Taming Discretised PDDL+ through Multiple Discretisations

Matteo Cardellini[1,5], **Marco Maratea**[2], **Francesco Percassi**[3], **Enrico Scala**[4], **Mauro Vallati**[3]
[1]University of Genova, Genova, Italy
[2]DeMaCS, University of Calabria, Rende, Italy
[3]School of Computing and Engineering, University of Huddersfield, UK
[4]University of Brescia, Brescia, Italy
[5]Polytechnic Turin, Turin, Italy
matteo.cardellini@polito.it, marco.maratea@unical.it, f.percassi@hud.ac.uk,
enrico.scala@unibs.it, m.vallati@hud.ac.uk

Abstract

The PDDL+ formalism supports the use of planning in applications that require the ability to perform hybrid discrete-continuous reasoning. PDDL+ problems are notoriously challenging to tackle, and discretisation is a well-established approach to reason upon them. Existing systems rely on a single discretisation delta or, at most, two: a simulation delta to model the dynamics of the environment, and a planning delta, that is used to specify when to take decisions. However, there exist cases where this rigid schema is inefficient. To address the needs of the mentioned class of hybrid planning problems, in this paper we introduce a reformulation approach that allows to encapsulate different levels of discretisation in PDDL+ models, hence allowing any domain-independent planning engine to reap the benefits.

Introduction

The ability to represent hybrid discrete-continuous changes is crucial to exploit automated planning in many real-world applications. The PDDL+ language has been introduced to support the compact encoding of models involving hybrid changes, via the use of specialised constructs such as events and processes (Fox and Long 2006).

Hybrid PDDL+ problems are notoriously challenging to tackle, due to the intertwined nature of numeric variables and time. A well-established approach to reason upon hybrid PDDL+ problems is discretisation (Percassi, Scala, and Vallati 2022; Della Penna, Magazzeni, and Mercorio 2012), which allows breaking down complexity by assuming the time is discrete, and so are the actual numeric changes. An important aspect of this approach is the ability to re-use well-established and general search techniques based on forward state-based exploration to tackle PDDL+ problems; it is indeed widely exploited by existing domain-independent planning engines such as ENHSP (Scala et al. 2016) and UPMurphi (Della Penna, Magazzeni, and Mercorio 2012).

Existing approaches that leverage discretisation generally consider a single delta, used for both simulating the evolution of the dynamic environment and for identifying decision points for planning. A more advanced approach, presented by (Ramirez et al. 2017), and supported by ENHSP, is to consider two deltas: a simulation delta, to be as small as possible to better approximate complex hybrid dynamics, and a planning delta, that can be discretionally large, to reduce the burden on the planning process by avoiding decision points when no actions are likely to be applicable.

Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Notably, there can be cases where even the advanced technique of using two different discretisation deltas does not allow to efficiently reason upon the dynamics of the problem at hand. In logistics, for example, it is common to have different means of transport each having different speeds and different granularity of timings in which actions must be performed (e.g. a plane is faster than a truck which is faster than a delivery man) and if the different agents involved have to coordinate, they must necessarily do it at the discretisation step of the slowest one, making the solving extremely challenging. Even the same agent could require a different granularity in different moments of the plan: for example, a ship must be finely controlled while manoeuvring in the harbour, but its course can be sporadically altered while at open sea. In this paper, as a running example, we introduce a model consisting of two agents with different speeds that need to interact to move an object between different locations.

To effectively deal with the described class of hybrid problems, there is the need for approaches that can support a variable number of discretisation deltas. In this work, we address this need by introducing a reformulation approach that encapsulates such multiple deltas straight into the PDDL+ models, hence allowing domain-independent planning engines to exploit the benefits. Any planning engine that supports PDDL+ can reason upon the reformulated models, thus extending the ability of existing systems to solve challenging hybrid problems.

Motivating Example

In this section, we present a novel domain, COOPROVERS, in which two agents operate at different speeds and need to coordinate in order to reach the stated goals. Figure 1 provides an example of an initial state (top) and a goal condition (bottom) in which two rovers (`Red` and `Green`) are performing two experiments (`A` and `B`) in two separate locations and need to exchange a tool. For safety reasons, the rovers are only allowed to move from the base camp to their working zone, hence they can only meet at base camp to exchange the mentioned tool. The two rovers are equipped with a bat-

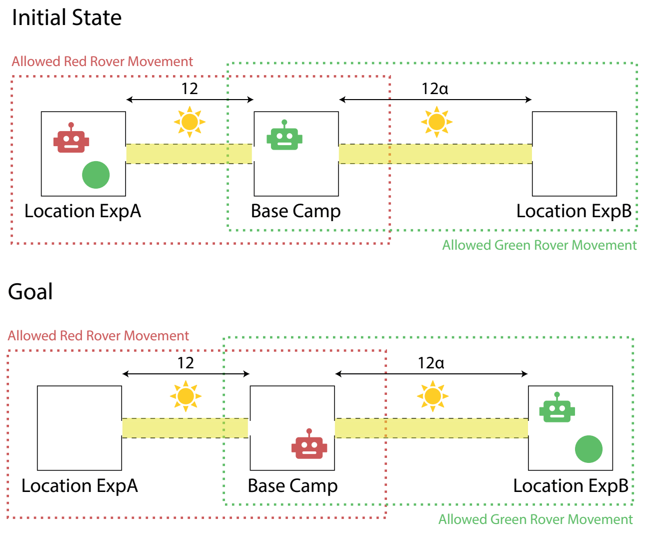
Initial State
Allowed Red Rover Movement
12
12α
Location ExpA
Base Camp
Location ExpB
Allowed Green Rover Movement
Goal
Allowed Red Rover Movement
12
12α
Location ExpA
Base Camp
Location ExpB
Allowed Green Rover Movement

Figure 1: A representation of the initial state and goal condition of the CoOpRovers motivating example.

tery and solar panels that allow them to recharge. Since the location of Experiment `B` is α times more distant than Experiment `A`, the `Green` rover has been equipped with a more efficient and light battery (discharges at $\frac{20}{\alpha}\%/m$), consuming less than the `Red` rover ($20\%/m$) and allowing longer trips. At any point, while moving between the locations, the rovers can deploy their solar panels and recharge (at the speed of $1\%/s$) for some time before restarting their trip. The battery must always stay above 20% to allow emergency operations and the deployment of solar panels. The rovers are also equipped with a holding container for transporting tools. The speed of the two rovers is the same ($1m/s$) but, given the differences in distances to be covered and discharging rates, their movements should be modelled and controlled with different granularities. For example, with $\alpha = 100$ the `Green` rover would need to move 1.2 km at a velocity of 1m/s, discharging at 0.2%/m. In this case, it is easy to notice that the `Green` robot would better benefit from a discretisation step one hundred time bigger than the `Red` one.

Background

A PDDL+ problem is a tuple $\Pi = \langle F, X, I, G, A, E, P\rangle$, where:

- F and X are the sets of Boolean and numeric variables, respectively.
- I and G are the description of the initial state, expressed as a set containing the full assignment to all variables in X and F, and the description of the goal, expressed as a formula, respectively.
- A and E are the sets of actions and events, respectively. Actions and events are pairs $\langle p, e\rangle$, where p is a propositional formula and e is a set of conditional effects of the form $c \triangleright e$, where c is a formula and e is a set of Boolean ($f = \{\bot, \top\}$) or numeric assignments $(\langle\{asgn, inc, dec\}, x, \xi\rangle$, with $x \in X$, ξ being an expression over X and $\mathbb{Q}$) .
- P is a set of processes. A process is a pair $\langle p, e'\rangle$, where p is a formula and e' is a set of numeric continuous effects expressed as pairs $\langle x, \xi\rangle$ with the meaning that ξ represents the net derivative of $x \in X$ when time flows.

Let $a = \langle p, e\rangle$ be an action or an event or a process, we use $pre(a)$ to denote the precondition p of a, and $\mathit{eff}(a)$ the effect e of a. In the following we will use a, ρ and ε for denoting an action, process and event, respectively.

Motivating example (cont'd). We are now in the position to illustrate how to model CoOpRovers using PDDL+. The movement of a rover `r` from two connected locations `a` and `b` is managed trough the usual triplet of action `startMoving(r,a,b)`, process `moving(r,a,b)` and event `endMovement(r,a,b)`. `moving(r,a,b)` is active only when the battery is above the threshold of 20% and keeps updating a variable $dRun$(`r,a,b`) whose role is to track the progress of the rover in going from `a` to `b`. During the movement, process `discharge(r)` models the draining of the battery, and does so with a rate of $cRate(r)$. The planning engine can decide to interrupt and restart the movement through action `startCharging(r)` and action `stopCharging(r)`, respectively. Between these two actions, the process `charging(r)` gets activated, and the rover battery charges with a rate of $1\%/s$. Since a large discretisation can cause the battery of the rover to go above 100%, an event `overflowBattery(r)` sets it back to 100% when this happens. In order to collect and exchange tools, the actions `drop(r,o)` and `pick(r,o)` model the handling of object `o` by rover `r`. The full PDDL+ formulation is available at https://anonymous.4open.science/r/deltaExperiments-F380.

Dynamic Planning-Discretised PDDL+

To address the kind of hybrid problems that require the ability to deal with different dynamics, here we characterise the dynamic planning-discretised PDDL+ problem.

A dynamic planning-discretised PDDL+ problem (PDDLδ+) is the tuple $\langle \Pi, K_\delta = \langle J, \nabla\rangle\rangle$, where Π is a PDDL+ problem defined as above and K_δ is the *discretisation knowledge* detailed as follows. J is a function $A \cup E \rightarrow \{1, ..., m\}$ which partitions the set of actions and events in m classes such that $A = \bigcup_{j\in\{1,...,m\}} A_j$ and $E = \bigcup_{i\in\{1,...,m\}} E_j$, where $A_j = \{a \in A \mid J(a) = j\}$ and $E_j = \{\varepsilon \in E \mid J(\varepsilon) = j\}$. The number of partitions induced by J defines the number of discretisation variables, i.e., $\delta^m = \{\delta_1, ..., \delta_m\}$, which take value in $\mathbb{Q}_{>0}$. Intuitively, every δ_j manages a different aspect of the problem by controlling when actions from A_j can be executed.

∇ is the function which controls the dynamic of the discretisation steps, that is, how the δ^m variables change according to the actions applied and the triggered events. Such a function maps every action and event into a positive rational number plus a special symbol κ, i.e., $\nabla : A \cup E \rightarrow \mathbb{Q}_{>0} \cup \{\kappa\}$; the special symbol κ is the *persist* value and it represents that the affected discretisation variable remains unchanged. With a little abuse of notation, we allow the ∇ function to also accept the initial state as input and return a full assignment of δ_m, i.e., $\nabla(I) = \{\langle \delta_i := \delta_i^0 \rangle \mid \delta_i \in \delta^m\}$ where $\langle \delta_1^0, ..., \delta_m^0 \rangle \in \mathbb{Q}_{>0}^m$. This allows us to initialise the discretisation variables in the initial state.

A discretisation knowledge K_δ may induce a non-deterministic behaviour w.r.t. events. In particular, it is known that events can generate non-determinism in PDDL+ problems (Fox and Long 2006) and this can also affect the discretisation variables δ^m. That said, we define a discretisation knowledge K_δ as *well-defined* iff for each state s, $\varepsilon, \varepsilon' \in E$ and $J(\varepsilon) = J(\varepsilon')$, such that $s \models \mathit{pre}(\varepsilon) \wedge \mathit{pre}(\varepsilon')$, then $\nabla(\varepsilon) = \nabla(\varepsilon')$.

Intuitively, solving a PDDLδ+ problem $\langle \Pi, K_\delta \rangle$ consists in finding a PDDL+ plan for Π such that every executed action is compatible with the discretisation steps prescribed by K_δ.

Motivating example (cont'd). We now show how the discretisation knowledge K_δ can be expressed in the CoOpRovers domain. Intuitively, the actions and events can be partitioned by the rover which performs the action or is subject to the events. For example, $J($`startCharging(red)`$)$ and $J($`startMoving(red,expA,bc)`$)$ are set equal to 1 and $J($`startCharging(green)`$)$ and $J($`startMoving(green,expB,bc)`$)$ equal to 2. This partition induces the set $\delta^2 = \{\delta_1, \delta_2\}$. The function ∇ is set allowing for (i) differentiating the different time scales of the two rovers when they are moving, and (ii) allowing for the same timescale when the two rovers are charging. For this reason, $\nabla($`startMoving(red,expA,bc)` (and the symmetric action for moving from `bc` to `expA`) is set to be 3 while $\nabla($`startMoving(green,expB,bc)` (and symmetric) is set to be 3α, allowing for (i). $\nabla($`startCharging(red)`$)$ and $\nabla($`startCharging(green)`$)$ are both set to 30, allowing for (ii). For all the other actions and events, ∇ returns κ. The initial condition sets the initial deltas to their respective delta of movements: $\nabla(I) = \{\langle \delta_1 := 3 \rangle, \langle \delta_2 := 3\alpha \rangle\}$.

Encoding of K_δ in PDDL+

Let $\langle \Pi = \langle F, X, I, G, A, E, P \rangle, K_\delta = \langle J, \nabla \rangle \rangle$ be a PDDLδ+ problem. To address such problem, here we introduce the FLAT translation, that produces an equivalent PDDL+ problem $\Pi_{\text{FLAT}} = \langle F, X_\delta, I_\delta, G, A_\delta, E_\delta, P_\delta \rangle$, where:

$$X_\delta = X \cup \{\mathtt{ck}\} \cup \bigcup_{j=1}^{m} \{\delta_j, \mathtt{tk}_j\}$$

$$I_\delta = I \cup \nabla(I) \cup \{\langle \mathtt{ck} := 0 \rangle\} \cup \bigcup_{j=1}^{m} \{\langle \mathtt{tk}_j := 0 \rangle\}$$

$$A_\delta = \bigcup_{a \in A} \{\langle \mathit{pre}(a) \wedge \langle \mathtt{ck} = \mathtt{tk}_{J(a)} \rangle, \mathit{eff}(a) \cup u(a) \rangle\}$$

$$E_\delta = \bigcup_{\varepsilon \in E} \{\langle \mathit{pre}(\varepsilon), \mathit{eff}(\varepsilon) \cup u(\varepsilon) \rangle\} \cup \bigcup_{j=1}^{m} \{\mathtt{tic}_j\}$$

$$u(h) = \begin{cases} \emptyset \text{ } \mathit{if}\, \nabla(h) = \kappa \\ \{\langle \delta_{J(h)} := \nabla(h) \rangle, \langle \mathtt{tk}_{J(h)} := \mathtt{ck} \rangle\} \text{ } \mathit{otherwise} \end{cases}$$

$$\mathtt{tic}_j = \langle \langle \mathtt{ck} = \mathtt{tk}_j + \delta_e \rangle, \{\langle \mathtt{tk}_j := \mathtt{ck} + \delta_j - \delta_e \rangle\} \rangle$$

$$P_\delta = P \cup \{\mathtt{t}\}$$

$$\mathtt{t} = \langle \top, \{\langle \mathit{inc}, \mathtt{ck}, \delta_e \rangle\} \rangle$$

The first Equation augments the set of numeric predicates X with new fluents, producing X_δ. The fluent `ck` represents the clock of the system, which keeps track of the flowing of time. Two fluents δ_j and $\mathtt{tk}_j$ are inserted for every partition induced by J: the fluent δ_j keeps track of the value of the discretisation step of the actions of the partition j during the plan and $\mathtt{tk}_j$ keeps track of the next time an action will be applicable in that partition (`tk` stands for *tick*). The I_δ Equation expands the initial state of the original problem with (i) the set given by $\nabla(I)$ which states the initial value of δ^m, and (ii) the initialisation of the clock and all the ticks of the system to zero. The next equation redefines every original action a of Π, augmenting its precondition with a condition enforcing the action to be applicable only when the clock reaches the correct point, established by the value of $\mathtt{tk}_{J(a)}$; here $J(a)$ returns the index of the partition containing a. The effects set of an action or an event h is augmented with the set $u(h)$, in which (i) the value of $\delta_{J(h)}$ is changed to its respective value defined by $\nabla(h)$ only if its value is different from the *persist* value κ, and (ii) the value of $\mathtt{tk}_{J(h)}$ is reset in order to realign the ticks with the correct value of $\delta_{J(h)}$ set by $\nabla(h)$ (this because events are not constrained by the ticks). The Equation which defines E_δ adds also n novel events $\mathtt{tic}_j$ which represent the metronome of the system: the event $\mathtt{tic}_j$ is fired when the value of the clock `ck` has just surpassed the value of $\mathtt{tk}_j$ by the *simulation delta* of the planner δ_e (i.e., we are in the *falling edge*) and, in the effects, it sets the timing ($\mathtt{tk}_j$) in which the raising edge will happen again, taking into consideration the already passed *simulation delta*. Finally, a new process `t` is added, whose job is to increase the value of `ck` by the *simulation delta* δ_e.

It is worth noticing that the FLAT translation yielding Π_{FLAT} is polynomial on the size of $\langle \Pi, K_\delta \rangle$. Specifically, FLAT introduces $2m + 1$ numerical variables (δ_j, $\mathtt{tk}_k$ for each $j \in \{1, ..., m\}$ and `ck`), m events $\mathtt{tic}_j$, where m is the number of partitions of $A \cup E$ induced by J, and a single process, i.e., `t`. Additionally, the preconditions and effects of action/events are extended with at most 2 numeric conditions and effects. Also, it is easy to see that FLAT preserves the length of a plan exactly; as highlighted by (Nebel 2000), this is a desired property when we talk about compilation from one planning problem into another. Let $\langle \Pi, K_\delta \rangle$ be a PDDLδ+ problem and let Π_{FLAT} be the PDDL+ obtained by using FLAT. We expect that $\langle \Pi, K_\delta \rangle$ admits a solution iff Π_{FLAT} does so. The demonstration is left for future work.

Experimental Analysis

Our experimental analysis aims at assessing how the proposed encoding can affect the performance of PDDL+ domain-independent planning engines. This is done by focusing on the CoOpRovers, where the use of the proposed encoding is expected to deliver a significant performance boost due to the characteristics of the domain.

We consider two state-of-the-art domain-independent planning engines: ENHSP (Scala et al. 2016) and UPMurphi

	$\alpha = 1$			$\alpha = 10$			$\alpha = 100$			$\alpha = 500$			$\alpha = 1k$			$\alpha = 5k$			$\alpha = 10k$		
	1δ	2δ	$K\delta$	1δ	2δ	$K\delta$	1δ	2δ	$K\delta$	1δ	2δ	$K\delta$	1δ	2δ	$K\delta$	1δ	2δ	$K\delta$	1δ	2δ	$K\delta$
E+HADD	0.7	**0.3**	0.5	4.5	**0.3**	0.5	TO	40.2	**0.7**	TO	TO	**7.5**	TO	TO	**16.8**	TO	TO	**95.4**	TO	TO	**195.6**
E+AIBR	0.6	**0.3**	2.7	1.4	**0.4**	4.2	10.7	**1.1**	12.2	99.0	**8.2**	151.3	290.2	**27.2**	TO	TO	TO	TO	TO	TO	TO
E+BLIND	0.5	**0.3**	0.5	2.8	**0.4**	0.5	TO	TO	**2.9**	TO	TO	**24.2**	TO	TO	TO	TO	TO	TO	TO	TO	TO
U+BLIND	TO	-	**18.3**	TO	-	**18.6**	TO	-	**65.5**	TO	-	TO	TO	-	TO	TO	-	TO	TO	-	TO

Table 1: Average runtime (CPU-time seconds) achieved by informed and uninformed search approaches implemented in ENHSP (E) and UPMurphi (U) while relying on different discretisation approaches on the CoOpRovers benchmarks. TO indicates that no solution was found in the cut-off of 300s, - indicates that the approach is not available for the planner. Bold indicates the best results.

(Della Penna, Magazzeni, and Mercorio 2012). ENHSP incorporates a range of heuristics and search techniques, hence providing the ideal ground to compare them within the same infrastructure. In our analysis, we used the default A^* search paired with the default *AIBR* heuristic (Scala et al. 2016), the *add* heuristic (Scala, Haslum, and Thiébaux 2016), and a traditional blind search. UPMurphi is a very different planning engine, based on the planning via model-checking paradigm. It automatically translates discretised PDDL+ to a model-checking formulation, and then uses blind search to find a solution. Experiments are run on Intel Xeon Gold 6140M CPUs with 2.3 GHz, with a cut-off time of 300 CPU-time seconds, and 8 GB of RAM.

In the CoOpRovers domain model, choosing the right discretisation step is crucial to efficiently generate a valid plan: larger discretisation can lead to draining the battery of the fastest robot, while the use of a smaller discretisation step makes the search space deeper and requires more resources to be explored. Table 1 shows the results achieved by the considered planning engines when a range of discretisation options are exploited: 1δ, the traditional approach in which there is a unique discretisation step to model the granularity of the agent and the environment, i.e., $\delta_e = \delta_p = 1$; 2δ, the approach where environment and agent are natively decoupled by the planner by using $\delta_e = 1$ and $\delta_p = 3$ (available only in ENHSP), and the proposed K_δ approach. The K_δ approach is run with $\delta_e = 1$ over the PDDL+ problems obtained by using the FLAT translation and exploiting the discretisation knowledge provided for the motivating example in the corresponding section. The experiments show that in the informed search settings (E+HADD and E+AIBR), the proposed approach is beneficial in large and challenging problems. These are indeed the cases where the K_δ approach shows its potential, as the gap between using a small and a large discretisation step increases. In fact, K_δ is the only discretisation approach that allows solving instances with $\alpha \geq 5k$. The performance improvement is more pronounced when an uninformed search is used, where the improvements are noticeable with smaller values of α. The displayed results confirm that the proposed approach can effectively support the reasoning of domain-independent planning engines in cases where dynamics evolving at different speeds are present in a single planning problem.

Conclusion

Discretisation is a well-established approach to reason upon challenging hybrid PDDL+ problems. The vast majority of existing approaches are based on a single discretisation step, and only ENHSP can leverage two different discretisation steps in a domain-independent fashion. With the aim of taming complex PDDL+ problems where multiple deltas are needed to efficiently generate solutions, in this paper we formalised the notion of dynamic planning-discretised PDDL+ problem and presented a reformulation approach that allows any domain-independent planning engine to exploit multiple discretisation deltas. The formalised notion also let us categorise different levels of discretisation control. The performed experimental analysis highlights the benefits of K_δ in instances where different dynamics are included.

As future work, we are interested in (i) demonstrate that FLAT is sound and complete w.r.t the PDDLδ+ problem $\langle \Pi, K_\delta \rangle$, (ii) show the merits of the introduced reformulation on a broader range of PDDL+ benchmarks and planning approaches, and (iii) investigate synergies that can be generated between multiple discretisation reformulations and domain-independent heuristics, to design models that can generate search spaces easier to be navigated by search engines.

References

Della Penna, G.; Magazzeni, D.; and Mercorio, F. 2012. A universal planning system for hybrid domains. *Applied Intelligence*, 36(4): 932–959.

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

Nebel, B. 2000. On the Compilability and Expressive Power of Propositional Planning Formalisms. *J. Artif. Intell. Res.*, 12: 271–315.

Percassi, F.; Scala, E.; and Vallati, M. 2022. The Power of Reformulation: From Validation to Planning in PDDL+. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS 2022)*, 288–296.

Ramirez, M.; Scala, E.; Haslum, P.; and Thiebaux, S. 2017. Numerical integration and dynamic discretization in heuristic search planning over hybrid domains. *arXiv preprint arXiv:1703.04232*.

Scala, E.; Haslum, P.; and Thiébaux, S. 2016. Heuristics for numeric planning via subgoaling. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 3228–3234.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramírez, M. 2016. Interval-Based Relaxation for General Numeric Planning. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016)*, 655–663.