Heuristic Planning for Hybrid Systems

Wiktor Piotrowski Daniele Magazzeni Maria Fox Derek Long
Department of Informatics
King's College London
London, UK

Fabio Mercorio
Department of Statistics and Quantitative Methods
C.R.I.S.P. Research Centre
University of Milan-Bicocca
Milan, Italy

Abstract

Planning in hybrid systems has been gaining research interest in the Artificial Intelligence community in recent years. Hybrid systems allow for a more accurate representation of real world problems, though solving them is very challenging due to complex system dynamics and a large model feature set. 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. 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 been evolving to accommodate new concepts and operations, enabling research to tackle problems more accurately representing real-world scenarios. Recent versions of the language, PDDL2.1 and PDDL+ (Fox and Long (2003, 2006)), enabled the most accurate standardised way yet, of defining hybrid problems as planning domains.

Planning in hybrid domains (also known as hybrid systems) is currently one of the fastest growing and most interesting subfields of Artificial Intelligence planning. 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 behaviours and immense search spaces. Indeed, planning in hybrid domains is challenging because apart from the state explosion caused by discrete state variables, the continuous variables cause the reachability problem to become undecidable (Alur et al. (1995)).

In this paper, we introduce DiNo, a new planner designed to tackle problems set in hybrid domains with mixed discrete-continuous system dynamics. DiNo 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 in a finite search space.

DiNo, based on UPMurphi, extends its capabilities by using a novel relaxation-based domain-independent heuristic, 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 improves on the Temporal Relaxed Planning Graph and extends its functionality to include information gained from PDDL+ features, namely the processes and events.

DiNo is currently 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 experiments conducted to test DiNo against its competitor planners. We also describe Powered Descent, a new hybrid domain we designed to further test DiNo's capabilities and performance. Section 6 concludes the paper.

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)). 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, and (Bryce et al. (2015)) where PDDL+ models are handled using SMT.

Nevertheless, none of these approaches are able to handle the full set of PDDL+ features, namely nonlinear domains with processes and events. To date, the only viable approach in this direction is PDDL+ planning via *discretisation*. UPMurphi (Della Penna, Magazzeni, and Mercorio (2012)), which implements the discretise and validate ap-

proach, is able to deal with the full range of PDDL+ features. The main drawback of UPMurphi is the lack of heuristics, and this strongly limits its scalability. However, UPMurphi was successfully used in the multiple-battery management domain (Fox, Long, and Magazzeni (2012)), where a domain-*specific* heuristic was used.

3 Discretise & Validate Approach

As a successor to UPMurphi (Della Penna et al. (2009)), DiNo relies on the Discretise & Validate approach 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, Long, and Fox (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 Fig. 1.

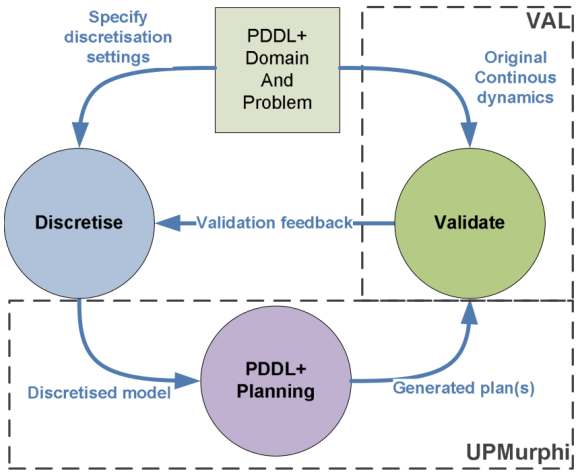


Figure 1: The Discretise & Validate diagram

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

Definition 1. $\Delta-$***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. The time-passing action implements the step function used to discretise the system dynamics, and its effects are: (i) to update all numeric state variables affected by the combined effect of all processes that are active at the time of application, (ii) to update all state variables affected by events, and (iii) to advance time by Δt.

Definition 2. ***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. $F : S \times \Delta\mathcal{A} \times \mathcal{D} \rightarrow S$ is the transition function, i.e. $F(s, \Delta a, d) = s'$ iff applying a Δ-action Δa with a duration 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 3. ***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, 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 $\tilde{\pi} = \sum_{i=0}^{|\pi|-1} \pi_d(i)$.*

Each state s contains a temporal clock $t : S \rightarrow \mathbb{R}^+$, 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, $t(s) \leq T$.

An example of a trajectory π is shown in the following:

$$(s_0, a_1, 0)(s_1, tp, 1)(s_2, a_2, 0)(s_3, tp, 1)(s_4, e_1, 0)$$

$$(s_5, tp, 1)(s_6, tp, 1)(s_7, e_2, 0)(s_8, tp, 1)(s_9, a_3, 0)$$

where $\tilde{\pi} = 5$, and the corresponding graphical representation is reported in Figure 2.

Definition 4. ***Planning Problem***. *In terms of a FSTS, a planning problem $\mathcal{P}$ is defined as a tuple $\mathcal{S} = ((S, s_0, \Delta\mathcal{A}, \mathcal{D}, F, T), G)$ 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.

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

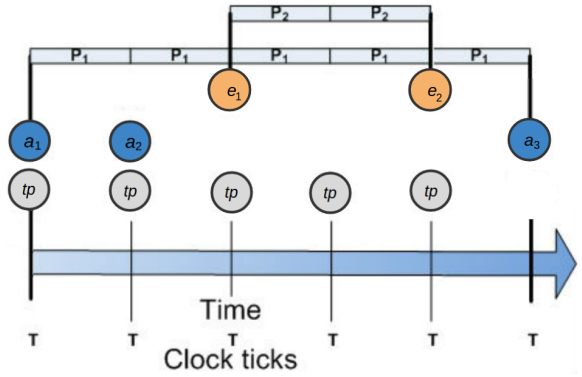


Figure 2: Example of processes and events interaction in the discretised plan timeline

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 1, 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, Howey, and Long (2005)), for avoiding cyclic triggering of events.

4 Staged Relaxed Planning Graph

This section describes the Staged Relaxed Planning Graph (SRPG) domain-independent heuristic implemented in DiNo. It is designed for hybrid domains with continuous change and PDDL+ features. The heuristic is based on the successful and well-known Temporal Relaxed Planning Graph (TRPG), but it significantly differs in time handling.

The SRPG heuristic follows from Propositional (Hoffmann and Nebel (2001)), Numeric (Hoffmann (2003, 2002)) and Temporal RPGs (Coles et al. (2012, 2008); Coles and Coles (2013)). The original problem is relaxed and does not account for 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 the time at which it occurs.

4.1 Time-Staging

The Temporal RPG takes time constraints into account by time-stamping each fact layer at the earliest possible occurrence of a happening. This means that only fact layers where values are directly affected by actions are contained in the Relaxed Planning Graph.

The Staged RPG differs from the TRPG in that it explicitly represents every fact and action layer separated by a predefined time interval. The RPG is "staged" in the sense that the finite set of fact layers are separated by Δt.

The SRPG follows the priority of happenings from VAL, i.e. each new fact layer f' is generated by applying the effects of active processes in f, 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.

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

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.

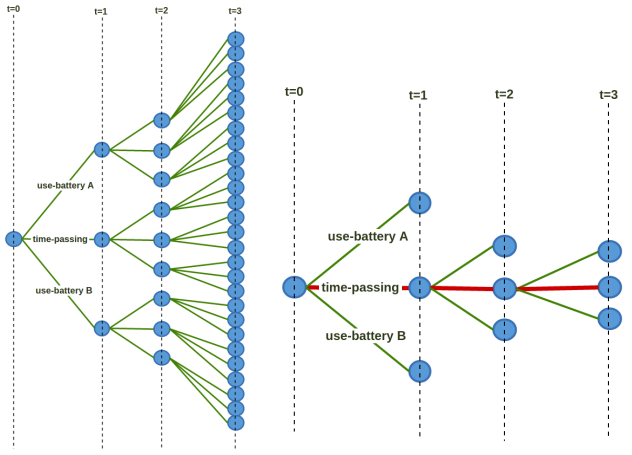
Definition 5. ***SRPG*** *Let $\mathcal{S} = (S, s_0, \mathcal{A}, \mathcal{D}, F, T)$ be an FSTS, then a Staged Relaxed Planning Graph $\widehat{\mathcal{S}}$ is a tuple $(\widehat{S}, \widehat{s}_0, \widehat{\Delta A}, \Delta t, \widehat{F}, T)$ where $\widehat{S}$ is a finite set of fact layers, $\widehat{s}_0$ 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 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.

Note that the time horizon T bounds the expansion of the SRPG. The evaluated state's heuristic estimate will be set to a very high threshold value if the SRPG is unable to satisfy the goal conditions within the time horizon.

4.2 Time-passing

The time-passing action plays an important role as it propagates the search in the discretised timeline. During the nor-



(a) UPMurphi (b) DiNo

Figure 3: Branching of search trees (red edges correspond to the helpful actions in the SRPG)

mal 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 when its effects achieve some goal conditions (or intermediate goal facts). Furthermore, if at a given time t there are no helpful actions 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, the SRPG allows DiNo to recognise when time-passing is a helpful action (as it is the case during the idle periods) and thus advance time mitigating the state explosions.

An illustrative example is shown in Figure 3, 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. In such a case, UPMurphi has no information about the need to wait for the TIL, therefore it tries to use the batteries at each time step. On the contrary, DiNo recognises the time-passing as an helpful action, and this prunes the state space dramatically.

4.3 Processes and Events in the SRPG

As the SRPG is tailored for PDDL+ domains, it takes into account processes and events. In the SRPG, the 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. To this end, Table 1 summarises the number of states visited by DiNo and UPMurphi in each test, in order to provide a clear overview of the pruning effect of the SRPG heuristic. However, for sake of completeness, where possible we also provide a comparison with other competitors able to handle (sub-class of) PDDL+ features, namely POPF (Coles et al. (2010); Coles and Coles (2013)) and dReach (Bryce et al. (2015)).

Each domain in our test suite was chosen to highlight a specific aspect of DiNo: Generator is the benchmark domain, Solar Rover shows how DiNo handles Timed Initial Literals, Powered Descent further tests DiNo's non-linear capabilities, and the Car domain shows the overhead generated by the SRPG under shortage of heuristic information.

All test domains, problems and plans are available at: *[Website ommitted for author anonymity]*. Note that all figures in this sections have their Y axis in a logarithmic scale and for all domains we used the same time discretisation $\Delta t = 1.0$.

For a fair comparison, all reported 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.

Generator The generator domain 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 the linear variant we increase the number of tanks available to the planner while decreasing the initial generator fuel level. The non-linear problems have an increasing duration of the generator action and increasing number of refueling tanks available.

The results for the linear generator problems (Figure 4 and Table 2) 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 time horizon was set to $T = 1000$, that is the duration for which the generator is requested to run.

DOMAIN	PLANNER	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LINEAR GENERATOR	DiNo	1,990	2,957	3,906	4,837	5,750	6,645	7,522	8,381	9,222	10,045	10,850	11,637	12,406	13,157	13,890	14,605	15,302	15,981	16,642	17,285
	UPMurphi	7,054,713	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
NON-LINEAR GENERATOR	DiNo	374	1,631	9,735	19,165	99,602	222,144	2,491,141	1,046,579	X	X	-	-	-	-	-	-	-	-	-	-
	UPMurphi	1,768,138	X	X	X	X	X	X	X	X	X	-	-	-	-	-	-	-	-	-	-
LINEAR SOLAR ROVER	DiNo	211	411	611	811	1,011	1,211	1,411	1,611	1,811	2,011	2,211	2,411	2,611	2,811	3,011	3,211	3,411	3,611	3,811	4,011
	UPMurphi	9,885,372	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
NON-LINEAR SOLAR ROVER	DiNo	231	431	631	831	1,031	1,231	1,431	1,631	1,831	2,031	2,231	2,431	2,631	2,831	3,031	3,231	3,431	3,631	3,831	4,031
	UPMurphi	14,043,452	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
POWERED DESCENT	DiNo	582	942	2,159	4,158	3,161	3,345	4,552	3,850	1,180	1,452	9,498	44,330	12,570	55,265	15,115	64,996	1,587	2,426,576	X	X
	UPMurphi	1,082	30,497	134,135	361,311	1,528,321	6,449,518	16,610,175	44,705,509	45,579,649	X	X	X	X	X	X	X	X	X	X	X
CAR	DiNo	2,974	10,435	18,800	26,132	31,996	36,409	36,504	41,483	42,586	43,112	-	-	-	-	-	-	-	-	-	-
	UPMurphi	4,124	10,435	18,800	26,132	31,996	36,409	36,504	41,483	42,586	43,112	-	-	-	-	-	-	-	-	-	-

Table 1: Number of states explored for each problem in our test domains ("X" - planner ran out of memory, "-" - problem not tested)

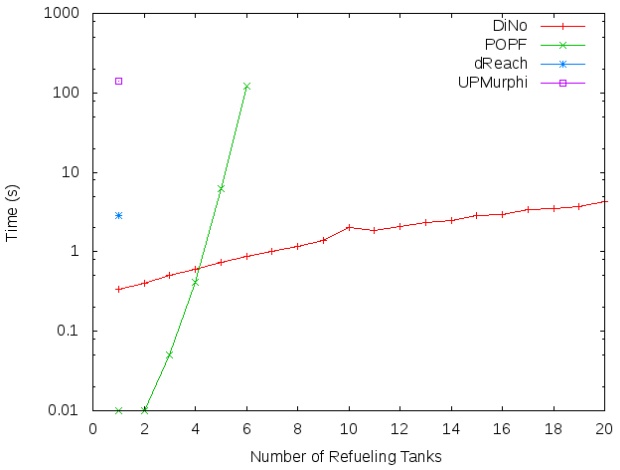


Figure 4: Linear Generator results

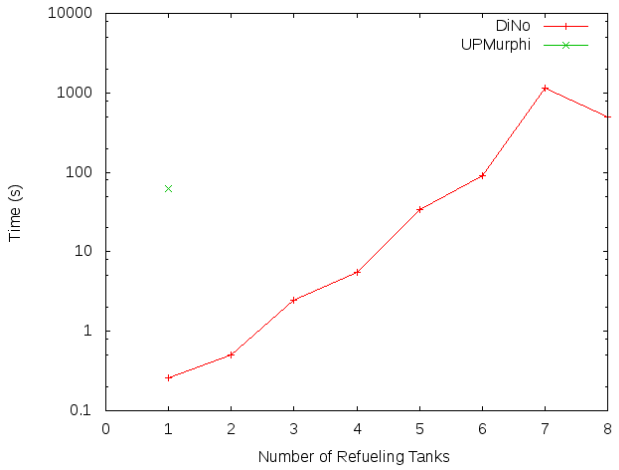


Figure 5: Non-linear Generator results

Figure 5 shows the results for the non-linear generator problem. Also in this case, the time horizon was set as a function of the duration for which the generator is requested to run in each test. The domain could only be tested on DiNo and UPMurphi as the remaining planners do not support non-linear behaviour. dReach results presented in (Bryce et al. (2015)) show that it is only able to solve instances of the non-linear generator problem with up to 3 tanks. However, the dReach results have been left out of our comparison as Bryce et al. tested a much simpler version of the problem compared to us and difficulty with reproducing our domain (written in PDDL+) using the dReach modelling language. The dReach domain and problem description is not standardised and extremely difficult to code as each mode has to be explicitly defined (i.e. the files for 1, 2, 3 and 4 tanks problems are respectively 91, 328, 1350, 5762 lines long). Furthermore, Bryce et al. use a much simplified version of the problem where the generator can never overflow, the refueling action duration is fixed (refueling tanks have no defined capacity), and the flow rate formula is defined as $(0.1 * (tank\_refuel\_time^2))$.

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

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.

In order 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 (between 50 and 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 at a certain time point, the TIL now triggers a process charging the rover's battery at an exponential rate. 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).

The results (Figures 6 and 7) show that DiNo can easily handle this domain and solve all test problems. UPMurphi struggles and is only able to solve the smallest problem in-

DOMAIN	PLANNER	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LINEAR GENERATOR	DiNo	0.34	0.40	0.50	0.60	0.74	0.88	1.00	1.16	1.38	2.00	1.84	2.06	2.32	2.46	2.88	2.94	3.42	3.54	3.76	4.26
	POPF	0.01	0.01	0.05	0.41	6.25	120.49	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	UPMurphi	140.50	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	dReach	2.87	X	X	X	X	X	X	X	-	-	-	-	-	-	-	-	-	-	-	-

Table 2: Time taken to find a solution for the linear generator domain. Problem number corresponds to number of available refueling tanks ("X" - planner ran out of memory, "-" - problem not tested).

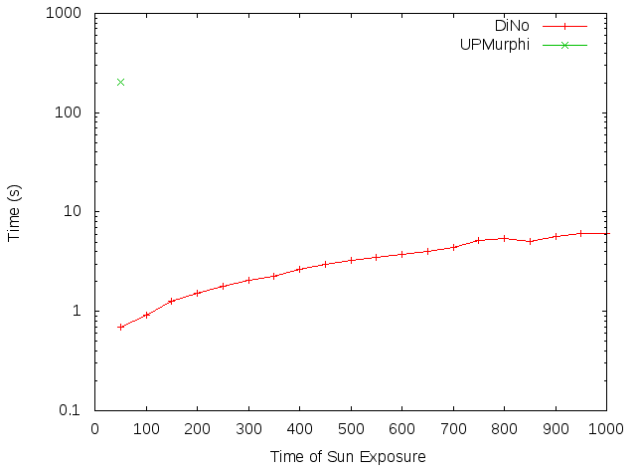


Figure 6: Solar Rover results

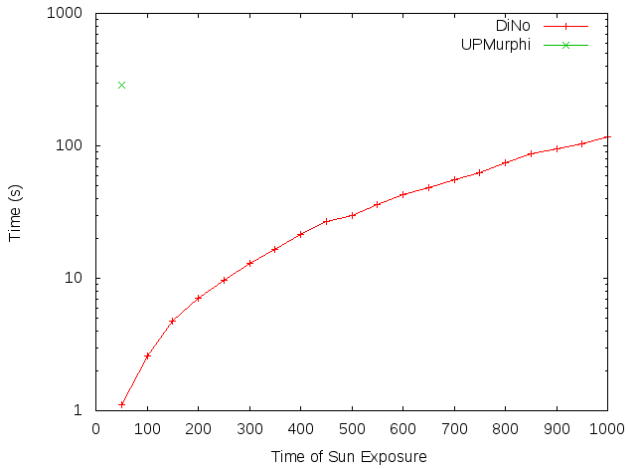


Figure 7: Non-linear Solar Rover

stance 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)):

$$\Delta v = I_{sp}\, g \ln \frac{m_0}{m_1} \tag{1}$$

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 reach the ground with a low enough velocity to make a controlled landing and not crash. The spacecraft has been modelled after the Lunar Descent Module used in NASA's Apollo missions.

Figure 8 shows results for the Powered Descent problems with increasing initial altitude of the spacecraft (from 100 to 1800 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. It has to be said that even a minimal change in the initial conditions can vastly affect the complexity of the problem.

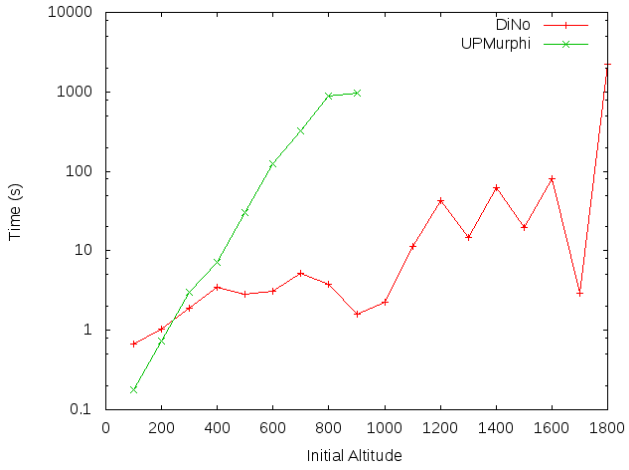


Figure 8: Powered Descent results

Car The Car domain (Fox and Long (2006)) shows that DiNo does not perform well on all types of problems, the heuristic cannot extract enough information from the domain and as a result loses out to UPMurphi by approximately one order of magnitude. Figure 9 shows results for problems with processes and events. This variant of the Car domain has its overall duration and acceleration limited, and the problems are set with increasing bounds on the acceleration (corresponding to the problem number). The SRPG time horizon was set to $T = 15$ based on the goal conditions.

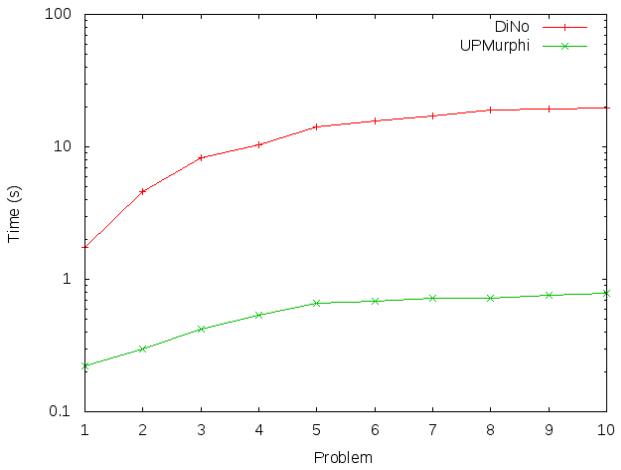


Figure 9: Car (with processes and events)

The reason why the SRPG heuristic does not help in this case it that there is no direct link between any action in the domain 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. The overhead 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 Staged Relaxed Planning Graph domain-independent heuristic that we have introduced in this paper. We have empirically proved DiNo's superiority over its competitors for 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 to deal with larger problem instances and infer more information about effects of events.

References

Alur, R.; Courcoubetis, C.; Halbwachs, N.; Henzinger, T.; Ho, P.; Nicolin, X.; Olivero, A.; Sifakis, J.; and Yovine, S. 1995. The Algorithmic Analysis of Hybrid Systems. *Theoretical Computer Science* 138:3–34.

Bogomolov, S.; Magazzeni, D.; Podelski, A.; and Wehrle, M. 2014. Planning as Model Checking in Hybrid Domains. In *Proceedings of the Twenty Eighth Conference on Artificial Intelligence (AAAI-14)*. AAAI Press.

Bryce, D.; Gao, S.; Musliner, D. J.; and Goldman, R. P. 2015. SMT-Based Nonlinear PDDL+ Planning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, 3247–3253.

Cimatti, A.; Giunchiglia, E.; Giunchiglia, F.; and Traverso, P. 1997. Planning via model checking: A decision procedure for ar. In *Recent Advances in AI planning*. Springer. 130–142.

Coles, A., and Coles, A. 2013. PDDL+ Planning with Events and Linear Processes. *PCD 2013* 35.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. Planning with Problems Requiring Temporal Coordination. In *AAAI*, 892–897.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *ICAPS*, 42–49.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2012. COLIN: Planning with Continuous Linear Numeric Change. *Journal of Artificial Intelligence Research (JAIR)* 44:1–96.

Della Penna, G.; Magazzeni, D.; Mercorio, F.; and Intrigila, B. 2009. UPMurphi: A Tool for Universal Planning on PDDL+ Problems. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*. AAAI.

Della Penna, G.; Magazzeni, D.; and Mercorio, F. 2012. A Universal Planning System for Hybrid Domains. *Appl. Intell.* 36(4):932–959.

Edelkamp, S., and Hoffmann, J. 2004. Pddl2. 2: The language for the classical part of the 4th international planning competition. *4th International Planning Competition (IPC'04), at ICAPS'04*.

Fox, M., and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research* 20:61–124.

Fox, M., and Long, D. 2006. Modelling Mixed Discrete-Continuous Domains for Planning. *Journal of Artificial Intelligence Research* 27:235–297.

Fox, M.; Howey, R.; and Long, D. 2005. Validating Plans in the Context of Processes and Exogenous Events. In *AAAI*, volume 5, 1151–1156.

Fox, M.; Long, D.; and Magazzeni, D. 2012. Plan-based Policies for Efficient Multiple Battery Load Management. *J. Artif. Intell. Res. (JAIR)* 44:335–382.

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. 2002. Extending FF to Numerical State Variables. In *ECAI*, 571–575. Citeseer.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating"Ignoring Delete Lists"to Numeric State Variables. *Journal of Artificial Intelligence Research* 20:291–341.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*, 294–301. IEEE.

Li, H. X., and Williams, B. C. 2008. Generative Planning for Hybrid Systems Based on Flow Tubes. In *ICAPS*, 206–213.

Long, D., and Fox, M. 2003. Exploiting a graphplan framework in temporal planning. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003), June 9-13, 2003, Trento, Italy*, 52–61.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - The Planning Domain Definition Language.

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

Penberthy, J. S., and Weld, D. S. 1994. Temporal Planning with Continuous Change. In *AAAI*, 1010–1015.

Richter, S., and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research* 39(1):127–177.

Shin, J.-A., and Davis, E. 2005. Processes and Continuous Change in a SAT-based Planner. *Artif. Intell.* 166(1-2):194–253.

Turner, M. J. 2008. *Rocket and spacecraft propulsion: principles, practice and new developments*. Springer Science & Business Media.