Heuristic Planning for PDDL+ Domains

Wiktor Piotrowski,[1] **Maria Fox,**[1] **Derek Long,**[1] **Daniele Magazzeni,**[1] **Fabio Mercorio**[2]
[1]Department of Informatics, King's College London, United Kingdom
[2]Department of Statistics and Quantitative Methods,
CRISP Research Centre, University of Milan-Bicocca, Italy

Abstract

Planning with hybrid domains modelled in PDDL+ has been gaining research interest in the Automated Planning community in recent years. Hybrid domain models capture a more accurate representation of real world problems that involve continuous processes than is possible using discrete systems. However, solving problems represented as PDDL+ domains is very challenging due to the construction of complex system dynamics, including non-linear processes and events. In this paper we introduce DiNo, a new planner capable of tackling complex problems with non-linear system dynamics governing the continuous evolution of states. DiNo is based on the discretise-and-validate approach and uses the novel Staged Relaxed Planning Graph+ (SRPG+) heuristic, which is introduced in this paper. Although several planners have been developed to work with subsets of PDDL+ features, or restricted forms of processes, DiNo is currently the only heuristic planner capable of handling non-linear system dynamics combined with the full PDDL+ feature set.

1 Introduction

Over the years, Automated Planning research has been continuously attempting to solve the most advanced and complex planning problems. The standard modelling language, PDDL [McDermott *et al.*, 1998], has evolved to accommodate new concepts and operations, enabling research to tackle problems which more accurately represent real-world scenarios. PDDL2.1 [Fox and Long, 2003] enabled modelling numeric variables and temporal information in the domains, while PDDL+ [Fox and Long, 2006] introduced exogenous events and continuous processes in the models. PDDL+ is the most accurate standardised way yet, of defining hybrid problems as planning domains.

Planning with PDDL+ has been gaining research interest in the Automated Planning community in recent years. Hybrid domains are models of systems which exhibit both continuous and discrete behaviour. They are amongst the most advanced models of systems and the resulting problems are notoriously difficult for planners to cope with due to non-linear behaviour and immense search spaces.

UPMurphi [Della Penna *et al.*, 2009], based on the Discretise & Validate approach, is the only planner able to handle the full range of PDDL+. However, the main drawback of UPMurphi is the lack of heuristics, which critically limits its scalability. In this paper, we fill the gap, and introduce DiNo, a new planner for PDDL+ problems with mixed discrete-continuous non-linear dynamics. DiNo is built on UPMurphi. It uses the planning-as-model-checking paradigm [Cimatti *et al.*, 1997; Bogomolov *et al.*, 2014], and relies on the Discretise & Validate approach [Della Penna *et al.*, 2009] to handle continuous change and non-linearity.

DiNo uses a novel relaxation-based domain-independent heuristic for PDDL+, Staged Relaxed Planning Graph+ (SRPG+). The heuristic guides the Enforced Hill-Climbing algorithm [Hoffmann and Nebel, 2001]. In DiNo we also exploit the deferred heuristic evaluation [Richter and Westphal, 2010] for completeness (in a discretised search space with a finite horizon). The SPRG+ heuristic which improves on the Temporal Relaxed Planning Graph and extends its functionality to include information gained from PDDL+ features, namely the processes and events.

The domain-independent SRPG+ heuristic makes DiNo the only heuristic planner capable of handling non-linear system dynamics combined with the full PDDL+ feature set.

We begin by discussing the related work done in the area of PDDL+ planning in section 2. We then outline the basis of the Discretise & Validate method on which DiNo is based and the underlying UPMurphi architecture in section 3. In section 4 we describe the SRPG+ heuristic. Section 5 describes the experimental evaluation. Section 6 concludes the paper[1].

2 Related Work

Various techniques and tools have been proposed to deal with hybrid domains [Penberthy and Weld, 1994; McDermott, 2003; Li and Williams, 2008; Coles *et al.*, 2012; Shin and Davis, 2005]. Nevertheless, none of these approaches are able to handle the full set of PDDL+ features, i.e. non-linear domains with processes and events. More recent approaches in this direction have been proposed by [Bogomolov *et al.*, 2014], where the close relationship between hybrid planning domains and hybrid automata is explored. [Bryce

[1]Research leading to these results has received funding from the European Commission under contract No. FP7-610532-SQUIRREL

et al., 2015] use dReach with a SMT solver to handle hybrid domains. However, dReach does not use PDDL+, and cannot handle exogenous events.

On the other hand, many works have been proposed in the model checking and control communities to handle hybrid systems. Some examples include [Cimatti *et al.*, 2015; Cavada *et al.*, 2014; Tabuada *et al.*, 2002; Maly *et al.*, 2013], sampling-based planners [Karaman *et al.*, 2011; Lahijanian *et al.*, 2014]. Another related direction is *falsification* of hybrid systems (i.e., guiding the search towards the error states, that can be easily cast as a planning problem) [Plaku *et al.*, 2013; Cimatti *et al.*, 1997]. However, while all these works aim to address a similar problem, they cannot reason with PDDL+ domains. Some recent works [Bogomolov *et al.*, 2014; 2015] are trying to define a formal translation between PDDL+ and standard hybrid automata, but so far only an over-approximation has been defined, that only allows proving plan non-existence. PDDL+ also presents important features, including Timed Initial Literals/Fluents, no-moving target rule, epsilon separation of actions. Currently, no control-based approaches can handle PDDL+ models.

To date, the only viable approach in this direction is PDDL+ planning via *discretisation*. UPMurphi [Della Penna *et al.*, 2012], which implements the Discretise & Validate approach, is able to deal with the full range of PDDL+ features. UPMurphi's main drawback is the lack of heuristics which strongly limits its scalability. However, UPMurphi was successfully used in the multiple-battery management domain [Fox *et al.*, 2012], for urban traffic control [Vallati *et al.*, 2016], and for unit commitment problem [Piacentini *et al.*, 2016] . In all cases, a domain-*specific* heuristic was used.

3 Discretise & Validate Approach

As a successor to UPMurphi, DiNo relies on the Discretise & Validate approach [Della Penna *et al.*, 2012] which approximates the continuous dynamics of systems in a discretised model with uniform time steps and step functions. Using a discretised model and a finite-time horizon ensures a finite number of states in the search for a solution. Solutions to the discretised problem are validated against the original continuous model using VAL [Howey *et al.*, 2004]. If the plan at a certain discretisation is not valid, the discretisation can be refined and the process iterates. An outline of the Discretise & Validate process is shown in Figure 1.

In order to plan in the discretised setting, PDDL+ models are translated into *finite state temporal systems*, as formally described in the following[2].

Definition 1. ***State***. *Let P be a finite set of propositions and $V = \{v_1, ..., v_n\}$ a set of real variables. A state s is a triple $s = (p(s), v(s), t(s))$, where $p(s) \subseteq P$, $v(s) = (v_1(s), ..., v_n(s)) \in \mathbb{R}^n$ is a vector of real numbers, and $t(s)$ the value of the temporal clock in state s. We also denote with $v_i(s)$ the value of variable at the i-th position in $v(s)$.*

Note that real variables and temporal clock are discretised, according to the Discretise & Validate approach.

[2]Our notation was inspired by Metric-FF [Hoffmann, 2003].

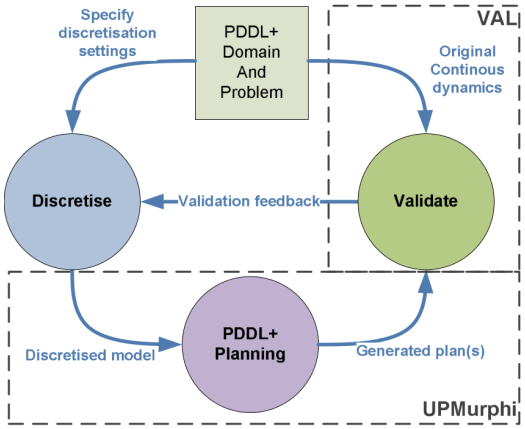


Figure 1: The Discretise & Validate process

Definition 2. ***Δ−Action***. *A Δ-action updates the state during the search. It can be of three types: an instantaneous PDDL action, a* snap action *[Long and Fox, 2003], or a time-passing action, tp.*

The effect of instantaneous actions and snap actions, is to update the state variables in the state resulting from their application, and to start/end a durative action, respectively.

The time-passing action implements the step function used to discretise the system dynamics, its effects are: (i) updating all numeric state variables affected by the combined effect of all processes that are active at the time of application, (ii) updating all state variables affected by events, and (iii) advancing time by Δt.

Definition 3. ***Finite State Temporal System (FSTS)***. *Let a Finite State Temporal System $\mathcal{S}$ be a tuple $(S, s_0, \Delta\mathcal{A}, \mathcal{D}, F, T)$ where S is a finite set of states, $s_0 \in S$ the initial state, $\Delta\mathcal{A}$ is a finite set of Δ-actions and $\mathcal{D} = \{0, \Delta t\}$ where Δt is the discretised time step. $F : S \times \Delta\mathcal{A} \times \mathcal{D} \to S$ is the transition function, i.e. $F(s, \Delta a, d) = s'$ iff applying Δ-action Δa with a duration $d \in \mathcal{D}$ to a state s yields a new reachable state s'. T is the finite temporal horizon.*

Note that d can be 0 to allow for concurrent plans and instantaneous actions. In fact, d will equal Δt only in the case of the tp action. The finite temporal horizon T makes the set of discretised states S finite.

Definition 4. ***Trajectory***. *A trajectory, π, in an FSTS $\mathcal{S} = (S, s_0, \Delta\mathcal{A}, \mathcal{D}, F)$ is a sequence of states, Δ-actions and durations ending with a state, i.e. $\pi = s_0, \Delta a_0, d_0, s_1, \Delta a_1, d_1, ..., s_n$ where $\forall i \geq 0, s_i \in S$ is a state, $\Delta a_i \in \Delta\mathcal{A}$ is a Δ-action and $d_i \in \mathcal{D}$ is a duration. At each step i, the transition function F yields the subsequent state: $F(s_i, \Delta a_i, d_i) = s_{i+1}$.*

Given a trajectory π, we use $\pi_s(k), \pi_a(k), \pi_d(k)$ to denote the state, Δ-action and duration at step k, respectively. The length of the trajectory based on the number of actions it contains is denoted by $|\pi|$ and the duration of the trajectory is denoted as $\tilde{\pi} = \sum_{i=0}^{|\pi|-1} \pi_d(i)$ or, simply, as $\tilde{\pi} = t(\pi_s(n))$

Following from Definition 1, each state s contains the temporal clock t, and $t(s)$ counts the time elapsed in the

current trajectory from the initial state to s. Furthermore, $\forall s_i, s_j \in S : F(s_i, \Delta a, d) = s_j,\ t(s_j) = t(s_i) + d$. Clearly, for all states s, $t(s) \leq T$.

Definition 5. ***Planning Problem***. *In terms of a FSTS, a planning problem $\mathcal{P}$ is defined as a tuple $\mathcal{P} = ((S, s_0, \Delta\mathcal{A}, \mathcal{D}, F, T), G)$ where $G \subseteq S$ is a finite set of goal states. A solution to $\mathcal{P}$ is a trajectory π^* where $|\pi^*| = n, \tilde{\pi} \leq T, \pi^*_s(0) = s_0$ and $\pi^*_s(n) \in G$.*

3.1 Handling the PDDL+ Semantics through Discretisation

In the following we show how FSTS are used to handle the PDDL+ semantics, and describe how this approach has been first implemented in UPMurphi[3].

Time and Domain Variable Discretisation. UPMurphi discretises hybrid domains using discrete uniform time steps and corresponding step functions. The discretisations for the continuous time and the continuous variables are set by the user. Timed Initial Literals and Fluents are variables whose value changes at a predefined time point [Edelkamp and Hoffmann, 2004]. UPMurphi can handle Timed Initial Literals and numeric Timed Initial Fluents to the extent that the discretisation used is fine enough to capture the happenings of TILs and TIFs. On the other hand, the time granularity of TILs and TIFs can be used as a guidance for choosing the initial time discretisation.

Actions and Durative-Actions. Actions are instantaneous, while durative-actions are handled following the start-process-stop model introduced by [Fox and Long, 2006]. A durative-action is translated into: (i) two snap actions that apply the discrete effects *at start* and *at end* of the action; (ii) a process that applies the continuous change over the action execution (iii) and an event that checks whether all the *overall* conditions are satisfied in the open interval of the durative-action execution. Following Definition 2, actions in UPMurphi are used to model instantaneous and snap actions, while the special action time-passing tp is used to advance time and handle processes and events.

Processes and Events. UPMurphi uses the tp action to check preconditions of processes and events at each clock-tick, and then apply the effects for each triggered event and active process. Clearly, the processes/events interleaving could easily result in a complex scenario to be executed, as for example an event can initiate a process, or multiple events can be triggered at a single time point. To address this kind of interaction between processes and events, UPMurphi works as follows: first, it applies the continuous changes for each active process and the effects of each triggered event. Second, it assumes that no event can affect the parts of the state relevant to the preconditions of other events, according to the *no moving target* definition provided by [Fox and Long, 2003]. In this way, the execution of events is mutually-exclusive, and their order of execution does not affect the final outcome of the plan. Third, UPMurphi imposes that, at each clock tick, any event can be fired at most once, as specified by [Fox *et al.*, 2005], for avoiding cyclic triggering of events.

[3]UPMurphi can also be used as Universal Planner, where a policy is generated, while we focus here on finding single plans.

4 Staged Relaxed Planning Graph+

This section describes the Staged Relaxed Planning Graph+ (SRPG+) domain-independent heuristic designed for PDDL+ domains and implemented in DiNo.

The SRPG+ heuristic is based on Propositional [Hoffmann and Nebel, 2001], Numeric [Hoffmann, 2003] and Temporal RPGs [Coles *et al.*, 2010; 2012]. Like its predecessors, SRPG+ relaxes the original problem and ignores the delete effects of actions on propositional facts. Numeric variables are represented as upper and lower bounds which are the theoretical highest and lowest values each variable can take at the given fact layer. Each layer is time-stamped to keep track of its time of occurrence.

The SRPG+, however, extends the capability of its RPG predecessors by tracking processes and events to capture the continuous and discrete evolution of the system.

For clarity, we denote SRPG+ as the heuristic and SRPG as the internal relaxed planning graph structure.

4.1 Building the SRPG

Apart from the inclusion of processes and events, the SRPG+ significantly differs from the Temporal RPG in time-handling. The SRPG contains ***every*** fact layer with the corresponding time clock, and in this sense the RPG is "staged", as the finite set of fact layers are separated by Δt. In contrast, the TRPG takes time constraints into account by time-stamping each fact layer at the earliest possible occurrence of a happening. Only fact layers where values are directly affected by actions are contained in the TRPG.

Definition 6. ***Fact Layer***. *A fact layer $\widehat{s}$ is a tuple $(p^+(\widehat{s}), v^+(\widehat{s}), v^-(\widehat{s}), t(\widehat{s}))$ where $p^+(\widehat{s}) \subseteq P$ is a finite set of true propositions, $v^+(\widehat{s})$ is a vector of real upper-bound variables, $v^-(\widehat{s})$ is a vector of real lower-bound variables, and $t(\widehat{s})$ is the value of the temporal clock.*

Notationally, $v_i^+(s)$ and $v_i^-(s)$ are, respectively, the upper and lower-bound values of the variable at position i in $v(s)$.

In the following we give a formal definition of an SRPG, starting from the problem for which it is constructed.

Definition 7. ***SRPG.*** *Let $\mathcal{P} = (\mathcal{S}, G)$ be a planning problem in the FSTS $\mathcal{S} = (S, s_0, \Delta\mathcal{A}, \mathcal{D}, F, T)$, then a Staged Relaxed Planning Graph $\widehat{\mathcal{S}}$ is a tuple $(\widehat{S}, \widehat{s}_0, \widehat{\Delta\mathcal{A}}, \Delta t, \widehat{F}, T)$ where $\widehat{S}$ is a finite set of fact layers, $\widehat{s}_0$ is the initial fact layer, $\widehat{\Delta\mathcal{A}}$ is a set of relaxed Δ actions, Δt is the time step. $\widehat{F} : \widehat{S} \times 2^{\widehat{\Delta\mathcal{A}}} \times \Delta t \rightarrow \widehat{S}$ is the SRPG transition function. T is the finite temporal horizon.*

The SRPG follows the priority of happenings from VAL, i.e. each new fact layer $\widehat{s}'$ is generated by applying the effects of active processes in $\widehat{s}$, applying the effects of any triggered events and firing all applicable actions, respectively. Note that, as for the FSTS, also in the SRPG the time passing action tp is used for handling processes and events effects and for advancing time by Δt.

Fact layers and relaxed actions in the SRPG are defined as in the standard numeric RPG, except for the fact that each fact layer includes also the temporal clock.

Effects are defined as a tuple $\mathit{eff}(x) = (p^+(\mathit{eff}(x)), p^-(\mathit{eff}(x)), v^+(\mathit{eff}(x)), v^-(\mathit{eff}(x)))$ where

Algorithm 4.1: Building the SRPG

Data: $\mathcal{P} = ((S, s_0, \Delta\mathcal{A}, \mathcal{D}, F, T), G)$;
$\widehat{Proc}$ = Set of processes;
$\widehat{Ev}$ = Set of events;
Result: return a constructed SRPG object if exists

1 $\widehat{s} := s_0$;
2 $\widehat{S} := \{s_0\}$;
3 $\widehat{\mathcal{S}} := (\widehat{S}, \widehat{s}, \widehat{\Delta\mathcal{A}}, \Delta t, \widehat{F}, T)$;
4 **while** $(\forall g \in G : p^+(g) \not\subseteq p^+(\widehat{s})) \vee (\exists v_i \in v(g) : v_i < v_i^-(\widehat{s}) \vee v_i > v_i^+(\widehat{s}))$ **do**
5 **if** $t(\widehat{s}) > T$ **then**
6 return $fail$;
7 **forall** $\widehat{proc} \in \widehat{Proc}$ **do**
8 **if** $p(pre(\widehat{proc})) \subseteq p^+(\widehat{s}) \wedge (\forall v_i \in v(pre(\widehat{proc})) : v_i \geq v_i^-(\widehat{s}) \wedge v_i \leq v_i^+(\widehat{s}))$ **then**
9 $\forall v_i \in v^+(\mathit{eff}(\widehat{proc})) : v_i^+(\widehat{s}) := v_i^+(\widehat{s}) + v_i$;
10 $\forall v_i \in v^-(\mathit{eff}(\widehat{proc})) : v_i^-(\widehat{s}) := v_i^-(\widehat{s}) - v_i$;
11 **forall** $\widehat{ev} \in \widehat{Ev}$ **do**
12 **if** $p(pre(\widehat{ev})) \subseteq p^+(\widehat{s}) \wedge (\forall v_i \in v(pre(\widehat{ev})) : v_i \geq v_i^-(\widehat{s}) \wedge v_i \leq v_i^+(\widehat{s}))$ **then**
13 $p^+(\widehat{s}) := p^+(\widehat{s}) \cup p^+(\mathit{eff}(\widehat{ev}))$;
14 $\forall v_i \in v^+(\mathit{eff}(\widehat{ev})) : v_i^+(\widehat{s}) := max(v_i^+(\widehat{s}), v_i)$
 $\forall v_i \in v^-(\mathit{eff}(\widehat{ev})) : v_i^-(\widehat{s}) := min(v_i^-(\widehat{s}), v_i)$
15 $\widehat{s}_c := \widehat{s}$;
16 **forall** $\widehat{a} \in \widehat{\Delta\mathcal{A}}$ **do**
17 **if** $p(pre(\widehat{a})) \subseteq p^+(\widehat{s}_c) \wedge (\forall v_i \in v(pre(\widehat{a})) : v_i \geq v_i^-(\widehat{s}_c) \wedge v_i \leq v_i^+(\widehat{s}_c))$ **then**
18 $p^+(\widehat{s}) := p^+(\widehat{s}) \cup p^+(\mathit{eff}(\widehat{a}))$;
19 $\forall v_i \in v^+(\mathit{eff}(\widehat{a})) : v_i^+(\widehat{s}) := max(v_i^+(\widehat{s}), v_i)$;
20 $\forall v_i \in v^-(\mathit{eff}(\widehat{a})) : v_i^-(\widehat{s}) := min(v_i^-(\widehat{s}), v_i)$;
21 $t(\widehat{s}) := t(\widehat{s}) + \Delta t$;
22 $\widehat{S} := \widehat{S} \cup \widehat{s}$;
23 return $\widehat{\mathcal{S}}$;

$p^+(\mathit{eff}(x)), p^-(\mathit{eff}(x)) \subseteq P$ (add and delete effects respectively), $v^+(\mathit{eff}(x))$ and $v^-(\mathit{eff}(x))$ are effects on numeric values (increasing and decreasing, respectively), and x can be any Δ-action, process, or event. Preconditions are defined analogously: $pre(x) = (p(pre(x)), v(pre(x)))$ where $p(pre(x)) \subseteq P$ is a set of propositions and $v(pre(x))$ is a finite set of numeric constraints. $p_i^+(\mathit{eff}(x)) \in p^+(\mathit{eff}(x))$ is effect on the i-th proposition in $p(s)$, $v_i^+(\mathit{eff}(x))$ and $v_i^-(\mathit{eff}(x))$ are the real values of the i-th increasing and decreasing effects affecting upper bound $v_i^+(s)$ and lower bound $v_i^-(s)$, respectively.

The SRPG transition function $\widehat{F}$ is a relaxation of the original FSTS transition function F, and follows the standard RPG approach: effects deleting any propositions are ignored and the numeric effects only modify the appropriate bounds for each numeric variable. Note that the set $\widehat{\Delta\mathcal{A}}$ of relaxed Δ actions includes the time passing action tp as from Definition 1. Also in the SRPG the tp is responsible for handling processes and events, whose effects are relaxed in the standard way. The construction of SRPG is shown in Algorithm 4.1. The first fact layer consists of the initial state (lines 1-3). Then the SRPG is updated until a goal state is found (line 4) or the time horizon is reached (line 5). In the former case, the SRPG constructed so far is returned, and the relaxed plan is extracted using backwards progression mechanism introduced in [Hoffmann, 2003]. In the latter case, a heuristic value of infinity ($h(s) = \infty$) is assigned to the current state. To construct the next fact layer, first the active processes are considered (lines 7-8) and the relaxed effects are applied to update upper and lower bounds of variables (lines 9-10). The same is then applied for events (lines 11-15) and instantaneous actions (lines 16-20), that can also add new propositions to the current fact layer. The last step is to increment the temporal clock (line 21), and the new fact layer is then added to the SRPG (line 22).

Note that, as a back-up strategy, DiNo reverts to a breadth-first search if the SRPG+ is unable to extract sufficient information from the domain to reach the relaxed goal within the set temporal horizon T (i.e. $h(s) = \infty$ for all states).

4.2 Time Handling in the SRPG

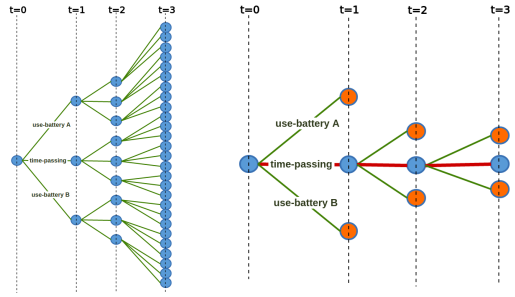
The time-passing action plays an important role as it propagates the search in the discretised timeline. During the normal expansion of the Staged Relaxed Planning Graph, the time-passing is one of the Δ-actions and is applied at each fact layer. Time-passing can be recognised as a helpful action [Hoffmann and Nebel, 2001] when its effects achieve some goal conditions (or intermediate goal facts). However, if, at a time t, no helpful actions are available to the planner, time-passing is assigned highest priority and used as a helpful action. This allows the search to quickly manage states at time t where no happenings of interest are likely to occur.

This is the key innovation with respect to the standard search in the discretised timeline performed, e.g., by UPMurphi. Indeed, the main drawback of UPMurphi is in that it needs to expand the states at each time step, even during the *idle periods*, i.e., when no interesting interactions or effects can happen. Conversely, SRPG+ allows DiNo to identify time-passing as a helpful action during *idle periods* and thus advance time, mitigating the state explosions.

An illustrative example is shown in Figure 2, that compares the branching of the search in UPMurphi and DiNo when planning with a Solar Rover domain. The domain is described in detail in Section 5. Here we highlight that the planner can decide to use two batteries, but the goal can only be achieved thanks to a Timed Initial Literal that is triggered only late in the plan. UPMurphi has no information about the future TIL, therefore it tries to use the batteries at each time step. On the contrary, DiNo recognises the time-passing as a helpful action, and this prunes the state space dramatically.

4.3 Processes and Events in SRPG+

As the SRPG+ heuristic is tailored for PDDL+ domains, it takes into account processes and events. In the SRPG, the



(a) UPMurphi (b) DiNo

Figure 2: Branching of search trees (Blue states are explored, orange are visited. Red edges correspond to helpful actions)

continuous effects of processes are handled in the same manner as durative action effects, i.e. at each action layer, the numeric variables upper and lower bounds are updated based on the time-step functions used in the discretisation to approximate the continuous dynamics of the domain.

Events are checked immediately after processes and their effects are relaxed as for the instantaneous actions. The events can be divided into "good" and "bad" categories. "Good" events aid in finding the goal whereas "bad" events either hinder or completely disallow reaching the goal. Currently, DiNo is agnostic about this distinction. However, as a direct consequence of the SRPG+ behaviour, DiNo exploits good events and ignores the bad ones. Future work will explore the possibility of inferring more information about good and bad events from the domain.

5 Evaluation

In this section we evaluate the performance of DiNo on PDDL+ benchmark domains. Note that the only planner able to deal with the same class of problems is UPMurphi, which is also the most interesting competitor as it can highlight the benefits of the proposed heuristic. For sake of completeness, where possible, we also provide a comparison with other planners able to handle (sub-class of) PDDL+ features, i.e. POPF [Coles *et al.*, 2010] and dReach [Bryce *et al.*, 2015][4].

For our experimental evaluation, we consider two benchmark domains: Generator and Car. In addition, we also consider two more domains that highlight specific aspects of DiNo: Solar Rover that shows how DiNo handles TILs, and Powered Descent that further tests its non-linear capabilities.

Note that to achieve the results a discretisation of $\Delta t = 1.0$ was chosen, except non-linear Generator where some problems required refinement to $\Delta t = 0.5$.

For a fair comparison, all results were achieved by running the competitor planners on a machine with an 8-core Intel Core i7 CPU, 8GB RAM and Ubuntu 14.04 operating system. For more information visit *kcl-planning.github.io/DiNo*

Generator. This domain [Howey and Long, 2003] is well-known across the planning community and has been a test-bed for many planners. The problem revolves around refueling a diesel-powered generator which has to run for a given duration without overflowing or running dry. We evaluate DiNo on both the linear and non-linear versions of the problem. In both variants, we increase the number of tanks available to the planner while decreasing the initial generator fuel level for each subsequent problem.

The non-linear Generator models fuel flow rate using Torricelli's Law which states: *Water in an open tank will flow out through a small hole in the bottom with the velocity it would acquire in falling freely from the water level to the hole*. The fuel level in a refueling tank (V_{fuel}) is calculated by:

$$V_{fuel} = (-kt_r + \sqrt{V_{init}})^2 \qquad t_r \in \left[0, \frac{\sqrt{V_{init}}}{k}\right] \quad (1)$$

V_{init} the initial volume of fuel in the tank, k the fuel flow constant (which depends on gravity, size of the drain hole, and the cross-section of the tank), and t_r is the time of refueling (bounded by the fuel level and the flow constant). The rate of change in the tank's fuel level is modelled by:

$$\frac{dV_{fuel}}{dt} = 2k(kt_r - \sqrt{V_{init}}) \qquad t_r \in \left[0, \frac{\sqrt{V_{init}}}{k}\right] \quad (2)$$

This domain has been previously encoded in PDDL by [Howey and Long, 2003].

The results for the linear Generator problems show that DiNo clearly outperforms its competitors and scales really well on this problem whereas UPMurphi, POPF and dReach all suffer from state space explosion relatively early.

The non-linear version could only be tested on DiNo and UPMurphi as the remaining planners do not support non-linear behaviour. However, the search space proved too large for UPMurphi, it failed to solve any of our test problems. DiNo found solutions to problems using $\Delta t = 1.0$. However, some of the found plans were invalid. Applying the Discretise & Validate approach, we refined the discretisation to $\Delta t = 0.5$ and valid solutions were returned. In both variants of the domain, time horizon was set to $T = 1000$, that is the duration for which the generator is requested to run.

Though dReach is able to reason with non-linear dynamics, their results have been left out of our comparison due to the difficulty with reproducing our domain (written in PDDL+) using the dReach modelling language. The dReach domain and problem descriptions are not standardised and extremely difficult to formulate. Each mode has to be explicitly defined, meaning that the models are excessive in size (i.e. the files for 1, 2, 3 and 4-tank problems are respectively 91, 328, 1350, 5762 lines long). Furthermore, compared to our model, Bryce et al. use a much simplified version of the problem where the generator can never overflow, the refueling action duration is fixed (tanks have no defined capacity), and the flow rate formula is defined as $(0.1 * (tank\_refuel\_time^2))$. Still, in this simplified domain, dReach could only scale up to 3 tanks.

In contrast, our variant of the non-linear Generator problem uses the Torricelli's Law to model the refueling flow rate (2), the refueling actions have inequality-based duration dependent on the tanks' fuel levels (1), and the generator can easily overflow. As a result, our domain is far more complex and further proves our improvement.

[4] We do not consider [Bogomolov *et al.*, 2014] as they only focus on proving plan-non-existence.

	LINEAR GENERATOR				NON-LINEAR GENERATOR		LINEAR SOLAR ROVER		NON-LINEAR SOLAR ROVER		POWERED DESCENT		CAR	
PROBLEM	**DiNo**	**POPF**	**dReach**	**UPMurphi**	**DiNo**	**UPMurphi**	**DiNo**	**UPMurphi**	**DiNo**	**UPMurphi**	**DiNo**	**UPMurphi**	**DiNo**	**UPMurphi**
1	0.34	0.01	2.87	140.50	3.62	X	0.70	203.26	1.10	288.94	0.68	0.18	1.74	0.22
2	0.40	0.01	X	X	0.78	X	0.92	X	2.58	X	1.04	0.74	4.56	0.30
3	0.50	0.05	X	X	2.86	X	1.26	X	4.74	X	1.88	2.98	8.26	0.42
4	0.60	0.41	X	X	59.62	X	1.52	X	7.10	X	3.52	7.18	10.28	0.54
5	0.74	6.25	X	X	1051.84	X	1.80	X	9.58	X	2.88	30.08	14.16	0.66
6	0.88	120.49	X	X	X	X	2.04	X	12.86	X	3.14	126.08	15.78	0.68
7	1.00	X	X	X	X	X	2.28	X	16.48	X	5.26	322.16	17.08	0.72
8	1.16	X	X	X	X	X	2.64	X	21.38	X	3.82	879.52	18.90	0.72
9	1.38	X	X	X	X	X	2.98	X	26.74	X	1.58	974.60	19.30	0.76
10	2.00	X	X	X	X	X	3.30	X	29.90	X	2.26	X	19.50	0.78
11	1.84	X	X	X	N/A	N/A	3.50	X	35.96	X	11.24	X	N/A	N/A
12	2.06	X	X	X	N/A	N/A	3.74	X	42.54	X	42.24	X	N/A	N/A
13	2.32	X	X	X	N/A	N/A	4.00	X	48.06	X	14.90	X	N/A	N/A
14	2.46	X	X	X	N/A	N/A	4.38	X	55.46	X	61.94	X	N/A	N/A
15	2.88	X	X	X	N/A	N/A	5.20	X	62.84	X	19.86	X	N/A	N/A
16	2.94	X	X	X	N/A	N/A	5.40	X	74.50	X	80.28	X	N/A	N/A
17	3.42	X	X	X	N/A	N/A	5.08	X	86.96	X	2.94	X	N/A	N/A
18	3.54	X	X	X	N/A	N/A	5.64	X	95.66	X	2234.88	X	N/A	N/A
19	3.76	X	X	X	N/A	N/A	6.12	X	102.86	X	X	X	N/A	N/A
20	4.26	X	X	X	N/A	N/A	6.02	X	117.48	X	X	X	N/A	N/A

Table 1: Run time in seconds for each problem in our test suite ("X" - planner ran out of memory, "N/A" - problem not tested)

As can be noticed, DiNo scales very well on these problems, and drastically reduces the number of explored states and the time to find a solution compared to UPMurphi.

Solar Rover. We developed the Solar Rover domain to test the limits and potentially overwhelm discretisation-based planners, as finding a solution to this problem relies on a TIL that is triggered only late in the plan.

The task revolves around a planetary rover transmitting data which requires a certain amount of energy. To generate enough energy the rover can choose to use its batteries or gain energy through its solar panels. However, the initial state is at night time and the rover has to wait until daytime to be able to gather enough energy to send the data. The sunshine event is triggered by a TIL at a certain time. The set of problem instances for this domain has the trigger fact become true at an increasingly further time point (50 to 1000 time units).

This problem has also been extended to a non-linear version to further test our planner. Instead of instantaneous increase in rover energy, the TIL triggers a process charging the rover's battery at an exponential rate: $\Delta E = 0.0025E^2$.

For both variants of the domain, the time horizon is set depending on the time point at which the sunexposure TIL is triggered (as defined in the problems).

DiNo can easily handle this domain and solve all test problems. UPMurphi struggles and can only solve the smallest problem instance of either variant. POPF and dReach could not solve this domain due to problems with handling events.

Powered Descent. We developed a new domain which models a powered spacecraft landing on a given celestial body. The vehicle gains velocity due to the force of gravity. The available action is to fire thrusters to decrease its velocity. The thrust action duration is flexible and depends on the available propellant mass. The force of thrust is calculated via Tsiolkovsky rocket equation [Turner, 2008]:

$$\Delta v = I_{sp}\, g \ln \frac{m_0}{m_1} \tag{3}$$

Δv is the change in spacecraft velocity, I_{sp} is the specific impulse of the thruster and g is the gravitational pull. m_0 is the total mass of the spacecraft before firing thrusters and $m_1 = m_0 - qt$ is the mass of the spacecraft afterwards (where q is the rate at which propellant in consumed/ejected and t is the duration of the thrust). The goal is to make a controlled landing from the initial altitude within a given time-frame.

Powered Descent problems were set with increasing initial altitude of the spacecraft (from 100 to 2000 metres) under Earth's force of gravity. The SRPG time horizon was set to $T = 20$ for the first 3 problems and $T = 40$ for the remaining problem instances based on the equations in the domain.

DiNo clearly outperforms UPMurphi which suffers from state explosion relatively early.

Car. The Car domain [Fox and Long, 2006] shows that DiNo does not perform well on all types of problems. SRPG+ cannot extract enough information from the domain and as a result loses out to UPMurphi by approximately one order of magnitude. Table 1 shows results for problems with processes and events. The plan duration and acceleration are limited, and the problems are set with increasing bounds on acceleration (corresponding to the problem number). The SRPG+ time horizon was set to $T = 15$ based on the goal conditions.

The reason behind our heuristic struggling in this case is that the domain is focused on continuous dynamics and, in fact, little search is required. Also, there is no direct link between any action and the goal conditions, since only the processes affect the necessary variables. As a consequence, DiNo reverts to a blind search and explores the same number of states as UPMurphi. The results show the overhead generated by the SRPG+ heuristic in DiNo that fundamentally depends on the sizes of states and the length of the solution.

6 Conclusion

We have presented DiNo, the first ***heuristic*** planner capable of reasoning with the full PDDL+ feature set and complex non-linear systems. DiNo is based on the Discretise & Validate approach, and uses the novel SRPG+ domain-independent heuristic that we have introduced in this paper. We have empirically proved DiNo's superiority over its competitors on benchmark problems set in hybrid domains. Enriching discretisation-based planning with an efficient heuristic that takes processes and events into account is an important step in PDDL+ planning. Future research will concentrate on expanding DiNo's capabilities for inferring more information from the PDDL+ models.

References

[Bogomolov *et al.*, 2014] Sergiy Bogomolov, Daniele Magazzeni, Andreas Podelski, and Martin Wehrle. Planning as Model Checking in Hybrid Domains. In *AAAI*, 2014.

[Bogomolov *et al.*, 2015] Sergiy Bogomolov, Daniele Magazzeni, Stefano Minopoli, and Martin Wehrle. PDDL+ planning with hybrid automata: Foundations of translating must behavior. In *ICAPS*, pages 42–46, 2015.

[Bryce *et al.*, 2015] Daniel Bryce, Sicun Gao, David J. Musliner, and Robert P. Goldman. SMT-Based Nonlinear PDDL+ Planning. In *AAAI*, pages 3247–3253, 2015.

[Cavada *et al.*, 2014] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuXmv symbolic model checker. In *CAV*, pages 334–342, 2014.

[Cimatti *et al.*, 1997] Alessandro Cimatti, Enrico Giunchiglia, Fausto Giunchiglia, and Paolo Traverso. Planning via model checking: A decision procedure for AR. In *Recent Advances in AI planning*, pages 130–142. Springer, 1997.

[Cimatti *et al.*, 2015] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. HyComp: An SMT-based model checker for hybrid systems. In *ETAPS*, pages 52–67, 2015.

[Coles *et al.*, 2010] Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long. Forward-Chaining Partial-Order Planning. In *ICAPS*, pages 42–49, 2010.

[Coles *et al.*, 2012] Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long. COLIN: Planning with Continuous Linear Numeric Change. *J. Artif. Intell. Res.*, 44:1–96, 2012.

[Della Penna *et al.*, 2009] Giuseppe Della Penna, Daniele Magazzeni, Fabio Mercorio, and Benedetto Intrigila. UPMurphi: A Tool for Universal Planning on PDDL+ Problems. In *ICAPS*, 2009.

[Della Penna *et al.*, 2012] Giuseppe Della Penna, Daniele Magazzeni, and Fabio Mercorio. A Universal Planning System for Hybrid Domains. *Appl. Intell.*, 36(4):932–959, 2012.

[Edelkamp and Hoffmann, 2004] Stefan Edelkamp and Jörg Hoffmann. PDDL2.2: The language for the classical part of the 4th international planning competition. *IPC at ICAPS*, 2004.

[Fox and Long, 2003] Maria Fox and Derek Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.*, 20:61–124, 2003.

[Fox and Long, 2006] Maria Fox and Derek Long. Modelling Mixed Discrete-Continuous Domains for Planning. *J. Artif. Intell. Res.*, 27:235–297, 2006.

[Fox *et al.*, 2005] Maria Fox, Richard Howey, and Derek Long. Validating Plans in the Context of Processes and Exogenous Events. In *AAAI*, pages 1151–1156, 2005.

[Fox *et al.*, 2012] Maria Fox, Derek Long, and Daniele Magazzeni. Plan-based Policies for Efficient Multiple Battery Load Management. *J. Artif. Intell. Res.*, 44:335–382, 2012.

[Hoffmann and Nebel, 2001] Jörg Hoffmann and Bernhard Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *J. Artif. Intell. Res.*, 14:253–302, 2001.

[Hoffmann, 2003] Jörg Hoffmann. The Metric-FF Planning System: Translating"Ignoring Delete Lists"to Numeric State Variables. *J. Artif. Intell. Res.*, 20:291–341, 2003.

[Howey and Long, 2003] Richard Howey and Derek Long. VAL's progress: The automatic validation tool for PDDL2. 1 used in the international planning competition. In *IPC at ICAPS*, 2003.

[Howey *et al.*, 2004] Richard Howey, Derek Long, and Maria Fox. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *ICTAI*, pages 294–301. IEEE, 2004.

[Karaman *et al.*, 2011] Sertac Karaman, Matthew R. Walter, Alejandro Perez, Emilio Frazzoli, and Seth J. Teller. Anytime motion planning using the RRT. In *IEEE-ICRA*, 2011.

[Lahijanian *et al.*, 2014] Morteza Lahijanian, Lydia E. Kavraki, and Moshe Y. Vardi. A sampling-based strategy planner for nondeterministic hybrid systems. In *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*, pages 3005–3012, 2014.

[Li and Williams, 2008] Hui X. Li and Brian C. Williams. Generative Planning for Hybrid Systems Based on Flow Tubes. In *ICAPS*, pages 206–213, 2008.

[Long and Fox, 2003] Derek Long and Maria Fox. Exploiting a graphplan framework in temporal planning. In *ICAPS*, pages 52–61, 2003.

[Maly *et al.*, 2013] Matthew R. Maly, Morteza Lahijanian, Lydia E. Kavraki, Hadas Kress-Gazit, and Moshe Y. Vardi. Iterative temporal motion planning for hybrid systems in partially unknown environments. In *HSCC*, pages 353–362, 2013.

[McDermott *et al.*, 1998] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL - The Planning Domain Definition Language. 1998.

[McDermott, 2003] Drew V. McDermott. Reasoning about Autonomous Processes in an Estimated-Regression Planner. In *ICAPS*, pages 143–152, 2003.

[Penberthy and Weld, 1994] J. Scott Penberthy and Daniel S. Weld. Temporal Planning with Continuous Change. In *AAAI*, pages 1010–1015, 1994.

[Piacentini *et al.*, 2016] Chiara Piacentini, Daniele Magazzeni, Derek Long, Maria Fox, and Chris Dent. Solving Realistic Unit Commitment Problems using Temporal Planning: Challenges and Solutions. In *ICAPS*, 2016.

[Plaku *et al.*, 2013] Erion Plaku, Lydia E. Kavraki, and Moshe Y. Vardi. Falsification of LTL safety properties in hybrid systems. *STTT*, 15(4):305–320, 2013.

[Richter and Westphal, 2010] Silvia Richter and Matthias Westphal. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.*, 39(1):127–177, 2010.

[Shin and Davis, 2005] Ji-Ae Shin and Ernest Davis. Processes and Continuous Change in a SAT-based Planner. *Artif. Intell.*, 166(1-2):194–253, 2005.

[Tabuada *et al.*, 2002] Paulo Tabuada, George J. Pappas, and Pedro U. Lima. Composing abstractions of hybrid systems. In *HSCC*, pages 436–450, 2002.

[Turner, 2008] Martin JL Turner. *Rocket and spacecraft propulsion: principles, practice and new developments*. Springer Science & Business Media, 2008.

[Vallati *et al.*, 2016] M. Vallati, D. Magazzeni, B. D. Schutter, L. Chrpa, and T. L. Mccluskey. Efficient Macroscopic Urban Traffic Models for Reducing Congestion: A PDDL+ Planning Approach. In *AAAI*. AAAI Press, 2016.