns of TILs and translate these into global lower- and upper-bounds on the times at which snap actions can occur (Tierney et al. 2012). In short, if a fact $v = k$ becomes true exactly once at time t, and false exactly once at time t', and no actions have effects on v, then any snap-action with a precondition $v = k$ must occur between t and t'. As in OPTIC, if TILs obey this pattern, their dummy snap-actions can be removed from search, so long as the temporal constraints of actions added to the plan are updated to reflect their bounds t, t' due to TILs. Unlike OPTIC, to conserve memory, we do not explicitly record these temporal constraints, and instead reconstruct them only at the point of temporal consistency checking, by iterating through the steps in the plan.

4.5 External Function Interface

Some important quantities are computed by other reactors in the overall architecture. For example, traversal costs between waypoints is estimated by a path planning reactor. We introduced an interface for external functions that allows values in the PDDL model to be defined as the result of an external function. We view external functions as functions that map PDDL objects of a given type to real numbers. These functions can then be used in the PDDL model in places where constants could otherwise be used. One option to deal with the dependency on other reactors for the values of these functions would be to make the planning reactor query the other reactors for all function values before building the PDDL model and compile the constants into the model. We decided to extend the PDDL language with an interface for such functions instead because it opens up possibilities to optimize function calls from within the search algorithm of the planner. So far, we evaluate all external functions when loading them but an on-demand evaluation would not be difficult to add.

[1]In theory, this restricts us to plans of at most 32,767 steps. In practice, if we had enough memory to store a search space that can reach a plan of this length, we would not be so concerned with memory efficiency.

```
(:modules (:module ef_orbitalrarmplanner
(:function (ra_move_dur ?from ?to - slot))
(:function (ra_move_energy ?to - slot))
))
```

Figure 2: Example for defining a module with two external functions

We follow the approach of Dornhege et al. (2009) who allow the definition of external solvers as modules. On the technical side, we added an optional section `:modules` to the PDDL model that can contain any number of modules, each consisting of a name and a list of parametrized functions (see Figure 2). When loading the PDDL file Stellar looks for a dynamic library with the name of the module and imports all functions from it. The dynamic library can be implemented independently of the planning algorithm and make calls to other reactors. After loading the functions, the planner evaluates all of them on all possible parameter choices and stores the result in a numeric variable that can be used transparently in the planning algorithms. As the parameters are typed and the types only contain a small amount of objects, the overhead of evaluating the functions for all parameter choices is manageable in our case.

We considered some interesting extensions to this idea but did not implement them so far: we could (i) consider functions that estimate action durations at different levels of accuracy and use a deferred evaluation search to evaluate expensive high-accuracy estimations only when needed; (ii) add functions that depend on values in the current state and that are re-evaluated in every state; (iii) investigate how to include a black-box function value in the heuristics; or (iv) extend our approach with planning modulo theories (Gregory et al. 2012) where new types like sets or vectors can be added as modules and functions operating over those types can then be used in actions.

5 The Stellar Reactor: Integration with the Executive

As noted in Section 3.1, the Stellar planner is used with an architecture based on T-REX, within the task planning reactor. Wrapper code in the reactor allows the planner itself to work in a conventional 'PDDL in, solution plan out' manner, by abstracting the details needed for system integration.

5.1 PDDL Generation

At an architectural level, no distinction is made between PDDL generation for planning (new goals have been sent) and re-planning (an unexpected scenario was discovered, so a new plan is needed). In both cases, the wrapper needs to produce an initial state and a goal state. The assumption is made that planning occurs from an 'inactive' initial state, i.e. mission operations are paused while planning takes place. To this end, each system component has one or more 'inactive' states, in which operations can be paused; and zero or more 'active' states, which can arise during plan execution. For instance, in the planetary mission scenario:

- Navigation is either *idle* at a location (inactive); or *going to* a location (active).
- The camera is either *idle* (inactive); or in states corresponding to taking an image (active).
- The robot arm is either *idle* in some pose (inactive); or *going to* another pose (active).
- Each science gathering goal has two states, both of which are inactive: the goal is not true, or the goal is true.

In the case where planning is triggered while components are in an active state, they fall back to an inactive state: the rover stops moving, the arm stops moving, the camera is deactivated; and the state of science-gathering goals persists.

The active/inactive definitions for each domain are defined as metadata outside the PDDL, allowing this approach to be reusable across different application domains.

5.2 Plan Execution Monitoring

The plans produced by the planner have straight-forward execution semantics in terms of PDDL durative actions: at the start of a durative action, the reactor commands the executive to begin the respective action; and the reactor then waits to be notified that the action has completed. The caveat is that these plans, as per the conventions of PDDL, give each action a fixed start time and duration, and any variation from the (estimated) durations used at planning-time will result in the plan being ostensibly invalid.

To manage this source of variability, due to the underlying uncertainty in actions' durations (for instance, rover traversal time is highly unpredictable) the reactor post-processes each plan into a *flexible* plan. As per Section 4.3, each plan produced has an associated Simple Temporal Network. This STN tracks the ordering dependencies between actions, their estimated durations, and any time window constraints. Ordinarily, the planner will schedule each action at its earliest time, with its duration set to the minimum time in the STN between its start and its end. However, this STN can be seen as a *template* for a space of possible plans: any assignment of timestamps to the actions in the STN that obey its constraints is a valid plan. Hence, to retain flexibility, instead the earliest *and latest* time at which each action can start is retained, giving a valid start time window $[lb, ub]$. Then, when executing the plan:

- If at time t the predecessors of an action have finished, and $t \leq lb$ the action executes at time lb, as planned;
- If at time t the predecessors of an action have finished, and $t \in (lb, ub]$ the action executes immediately – the action has been delayed, but this does not violate the constraints in the STN;
- If at time $t > ub$ the predecessors of an action have not yet finished, the plan is temporally invalid – delaying it to time t or later would violate the constraints in the STN.

In the latter case, replanning is triggered – it may be possible to find an alternative solution plan, placing actions in other time windows.

5.3 Opportunistic Science Goals and Replanning

In the planetary mission scenario, the rover is equipped with a panoramic camera. An on-board image analysis component periodically monitors the camera images, using an image classifier to identify novel geological targets. This functionality is embedded into a reactor that sends additional opportunistic science goals to the planning reactor.

In the reactor, when an opportunistic science goal is received, replanning is triggered. As discussed in Section 5.1 the current plan execution is paused; a new PDDL problem is then generated containing the existing goals and this additional goal, and the planner is invoked. In the ideal case, a new plan can be found, and execution re-starts. If a new plan cannot be found (either provably, or empirically if the planner has not found a solution after a minute), the opportunistic science goal is dropped, and replanning is triggered again, generating PDDL without the new opportunistic goal.

One remaining consideration is how to treat opportunistic science goals when replanning in general. If replanning fails to find a solution plan, the most recent opportunistic goal is discarded, and replanning attempted again. The motivation for this is whilst the opportunistic goals are nice to have, they are obvious candidates to discard if they would otherwise compromise achieving the hard goals for the day specified by ground control staff.

6 Evaluation

Our evaluation comprises two parts. First we discuss the use of our planner running as part of the complete system, executing on a rover in field trials in the Western Sahara. Second, we consider a theoretical evaluation of the planner on traditional planning benchmarks to demonstrate its efficiency in terms of memory usage and performance compared to existing PDDL planners with similar expressivity.

6.1 Rover Field Trials

To test the capabilities of planning in the planetary scenario, along with other technologies developed within our strategic research cluster, a four-week field trip was undertaken in the western Sahara, near Erfoud in Morocco.[2] This location was chosen due to its similarity to Martian terrain. The SherpaTT rover (Cordes and Babu 2016) was used for the experiments, providing a flexible four-wheeled rover platform. A number of scenarios were tested and validated as part of this trip:

[2]See `https://youtu.be/-zqve9baOzM` for an overview of the whole field trip.

Figure 3: SherpaTT Rover in the Western Sahara

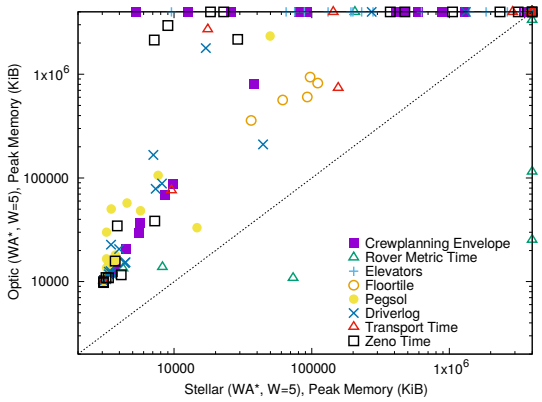

Figure 4: Peak memory usage on IPC benchmarks

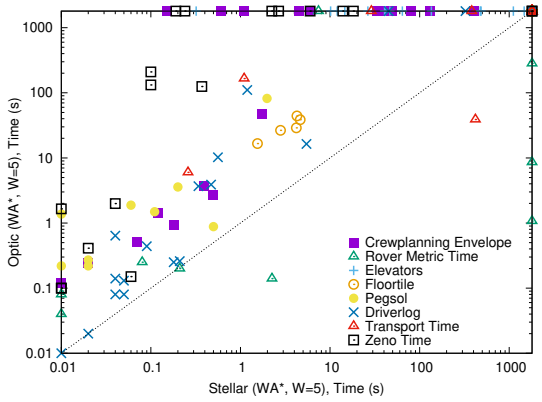

Figure 5: Time taken to solve IPC benchmarks

- Three goals were uploaded through the ground control interface: to take a photo, pick and drop a sample in three different locations. Planning took less than one second, and all three goals were successfully achieved.
- Three goals were uploaded including an optional goal to take an image with a narrow time window of opportunity. The latter was blocked by obstacles not known at planning time. When too much time had passed attempting to reach the imaging location replanning was triggered correctly. No new plan existed, so the goal was dropped, and replanning found a way to achieve the remaining goals.
- Opportunistic science goals were successfully generated, and replanning incorporated additional actions into the plan to achieve these. When too many soft goals were generated and/or terrain difficulties were encountered, these soft goals were successfully dropped.

The planner was invoked over 400 times over the duration of the trip. It never failed to solve a problem that had a solution and never took more than two seconds to do so.

The main limitation of the planner observed in the field is that the order in which locations were visited was not necessarily optimal, because the first plan found for each given problem was executed regardless of quality. This was a deliberate trade-off: finding a plan takes time, especially on modest hardware; and PDDL planners typically need a known, fixed initial state from which to plan. As future work, we will adapt recent techniques for situated replanning, where the planner can search for a new plan in parallel to the execution of an existing one; and preempt it when a new better plan is found (Cashmore et al. 2019).

6.2 Efficiency Compared to OPTIC

The field trials showed that, broadly, Stellar is sufficient for our mission purposes. We now perform a more detailed comparison to investigate our claims that the techniques deployed in Stellar reduce computational overheads compared to existing PDDL planners. For these experiments we use Intel Core i5-4690 3.50 GHz machines, with time and memory limits of 30 minutes and 4 GiB.

We were unable to compare to any planner on the domains for our orbital and planetary scenarios as no existing planner supports all required features (time windows and external functions). Instead, for these experiments we use a diverse range of temporal PDDL problems from previous International Planning Competitions (IPC) and the existing literature. We selected OPTIC as the best representative PDDL planner for comparison to Stellar as it is the most closely comparable in terms of supported features: OPTIC is the latest implementation in the family of planners that includes CRIKEY3, Colin and POPF: it has support for temporal time-windows and uses a STN scheduler similar to the one in Stellar in domains without continuous numeric effects.

Figures 4 and 5 show a comparison of peak memory usage and time taken to solve problems in our evaluation domains respectively for Stellar and OPTIC. Points above the line indicate that Stellar's performance was better (lower memory use/shorter solution time); those on the axes indicate that the planner failed to find a solution within the given time/memory limit. It is clear that Stellar is able to solve problems with a much lower memory consumption, often an order of magnitude. On most of the problems solved by Stellar but not by OPTIC the reason for OPTIC's failure was that it exceeded the available memory.

Whilst Stellar is efficient and lightweight, it can reach a solution within the given time limits to most problem in-

stances (including the ones that are highly challenging). Interestingly, of the 5 problems OPTIC did manage to solve more quickly, most were in the Rovers Metric Time domain; note that this is the IPC benchmark domain and not the planetary rover scenario for which Stellar was developed. The reason for this discrepancy in performance is that OPTIC makes use of numeric dominance constraints to prune states. That is, if the planner is in the same propositional state, but the value of a numeric fluent is different: e.g. when the communicate action is applied twice, the rover remains in the same propositional state, but has less battery remaining, then this state is not interestingly different because bigger values of battery charge are always better. The ability to recognise this boosts OPTIC's performance in this domain. This feature could be ported to Stellar, but it was not implemented in the project as it was not needed for the scenarios covered. In other domains Stellar generally solved problems more quickly than OPTIC, demonstrating that its tightly optimised code based on the Fast Downward planning architecture allows for efficient planning.

7 Conclusions

In this paper we discuss the development of a PDDL planner aimed at supporting on-board planning for robotic space missions. The planner was found to be reliable and capable when used for on-board planning for a rover during field trials; and empirically, has been shown to be efficiently implemented, in terms of time and memory requirements.

We continue to develop the ideas in this paper, with a focus on plan quality and supporting mixed-initiative planning with the role of the planner being to find minimal diversions to accommodate opportunistic science goals.

Acknowledgements

This work has received funding from the European Union's Horizon 2020 Research and Innovation programme (Grant Agreement 730086, ERGO; and Grant Agreement 821988, ADE).

References

Aldinger, J.; Mattmüller, R.; and Göbelbecker, M. 2015. Complexity of interval relaxed numeric planning. In *KI 2015: Advances in Artificial Intelligence (KI)*.

Benton, J.; Coles, A. J.; and Coles, A. I. 2012. Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 2–10.

Cashmore, M.; Coles, A. I.; Cserna, B.; Karpas, E.; Magazzeni, D.; and Ruml, W. 2019. Replanning for situated robots. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*. Accepted for publication.

Cesta, A.; Fratini, S.; Orlandini, A.; and Rasconi, R. 2012. Continuous planning and execution with timelines. In *Proceedings of the Eleventh International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS 2012)*.

Chien, S.; Rabideau, G. R.; Knight, R. L.; Sherwood, R.; Engelhardt, B.; Mutz, D.; Estlin, T. A.; Smith, B.; Fisher, F. W.; Barrett, A. C.; Stebbins, G.; and Tran, D. 2000. ASPEN – automated planning and scheduling for space mission operations. In *Proceedings of the International Conference on Space Operations (SpaceOps 2000)*.

Chien, S. A.; Tran, D.; Rabideau, G.; Schaffer, S. R.; Mandl, D.; and Frye, S. 2010. Timeline-based space operations scheduling with external constraints. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 34–41.

Chien, S.; Johnston, M.; Frank, J.; Giuliano, M.; Kavelaars, A.; Lenzen, C.; and Policella, N. 2012. A generalized timeline representation, services, and interface for automating space mission operations. In *Proceedings of the International Conference On Space Operations (SpaceOps 2012)*, 1–17.

Coles, A. I.; Fox, M.; Long, D.; and Smith, A. J. 2008. Planning with problems requiring temporal coordination. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, 892–897.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2009. Extending the use of inference in temporal planning as forwards search. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 66–73.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 42–49.

Coles, A. J.; Coles, A. I.; Martinez, M.; Savas, O. E.; Delfa, J. M.; de la Rosa, T.; E-Martín, Y.; and García Olaya, A. 2019. Efficiently reasoning with interval constraints in forward search planning. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019)*.

Cordes, F., and Babu, A. 2016. SherpaTT: A versatile hybrid wheeled-leg rover. In *Proceedings of the Thirteenth International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS 2016)*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Do, M. B., and Kambhampati, S. 2003. SAPA: Multi-objective Heuristic Metric Temporal Planner. *Journal of Artificial Intelligence Research* 20:155–194.

Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2009. Semantic attachments for domain-independent planning systems. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 114–121.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 130–137.

Fox, M., and Long, D. 2003. PDDL2.1: An extension of PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Frank, J. D., and Jonsson, A. K. 2003. Constraint-based attribute and interval planning. *Journal of Constraints Special Issue on Constraints and Planning* 8(4):339–364.

Gerevini, A.; Saetti, A.; and Serina, I. 2006. An Approach to Temporal Planning and Scheduling in Domains with Predictable Exogenous Events. *Journal of Artificial Intelligence Research* 25:187–231.

Gigante, N.; Montanari, A.; Mayer, M. C.; and Orlandini, A. 2016. Timelines are expressive enough to capture action-based temporal planning. In *Proceedings of the Twenty-Third International Sym-*

posium on Temporal Representation and Reasoning (TIME 2016), 100–109.

Gregory, P.; Long, D.; Fox, M.; and Beck, C. 2012. Planning modulo theories: Extending the planning paradigm. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hertle, A.; Dornhege, C.; Keller, T.; and Nebel, B. 2012. Planning with semantic attachments: An object-oriented view. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 402–407.

Hoffmann, J., and Edelkamp, S. 2005. The deterministic part of IPC-4: An overview. *Journal of Artificial Intelligence Research* 24:519–579.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research* 20:291–341.

McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2007. T-REX: A model-based architecture for AUV control. In *Proceedings of the Third Workshop on Planning and Plan Execution for Real-World Systems (PlanEx 2007)*.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence* 103(1–2):5–47.

Ocon, J.; Delfa, J. M.; de la Rosa, T.; García, A.; and Escudero, Y. 2017. GOTCHA: An autonomous controller for the space domain. In *Proceedings of the Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA 2017)*.

Piacentini, C.; Alamisis, V.; Fox, M.; and Long, D. 2013. Combining a temporal planner with an external solver for the power balancing problem in an electricity network. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 398–406.

Smith, D. E.; Frank, J.; and Cushing, W. 2008. The ANML language. In *ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS 2008)*.

Tierney, K.; Coles, A. J.; Coles, A. I.; Kroer, C.; Britt, A.; and Jensen, R. M. 2012. Automated planning for liner shipping fleet repositioning. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 279–287.