PDDL+ Planning with Events and Linear Processes

Amanda Coles and Andrew Coles
Department of Informatics,
King's College London, WC2R 2LS UK
email: firstname.lastname@kcl.ac.uk

Abstract

In this paper we present a scalable fully-automated PDDL planner capable of reasoning with PDDL+ events and linear processes. Processes and events model (respectively) continuous and discrete exogenous activity in the environment, occurring when certain conditions hold. We discuss the significant research challenges posed in creating a forward-chaining planner that can reason with these, and present novel state-progression and consistency enforcing techniques that allow us to meet these challenges. Finally we present results showing that our new planner, using PDDL+ domain models, is able to solve realistic expressive problems more efficiently than the current state-of-the-art alternative: a compiled PDDL 2.1 representation with continuous numeric effects.

1 Introduction

Classical planning has traditionally been concerned with reasoning about a static world in which the effects of actions occur instantaneously. The reality of the world in which plans must be executed is, however, often different to this: numeric quantities change over time and exogenous happenings occur, both in response to, and independently of, the actions carried out in the plan. For example, at sunrise the battery charge of a space vehicle begins to increase continuously over time, this increase does not depend upon the vehicle taking any specific action, it happens automatically.

Even in the absence of exogeny, scalable automated planning in the presence of continuous numeric change has only recently become a possibility, due to advances in classical and temporal planning. While there was some early work on planning with such models, notably the planners Zeno (Penberthy and Weld 1994) and OPTOP (McDermott 2003b), the challenge of efficiently computing effective heuristics severely restricted scalability. Following the introduction of continuous numeric change into version 2.1 of the standard planning domain modelling language, PDDL, (Fox and Long 2003) a number of modern planners began to address the challenge of reasoning with continuous numeric change.

The planner COLIN (Coles et al. 2012) performs forward-chaining search and uses a mixed integer program (MIP) to ensure that the constraints arising due to the interaction of continuous numeric variables are met. POPF (Coles et al. 2010) extends COLIN to reason with partially ordered plans, and forms the basis for this work. Kongming (Li and Williams 2011) uses a planning graph based structure to build plans, making use of a proprietary language to specify continuous dynamics. It also uses a MIP to manage temporal and numeric constraints, but is less expressive than COLIN in the sense that it does not allow two actions to simultaneously change the value of a variable.

To date there are only two planners that are capable of reasoning with discrete and continuous change caused by both actions and exogenous happenings as described in PDDL+ (Fox and Long 2006). TM-LPSAT (Shin and Davis 2005) is a fully automated planner that can solve PDDL+ planning problems with linear continuous change. It uses a SAT-based compilation, giving a discrete set of time points; and, like COLIN, uses an LP solver to manage numeric constraints. Its approach shows promise, but empirically, suffers from scalability issues. UPMurphi (Penna et al. 2009) takes a model-checking approach but relies on a hand-crafted discretisation of time to reason with continuous change. The use of a discretisation allows it to handle non-linear continuous change, the only planner to do so, but of course requires human expertise. The main challenge for UPMurphi is scalability as it it has no heuristic for guidance.

In this paper we present a scalable forward-chaining planner capable of reasoning with linear continuous change and exogenous happenings. By building on state-of-the-art approaches to planning with continuous numeric change, we avoid the need to discretise time, with the consequence of improved scalability. Avoiding discretisation introduces new challenges in ensuring that exogenous happenings occur immediately when their conditions hold and that their conditions are avoided if they are not desired. We discuss how we overcome these challenges and empirically demonstrate the scalability of our planner on PDDL+ problems.

2 Problem Definition

The logical basis for temporal planning, as modelled in PDDL 2.1 (Fox and Long 2003), is a collection of propositions P, and a vector of numeric variables $\mathbf{v}$. These are manipulated and referred to by actions. The executability of actions is determined by their preconditions. A single *condition* is either a single proposition $p \in P$, or a numeric constraint over $\mathbf{v}$. We assume all such constraints are linear, and hence can be represented in the form:

$$\mathbf{w}.\mathbf{v}\{>, \geq, <, \leq, =\}c$$

(**w** is a vector of constants and c is a constant). A *precondition* is a conjunction of zero or more conditions.

Each durative action A has three sets of preconditions: $\text{pre}_{\vdash} A$, $\text{pre}_{\leftrightarrow} A$, $\text{pre}_{\dashv} A$. These represent the conditions that must hold at its start, throughout its execution, and at the end of the action, respectively. Instantaneous effects can then be bound to the start or end of the action. $\text{eff}^{+}_{\vdash} A$ and $\text{eff}^{-}_{\vdash} A$ denote the propositions added and deleted at the start of A, and $\text{eff}^{num}_{\vdash} A$ denotes any numeric effects. Similarly, $\text{eff}^{+}_{\dashv} A$, $\text{eff}^{-}_{\dashv}$ and $\text{eff}^{num}_{\dashv}$ record effects at the end. We assume all such effects are linear, i.e. are of the form:

$$v\{\text{+=}, \text{-=}, \text{=}\}\mathbf{w}.\mathbf{v} + c \qquad \text{where } v \in \mathbf{v}$$

Semantically, the values of these instantaneous effects become available small amount of time, ϵ, after they occur.

Each action additionally has a conjunction of *continuous* numeric effects $\text{eff}_{\leftrightarrow}$, of the form dv/dt=$c, c \in \Re$, that occur while it is executing[1]. Finally, the action has a duration constraint: a conjunction of (assumedly linear) numeric constraints applied to a special variable dur_A corresponding to the duration of A. As a special case, *instantaneous* actions have duration ϵ, and have only one set of preconditions pre A and effects $\text{eff}^{+} A$ and $\text{eff}^{-} A$. For use in reasoning a durative action A can be split into two instantaneous *snap* actions, $A_{\vdash}$ and $A_{\dashv}$, representing the start and end of the action respectively, and a set of constraints (invariant and duration constraints and continuous numeric effects). Action $A_{\vdash}$ has precondition $\text{pre}_{\vdash} A$ and effects $\text{eff}^{+}_{\vdash} A \cup \text{eff}^{-}_{\vdash} A \cup \text{eff}^{num}_{\vdash} A$. Likewise, $A_{\dashv}$ is the analogous action for the end of A.

A PDDL+ planning problem augments a PDDL planning problem with processes and events. Like actions, these have preconditions, and effects. As an analogue, events are akin to instantaneous actions: if an event's preconditions pre A are satisfied, it occurs, yielding the event's instantaneous effects. Similarly, processes are akin to durative actions, with $\text{pre}_{\leftrightarrow} A$ corresponding to the process' precondition, and $\text{eff}_{\leftrightarrow}$ containing its continuous numeric effects. Then, while $\text{pre}_{\leftrightarrow} A$ is satisfied, the continuous numeric change occurs. Thus, the critical distinction between processes and events, and actions, is that a process/event will *automatically* occur as soon as its precondition is satisfied, modelling exogenous activity in the environment; whereas an action will only happen if chosen to execute in the plan.

PDDL+ has a number of problematic features that make the plan validation problem is intractable, even when the language is restricted to linear continuous change. In particular, in theory, events can infinitely cascade, repeatedly firing and self-supporting. Or, having reached the goals, it is challenging to determine whether they persist from then onwards, given future processes and events that may occur. To address the former of these issues, we make the restriction proposed by Fox and Long (2006) that events must delete one of their own preconditions. For the latter, we require that, if persistence is desired, the goal specified is sufficient to ensure the desired goals persist. Note that a goal required to be true beyond a specified fixed time, but not necessarily persist, can be modelled by using a process to count time and adding $time > time\_required$ to the goal.

[1]We allow c to be derived from mathematical operations on constant-valued state variables.

3 Running Example

We introduce a simple small example problem based on the use of a mobile phone (cellular phone). The scenario is as follows: a person initially in the countryside with his phone switched off must go to the city and make a call from there using his mobile phone (i.e. the goal is *called*[2]). The domain has three durative actions:

- **travel**: dur = 15; $\text{pre}_{\vdash}$ = {at country}; $\text{eff}^{-}_{\vdash}$ = {at country}; $\text{eff}^{+}_{\dashv}$ = {at city}; $\text{eff}_{\leftrightarrow}$ = $\{d(signal)/dt = 0.5\}$
- **turn_on**: dur > 0; $\text{pre}_{\vdash}$ = $\{\neg\text{on}\}$; $\text{eff}^{+}_{\vdash}$ = {on}; $\text{eff}^{-}_{\dashv}$ = {on}; $\text{eff}_{\leftrightarrow}$ = $d(battery)/dt$ = -1
- **call**: dur=1; $\text{pre}_{\vdash}$={at city $\wedge$ battery > 1}; $\text{eff}^{+}_{\dashv}$={called};

There is also a process, which models the transfer of data over the network at a fixed rate, if certain conditions are met:

- **transfer**: $\text{pre}_{\leftrightarrow}$ = {on $\wedge$ battery $> 10 \wedge$ signal > 5}; $\text{eff}_{\leftrightarrow}$ = $d(data)/dt$ = 1

Finally, an event models a low-battery warning:

- **warning**: pre = {$\neg$warned$\wedge$battery< 8}; eff = {warned}

4 PDDL+ versus PDDL 2.1

In this section we further explore the relationship between PDDL+ processes and events and their PDDL 2.1 counterparts: durative-actions and (instantaneous) actions.

First, we observe that, at any time, each process p_i (with precondition C_i and effects $\text{eff}_{\leftrightarrow} p_i$) is either executing, or not; i.e. either C_i or $\neg C_i$. We might therefore consider there to be two durative-actions for p_i, rather than one:

- run_p_i, with $\text{pre}_{\leftrightarrow}$run_$p_i$=$C_i$, and $\text{eff}_{\leftrightarrow}$run_$p_i$=$\text{eff}_{\leftrightarrow} p_i$;
- not-run_p_i, with $\text{pre}_{\leftrightarrow}$not-run_$p_i$=$\neg C$, and no effects;
- in both cases, the duration of the action is in $[\epsilon, \infty]$.

If we could somehow then ensure that we only ever apply run_$p_{i\dashv}$ (the end of run_p_i) if we simultaneously apply not-run_$p_{i\vdash}$ – and vice-versa – then the behaviour of the process has been simulated with actions. Whenever the truth value of C_i changes (which may be many times) we simply switch which one of these two actions is executing. Ensuring this switch happens simultaneously is crucial: if time was allowed to pass between e.g. not-run_$p_{i\dashv}$ and run_$p_{i\vdash}$ then there would be a period during which C_i might be true, but the effect of p_i is not being captured by any executing action.

We also observe, that each event e_j with precondition C_j and effects $\text{eff}\, e_j$, at any point it is either happening at that time, instantaneously; or its conditions are false. Precisely:

- As events occur as soon as their preconditions are satisfied, there is a period prior to e_j during which $\neg C_j$ holds;
- When the event occurs, C_j is true – and, as noted earlier, events must delete one of their own preconditions;
- Thus, begins again a period in which $\neg C_j$ holds.

This could be captured with the use of synchronisation actions, similar to the mechanism postulated for processes. A non-temporal action e_j, with the preconditions and effects of e_j represents the event itself. Then, only one durative action is needed – not-do_e_j, with $\text{pre}_{\leftrightarrow}$not-do_$e_j$=$\neg C_j$. This

[2]Persistence is guaranteed: no action or event deletes this fact.

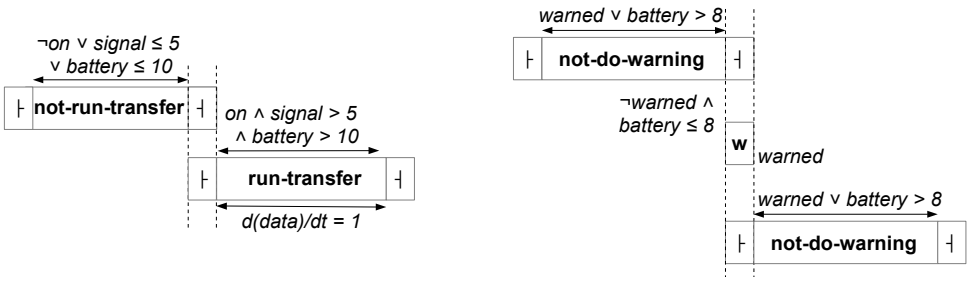

Figure 1: Representing Processes (left) and Events (right). Dotted lines denote synchronised actions.

captures the C_j intervals, with not-do_e_j ending and immediately re-starting at exactly the time e_j occurs.

Returning to our running example, the left of Figure 1 shows how synchronised actions can represent the 'transfer' process transitioning from not-running to running. To be not running, one of the terms in its precondition must be false; to be running, they must all be true. Synchronisation (dotted lines) then ensures this transition occurs at the right time. The right of Figure 1 shows the 'warning' event, abbreviated to 'w'. (The fact 'warned' ensures the event only fires once.) As can be seen, w is synchronised with the 'not-do-warning' durative-actions, again ensuring it occurs at the right time: the first point at which its precondition was met, and no later.

4.1 Achieving Synchronisation in PDDL 2.1

If we wish to reason with processes and events using a PDDL 2.1 planner we must use a compilation to enforce correct synchronisation. In fact, there are three requirements:

1. Synchronisation (as in Figure 1);
2. Ensuring that not-run_p_i (or run_p_i) and not-do_e_j are started *immediately*, at the start of the plan;
3. Allowing processes/event actions to finish in goal states.

We can achieve **synchronisation (1)**, through the use of *clip* actions (Fox, Long, and Halsey 2004):

Action 4.1 — clip_f
A (tight) clip for fact f, and auxiliary facts $\{fe_0, \ldots, fe_n\}$, is a durative action A, with duration 2ϵ, where:

- pre$_{\vdash}A = \neg f$, eff$^{+}_{\vdash}A = f$;
- pre$_{\dashv}A = \{fe_0, \ldots, fe_n\}$, eff$^{-}_{\dashv}A = \neg f, \{\neg fe_0, \ldots, \neg fe_n\}$.

f is thus only available, instantaneously, at time ϵ after starting clip_f; before immediately being deleted. If two snap-actions with a condition on f are placed inside the clip, they must therefore be synchronised: there is only one point during the clip at which their condition is met. The other aspect of a clip is its auxiliary facts, which must be true before it ends. If there are n+1 snap-actions that must occur during the clip, then each of these is given a distinct fe fact as an effect. Thus, not only must the snap-actions occur at the same time; but also, no necessary snap-action can be omitted.

To synchronise the actions from Section 4 in PDDL2.1, we use a clip for each process or event. For each process p_i:

- Create clip_f_i, with auxiliary facts $f_i e_0$, $f_i e_1$;
- Add f_i to $pre_{\vdash}$ and $pre_{\dashv}$ of run_p_i and not-run_p_i;
- Add $f_i e_0$ to $eff^{+}_{\dashv}$ of run_p_i and not-run_p_i.
- Add $f_i e_1$ to $eff^{+}_{\vdash}$ of run_p_i and not-run_p_i;

Similarly, for event e_j:

- Create clip_f_j, with auxiliary facts $f_j e_0$, $f_j e_1$, $f_j e_2$;
- Add f_i to $pre\ e_j$, and to $pre_{\vdash}$ and $pre_{\dashv}$ of not-do_e_j;
- Add $f_j e_0$ to $eff^{+}_{\dashv}$ not-do_e_j;
- Add $f_j e_1$ to $eff^{+}_{\vdash}$ not-do_e_j;
- Add $f_i e_2$ to $eff^{+}\ e_j$.

In both of these cases, the clip meets the objectives of Figure 1: by the use of 2 (resp. 3) auxiliary facts, the correct actions must occur within the clip; and due to the tight availability of f_i (f_j), the actions must be synchronised. Note that search using this compilation can be made slightly more efficient through the use of two clip actions for each process: one forcing a change from run-p_i to not-run-p_i and the other vice-versa. (The clip, as presented here, permits clipping run-p_i to run-p_i, and not-run-e_j to itself, which is pointless.)

The issue with the compilation as it stands is that a clip cannot end, unless an already-executing 'run', 'not-run' or 'not-do' action ends inside it: each of these adds the zero'th auxiliary fact of the clip, which is an end-condition of its execution. This is desirable in the general case, but in the initial state no such actions are executing. This brings us to the second of our requirements here, *viz.* **starting the actions immediately (2)**. For this, we use:

- A Timed Initial Literal (TIL) (Hoffmann and Edelkamp 2005) go, which is true initially and deleted at time ϵ, creating a small window at the start of the plan in which go is available. (NB go is not added by any action/event/TIL.)
- A TIL $exec$, appearing at time ϵ, and added as an 'at start' condition of every non-clip action in the plan.

We create a single 'go' clip allowing all process/event tracking actions to begin. We collate all the clip facts into a set F. We define the set FE_1 as the set of all '1' auxiliary facts (i.e. each $f_i e_1$ or $f_j e_1$). The go clip is defined thus:

Action 4.2 — go-clip
A 'go' clip for clip facts F, and 1-auxiliary fact set FE_1, is a durative action A, with duration 2ϵ where:

- pre$_{\vdash}A = go \wedge \forall f \in F\ \ \neg f$, eff$^{+}_{\vdash}A = F$;
- pre$_{\dashv}A = FE_1$, eff$^{-}_{\dashv}A = \forall f \in F\ \ \neg f$.

The condition go ensures the go-clip can only occur at time zero; and $exec$ ensures it precedes all other actions.

The final piece of the puzzle is to allow the run/not-run/do actions to **terminate once the goals have been met (3)** –

the semantics of PDDL 2.1 require there to be no executing actions in goal states. For this, we use a final, modified clip action – a 'goal-clip'. FE_0 is analogous to FE_1 – but for '0' facts – and $FE_{>0}$ contains all auxiliary facts not in FE_0.

Action 4.3 — goal-clip
A 'goal' clip for clip facts F, auxiliary fact sets FE_0 and $FE_{>0}$, in a problem with goal G, is a durative action A, with duration 2ϵ, where:

- $\text{pre}_{\vdash} A = \forall f \in F \ \ \neg f$, $\text{eff}^{+}_{\vdash} A = F$, $\text{eff}^{-}_{\vdash} A = exec$;
- $\text{pre}_{\dashv} A = G \ \ \wedge FE_0 \ \ \wedge \forall fe \in FE_{>0} \ \ \neg fe$, $\text{eff}^{+}_{\dashv} A = goal\_reached$, $\text{eff}^{-}_{\dashv} A = \forall f \in F \ \ \neg f$.

As a final note we observe that the negation of conjunctive conditions on processes/events results in disjunctive invariants on the not-run/not-do actions. In the absence of a planner supporting these, it is possible to create several not-run actions, each with an invariant comprising a single condition from the disjunction. Clips can then be used to switch between not-run actions to change which condition in the disjunct is satisfied. This has implications when using the efficient clip model (distinct clips for switching from run to not-run, and vice versa) – we must also allow different not-run actions to be clipped to each other. This does not, however, negate the benefits of the efficient clip model.

While this compilation to PDDL 2.1 is possible, it is clearly a very unnatural model, and still requires a highly expressive planner. Indeed several authors have argued that the model adopted in PDDL+ is much more natural than the previous model (McDermott 2003a; Boddy 2003). Further, it is likely to make search computationally inefficient: not only is the planner forced to reason about the exogenous actions within the environment as if they were real planning actions, many extra 'book-keeping' actions are added to the domain. If there are n processes and m events then $3n + 2m + 2$ actions are added to the planning problem, of which $(m + n)$ are applicable in each state. This massively the branching factor and the length of solution plans. Permutations of such actions can also cause significant problems in temporal planning (Tierney et al. 2012). It therefore seems that the native handling of processes and events is, in theory, far more efficient – and this forms the focus of the rest of the paper. We will, of course, return to this point in our evaluation.

5 Forward Chaining Partial-Order Planning

In this work, we build upon the planner POPF (Coles et al. 2010). POPF uses an adaptation of a forward-chaining planning approach where, rather than placing a total-order on plan steps, the plan steps are partially ordered: ordering constraints are inserted on an as-needed basis. To support this partial-ordering, additional information is stored in states, associated with the facts and variable values. For facts p:

- $F^{+}(p)$ ($F^{-}(p)$) records the index of the step that last added (deleted) p;
- $FP^{+}(p)$, a set of pairs, each $\langle i, d\rangle$, notes steps with a precondition p: i is the plan step index, and $d \in \{0, \epsilon\}$. If d=0, p can be deleted at or after step i; if d=ϵ, p can be deleted from ϵ after i.
- $FP^{-}(p)$, similarly, records negative preconditions on p.

For the vector of state variables $\mathbf{v}$, the state records lower- and upper-bound vectors, V^{min} and V^{max}. These reflect the fact that in the presence of continuous numeric change, a variable's value depends on the time; and hence, having applied some actions, a range of variable values are possible. With each $v \in \mathbf{v}$ the state also notes:

- $V^{\mathit{eff}}(v)$, the index of the most recent step to affect v;
- $VP(v)$, a set of plan step indices, of steps that have referred to v since the last effect on v. A step depends on v if either: it has a precondition on v; an effect whose outcome depends on v; or is the start of an action with a duration depending on v.
- $VI(v)$, a set of plan step indices, of the start of actions that are currently executing but have not yet finished; and have an over all condition on v.

The actions applied during search are snap-actions, corresponding to either instantaneous actions; the start of a durative action; or ending a (currently executing) durative action. When a snap-action is applied, the temporal constraints recorded are derived from the annotations: the new plan step is ordered after each fact or variable the action refers in its conditions, effects, and duration. Similarly, to avoid interference with existing plan steps, if the action deletes (adds) p it is ordered after each $FP^{+}(p)$ (resp. $FP^{-}(p)$); or if it affects v (either instantaneously, or by starting/ending a continuous numeric effect on v) it is ordered after each $VP(v)$, and after each $VI(v)$. Finally, in addition to these, constraints are added to respect the duration constraints on actions.

In the absence of continuous (or duration-dependent) numeric effects, the temporal-constraint consistency in POPF can be checked with a simple temporal network. However, with linear continuous numeric effects, a MIP solver is required for the resulting temporal–numeric constraints.

For each step i, $t(i)$ records its time-stamp, and the temporal constraints are encoded directly over these variables. Additionally, for each i, the variables $v_i \in V_i$ record the values of each of $\mathbf{v}$ prior to i for variables referred to in the preconditions/effects/duration constraints of step i). Likewise, $v'_i \in V'_i$ record the variable values immediately following i. The numeric preconditions and effects of actions are then added as constraints on these:

- preconditions at i form constraints over V_i;
- the invariants of i form constraints over V'_i if i is a start snap-action; or over V_i if it is an end snap-action.
- instantaneous numeric effects form constraints relating V_i to V'_i; for instance, $v'_i = v_i + x_i$ records that at step i, v is increased by the value of variable x.

In addition to the constraints for i itself, if i has an effect on v, it will be ordered after each $VI(v)$: the steps that have invariants on v. These invariants need to be enforced at step i: although they do not belong to i itself, they belong to currently executing actions, and we need to ensure i does not adversely interfere with these. Thus, v_i and v'_i are constrained to obey these invariants. This may necessitate the addition of extra ordering constraints, if an invariant to be enforced refers to a variable not otherwise relevant to the action. Hereon, if we state that an invariant must be *enforced* at step i, we mean that v_i and v'_i must be constrained

step	variables	constraint
turn_on$_\vdash$	t_0	≥ 0
	$battery_0$	$= 30$
	$battery'_0$	$= battery_0 \wedge > 0$
travel$_\vdash$	t_1	≥ 0
	$signal_1$	$= 0$
	$signal'_1$	$= signal_1$
travel$_\dashv$	t_2	$= t_0 + 15$
	$signal_2$	$= signal'_1 + 0.5 * (t_2 - t_1)$
	$signal'_2$	$= signal_2$
now	$battery_{now}$	$= battery'_0 - 1 * (t_{battery-now} - t_0)$
	$t_{battery-now}$	$> t_0$
	$signal_{now}$	$= signal'_2$
	$t_{signal-now}$	$> t_2$

Table 1: Example POPF MIP

to obey the invariant, and any extra ordering constraints must be added. For conjunctive invariants this neatly exploits the monotonicity of the continuous numeric change supported: for some interval in which the invariant on v must hold, it suffices to check the condition at the start and end of consecutive intervals, bounded by either the action to which the invariant belongs, or between successive effects on v. If, for example, $v \geq 5$ at the start and end of an interval, then under monotonic change it must have been true throughout.

To capture the interaction between time and variable values, the final consideration is the continuous numeric change that occurs over time. During MIP construction, at each point referring to v, the sum of the gradient effects δv acting on a variable v are noted. As the continuous numeric change is linear, and any changes to δv are totally ordered, at each point this is a constant value, known at the time the MIP is built. With $\delta v'_i$ denoting the gradient active after step i (assuming i refers to v), the value of v at a future step j is:

$$v_j = v'_i + \delta v'_i(t(j) - t(i)) \quad (1)$$

To illustrate the MIP built we return to our running example; since POPF does not handle processes/events we remove transfer and warning to demonstrate POPF's MIP. Table 1 shows the MIP that would be built in the state following the addition of travel$_\vdash$, turn_on$_\vdash$ and travel$_\dashv$ to the plan. Notice that each step only has MIP variables representing variables in its conditions/effects, or if an invariant is enforced; and it is only ordered w.r.t. other steps that affect or condition on the same variables/propositions. Because this ordering is guaranteed it is possible to compute $\delta' v_i$ at each t_i that has an effect on v, by working through the plan. In this example, $\delta signal'_0$=0.5, $\delta battery'_1$=-1 and $\delta signal'_2$=0.

The *now* steps are additionally added to the MIP, to allow the computation of the upper and lower bounds on state variables: for each numeric problem variable we ask the MIP solver to maximise and then to minimise its corresponding MIP variable and use these as the bounds (for brevity, we omit these from future MIPs in the paper). A solution to the MIP represents a valid setting of the timestamps of plan actions $t_0 ... t_n$ that respects all of the temporal and numeric constraints. If no such solution exists the plan to reach that state cannot be scheduled and the state can be pruned.

6 Search with Processes and Events

In Section 4 we detailed how synchronised actions can in principle be used for processes and events; and then in Section 4.1 detailed how this forms the basis of a compilation to PDDL 2.1 The drawback of this compilation is the search-space blow-up it entails. In this section we present an alternative approach: modifying a forward-chaining planning approach, to eliminate many of the artificial planning decisions entailed by the compilation.

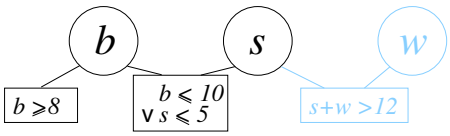

Figure 2: Condition–Variable Dependency Graph

6.1 Managing Invariants of Processes/Events

The run, not-run and not-do actions introduced by processes and events are, for many purposes, normal durative actions, with invariants – the only difference is the necessary synchronisation constraints. Thus, first, we need to consider how to decide when such invariants need to be enforced during planning. The basic approach in POPF was described in Section 5 – we build on this here.

The subset of invariants chosen by POPF to be enforced at a given step is sound if all invariants are conjuncts of single-variable constraints, e.g. $(battery > 10) \wedge (signal > 5)$. In other words, there is a direct relationship between the variables an action affects, and the constraints that need to be enforced. However, there are two cases where POPF, as it stands, cannot handle numeric invariants. These limitations only arise if variables referred to in the invariant have been subject to continuous numeric change, but in the context of processes, this is almost a certainty. POPF cannot handle:

- Invariants that are multi-variate e.g. $signal + wifi > 12$;
- Disjunctive invariants e.g. $(signal \leq 5) \vee (battery \leq 10)$.

The latter of these is particularly problematic with processes and events: even if a process/event has a condition that is a conjunct of terms, taking the negation of this, to mark the intervals during which the process/event is not occurring, leads to a disjunction (c.f. De Morgan's laws).

To handle such invariants, we require a more general solution to numeric invariants in POPF. Fundamental to our approach is a condition–variable dependency graph, built from the invariants of the currently executing actions. An example of such a graph is shown in Figure 2 – this is based on our running example, with an additional action whose invariant is $signal + wifi > 12$, the variable names are abbreviated to b,s, and w. The vertices are:

- One variable vertex for each numeric variable in the problem (in our example, b, s, w);
- $n+1$ constraint vertices for each invariant $C = (c_0 \wedge \ldots \wedge c_n)$, one for each term $c_i \in C$;
- One constraint vertex for each invariant $C = (c_0 \vee \ldots \vee c_n)$, containing the entire constraint C.

An edge is added between a constraint vertex and a variable vertex if the constraint refers to that variable. With this graph, we then have a straightforward way of ascertaining the *indirect* relationships between variables, that arise due to

disjunctive and multi-variate invariants. Simply: if a snap-action has an effect on a variable v, then any condition (invariant) that can be reached in the graph from v needs to be enforced at the point when the snap-action is applied.

We return to our running example to illustrate why this mechanism is necessary. Suppose we are in a state where the currently executing actions are turn_on, travel, not-run_transfer and not-do_warning. The condition–variable dependency graph comprises the dark (black) portion on the left of Figure 2. As turn_on and travel both have continuous numeric effects, the values of *battery* and *signal* are not fixed – they depend on the timestamps given to the actions in the plan, their ranges, as evaluated by the MIP, are $battery \in [0, 30]$ and $signal \in [0, 7.5]$. Suppose the action travel$_\dashv$ is then to be applied – which refers to the variable *signal* in its effects. With the prior mechanism of POPF:

- the condition that would be enforced is $(battery \leq 10) \vee (signal \leq 5)$ – as it refers to *signal* (we omit '$\ldots \vee \neg on$' from this discussion since *on* is known to be true);
- as the constraint is disjunctive, it could be satisfied by assuming for instance, *battery*=5: a value that lies within its range in the current state.

However, from the graph we see that if restrictions are made on the value of b, this may impact other conditions; in fact, assuming b=5 is incompatible with the invariant $b \geq 8$. This would, however, be captured by the new mechanism: upon referring to s, all reachable conditions are enforced, including those on b, due to the disjunctive constraint.

As an illustration of why the new mechanism is needed for multi-variate conditions too: we have the additional invariant shown on the right of Figure 2. Suppose an action is being applied that assigns w=5. This necessitates enforcing the invariant s+$w > 12$. With the prior mechanism of POPF, we could assume $s > 7.1$, which is within its range in the current state. But, from the graph we can see that constraints on s also need to be enforced. Notably, $s \leq 5$ can no longer be true if $s > 7.1$; and hence $b \leq 10$ has to be true; which, in turn, may impact whether $b \geq 8$ can be true (indeed had the condition on warning been, for example, $b \geq 12$ search would need to backtrack at this point). Thus, even though the action only affected w, the multi-variate and disjunctive invariants lead to indirect relationships with s and b, too.

6.2 Ordering Implications

A side effect of enforcing extra invariants is the addition of extra ordering constraints when adding actions to the plan. In our running example, when the action turn_on$_\vdash$ has been applied, and we consider applying travel$_\vdash$, the disjunctive invariant condition $\neg on \vee battery \leq 10 \vee signal \leq 5$ must be enforced. This leads to additional ordering constraints: using the POPF state-update rules, to establish the value of *battery* for the purposes of enforcing this invariant, travel$_\vdash$ will be ordered after turn_on$_\vdash$. This would not have been the case had the disjunctive invariant not been enforced, as travel$_\vdash$ does not otherwise refer to *battery*. Note that completeness is not compromised as the state arising from applying these actions in the opposite order still appears as a distinct state in the search space. The practical effect of the ordering constraints is to impose a total order on actions affecting any variable in a set of connected variables in the condition-variable dependency graph. In the running example that means any action affecting b, s or w will be ordered with respect to any other action affecting b, s or w.

This has useful implications on our obligations to enforce disjunctive invariants. As a result of these orderings, we guarantee that we only need to maintain an invariant C=$(c_0 \vee \ldots \vee c_n)$ between plan steps at which C is enforced, and between which no step exists that could affect of the set of variables V_C (those referred to by any $c_f \in C$).

To understand this, suppose this invariant C became active at step i. Any action a_k with an effect on any $v \in V_C$ is totally ordered after the previous such action (c.f constraints introduced from condition-variable dependency graph). Let us name this totally ordered collection of actions A_C=$[a_0, \ldots, a_m]$, where $t(a_k) < t(a_{k+1})$, and $t(i) < t(a_0)$. At each a_k, C is enforced (when a_k is added to the plan). Therefore, when adding a new action a to the plan, we need only record the obligation to enforce the invariant at $t(a_m)$ (if A_C is not empty), and at $t(a)$: where $t(a_m)$ and $t(a)$ are adjacently ordered steps. Further, we know that no other action affecting any $v \in V_C$ occurs between $t(a_m)$ and $t(a)$: if such an action was added to the plan before a_m it would be ordered in A_C before a_m; and if it is later added, after a, it will be ordered in A_C after a.

6.3 Maintaining Disjunctive Invariants

In Section 5 we observed that in order to enforce a conjunctive invariant it is sufficient to *enforce* the invariant at the start and end of the interval over which it is required. This is not, however, sufficient for disjunctive invariants.

Consider, for example, meeting the invariant $(battery \leq 10) \vee (signal \leq 5)$ (the condition of not-run_transfer) during the interval between the actions travel$_\vdash$ and travel$_\dashv$, hereon step i and step j. This scenario is shown in Table 2. At this point in the plan, *battery* is decreasing and *signal* is increasing. If we simply insist that the disjunction is true at each end, we can rely on $(signal \leq 5)$ at the start and $(battery \leq 10)$ at the end but in fact both constraints could be false at some time during the interval: signal could become too large before battery becomes sufficiently small. Conversely, if we were to insist that either one of the conditions hold at both i *and* at j, we would preclude the possibility that for the first part of the interval we can rely on $(signal \leq 5)$; and then later, but before step j, rely on $(battery \leq 10)$. That is, we must allow changing of which condition we rely on part way through the interval.

Allowing for a potentially infinite number of such changes would be infeasible. Fortunately, in the general case for a disjunction C of $|C|$ numeric terms $c_1 \ldots c_{|C|}$ we need only include $|C|$-1 possible changing points. This result arises from the monotonicity of continuous numeric change: if we rely on a condition c_i until it becomes false, we will never be able to later rely on c_i as it cannot become true again. In our example, when $(signal \leq 5)$ becomes false, it cannot become true again until some later action affects *signal* or the gradient on *signal*. As we saw in the previous section, there is a guarantee that between two adjacently ordered plan

step	variable	constraints
turn_on$_\vdash$	t_0	≥ 0
	$battery_0$	= 30
	$battery'_0$	= $battery_0 \wedge \geq 8$
travel$_\vdash$	t_1	$> t_0$
	$signal_1$	= 0
	$signal'_1$	= $signal_1$
	$battery_1$	$= battery'_0 - 1 * (t_1 - t_0) \wedge \geq 8$
	$battery'_1$	= $battery_1 \wedge \geq 8$
	$battery_1 \leq 10 \vee signal_1 \leq 5$	
	$battery'_1 \leq 10 \vee signal'_1 \leq 5$	
travel$_\dashv$	t_2	= t_1 + 15
	$signal_2$	= $signal'_1$ + 0.5*(t_2-t_1)
	$signal'_2$	= $signal_2$
	$battery_2$	$= battery'_0 - 1 * (t_2 - t_0) \wedge \geq 8$
	$battery'_2$	= $battery_2 \wedge \geq 8$
	$battery_2 \leq 10 \vee signal_2 \leq 5$	
	$battery'_2 \leq 10 \vee signal'_2 \leq 5$	
$\psi_{0-transfer}$	$t_{\psi 0}$	$\geq t_1 \wedge \leq t_2$
	$battery_{\psi 0}$	= $battery'_0$ - 1*($t_{\psi 0} - t_0$) $\wedge \geq 8$
	$signal_{\psi 0}$	= $signal'_1$ + 0.5*($t_{\psi 0}$-t_1)
	($battery'_1 \leq 10 \wedge battery_{\psi 0} \leq 10 \vee$ $signal'_1 \leq 5 \wedge signal_{\psi 0} \leq 5$) $\wedge$ ($battery_{\psi 0} \leq 10 \wedge battery_2 \leq 10 \vee$ $signal_{\psi 0} \leq 5 \wedge signal_2 \leq 5$)	

Table 2: Example MIP featuring a Disjunctive Invariant

steps at which a disjunctive invariant is enforced, no actions affecting the variables referred to in that invariant are applied. Therefore, if we select a true condition to rely on, and maintain that for as long as possible before switching to another condition, we need only $|C|$-1 changing points for each adjacently ordered pair of plan steps.

To maintain a disjunctive invariant in the interval between two adjacently ordered plan steps i,j at which it was enforced, we add changing points, each ψ_m, as totally ordered time-points in the MIP such that:

$$t_i \leq t(\psi_0) \ldots \leq t(\psi|C|\text{-}1) \leq t(j)$$

Between each adjacent pair of steps $[y, z]$ in this total order, we insist that there is a condition $c_i \in C$ which is true at y *and* true at z. In doing so, at least one $c \in C$ is true at all points over the interval $[t(i), t(j))]$. Table 2 shows the changing point $\psi_{0-transfer}$ and its associated constraints that enforce the satisfaction of the disjunction $(battery \leq 10) \vee (signal \leq 5)$ between travel$_\vdash$ and travel$_\dashv$. Notice it is possible to either rely on one condition for the whole interval if desired; or to switch conditions at $\psi_{0-transfer}$.

6.4 Synchronising Process/Event Actions

As discussed in Section 4, the action analogues used for processes and events must be synchronised. By modifying the planner, we can achieve this quite readily. Suppose not-run_p_i is executing, and the decision is made to apply run_$p_{i\vdash}$, as step k of a plan. We can treat this as a special case, insisting that, at step k, we simultaneously apply not-run_$p_{i\dashv}$. The constraints on the resulting plan step are:

- Due to not-run_$p_{i\dashv}$, V_k (the values of the variables immediately at step k) must satisfy the invariants of not-run_p_i;
- Due to run_$p_{i\vdash}$, V'_k (the values of the variables immediately after step k) must satisfy the invariants of run_p_i.

This slightly abuses the PDDL 2.1 semantics: strictly, the invariants of not-run_p_i only need to hold to the end of the half-closed interval ending at step k, i.e. up to, but excluding, the values of the variables at V_k. This is not an issue *per se* when dealing with processes, but is an issue with events. Suppose not-do_e_j is executing, with invariants $\neg C_j$ – the negation of the condition C_j on event e_j. Then, suppose $e_{j\dashv}$ is applied as step k – we would want to synchronise this with ending and restarting not-do, as shown in Figure 1. But:

1. Due to not-do_$e_{j\dashv}$, we would say that V_k must satisfy $\neg C$: the invariants of not-do_e_j;
2. Due to e_j itself, V_k must also satisfy C, i.e. $pre\, e_j$;
3. Due to not-do_$e_{j\vdash}$, V'_k must then satisfy $\neg C$.

The first two of these are mutually exclusive: they require V_k to satisfy $C \wedge \neg C$. Thus, we tweak the constraints, creating variables ϵ prior to step k – denoted $V_k^{-\epsilon}$ – and apply the constraints of not-do_$e_{j\dashv}$ to $V_k^{-\epsilon}$ rather than V_k.

The constraints to give the values of $V_k^{-\epsilon}$ can be calculated *almost* in the same way as those for V_k, with a slight modification to calculate the values of variables ϵ in the past rather than now. With a small substitution into Equation 1 we get:

$$v_k^{-\epsilon} = v'_i + \delta v'_i((t(k) - \epsilon) - t(i)) \quad (2)$$

...where step i was the last step to have an effect on v. This is correct unless $t(k)$=$t(i)$+ϵ, in which case:

$$v_k^{-\epsilon} = v_i \quad (3)$$

Or, in other words, $v_k^{-\epsilon}$ takes the value of v immediately before the last effect on v (i.e. step i). To capture this choice over $v_k^{-\epsilon}$, each invariant of not-do_$e_{j\dashv}$ referring to v is replaced with a disjunction of two conditions:

- The invariant with $v_k^{-\epsilon}$ from Equation 2 $\wedge\, t(k) > t(i)$+ϵ;
- The invariant with $v_k^{-\epsilon}$ from Equation 3 $\wedge\, t(k)$=$t(i)$+ϵ;

Invariants referring to n variables are replaced with a disjoint of 2^n options: one for each combination of choosing some $v_k^{-\epsilon}$, and enforcing the appropriate temporal side-constraint.

So far, the focus has been on the general case: choosing to apply an event or to switch the running state of a process. In some cases such choices are inevitable, and for efficiency, we can exploit this. For instance, suppose run_p_i is executing, and has an invariant $battery \geq 8$, and an action assigns $battery$=0 – that invariant is immediately broken. In this case, we need not branch over what to apply next in the state reached: the only sensible thing to do is to force the (synchronised) application of run_$p_{i\dashv}$ and not-run_$p_{i\vdash}$, to change the process from running to not-running, or vice versa as appropriate. Similarly, if the invariant was attached to not-do_e_j, and is now broken, we can force the application of not-do_$e_{i\dashv}$, e_j, not-do_$e_{j\vdash}$. This eliminates the increase in branching factor due to process/event steps in such cases.

As a final note, we need to consider what happens at the start and end of the plan. In the compilation (Section 4.1), this was achieved with clips. Here, it is far easier:

- In the initial state I, for each p_i, if its condition C_i is satisfied, apply run_$p_{i\vdash}$; otherwise, apply not-run_$p_{i\vdash}$. In both cases, fix the time of this step to zero.
- In the initial state I, for each e_j, its condition C_j is false (c.f. PDDL semantics). Thus, apply not-do_$e_{j\vdash}$, at t=0.
- A state satisfies goals G, if an action with precondition G could be applied, and if there are no currently executing actions other than run, not-run or do actions.

Domain	Ver	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
#S/#G		1/3	1/5	2/5	2/8	3/8	3/7	4/9	4/11	4/12	4/13	4/14	4/15	4/16	4/17	5/3	5/6	5/9	6/4	7/4	8/4
Satellite	P/E	0.30	1.42	1.39	2.89	8.54	3.40	247.58	62.85	34.91	34.66	37.00	41.82	42.89	74.83	36.71	471.23	42.93	25.64	27.27	364.96
Satellite	comp	-	9.33	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Transformer	P/E	0.01	0.02	0.29	0.30	1.19	0.29	4.67	0.27	13.27	1.5	43.30	5.91	598.96	17.91	-	61.68	-	395.59	-	1465.89
Transformer	comp	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
LSFRP	P/E	0.03	0.04	2.33	5.32	5.58	2.88	5.58	5.58	5.25	5.74	5.52	x	x	x	x	x	x	x	x	x
LSFRP	comp	0.29	0.34	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	x

Table 3: Results of running the planner on PDDL+ Domains. P/E denotes 'Processes and Events'. comp denotes 'Compiled'. '-' indicates that the problem was unsolved, 'x' marks problems that do not exist.

Note that the initial state checks do not require recourse to the MIP, as variables (and hence the truth values of conditions) hold definite values, namely those of the initial state: there is initially no active continuous numeric change.

7 Evaluation

In this section we empirically demonstrate the performance of our implemented planner on PDDL+ domains. In domains without processes and events our planner will perform exactly as the planner POPF, runner up in the IPC2011 temporal track. Thus, we refer the reader to the published results for that planner (Coles et al. 2010) and the results of IPC2011 (Jimenez and Linares-Lopez 2011) for details of its performance on these domains, and comparisons to other existing planners. Unfortunately we are unable to compare the performance of our planner on PDDL+ domains with that of any other existing planner as TM-LPSAT, the only other fully-automated PDDL planner to support these features, is not available in a runnable form. As a guide to the reader, however, the limited published results for TM-LPSAT on available benchmarks (Shin 2004) report that the best configuration solves IPC2000 Driverlog Numeric Problems 2,3 and 4 in 149.82, 29.28 and 139.97 seconds respectively; whereas our planner solves these instances in 0.16, 0.01 and 0.05 seconds (albeit on slightly different hardware).

As a baseline for comparison we therefore use POPF, reasoning with the 'efficient' version of the clip compilation described in Section 4.1. POPF (and its siblings) are the only currently available systems for solving such problems, even when compiled. As there are no available standard PDDL+ problem sets we have created three domains and problem sets of our own based on those in the existing literature. Table 3 shows the time taken to solve these problems with processes and events (P/E) and using the compilation (comp).

The first of these is the cooled satellite domain described in (Coles et al. 2012). Originally based on the IPC2000 satellite domain, the cooled version allows active cooling of imaging sensors to be used to reduce the exposure time needed; at the expense of increased power demands. In our version of this domain, sunrise/sunset are processes, with preconditions on the time elapsed thus far, and that increase and decrease the (solar) power available to satellites. The results for this domain show that the compilation scales very poorly, indeed the planner using this solves only 1 problem. The #S/#G row shows the number of satellites and goals in each of the problem files: the P/E configuration scales well and to much larger problems than the compilation.

Our second domain is the transformer domain described in (Bell et al. 2009). This domain lends itself naturally to processes and events: the voltage on the circuit changes throughout the day due to customer demand. Our encoding uses processes based on the current time to update the voltage linearly over each half-hour period, using a piecewise-linear demand curve: an improvement on the original discrete model. The goal in this problem is to keep the voltage within a specified range, until a specified time, by switching transformers and capacitors in response to demand. We model the voltage going out of range using an event (one for each of bound) with the precondition $voltage > max$ (or $< min$), that deletes a fact required by the goal. From the table we see that the compilation leads to poor performance – no problems are solved – whilst the P/E configuration performs well. For guidance, within the problem set, even-numbered problems model winter demand, whilst odd problems model summer demand. Also, problem n+2 has one additional half-hour period of demand change compared to problem n. Performance on the winter configuration does not scale quite as far as summer, as more control actions are needed in winter to keep the voltage in range.

Finally, we consider the LSFRP domain (Tierney et al. 2012), based on the movement of ocean liners from one shipping service to another, around the world. The key aspect of this domain with respect to processes is the 'hotel-cost' of a vessel, paid from when it leaves one service until it reaches another. There may be several actions between these two points and the ship might have to wait to join its destination service at an appropriate point in the timetable. Further, if 'sail-on-service' actions are used, available on certain routes at certain times, hotel cost is not payable for the duration of these actions. The most natural model of this cost is as a process, which starts when the ship leaves its initial service; and stops when it either joins its destination service, or while sailing-on-service. Whilst the compilation successfully solves the 2 smallest problems in this domain it quickly becomes unable to scale. The P/E configuration solves all 11 problems in the suite – a set of real-world-sized problems developed working with industry – in under 6 seconds, although it is not attempting to optimise quality.

In conclusion, we have shown that direct handling of processes and events can lead to a significant advantage in terms of scalability in solving problems in which exogenous happenings are involved. Since the compilation solved so few problems it is difficult to make conclusions about solution quality, but on the 3 problems that were mutually solved, 2 had identical quality solutions, and in the other, the P/E configuration found the better solution. In future work we intend to consider optimising the quality of plans in the presence of processes and events and to extend our reasoning to consider a wider class of continuous functions.

References

Bell, K. R. W.; Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2009. The role of AI planning as a decision support tool in power substation management. *AI Communications* 22(1):37–57.

Boddy, M. S. 2003. Imperfect match: PDDL 2.1 and real applications. *Journal of Artificial Intelligence Research* 20:61–124.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

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

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

Fox, M., and Long, D. 2006. Modelling mixed discrete continuous domains for planning. *Journal of Artificial Intelligence Research* 27:235–297.

Fox, M.; Long, D.; and Halsey, K. 2004. An Investigation into the Expressive Power of PDDL2.1. In *Proceedings of the European Conference of Artificial Intelligence (ECAI)*.

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

Jimenez, S., and Linares-Lopez, C. 2011. IPC-2011 results. `http://www.plg.inf.uc3m.es/ipc2011-deterministic/Results`.

Li, H., and Williams, B. 2011. Hybrid planning with temporally extended goals for sustainable ocean observing. In *Proceedings of AAAI*.

McDermott, D. 2003a. PDDL2.1-The Art of the Possible? Commentary on Fox and Long. *Journal of Artificial Intelligence Research* 20:61–124.

McDermott, D. 2003b. Reasoning about Autonomous Processes in an Estimated Regression Planner. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Penberthy, S., and Weld, D. 1994. Temporal Planning with Continuous Change. In *Proceedings of AAAI*.

Penna, G. D.; Intrigila, B.; Magazzeni, D.; and Mercorio, F. 2009. UPMurphi: a tool for universal planning on PDDL+ problems. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS)*.

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

Shin, J. 2004. *TM-LPSAT: Encoding Temporal Metric Planning in Continuous Time*. Ph.D. Dissertation, New York University.

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