PDDL+ Planning with Temporal Pattern Databases

Wiktor Piotrowski[1], Maria Fox[1], Derek Long[1], Daniele Magazzeni[1], and 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

The introduction of PDDL+ allowed more accurate representations of complex real-world problems of interest to the scientific community. However, PDDL+ problems are notoriously challenging to planners, requiring more advanced heuristics. We introduce the Temporal Pattern Database (TPDB), a new domain-independent heuristic technique designed for PDDL+ domains with mixed discrete/continuous behaviour, non-linear system dynamics, processes, and events. The pattern in the TPDB is obtained through an abstraction based on time and state discretisation. Our approach combines constraint relaxation and abstraction techniques, and uses solutions to the relaxed problem, as a guide to solving the concrete problem with a discretisation fine enough to satisfy the continuous model's constraints.

1 Introduction

Automated planning is continuously evolving to tackle challenging problems emerging from various fields of science. The standardised planning language, PDDL (McDermott et al. 1998), has evolved accordingly to allow modelling of new concepts and constructs, and subsequently enable further research. PDDL+ (Fox and Long 2006) extended the language to include processes and events.

PDDL+ enabled modelling of hybrid systems (mixed discrete/continuous domains), and planning with PDDL+ domains has been gaining substantial research interest in the recent years. Problems set in hybrid systems are notoriously difficult to solve. Non-linear system dynamics, high branching factors, and vast search spaces can render even state-of-the-art heuristic planners ineffective. Hybrid systems have also been the subject of research in Model Checking for many years. Striking similarities between model checking and automated planning allowed symbiotic growth of both fields, through knowledge transfer of approaches and techniques (e.g. (Bogomolov et al. 2014; Bryce et al. 2015)).

One of the approaches successfully used in both planning and model checking is the Pattern Database (PDB)(Culberson and Schaeffer 1998). A Pattern Database is a look-up table indexed by a subset of the state and containing precomputed heuristic value that reflects the cost of solving the corresponding subproblem. In planning, PDBs are considered one of the most successful classes of heuristics for classical planning problems (Edelkamp 2014), while model checking approaches rely on PDBs for handling hybrid systems.

We build on research conducted in both fields to develop the Temporal Pattern Database (TPDB), a new domain-independent heuristic method that enables tackling complex planning problems containing non-linear dynamics and mixed discrete/continuous behaviour. The TPDB combines time and state abstraction with constraint relaxation, and uses the solutions to the relaxed problems as a guide to solving the concrete problem in a way which satisfies the constraints of the continuous model. Our heuristic prunes a substantial part of the search space, reducing memory requirements and execution time.

To further improve the efficiency and performance, we also introduce the Partial Temporal Pattern Database heuristic. The partial TPDB is a downscaled variant of the TPDB, which operates in the same fashion and uses the same mechanics to guide the search as the full TPDB. Our heuristic prunes a substantial part of the search space, reducing memory requirements and execution time.

We introduce DiNo-TPDB, a new planning extension using the Temporal Pattern Database heuristic. We implemented our heuristic into DiNo (Piotrowski et al. 2016), a discretisation-based heuristic planner, able to cope with non-linear dynamics and full PDDL+ semantics.

PDDL+ introduces significant complexity and a variety of features that traditional PDBs cannot handle. To the best of our knowledge, the TPDB is the first extension of the Pattern Database heuristic to temporal hybrid planning domains.

We begin by discussing related work in section 2. Next, in section 3, we give a background on DiNo and its discretised setting. Section 4 describes, and formally defines, the Temporal Pattern Database heuristic, and its variant, the Partial TPDB. Experimental results and comparison against other planners are shown in section 5. Section 6 concludes the paper and describes future research plans.

2 Related Work

The Pattern Databases (PDBs) are a successful class of abstraction heuristics, originating from classical planning (Culberson and Schaeffer 1998; Edelkamp 2014; 2002; Sievers, Ortlieb, and Helmert 2012; Haslum et al. 2007). While PDBs in planning are applied to propositional domains, research in model checking has concentrated on using PDBs for hybrid systems (Bogomolov et al. 2012; 2013). PDBs in planning applications work by obscuring part of the states' variable set, model checking approaches exploit PDBs by abstracting the continuous state variables.

Recently, the increasing complexity and growing search spaces of domains caused the corresponding Pattern Databases' sizes to significantly inflate. In some cases, informative PDBs with fine-grained abstraction can become unfeasible to construct. To maintain the memory efficiency and performance of the heuristic, Partial Pattern Databases have been introduced (Anderson, Holte, and Schaeffer 2007). Partial PDBs cover only a section of the abstract search space while pruning the unpromising parts. They have been successfully employed in both planning (Edelkamp and Kissmann 2008) and model checking (Bogomolov et al. 2013).

Over the years, there have been various restricted approaches in planning for dealing with hybrid domains (McDermott 2003a; Penberthy and Weld 1994; Li and Williams 2008; Coles et al. 2012; Shin and Davis 2005; Fernández-González, Karpas, and Williams 2015), though none of them use PDB heuristics. More recent attempts at dealing with PDDL+ domains include using SMT solvers such as SMTPlan+ (Cashmore et al. 2016), an efficient PDDL+ planner capable of dealing with all aspects of PDDL+ though limited by its solvers to nonlinear polynomials. UPMurphi (Della Penna et al. 2009) can reason with the full PDDL+ feature set and non-linear dynamics but suffers from scalability issues. DiNo (Piotrowski et al. 2016) extends UPMurphi, and alleviates scalability issues with the Staged Relaxed Planning Graph+ heuristic, specifically designed for PDDL+ domains.

3 PDDL+ Planning through Discretisation

DiNo (Piotrowski et al. 2016) and UPMurphi (Della Penna et al. 2009) are discretisation-based planners, that approximate the continuous dynamics of systems in a discretised model using uniform time steps and step functions. The use of a discretised model and a finite-time horizon ensures a finite number of states in the search for a solution, which can be validated against the original continuous model through the validator VAL (Howey, Long, and Fox 2004).

In order to plan in the discretised setting, PDDL+ models are translated into *finite state temporal systems*, as formally described in the following. The notation is inspired from (Piotrowski et al. 2016).

Definition 1. ***State***. *Let $P = \{p_1, ..., p_m\}$ be a finite set of discrete variables and $V = \{v_1, ..., v_n\}$ be a set of real variables. A state s is a triple $s = (p(s), v(s), t(s))$, where $p(s) = (p_1(s), ..., p_m(s)) \in \mathbb{Z}^m$ composes the discrete part of the state, $v(s) = (v_1(s), ..., v_n(s)) \in \mathbb{R}^n$ composes the continuous part of the state, and $t(s)$ is the value of the temporal clock in state s. We also denote with $v_i(s)$ ($p_i(s)$ respectively) the value of variable at the i-th position in $v(s)$ ($p(s)$ respectively).*

Here only real variables and temporal clock are discretised, according to the Discretise & Validate approach (Della Penna, Magazzeni, and Mercorio 2012).

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.*

Borrowing from (Hoffmann 2003), we also denote the set of action preconditions as $pre(\Delta a)$, and the set of action effects as $eff(\Delta a)$.

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 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.

A solution to a planning problem (i.e. a trajectory) is a path in the FSTS transition graph from a reachable state, and ending with a goal state. Therefore, a solution to a planning problem is a trajectory starting with the initial state.

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 $\tilde{\pi} = t(\pi_s(n))$. All states in any trajectory are reachable from states preceding them in the sequence, more formally we define reachable states in the following.

Definition 5. ***Reachable States***. *Let $\mathcal{S} = (S, s_0, \Delta\mathcal{A}, \mathcal{D}, F, T)$ be an FSTS. A state $s_i \in S$ is reachable from state $s_j \in S$ iff there exists a trajectory π in $\mathcal{S}$ s.t. $\pi_s(k) = s_i$ and $\pi_s(l) = s_j$, where $k \leq l$ and $l \leq T$. Therefore, the finite set of states reachable from state $s \in S$ is denoted $Reach(s)$. Conversely, $Reach^{-1}(s)$ is the set of all state from which s is reachable.*

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 6. ***Planning Problem***. *In terms of a FSTS, a planning problem $\mathcal{P}$ is defined as a tuple $\mathcal{P} = (\mathcal{S}, G)$ 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 $\pi^*_s(n) \in G$.*

4 Temporal Pattern Database

Temporal Pattern Database extends the PDB to cope with temporal information and continuous variables.

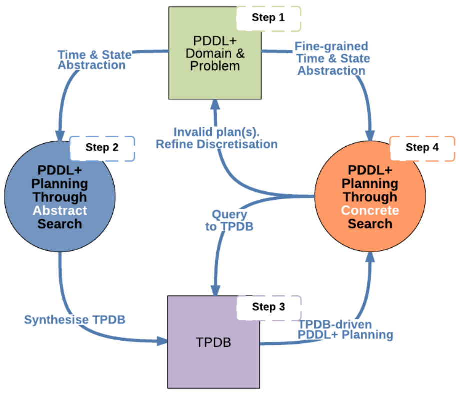

Figure 1: Outline of TPDB-based Planning

In Fig.1 we show a graphical representation of TPDB-based planning. Initially, the PDDL+ domain and problem are both discretised, according to the D&V approach, using *time* and *state* abstraction (see Sec. 4.1). Notice that we also use a *goal* and *action relaxation* to avoid mismatches between temporal clocks and the time discretisation (see Sec. 4.1). Then, we synthesise a TPDB for the abstract PDDL+ domain and problem as discussed in Sec. 4.2. Once the TPDB has been generated, we can solve the original PDDL+ planning problem by performing a concrete search using a fine-grained discretisation Δt. Specifically, the TPDB is queried for each explored concrete state to provide the next promising action which is likely to be on the path to the goal. The action is then applied to the concrete state and the process is repeated for the subsequently explored state. Effectively, the TPDB guides the concrete search to explore only the promising areas of the search space based on solutions found in the relaxed and abstracted setting. Finally, if the solution resulting from the concrete search is not valid, the discretisation should be refined and the process repeated.

4.1 Building the TPDB

The TPDB maintains the simplicity of the look-up table structure and is generated during the preprocessing stage. Search in abstract and relaxed space is conducted, and the results are compiled into the TPDB.

Abstraction. The abstraction is two-fold: time abstraction and state abstraction. Time abstraction works by scaling the concrete time-step Δt up to the abstract time step $\Delta t^{\#}$, using abstraction function ϕ.

Definition 7. ***Time Abstraction****. A time abstraction is a function $\phi : \mathcal{D} \rightarrow \mathcal{D}^{\#}$ which scales the time step Δt up. The function takes the form $\phi : \mathcal{D} \rightarrow \mathcal{D}^{\#}$ where $\mathcal{D} = \{0, \Delta t\}$ and $\mathcal{D}^{\#} = \{0, \phi(\Delta t)\}$, i.e. $\phi(\Delta t) = c * \Delta t = \Delta t^{\#}$ where $c \in \mathbb{R}_{\geq 1}$ is a scalar constant.*

While time abstraction only concerns the discretised time step, the state abstraction function reduces the precision of all continuous variables $v(s)$, where $s \in S$, to a given value.

Definition 8. ***State Abstraction****. The State Abstraction is a function $\psi : S \times \mathbb{R}^{+} \rightarrow S^{\#}$ where S is a finite set of concrete states, $S^{\#}$ is a finite state of abstracted states and $\mathbb{R}^{+}$ is the set of positive real numbers. $\psi(s, q) = s^{\#}$ s.t. $\forall v(s)$ the abstracted value $v^{\#}(s^{\#})$ is computed as follows: $\forall v \in v(s) : v^{\#} = (v + q/2) - ((v + q/2) \bmod q)$*

In practice, function ψ reduces the precision of real variables $v(s)$ for state $s \in S$ and $q \in \mathbb{R}^{+}$ and yields an abstract state $s^{\#} \in S^{\#}$, i.e. $\psi(s, q) = s^{\#}$. For example, applying state abstraction $\psi(s, 0.05)$ to a state s containing a variable $v_1 \in v(s)$, where $v_1 = 12.34567$, would yield an abstract state $s^{\#}$ with real variables scaled to precision up to 0.05, i.e. $v_1^{\#} = 12.35$. Applying $\psi(s, 2)$ would yield the abstracted variable $v_1^{\#} = 12$.

Discretising continuous system dynamics through step functions can sometimes under- or overapproximate the values of continuous variables (depending on the equations). This can be seen in the non-linear generator domain, where in the discretised model, the planner slightly overestimates the amount of fuel added to the generator (compared to values in the continuous model).

A concrete state s and an abstract state $s^{\#}$ are **corresponding** to each other if they agree on the discrete part of their state variables, and the continuous part of $s^{\#}$ equals to the continuous part of $\psi(s, q)$.

Choosing precision for the state abstraction is crucial for the Temporal Pattern Database. On the one hand, choosing a coarser precision for real variables will shrink the size of the TPDB (each abstract state will correspond to a larger number of concrete states). On the other hand, choosing finer precision will make the heuristic estimates more accurate. When choosing the precision value, one should aim to balance the two aspects.

Relaxation. In addition to time and state abstraction, when the TPDB is built, the problem is also relaxed. Firstly, the goal conditions are relaxed, and secondly, the duration of durative actions is relaxed. Both of these relaxation methods are designed to account for cases when the abstract coarse time discretisation mismatches the temporal clocks in the abstract space $S^{\#}$.

Goal relaxation is applied to account for the rigid constraints which can be problematic when working with abstract discretisation. The coarse dicretisation can cause certain values be eliminated from the domains of durative actions and process time-dependent effects.

The following example should clarify the matter. Consider a durative action whose continuous effect is $v_i(s) += \Delta t$, for some numeric variable $v_i(s) \in s$. If v_i is part of the goal condition, then the value of $v_i(s)$ needs to be a multiple of Δt for the problem to be solvable. To compensate for this issue in the abstract space, we set bounds of size $\Delta t^{\#} + 1$ on each numeric goal condition $v(s_G)$ which account for the discretised nature of transitions between states. The value of the concrete numeric goal lies in between the upper and lower bound, i.e. $lb(\overline{s_{\overline{G}}}) <= v(s_G)$ and $ub(\overline{s_{\overline{G}}}) >= v(s_G)$. The size of the bounds was chosen to contain two multiples of $\Delta t^{\#}$, surrounding the concrete goal condition value so that it accounts for both increasing and decreasing effects.

Definition 9. ***Goal Relaxation****. The goal relaxation is a function $\zeta : G \rightarrow \overline{G}$ that, for each goal state*

$s_G \in G$ *computes the set of relaxed goal states resulting from* s_G *by applying the relaxation to each variable of the continuous part as follows* $\forall v_i(s_G), v(\overline{s}_{\overline{G}}) = (v_1(s_G), \ldots, v_i(\overline{s}_{\overline{G}}), \ldots, v_n(s_G))$ *where* $v_i(\overline{s}_{\overline{G}})$ *ranges within* $\{lb(\overline{s}_{\overline{G}}), ub(\overline{s}_{\overline{G}})\}$ *with* $lb(\overline{s}_{\overline{G}}) = v_i(s_G) - (v_i(s_G) \bmod \Delta t^{\#})$ *and* $ub(\overline{s}_{\overline{G}}) = v_i(s_G) + (\Delta t^{\#} - (v_i(s_G) \bmod \Delta t^{\#}))$

In simple terms, function ζ amends any goal condition on $v(s_G)$ s.t. any value between the lower bound $lb(\overline{s}_{\overline{G}})$ and upper bound $ub(\overline{s}_{\overline{G}})$ satisfies the goal condition on $v(s_G)$.

In the case of propositional goal conditions, we relax the numeric preconditions of actions which achieve those goal conditions in the similar manner as Def. 9.

Definition 10. ***Action Precondition Relaxation***. *Extending the* ζ *function defined in Def. 9, by abuse of notation, we apply the function to the set of actions, such that:* $\zeta : \Delta\mathcal{A} \rightarrow \overline{\Delta\mathcal{A}}$, *i.e.* $\forall v \in pre(\Delta a)$ *iff* $\exists s$ *s.t.* $\exists p(s) \in eff(\Delta a) \cap (s_G) : v = \zeta(v) \mid \Delta a \in \Delta\mathcal{A}, s_G \in G, s \in S$.

Action Duration Relaxation. Another issue which arises from a coarse time discretisation, is to do with durative actions. A coarse abstract time step $\Delta t^{\#}$ can render some action a inapplicable if its fixed duration $d(a) \notin \mathcal{D}^{\#}$, or if its maximum flexible duration $d^{max}(a) < \Delta t^{\#}$.

Definition 11. ***Action Duration Relaxation***. *The action duration relaxation is an action which expands the domain of action durations in the abstract space* $\eta : \mathcal{D}^{\#} \rightarrow \overline{\mathcal{D}^{\#}}$, *where* $\mathcal{D}^{\#} = \{0, \Delta t^{\#}\}$ *and* $\overline{\mathcal{D}^{\#}} = \mathcal{D}^{\#} \cup \mathbb{R}^+$.

In essence, action durations are no longer bound to multiples of $\Delta t^{\#}$ but can have any duration, subject to $d(a) <= d^{max}(a)$.

4.2 Temporal Pattern Database

Using the abstraction and relaxation functions, we define the Abstract FSTS and the Abstract Planning Problem which, in turn, form the basis of the Temporal Pattern Database. The Abstract FSTS and Abstract Planning Problem are, in fact, both *relaxed* and *abstracted* by functions ζ, η, ϕ, and ψ.

Definition 12. ***Abstract FSTS***. *Let* $\mathcal{S} = (S, s_0, \Delta\mathcal{A}, \mathcal{D}, F, T)$ *be a FSTS, then an Abstract FSTS* $\overline{\mathcal{S}^{\#}}$ *is a tuple* $(S^{\#}, s_0, \overline{\Delta\mathcal{A}}, \overline{\mathcal{D}^{\#}}, F^{\#}, T)$ *where* $S^{\#}$ *is a finite set of abstract states under state abstraction* ψ, s_0 *is the initial state,* $\overline{\Delta\mathcal{A}}$ *is a finite set of* Δ*-actions under relaxation* ζ, *and* $\overline{\mathcal{D}^{\#}} = \{0, \Delta t^{\#}\} \cup \mathbb{R}^+$ *is a set of relaxed abstract action durations.* $F^{\#} : S^{\#} \times \overline{\Delta\mathcal{A}} \times \overline{\mathcal{D}^{\#}} \rightarrow S^{\#}$ *is a transition function, and* T *is a finite temporal horizon.*

Definition 13. ***Abstract Planning Problem***. *Let* $\mathcal{P} = (\mathcal{S}, G)$ *be a planning problem, then an abstract planning problem is a tuple* $\overline{\mathcal{P}^{\#}} = (\overline{\mathcal{S}^{\#}}, \overline{G})$, *where* $\overline{\mathcal{S}^{\#}}$ *is an Abstract FSTS, and* $\overline{G} = \zeta(g)$ *is the relaxed set of goal conditions under relaxation* ζ.

The Temporal Pattern Database is a structure which maps abstract states to actions applicable in the relaxed state space. A TPDB is built by using the Abstract FSTS and transition function $F^{\#}$ to generate the subsequent abstract states, until the abstract goal state is found, or the finite temporal horizon T is reached (meaning the bounded abstract problem is unsolvable with the current parameters, prompting a refinement of the abstract time step $\Delta t^{\#}$ and/or an increase of the temporal horizon).

Formally, we define the TPDB as follows[1].

Definition 14. ***Temporal Pattern Database (TPDB)***. *Let* $\overline{\mathcal{P}^{\#}} = \{\overline{\mathcal{S}^{\#}}, \overline{G}\}$ *be an abstract planning problem,* $\overline{\mathcal{S}^{\#}} = (S^{\#}, s_0, \overline{\Delta\mathcal{A}}, \overline{\mathcal{D}^{\#}}, F^{\#}, T)$ *be an Abstract FSTS, and* $\overline{\mathcal{R}^{\#}} = Reach(s_0) \cap \bigcup_{\overline{s}_{\overline{G}} \in \overline{G}} Reach^{-1}(\overline{s}_{\overline{G}})$. *Then a Temporal Pattern Database is a finite map TPDB, from* $\overline{\mathcal{R}^{\#}}$ *to* $\overline{\Delta\mathcal{A}}$ *such that* $\forall s^{\#} \in \overline{\mathcal{R}^{\#}}$ *there exist* k, *such that* $\pi_s(k) \in \overline{G}$, *and a trajectory* π *such that* $\pi_s(0) = s^{\#}$, $\forall t < k \; \exists \overline{d^{\#}} \in \overline{\mathcal{D}^{\#}} : \pi_s(t+1) = F^{\#}(\pi_s(t), TPDB(\pi_s(t)), \overline{d^{\#}})$ *and* $\tilde{\pi} \leq T$.

Informally, a TPDB stores all abstract state-action pairs which exist on some trajectory to the relaxed goal.

TPDB is implemented as a hash table with the abstract state acting as the key and the action as the value. During the concrete search each explored concrete state $s \in S$ is used to query the TPDB. Before the TPDB being queried, the explored state is abstracted through the state abstraction function $\psi(s, q) = s^{\#}$ where $s^{\#}$ is the abstracted state, $q \in \mathbb{R}^+$ is a positive real number to which the continuous variables' precision is abstracted. The TPDB is then queried to find the suggested action. The look-up method returns the suggested action which is paired with the corresponding abstract state in the TPDB. The suggested action is then applied to the concrete state, and the successor state is pushed to the top of the state queue. The process then repeats for the next explored state fetched from the top of the queue. A special case in the search comes when querying the TPDB yields time passing (tp) as the suggested action. To quickly forward the search, and prune the search space considerably, we employ Pruning Jump (described in Section 4.4).

4.3 Partial TPDB

PDDL+ planning problems often have high branching factors and long temporal horizons. A full-sized TPDB can increase the plan quality but when searching for a feasible solution only, the added overhead is in most cases redundant. Often, most of the search space is never explored during the concrete search, thus the TPDB is never queried for the majority of its elements. Exhaustively traversing the search space to build the full TPDB, even with a coarse discretisation and relaxation, can be very time- and memory-consuming. Because we are only concerned with finding a feasible solution to the given problems, generating a full-sized Temporal Pattern Database is often ineffective. The time to build an informative TPDB can disproportionately outweigh the run time of concrete search.

A solution to this issue is generating a Partial Temporal Pattern Database, pruning parts of the abstract search space and significantly reduces TPDB build time. The idea of Partial TPDB is based on the the concept introduced by

[1]Our TPDB notation was inspired by (Della Penna, Magazzeni, and Mercorio 2012)

(Edelkamp and Kissmann 2008), which only considers a perimeter of the abstract state space. In our settings, once a goal state s_G has been reached (i.e. $s_G \in Reach(s0)$), a partial TPDB can be synthesised by collecting all the encountered state-action pairs, reachable from s_0, that are on some path to *the single goal state* s_G (i.e. $Reach^{-1}(s_G)$).

It can be seen as a river, that originates from a single source s_0, then it grows by branching in several streams, it receives water from several tributaries, and finally ends in the mouth (i.e., goal state s_G). A partial TPDB would consider all streams generated from the source that are able to reach the mouth. Conversely, all the tributaries originating from outside the river would be discarded.

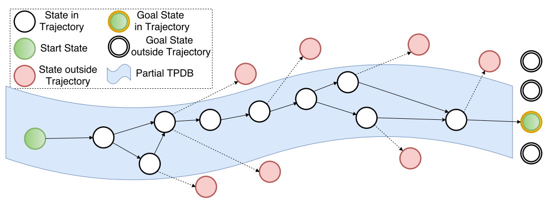

Figure 2: Building the Partial TPDB

We generate the Partial Temporal Pattern Database by limiting the abstract search to one relaxed goal state $\overline{s}_{\overline{G}} \in \overline{G}$, and by accepting subsets of the intersection of the set of abstract state reachable from the initial state ($Reach(s_0)$), and the set of states from which one goal state is reachable ($Reach^{-1}(\overline{s}_{\overline{G}})$).

Definition 15. ***Partial Temporal Pattern Database (TPDB).*** *Let TPDB= $\{(\overline{\mathcal{R}^{\#}}, \overline{\Delta\mathcal{A}})\}$ be a Temporal Pattern Database as in Def. 14, then a* Partial *Temporal Pattern Database TPDB$_{part}$ $= \{(\overline{\mathcal{R}^{\#}}_{part}, \overline{\Delta\mathcal{A}})\}$ is a map from $\overline{\mathcal{R}^{\#}}_{part}$ to $\overline{\Delta\mathcal{A}}$ where $\overline{\mathcal{R}^{\#}}_{part} \subseteq (Reach(s_0) \cap Reach^{-1}(\overline{s}_{\overline{G}}))$, s_0 is the initial state and $\overline{s}_{\overline{G}} \in \overline{G}$ is a* single *goal state.*

In the TPDB implementation in DiNo, Depth-First Search algorithm (DFS) is used to determine the goal state for which a Partial TPDB will be generated.

4.4 Concrete search with TPDB guidance

The concrete search algorithm is guided by the TPDB generated in the preprocessing stage. The TPDB is queried for every dequeued state and there are 3 possible outcomes:

1. Time-passing (*tp*) is returned, Pruning Jump is performed.
2. A Δ-action is returned (other than *tp*).
3. No action is returned, in which case BFS is executed.

The entire algorithm for the full concrete search through TPDB guidance is shown in Alg. 4.1. For sake of completeness, we also provide in Fig. 3 the workflow of point 2, where the TPDB returns a valid Δ-action.

Pruning Jump. Pruning Jump is a mechanism used in conjunction with the TPDB to skip parts of the search space which are likely yield unpromising states. Rather than exploring the generated states and assessing their heuristic values, the search algorithm chooses to advance time. Pruning Jump is triggered by the suggestion from TPDB (line 3 in Alg. 4.1). If the TPDB answers a query with the time-passing action (line 8), the Pruning Jump is executed. It iteratively advances time in the concrete search by a total duration of abstract time step $\Delta t^{\#}$ (line 10). Time passing is applied in increments of the concrete time step Δt, on the previously generated state (line 11) as, due to coarse discretisation, it is possible for an adverse happening, missed in abstract search, to occur in concrete search (e.g. an event prohibiting achieving goal conditions). Thus after each transition of Δt, a check is carried out on the resulting state s' to confirm its validity (line 12). If the goal state is no longer reachable, the Pruning Jump stops (line 13) and general exploration restarts from the last enqueued state (lines 20 & 2). States generated during each iteration of the Pruning Jump are added to the front of the queue (line 14), helping to avoid lengthy backtracking.

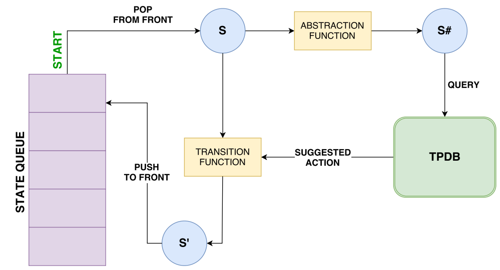

Figure 3: Process of generating new states via TPDB (in concrete search)

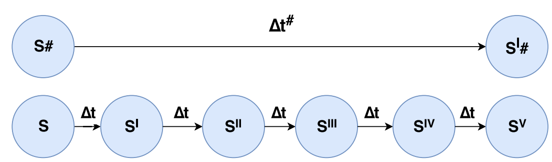

Figure 4: Pruning Jump depiction for $\Delta t^{\#} = 5 * \Delta t$

In essence, if the TPDB indicates that time-passing action is the most promising from the current state s, and no adverse happenings occur between the corresponding abstract state $s^{\#}$ at time $t(s^{\#})$ and the subsequent abstract state $s'^{\#}$ at time $t' = t(s^{\#}) + \Delta t^{\#}$, the time can be advanced by the abstract time discretised variable $\Delta t^{\#}$ in the concrete search. The number of iterations in a Pruning Jump is determined by the duration of the abstract and concrete time steps, i.e. Number of iterations $= \Delta t^{\#}/\Delta t$. An outline of the mechanism is shown in Fig. 4, where the top graph is shows the duration of the abstract time step $\Delta t^{\#}$ and the bottom graph shows the Pruning Jump matching the abstract time step (i.e. 5 iterations of duration Δt)

Effectively, the Pruning Jump acts as an adaptive time discretisation mechanism which helps in mitigating state explosion by skipping unpromising areas of the space.

Back-up Search Strategy. As a back-up strategy, if no corresponding state is found in the TPDB, the search is forwarded through breadth-first search, querying the TPDB for every visited state. If a corresponding state is found, the search is continued through the suggested actions again.

Algorithm 4.1: Concrete Search Algorithm with Pruning Jump and TPDB guidance

```
Data: s ∈ S = Currently explored state;
      a ∈ ΔA = Δ-action;
      tp ∈ ΔA = time-passing Δ-action;
      isValid(s) = true if no constraints are violated in s;
1  Q := ∅ s := s_0;
2  while s ∉ G do
3      a_best := TPDB(s)  ▷ returns suggested action querying the TPDB;
4      if isValid(s) then
5          if a_best = ∅ then
6              for s_succ ∈ Successors(s) do
7                  enqueue(Q, s_succ);
8          else if a_best = tp then
9              i := 0;
10             while i < Δt# do
11                 s := F(s, tp, Δt);
12                 if ¬isValid(s) then
13                     break;
14                 enqueueFront(Q, s);
15                 i := i + Δt;
16         else
17             ▷ Generate state through TPDB suggested action;
18             s := F(s, a_best, 0);
19             enqueueFront(Q, s);
20     s := dequeue(Q);
```

The back-up search strategy has been devised to account for situations when the TPDB fails to provide a feasible next step. This can occur for three reasons:

- TPDB-suggested action is inapplicable in the concrete setting.
- Discrepancies in continuous state variables, larger than the state abstraction (ψ) value, between the abstract states in the TPDB and the abstracted concrete states.
- Triggered events and processes, uncaught in the abstract search due to coarse discretisation, changed the state variable values beyond the scope of variables in the TPDB.

In those cases, uninformed search (BFS) is applied until a new concrete state, with a corresponding abstract state in the TPDB (or the goal), is found (lines 5-7 in Alg. 4.1).

5 Evaluation

We evaluate the Temporal Pattern Database implementation, and compare the results against other planners capable of handling the same class of PDDL+ domains. This includes DiNo-SRPG+, UPMurphi, and SMTPlan+. For the Linear Generator, we also compare against POPF (Coles et al. 2010) since it can handle the sub-class of PDDL+ required for this domain. For DiNo-TPDB and UPMurphi, the concrete search was conducted under discretisation $\Delta t = 1$, for all domains. For DiNo-SPRG+ all problems were also solved using default discretisation $\Delta t = 1$ except for the Non-linear Generator, where for problem instances 3, 4, and 5, the discretisation had to be refined to $\Delta t = 0.5$ since the found plans were invalid using a coarser discretisation.

For the abstract search in DiNo-TPDB, the abstraction settings were set as follows: Car - $\Delta t^{\#} = 4$, $\psi = 4$; Powered Descent - $\Delta t^{\#} = 2$, $\psi = 4$; Vertical Take-Off - $\Delta t^{\#} = 5$, $\psi = 5$. For all other domains: $\Delta t^{\#} = 10$, $\psi = 5$. To standardise the experiments, we used a partial TPDB for our heuristic guidance.

Tab.1 shows the results of experiments. Run times for DiNo-TPDB are the combined times of the concrete search and generating the TPDB. Tab.2 shows the number of states explored by planners when solving each individual problem from our test suite. Note that SMTPlan+ was not included in the comparison as it is not a forward search planner.

All results were obtained by running the experiments on a machine with 8-core Intel i7 CPU, 8GB RAM and Ubuntu 14.04 OS. Where possible, the solutions were validated by VAL. For Powered Descent and Vertical Take-Off, the validation was done through ad-hoc scripts, as system dynamics equations proved too complex for VAL. All our test domains are available at *https://goo.gl/CFAybW*.

Generator. The generator domain (Howey and Long 2003) is a well-established test domain for PDDL+ planners. It concerns a diesel power generator which has to run for a particular duration. The generator cannot run dry or overflow so the refueling actions have to be planned accordingly. There are two versions of the domain, linear and non-linear. In the linear version, the generator's fuel level increases linearly when refueling. On the other hand, the non-linear version of the domain models the flow rate using Torricelli's Law, through complex non-linear equations. Both variants are particularly difficult for planners to deal with because of high-branching factors and long temporal horizons.

Advanced Solar Rover. The Advanced Solar Rover is an extended version of the Solar Rover introduced in (Piotrowski et al. 2016). In comparison with its predecessor, the advanced version increases the role of batteries in the domain, they can be used more repeatedly depending on the batteries' states of charge. Some of the problem instances can actually be solved using batteries alone, rather than solely relying on the future *sunexposure* event to provide energy for the rover. This significantly increases the branching factor, and thus search space grows exponentially.

Vertical Take-Off. Vertical Take-Off domain models the initial stages of flying a tilt-wing rotor plane. The plane has to lift off and transition into fixed-wing flight above a given altitude and distance from the initial point on the ground without crashing. The plane can increase or decrease the angle of its wings in relation to the fuselage, which causes change in the plane's horizontal and vertical velocity modelled through $v_x = \sin\theta v_{eng}$ and $v_y = \cos\theta v_{eng} - g$, respectively (where θ is the wing angle, v_{eng} is the engine velocity, and g is gravity). At all times, the plane is under the influence of gravity which exerts a downward force on it, affecting its vertical velocity.

Powered Descent. The domain (Piotrowski et al. 2016) models a lunar descent module making a controlled landing on a given celestial body without crashing. The spacecraft can thrust in the opposite direction to its fall towards the

	LINEAR GENERATOR					NON-LINEAR GENERATOR				ADVANCED LINEAR SOLAR ROVER		
	DiNo-T	DiNo-S	SMTPlan+	POPF	UPM	DiNo-T	DiNo-S	SMTPlan+	UPM	DiNo-T	DiNo-S	UPM
1	0.38	0.34	0.04	0.01	140.50	0.54	3.62	0.04	X	0.38	0.54	X
2	0.34	0.40	0.04	0.01	X	0.88	0.78	0.06	X	0.44	140.24	X
3	0.36	0.50	0.04	0.05	X	1.50	2.86	0.09	X	0.42	X	X
4	0.38	0.60	0.05	0.41	X	2.44	59.62	0.18	X	0.48	X	X
5	0.38	0.74	0.08	6.25	X	3.62	1051.84	0.40	X	0.44	X	X
6	0.38	0.88	0.12	120.49	X	5.78	X	0.95	X	0.54	X	X
7	0.38	1.00	0.21	X	X	8.90	X	2.34	X	0.56	X	X
8	0.36	1.16	0.43	X	X	13.94	X	5.79	X	0.70	X	X
9	0.36	1.38	0.96	X	X	25.02	X	14.09	X	0.56	X	X
10	0.36	2.00	2.41	X	X	X	X	34.53	X	0.68	X	X
11	0.40	1.84	7.46	X	X	-	-	-	-	0.70	X	X
12	0.40	2.06	28.58	X	X	-	-	-	-	0.66	X	X
13	0.40	2.32	107.57	X	X	-	-	-	-	0.70	X	X
14	0.40	2.46	503.80	X	X	-	-	-	-	0.74	X	X
15	0.38	2.88	X	X	X	-	-	-	-	0.78	X	X
16	0.42	2.94	X	X	X	-	-	-	-	0.82	X	X
17	0.40	3.42	X	X	X	-	-	-	-	0.88	X	X
18	0.40	3.54	X	X	X	-	-	-	-	0.90	X	X
19	0.42	3.76	X	X	X	-	-	-	-	0.98	X	X
20	0.42	4.26	X	X	X	-	-	-	-	1.16	X	X

	ADVANCED NON-LINEAR SOLAR ROVER			POWERED DESCENT			VERTICAL TAKE-OFF			CAR		
	DiNo-T	DiNo-S	UPM	DiNo-T	DiNo-S	UPM	DiNo-T	DiNo-S	UPM	DiNo-T	DiNo-S	UPM
1	0.40	298.44	X	0.56	0.68	0.18	0.46	130.34	9.86	1.04	24.02	0.34
2	0.42	X	X	0.58	1.04	0.74	0.32	X	202.22	1.72	90.34	0.74
3	0.42	X	X	6.30	1.88	2.98	0.32	X	X	2.72	191.76	1.16
4	0.50	X	X	27.48	3.52	7.18	0.34	X	X	3.50	240.74	1.64
5	0.46	X	X	40.46	2.88	30.08	0.34	X	X	3.64	286.82	1.96
6	0.50	X	X	97.46	3.14	126.08	0.36	X	X	5.46	359.72	2.10
7	0.54	X	X	522.44	5.26	322.16	0.36	X	X	7.70	365.18	2.46
8	0.72	X	X	444.78	3.82	879.52	0.38	X	X	7.78	405.90	2.44
9	0.56	X	X	527.32	1.58	974.60	0.40	X	X	7.80	461.60	2.60
10	0.68	X	X	535.18	2.26	X	0.42	X	X	10.04	389.82	2.44
11	0.72	X	X	10.06	11.24	X	-	-	-	-	-	-
12	0.66	X	X	9.72	42.24	X	-	-	-	-	-	-
13	0.70	X	X	X	14.90	X	-	-	-	-	-	-
14	0.72	X	X	X	61.94	X	-	-	-	-	-	-
15	0.78	X	X	X	19.86	X	-	-	-	-	-	-
16	0.82	X	X	X	80.28	X	-	-	-	-	-	-
17	0.66	X	X	X	2.94	X	-	-	-	-	-	-
18	0.92	X	X	X	X	X	-	-	-	-	-	-
19	0.94	X	X	X	X	X	-	-	-	-	-	-
20	0.92	X	X	X	X	X	-	-	-	-	-	-

Table 1: Run time in seconds for each problem in our test suite (”X” - planner ran out of memory).
Labels: DiNo-T = DiNo with Partial TPDB, DiNo-S = DiNo with SRPG+, UPM = UPMurphi

	LINEAR GENERATOR				NON-LINEAR GENERATOR			ADVANCED LINEAR SOLAR ROVER		
	DiNo-T	DiNo-S	POPF	UPM	DiNo-T	DiNo-S	UPM	DiNo-T	DiNo-S	UPM
1	1,006	1,990	7	7,054,713	2,024	31,742	X	811	261	X
2	1,009	2,957	15	X	3,110	3,057	X	1,593	32,183	X
3	1,012	3,906	83	X	4,033	8,056	X	1,205	X	X
4	1,015	4,837	605	X	5,027	257,775	X	1,993	X	X
5	1,018	5,750	7,922	X	6,038	4,089,559	X	1,541	X	X
6	1,021	6,645	133,260	X	7,051	X	X	2,393	X	X
7	1,024	7,522	X	X	8,066	X	X	2,670	X	X
8	1,027	8,381	X	X	9,083	X	X	3,844	X	X
9	1,030	9,222	X	X	10,102	X	X	2,473	X	X
10	1,033	10,045	X	X	X	X	X	3,274	X	X
11	1,036	10,850	X	X	-	-	-	3,474	X	X
12	1,039	11,637	X	X	-	-	-	3,011	X	X
13	1,042	12,406	X	X	-	-	-	3,273	X	X
14	1,045	13,157	X	X	-	-	-	3,411	X	X
15	1,048	13,890	X	X	-	-	-	3,611	X	X
16	1,051	14,605	X	X	-	-	-	3,811	X	X
17	1,054	15,302	X	X	-	-	-	4,011	X	X
18	1,057	15,981	X	X	-	-	-	4,141	X	X
19	1,060	16,642	X	X	-	-	-	4,341	X	X
20	1,063	17,285	X	X	-	-	-	5,312	X	X

	ADVANCED NON-LINEAR SOLAR ROVER			POWERED DESCENT			VERTICAL TAKE-OFF			CAR		
	DiNo-T	DiNo-S	UPM	DiNo-T	DiNo-S	UPM	DiNo-T	DiNo-S	UPM	DiNo-T	DiNo-S	UPM
1	803	50,029	X	1,148	582	1,082	111	154,398	455,400	8,188	6,223	9,062
2	1,601	X	X	255	942	30,497	167	X	8,919,999	16,800	23,114	64,693
3	1,213	X	X	73,210	2,159	134,135	162	X	X	27,564	41,770	64,200
4	2,001	X	X	341,429	4,158	361,311	157	X	X	36,252	58,116	90,902
5	1,549	X	X	491,611	3,161	1,528,321	154	X	X	39,334	71,293	112,998
6	2,401	X	X	1,238,549	3,345	6,449,518	150	X	X	58,020	81,304	130,340
7	2,678	X	X	6,344,033	4,552	16,610,175	150	X	X	79,854	88,398	143,242
8	3,852	X	X	4,898,796	3,850	44,705,509	144	X	X	79,854	93,007	152,212
9	2,481	X	X	6,244,168	1,180	44,579,649	233	X	X	79,854	95,664	157,935
10	3,282	X	X	6,362,273	1,452	X	230	X	X	108,659	96,988	161,176
11	3,482	X	X	3,136	9,498	X	-	-	-	-	-	-
12	3,019	X	X	8,767	44,330	X	-	-	-	-	-	-
13	3,281	X	X	X	12,570	X	-	-	-	-	-	-
14	3,419	X	X	X	55,265	X	-	-	-	-	-	-
15	3,919	X	X	X	15,115	X	-	-	-	-	-	-
16	3,819	X	X	X	64,996	X	-	-	-	-	-	-
17	4,019	X	X	X	1,587	X	-	-	-	-	-	-
18	4,149	X	X	X	X	X	-	-	-	-	-	-
19	4,349	X	X	X	X	X	-	-	-	-	-	-
20	5,320	X	X	X	X	X	-	-	-	-	-	-

Table 2: Number of explored states for each problem in our test suite (”X” - planner ran out of memory).
Labels: DiNo-T = DiNo with Partial TPDB, DiNo-S = DiNo with SRPG+, UPM = UPMurphi

surface, the change in vertical velocity is dependent on the spacecraft's mass and duration of the thrust. The domain contains complex highly non-linear system dynamics modelled using Tsiolkovsky Rocket Equation (Turner 2008).

Car. The Car domain (Fox and Long 2006) is a well-known benchmark model in the planning community. It models a vehicle which can accelerate and decelerate, and whose goal is to travel a given distance and come to a complete stop at a precise point. The domain relies heavily on processes, and actions only affect the goal facts indirectly.

The results show that the partial TPDB heuristic enriches DiNo, and allows it to solve more constrained and complex problems. We notice that DiNo-TPDB does particularly well with domains heavily relaying on the Theory of Waiting (McDermott 2003b), such as the Generator and the Advanced Solar Rover. In those cases the Pruning Jump significantly reduces the size of the search space, improving the performance. Overall, DiNo either outperforms, or is competitive, on all domains in the extensive test suite. As an added note, SMTPlan+'s performance is very high, though some domains had system dynamics too complex for its solver or simply would not return a solution. Ultimately, DiNo-TPDB outperforms its predecessor (DiNo-SRPG+) on all but one test domain. Furthermore, DiNo-TPDB scales very well compared to other PDDL+ planners (Table 2).

Powered Descent is very time-sensitive and there are major discrepancies between the concrete states and the abstract states in the TPDB. In this domain missing a single time unit can render the plan invalid, though this will become apparent to the planner only at the end of the trajectory, inducing significant backtracking.

The car domain proved somewhat problematic for the TPDB, as the numeric values have very different rates of change, generating mismatches between theoretically corresponding states.In this domain, DiNo-TPDB makes heavy use of the back-up strategy to find a viable solution.

6 Conclusion

We presented Temporal Pattern Database (TPDB), a novel domain-independent heuristic capable of handling complex non-linear PDDL+ models exhibiting both discrete and continuous behaviour. The TPDB stores pairs of abstract states and actions, and uses a solution to the abstracted and relaxed version of the original problem as guidance to solving the concrete problem. We have also inherited from literature, and exploited, the concept of Partial Temporal Pattern Database, a downscaled variant of the TPDB, which improves the performance of the planner by focusing only on part of the abstract search space during the pre-processing phase. We have empirically shown that the DiNo-TPDB is competitive on benchmark domains and, in general, outperforms other PDDL+ planners. Our heuristic combines and extends approaches used in planning and model checking, and it is an important step in PDDL+ planning. Future research will concentrate on automating the selection of discretisation and abstraction settings.

References

Anderson, K.; Holte, R.; and Schaeffer, J. 2007. Partial Pattern Databases. In *International Symposium on Abstraction, Reformulation, and Approximation*, 20–34. Springer.

Bogomolov, S.; Frehse, G.; Grosu, R.; Ladan, H.; Podelski, A.; and Wehrle, M. 2012. A Box-Based Bistance Between Regions for Guiding the Reachability Analysis of SpaceEx. In *Computer Aided Verification*, 479–494. Springer.

Bogomolov, S.; Donzé, A.; Frehse, G.; Grosu, R.; Johnson, T. T.; Ladan, H.; Podelski, A.; and Wehrle, M. 2013. Abstraction-Based Guided Search for Hybrid Systems. In *Model Checking Software*. Springer. 117–134.

Bogomolov, S.; Magazzeni, D.; Podelski, A.; and Wehrle, M. 2014. Planning as model checking in hybrid domains. In *AAAI*, 2228–2234.

Bryce, D.; Gao, S.; Musliner, D. J.; and Goldman, R. P. 2015. SMT-Based Nonlinear PDDL+ Planning. In *AAAI*, 3247–3253.

Cashmore, M.; Fox, M.; Long, D.; and Magazzeni, D. 2016. A Compilation of the Full PDDL+ Language into SMT. In *ICAPS*.

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.

Culberson, J. C., and Schaeffer, J. 1998. Pattern Databases. *Computational Intelligence* 14(3):318–334.

Della Penna, G.; Magazzeni, D.; Mercorio, F.; and Intrigila, B. 2009. UPMurphi: A Tool for Universal Planning on PDDL+ Problems. In *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 Kissmann, P. 2008. Partial symbolic pattern databases for optimal sequential planning. In *Annual Conference on Artificial Intelligence*, 193–200. Springer.

Edelkamp, S. 2002. Symbolic Pattern Databases in Heuristic Search Planning. In *AIPS*, 274–283.

Edelkamp, S. 2014. Planning with Pattern Databases. In *Sixth European Conference on Planning*.

Fernández-González, E.; Karpas, E.; and Williams, B. C. 2015. Mixed discrete-continuous heuristic generative planning based on flow tubes. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 1565–1572.

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

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning. In *AAAI*, volume 7, 1007–1012.

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., and Long, D. 2003. VAL's Progress: The Automatic Validation Tool for PDDL2. 1 Used in the International Planning Competition. In *Proc. of ICAPS Workshop on the IPC*, 28–37.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *ICTAI 2004*, 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 *ICAPS*, 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. 2003a. Reasoning about Autonomous Processes in an Estimated-Regression Planner. In *ICAPS*, 143–152.

McDermott, D. V. 2003b. 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.

Piotrowski, W.; Fox, M.; Long, D.; Magazzeni, D.; and Mercorio, F. 2016. Heuristic Planning for PDDL+ Domains. In *IJCAI*, 3213–3219.

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

Sievers, S.; Ortlieb, M.; and Helmert, M. 2012. Efficient Implementation of Pattern Database Heuristics for Classical Planning. In *SOCS*.

Turner, M. J. 2008. *Rocket and Spacecraft Propulsion: Principles, Practice and New Developments*. Springer Science & Business Media.