Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling

PDDL+ Planning with Events and Linear Processes

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

Abstract

We present a scalable fully-automated forward-chaining 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 challenges posed in creating a planner that can reason with these, and present novel state-progression and consistency enforcing techniques that allow us to meet these challenges. We present results showing that our new planner, using PDDL+ models, is able to solve realistic expressive problems more efficiently than the state-of-the-art alternative: a compiled PDDL2.1 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. Reasoning with these problems was identified and strongly motivated as a key challenge to real-world deployment of planners in 2006 (Fox and Long), our evaluation adds to this by including two real application domains in which reasoning with such exogeny is crucial in solving problems.

Despite their importance, the difficulty of solving such problems means that developing scalable fully-automated planners capable of reasoning with them remains a significant challenge. Even in the absence of exogeny, scalable automated planning in the presence of continuous numeric change has only recently become possible, 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 planning domain definition language, PDDL, (Fox and Long 2003) a number of modern planners began to address the challenge of reasoning with continuous numeric change.

COLIN (Coles et al. 2012) performs forward-chaining search, using 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/numeric constraints, but is less expressive than COLIN in the sense that it does not allow two actions to simultaneously change a variable.

To date there are only two planners capable of reasoning with discrete and continuous change and exogenous happenings, as in PDDL+ (Fox and Long 2006). TM-LPSAT (Shin and Davis 2005) was designed to solve PDDL+ problems with linear continuous change. It uses a SAT-based compilation, giving a discrete set of time points; and also uses an LP to manage numeric constraints. Its approach shows promise but empirically suffers 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 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: 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.

Following background and a running example, the paper first considers whether native handling of our PDDL+ subset could be avoided via a novel compilation to PDDL2.1. This motivates the need for native PDDL+ planning, provides a benchmark against which to evaluate, and introduces some important concepts and properties of PDDL+ that are key to native handling. We go on to describe our new planner detailing how we overcome the identified challenges and empirically demonstrate its scalability on PDDL+ problems.

2 Problem Definition

The logical basis for temporal planning, as modelled in PDDL2.1 (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 $p \in P$, $\neg p$, or a numeric constraint over $\mathbf{v}$. We assume all such constraints are linear, and hence can be represented in the form:

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

($\mathbf{w}$ 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 (invariants), and at the end, respectively. Instantaneous effects can occur at the start or end of A: $\text{eff}^{+}_{\vdash}A$ ($\text{eff}^{-}_{\vdash}A$) denote propositions added (resp. deleted) at the start; $\text{eff}^{num}_{\vdash}A$ 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 of the form:

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

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

Each action also 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 numeric constraints applied to a special variable dur_A denoting its duration. As a special case, *instantaneous* actions have duration ϵ, and only one set of preconditions pre A and effects $\text{eff}^{+}A$, $\text{eff}^{-}A$, and $\text{eff}^{num}A$. 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 $\text{eff}^{+}_{\vdash}A$, $\text{eff}^{-}_{\vdash}A$, $\text{eff}^{num}_{\vdash}A$. $A_{\dashv}$ is the analogous action for the end of A.

PDDL+ augments PDDL problems with processes and events. Like actions, these have preconditions, and effects. Events are akin to instantaneous actions: if an event's preconditions are satisfied, it occurs, yielding the event's instantaneous effects. Processes are akin to durative actions, with $\text{pre}_{\leftrightarrow}A$ corresponding to the process' precondition, and $\text{eff}_{\leftrightarrow}$ as its continuous numeric effects. Then, while $\text{pre}_{\leftrightarrow}A$ is satisfied, $\text{eff}_{\leftrightarrow}$ occurs. 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 intractable, even when restricted to linear continuous change. In particular, in theory, events can infinitely cascade, repeatedly firing and self-supporting. Also, having reached the goals, it is challenging to determine whether they persist indefinitely, given future processes and events that may occur. To address the former of these, 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 t_r, but not necessarily persist, can be modelled by using a process to count time and adding $time > t\_r$ to the goal.

2.1 Running Example

We introduce a simple illustrative example problem based on the use of a mobile 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 (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; $\text{pre}_{\vdash}$ = {$\neg$on}; $\text{pre}_{\leftrightarrow}$ = {battery>0}; $\text{eff}^{+}_{\vdash}$ = {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}.

3 PDDL+ versus PDDL 2.1

We now explore the relationship between PDDL+ processes and events and their PDDL2.1 counterparts, durative-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 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_$p_i$=$\neg C_i$, and no effects;
- in both cases, the duration of the action is in $[\epsilon, \infty]$.

If we could 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. In other words, the start of one action must be always synchronised with the end of the other. Whenever the truth value of C_i changes (which may be many times) we simply switch which of these actions is executing. Ensuring this switch happens simultaneously is crucial: if time passed between e.g. not-run_$p_{i\dashv}$ and run_$p_{i\vdash}$ then there would be time when C_i might be true, but the effect of p_i is not being captured by any executing action.

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

- When the event occurs, C_j is true – and, as noted earlier, events must delete one of their own preconditions;

[1] c may be derived from operations on constant-valued variables

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

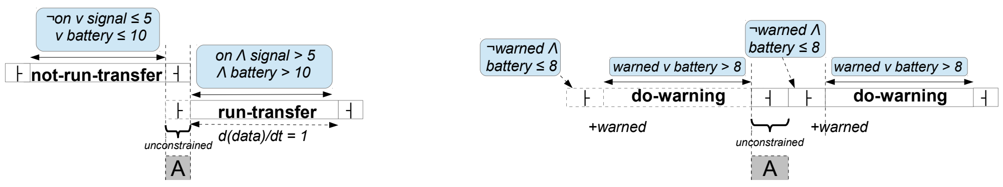

Figure 1: Representing Processes (left) and Events (right). Dotted lines denote clipping constraints (ϵ and 2ϵ, resp.)

- A period then begins in which $\neg C_j$ holds;
- If the event re-occurs, then beforehand there is an ϵ-long period in which neither $\neg C_j$ nor C_j: the conditions are mutually exclusive with any discrete effects that led to C_j.

We can capture this using a durative action do-e_j, that must restart *immediately after* it ends, i.e. meet with itself. pre$_{\vdash}$do-e_j=pre e_j and eff$_{\vdash}$do-e_j=eff e_j reflect that e_j occurs at its start. Then, pre$_{\leftrightarrow}$do-e_j=$\neg C_j$ enforces the subsequent period in which e_j does not fire. Finally, the absence of effects/conditions at the end gives an ϵ gap in which a snap-action can change the facts/variables in C_j. Defined thus, do-e_j must be restarted at exactly the times e_j would occur.

As a note for the astute, there is a subtle difference between the way in which process and event actions are clipped together: process actions are synchronised; event actions must meet each other. This is due to the treatment of closed and open intervals in which conditions must hold, in PDDL2.1. For a detailed treatment of these, see (Cushing 2012); to briefly review the relevant semantics here:

- When a fact is added (deleted) by an action in PDDL it becomes true (false) immediately, but only available to meet actions' start/end preconditions after time ϵ.
- When a durative action A is applied at time t and ended at time u pre$_{\vdash}A$ must be true at time t. pre$_{\leftrightarrow}A$, however, need only to hold in the open interval (t, u), that is *after* t (recall that eff$_{\vdash}A$ can achieve pre$_{\leftrightarrow}A$).

Strictly, in PDDL+, the ϵ-gap is needed for the conditions of actions, but not events. We can only approximate this in PDDL2.1, treating the event as an action, and hence needing an ϵ gap. This leads to an unconstrained region, of duration ϵ, in which the condition on the event is not being inspected, so that its truth value can be changed, before the event fires.

Processes can start and end either due to instantaneous (propositional or discrete numeric) or continuous numeric change. In the former case the following actions occur in parallel: run_$p_{i\dashv}$ (not-run_$p_{i\dashv}$); an action A (or several such actions) whose discrete effects change the truth value of C_i; and not-run_$p_{i\vdash}$ (run_$p_{i\vdash}$) – the invariant condition will be satisfied ϵ later as required. The sequence for events is similar, except that do_$e_{j\dashv}$ occurs in parallel with A and then do_$e_{j\vdash}$ occurs ϵ after these, as its start precondition (which is satisfied by A) is only available at this time. If the condition is satisfied by continuous numeric change then the action(s) A are not required: the ongoing continuous numeric change will cause the change in the truth value of C_i/C_j.

Returning to our running example, the left of Figure 1 shows how synchronised actions capture 'transfer' transitioning from not-running to running. To be not running, one of its preconditions must be false; to be running, all must be true. Clipping constraints (dotted lines) ensure this transition occurs at the right time. The right of Figure 1 shows the 'warning' event. (The fact 'warned' ensures it does not repeatedly fire.) The key detail is where the two instances of do_warning meet: clipping constraints ensure this period is only 2ϵ long, comprising the ϵ-long 'unconstrained' region; and the start of the next action, i.e. the event.

3.1 Clipping Constraints in PDDL 2.1

Reasoning with processes and events using PDDL2.1 requires a compilation that enforces three conditions:

1. Clipping together of process/event actions (as Figure 1);
2. Ensuring that not-run_p_i (or run_p_i), and a variant of do_e_j, start *immediately*, at the start of the plan;
3. Allowing processes/event actions to end in goal states.

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

Definition 3.1 — clip_$f(d)$
A (tight) clip for fact f, and auxiliary facts $\{r_0, \ldots, r_n\}$, is a durative action B, with duration $d + \epsilon$, where:

- pre$_{\vdash}B$ = $\neg f$, eff$^{+}_{\vdash}B$ = $\{f\}$;
- pre$_{\dashv}B$ = $\{r_0, \ldots, r_n\}$; eff$^{-}_{\dashv}B$ = $\{f\} \cup \{r_0, \ldots, r_n\}$.

f is thus only available ϵ after starting clip_f; before being deleted at the end. Snap-actions with a condition on f must then be placed inside the clip, and d bounds them to occur within the desired length of time. 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 r_i fact as an effect. Thus, not only must the snap-actions occur in the clip; but also, no necessary snap-action can be omitted.

To constrain the actions from Section 3 in PDDL2.1, we use a clip for each process or event. For each process p_i:

- Create clip_$f_i(\epsilon)$, with auxiliary facts $r_{i\vdash}$, $r_{i\dashv}$;
- Add f_i to $pre_{\vdash}$ and $pre_{\dashv}$ of run_p_i and not-run_p_i;
- Add $r_{i\dashv}$ to $\mathit{eff}^{+}_{\dashv}$ of run_p_i and not-run_p_i.
- Add $r_{i\vdash}$ to $\mathit{eff}^{+}_{\vdash}$ of run_p_i and not-run_p_i;

Similarly, for event e_j:

- Create clip_$f_j(2\epsilon)$, with auxiliary facts $r_{j\vdash}$, $r_{j\dashv}$;
- Add f_j to $pre_{\dashv}$do_e_j and $pre_{\vdash}$do_e_j;
- Add $r_{j\dashv}$ to $\mathit{eff}^{+}_{\dashv}$ do_e_j; add $r_{j\vdash}$ to $\mathit{eff}^{+}_{\vdash}$ do_e_j.

In both of these cases, the clip meets the objectives of Figure 1: by the use of auxiliary facts representing the action having started and ended, the action must end and the next one must start within the clip; and due to the tight availability of f_i (f_j), the actions cannot start or end outwith clips. 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 presented clip permits clipping run-p_i to run-p_i and not-run-p_i to not-run-p_i).

The compilation as it stands ensures that a clip cannot end, unless an already-executing 'run', 'not-run' or 'do-' action ends inside it: these add $r_{*\dashv}$, an end-condition of of the clip. This is desirable in the general case, but in the initial state no such actions are executing. Further, to start the 'do-' actions for an event, its conditions must be true – which in the initial state is necessarily not the case (c.f. PDDL+ semantics (Fox and Long 2006)). This brings us to our second requirement, **starting the actions immediately (2)**. For this, we use:

- Facts go and $init\_do$ (not added by any action, event or Timed Initial Literal (TIL)), which are true initially and deleted at time ϵ and 2ϵ respectively by TILs (Hoffmann and Edelkamp 2005). This creates a small window at the start of the plan in which they are available.
- A fact $exec$, added by a TIL at time ϵ, and added to $pre_{\vdash}$ of every non-clip action in the plan.
- Special 'initial-do-' actions, the same as the regular do-actions for events, modulo the start preconditions and effects are replaced with $init\_do$ and $\emptyset$, respectively.

We create a single 'go' clip allowing all process/initial event tracking actions to begin at the start of the plan (without needing to end first). We collate all clip facts (all f_i and f_j) into a set F. We define the set $R_{\vdash}$ as the set of all $r_{*\vdash}$ auxiliary facts, and $R_{\dashv}$ analogously contains all $r_{*\dashv}$ facts.

Definition 3.2 — go-clip
A 'go' clip for clip facts F, and auxiliary fact set $R_{\vdash}$, is a durative action B, with duration 2ϵ where:

- $\text{pre}_{\vdash}B = go \wedge \{\forall f \in F \;\; \neg f\}$; $\text{eff}^{+}_{\vdash}B = F$;
- $\text{pre}_{\dashv}B = R_{\vdash}$; $\text{eff}^{-}_{\dashv}B = F$.

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

Next we must allow the run/not-run/do actions to **terminate once the goals have been met (3)** – the PDDL2.1 semantics require there to be no executing actions in goal states. We use a final modified clip action – a 'goal-clip', and replace the goal G with a single fact $goal$. Note the addition of $R_{\dashv}$ to $\text{pre}_{\dashv}$ ensures the required run/not-run/do actions continue running until G is achieved.

Definition 3.3 — goal-clip
A 'goal' clip for clip facts F, and auxiliary fact set $R_{\dashv}$, is a durative action B, with duration 2ϵ, where:

- $\text{pre}_{\vdash}B = \{\forall f \in F \;\; \neg f\}$; $\text{eff}^{+}_{\vdash}B = F$; $\text{eff}^{-}_{\vdash}B = exec$;
- $\text{pre}_{\dashv}B = G \wedge R_{\dashv}$; $\text{eff}^{+}_{\dashv}B = goal$; $\text{eff}^{-}_{\dashv}B = F$.

Finally we note that negation of conjunctive conditions on processes/events yields disjunctive invariants on the not-run/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. It does not, however, negate its benefits.

While this compilation to PDDL2.1 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). The compilation is also 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 $3n + 3m + 2$ actions are added to the planning problem, of which $(m + n)$ are applicable in each state. This massively increases the branching factor and solution plan length. Permutations of such actions can also cause significant problems in temporal planning (Tierney et al. 2012). It therefore seems that native handling of processes and events is likely to be far more efficient – this forms the focus of the rest of the paper. We will, of course, return to this point in our evaluation.

4 Forward Chaining Partial-Order Planning

Our new planner builds 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. For a comprehensive explanation of the workings of POPF please refer to (Coles et al. 2010), we summarise the key details here.

State Representation: To support partial-ordering, additional information is stored in states. Each plan step is given a unique index to facilitate this. For each fact $p \in P$:

- $F^{+}(p)$ ($F^{-}(p)$) the index of the last step to add (delete) p;
- $FP^{+}(p)$, a set of pairs $\langle i, d\rangle$, steps with a precondition p: i is the 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; so having applied some actions, a range of values are possible. For each $v \in \mathbf{v}$:

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

Temporal Constraints: 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 A is applied, as step j, temporal constraints are added to enforce causal links and to prevent threats to earlier causal links. To satisfy each $p \in \text{pre}A$, $t(j) > t(F^{+}(p))$; and for each $p \in \text{eff}^{+}A$:

$$t(j) > t(F^{-}(p)); \;\; t(j) > t(F^{+}(p));$$
$$\forall \langle i, d\rangle \in (FP^{-}(p)) : \;\; t(j) \geq t(i) + d$$

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	$battery_{now}$	$= battery'_0 - 1.(t_{battery-now} - t_0) \wedge > 0$
	$t_{battery-now}$	$> t_0$
	$signal_{now}$	$= signal'_2$
	$t_{signal-now}$	$> t_2$

Table 1: Example POPF MIP

Analogous constraints order negative preconditions and delete effects after $F^-(p)$, $F^+(p)$, and $FP^+(p)$. To ensure consistency in numeric variable updates we totally order all effects on each variable and protect existing preconditions/invariants. For each v modified by eff^{num}A or eff$_{\leftrightarrow}A$:

$$(t(j) > t(V^{eff}(v))) \wedge (\forall i \in VP(v) \cup VI(v) : t(j) > t(i))$$

To ensure correct input values for all effect computations: for each variable v referred to by eff^{num} A, eff$_{\leftrightarrow}A$ or dur$A$:

$$t(j) > t(V^{eff}(v))$$

Finally constraints are added between action start/end points to represent duration constraints (e.g. $t(j) = t(i) + 10$).

In the absence of continuous (or duration-dependent) numeric effects, the temporal-constraint consistency in POPF can be checked with a simple temporal network.

Temporal-Numeric Constraints: In the presence of linear continuous numeric effects, a MIP solver is required for the resulting temporal–numeric constraints. To represent numeric constraints for each step i, the variables $v_i \in V_i$ record the values of each of $\mathbf{v}$ immediately 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. We use the notation $(V_i \vDash p)$ to denote a constraint added to ensure the numeric precondition p is satisfied by the values of 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.

Maintaining Invariants: 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 active invariants on v. These invariants need to be enforced at step i: although they do not belong to i, they belong to currently executing actions, and i must not adversely interfere with these. Thus, v_i and v'_i are constrained to obey these invariants. This may necessitate additional ordering constraints: if an invariant referring to v also refers to variables not otherwise relevant to i, i must be ordered after the last modifiers of these, too. Hereon, if we state that an invariant p is *enforced* at step i, we mean V_i and V'_i are constrained to obey the invariant (i.e. $V_i, V'_i \vDash p$), and any extra ordering constraints are added. For conjunctive invariants this approach exploits the monotonicity of linear continuous numeric change: for intervals in which an 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 successive effects on v. If, for example, $v \geq 5$ at the start and end of an interval, under monotonic change it must have been throughout.

Continuous Numeric Change: The final consideration is the continuous numeric change. During MIP construction, at each point referring to variable v, the sum of the gradient effects δv acting on 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:

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

Example: 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. We can compute $\delta' v_i$ at each t_i with an effect on v, by working through the plan: $\delta signal'_0$=0.5, $\delta battery'_1$=-1 and $\delta signal'_2$=0.

The *now* steps are added for computing upper and lower bounds on state variables: for each state variable we solve the MIP, maximising then minimising the 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.

5 Search with Processes and Events

In this section we describe how a forward-chaining planning approach can be modified to handle processes and events *natively*, eliminating many of the artificial planning decisions entailed by the Section 3.1 compilation. Two key modifications are made to POPF to achieve this: first we define how processes and events are started and ended at the appropriate times during search. Second we consider the major challenge that emerges from this of managing their invariants, which may be disjunctive or multi-variate.

5.1 Process/Event Steps in Plans

Here we show how actions representing process/event execution can be added to the plan, and how MIP constraints can be added to allow direct enforcement of correctness.

Definition 5.1 — Starting a process

A process p_i can be started as step k of a plan *iff* it is not currently executing. If so:

- k is ordered after the necessary existing plan steps and the invariants of existing plan steps are checked, in exactly the same way as if k was the start of a durative action with over all condition $\text{pre}_{\leftrightarrow}p_i$ and effects $\text{eff}_{\leftrightarrow}p_i$.

- In addition: $\neg(V_k \vDash \text{pre}_{\leftrightarrow}p_i) \wedge (V'_k \vDash \text{pre}_{\leftrightarrow}p_i)$

With reference to the compilation, we no longer need to explicitly stop the non-execution of the process: it instead suffices to ensure that this process marks a point where V_k does not satisfy the conditions of the process; but V'_k, infinitesimally later, does. In other words, run_$p_{i\vdash}$ and not-run_$p_{i\dashv}$ are one step. Stopping processes is similar:

Definition 5.2 — Stopping a process
A process p_i can be stopped as step k of a plan *iff* it is currently executing. If so:

- k is ordered after the necessary existing plan steps and the invariants of existing plan steps are checked, in exactly the same way as if k was the start of a durative action with over all condition $\neg\text{pre}_{\leftrightarrow}p_i$ and zero-gradient effects on the variables in $\text{eff}_{\leftrightarrow}p_i$.
- In addition: $(V_k \vDash \text{pre}_{\leftrightarrow}p_i) \wedge \neg(V'_k \vDash \text{pre}_{\leftrightarrow}p_i)$

In many cases, rather than using extra plan steps for processes, we can make them implicit. Suppose a process p_i is executing (with condition C_i), and after applying an action A at step k, C_i cannot be satisfied[3]. In this situation, we can insist that at step k, A is applied *and* p_i stops (by adding directly to step k the constraints above). The proof is by contradiction: if k comprised A alone, we would then be in a state where p_i was executing but C_i was false.

Definition 5.3 — Implied process steps
A snap-action A, applied as step k of a plan, and leading to state S', entails also applying within step k of the plan:

- Stopping all processes p_i where S' cannot satisfy C_i;
- Starting all processes p_i where S' cannot unsatisfy C_i.

Contrast this with the compilation in which, to apply A, the planner must anticipate p_i ending by starting a clip; then apply not-run_$p_{i\dashv}$; A; run_$p_{i\vdash}$; and finally end the clip – five search steps. Here, instead, the choice of explicitly starting/stopping a process remains only if S' could satisfy both C_i and $\neg C_i$: with the precise truth value depending on the values of numeric variables, and hence timestamps of actions. Only in this case is an explicit plan step required.

As events have discrete – not continuous – effects we can use a similar but slightly different approach:

Definition 5.4 — An Event occurring
An event e_j can occur as a pair of steps $(k, k+1)$, where:

- k is ordered after the necessary existing plan steps, and invariants of existing steps are checked, as if it were the end of a durative action with over all condition $\neg(\text{pre}\, e_j)$;
- $k+1$ is an instantaneous action, with the preconditions and effects of e_j plus,
- In addition: $\neg(V'_{k+1} \vDash \text{pre}\, e_j)$; and $t(k+1) = t(k) + 0$.

Note the importance of the last of these constraints. Whereas the compilation to PDDL 2.1 introduced an ϵ-gap between the preconditions of an event becoming true, and the event occurring, there is no such gap here: the preconditions become true at step k; and the event's effects occur at step $k+1$, zero time later.

As with processes, we can eliminate faux planning decisions in the case where an event e_j is implied by an actions. The choice only remains in states where the event's precondition could be either satisfied or unsatisfied:

Definition 5.5 — Implied event steps
A snap-action A, applied as step k of a plan, and leading to state S', entails also applying as steps $(k, k+1)$ of the plan, all events e_j where S' cannot unsatisfy $\text{pre}\, e_j$.

One final remark with reference to the compilation (Section 3.1): we can eliminate the need for go/goal clips:

- In the initial state I, for each process p_i, if C_i is satisfied, start p_i at t=0;
- in the initial state I, for each event e_j, C_j is unsatisfied (c.f. PDDL semantics);
- A state is a goal state for the goals G *iff* a dummy action with precondition G can be applied. Notice that in PDDL+ we need not insist that no processes are running.

Note the initial state checks do not require recourse to the MIP, as variables (hence the truth values of conditions) hold definite values: there is no active continuous numeric change. The goal state check does, however, require a MIP evaluation, where the final values of the variables are constrained to meet G.

5.2 Managing Invariants of Processes/Events

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. $(\mathit{battery} > 10) \wedge (\mathit{signal} > 5)$. 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 happens frequently. POPF cannot handle:

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

The latter of these is particularly prevalent when considering processes and events: even if a process/event's condition is a conjunct of terms, taking the negation of this, to enforce intervals during which it is not executing, yields a disjunction. In this section we introduce new invariant handling techniques that correctly identify all the necessary invariants to check at each plan step, discuss the extra implied ordering constraints and finally exploit these constraints in developing a mechanism for handling disjunctive invariants.

5.2.1 Identifying Necessary Invariants to Enforce

To identify all invariants that must be enforced when a snap-action is added to the plan, 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 currently executing actions.

[3]e.g. in the running example transfer is executing, with condition $\mathit{battery} \geq 10$, and some action A assigns $\mathit{battery}$=0.

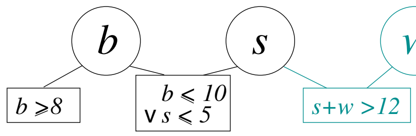

Figure 2: Condition–Variable Dependency Graph

Definition 5.6 — Condition–Variable Dependencies
A condition–variable dependency graph (CVDG) has:

- One variable vertex v_j for each numeric variable v_j;
- For each invariant $C = (c_1 \wedge \ldots \wedge c_n)$: one constraint vertex c_i for each $c_i \in C$; edges from each c_i to each variable vertex v_j where v_j affects the truth value of c_i.
- For each invariant $C' = (c'_1 \vee \ldots \vee c'_n)$: one constraint vertex C'; and for each c'_i, an edge from C' to each v_j where v_j affects the truth value of c'_i.

This graph gives us a straightforward way to ascertain 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 constraint (invariant) that can be reached in the graph from v needs to be enforced at the point when the snap-action is applied. Figure 2 shows an example based on our running example, with an additional action with invariant $signal + wifi > 12$ (the variable names are abbreviated to b, s, and w).

We return to our running example to illustrate why this mechanism is necessary. Suppose in a state S the currently executing actions are turn_on, travel, not-run_transfer and 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 actions, so 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 $signal$ 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 e.g. $battery$=5: a value within its range in S.

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.

To illustrate why the new mechanism is also needed for multi-variate conditions we have the additional invariant shown on the right of Figure 2. Suppose an action is 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 other 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 $b \geq 12$ 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.

5.2.2 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 turn_on$_\vdash$ has been applied, and we consider applying travel$_\vdash$, the disjunctive invariant $\neg on \vee battery \leq 10 \vee signal \leq 5$ must be enforced. This leads to additional ordering constraints: the standard POPF state-update rules now require travel$_\vdash$ to be ordered after turn_on$_\vdash$, as the value of $battery$ must known for the purposes of enforcing this additional invariant. This would not have been the case had the 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, 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 added orderings, we guarantee that we only need to maintain an invariant C=$(c_1 \vee ... \vee c_n)$ 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 $c_f \in C$).

Proof sketch:, suppose the 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 (following a_m), 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 a_m and 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 added later, after a, it will be ordered in A_C after a.

5.2.3 Maintaining Disjunctive Invariants

In Section 4 we noted that conjunctive invariants can be checked at a plan step by ensuring the variable values at that point satisfy the invariant; and further, if the invariants are checked at two successive plan steps, they will hold at all points in between. The latter of these statements is not, however, true 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, for a disjunction C of $|C|$ numeric terms $c_1 ... c_{|C|}$ we need only include $|C|$-1 possible changing points. This result arises from the monotonicity of continuous linear 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 until a 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 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 steps.

Definition 5.7 — Disjunctive invariant constraints
A disjunctive invariant $C = c_1 \vee \ldots \vee c_{|C|}$ can be maintained in the interval between two adjacently ordered plan steps i,j through intermediate points, each ψ_m s.t:

$$t(i) \leq t(\psi_1) \ldots \leq t(\psi_{|C|\text{-}1}) \leq t(j)$$

...where for each adjacent pair $[y, z]$ in this total order:

$$\exists c_i \in C \text{ s.t. } (V'_y \vDash c_i) \wedge (V_z \vDash c_i)$$

Thus, the interval $(t(i), t(j)]$ is broken down into piecewise-adjacent intervals, in each at least one $c \in C$ is true. The rules for when invariants need to be checked are unchanged – but these constraints give us the mechanism for doing so. These ψ points are introduced into the MIP as if they were plan steps, but not into search. Table 2 shows the intermediate point $\psi_{1-transfer}$ 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; or to switch conditions at $\psi_{1-transfer}$.

6 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 POPF, runner up in the IPC2011 temporal track. Thus, we refer the reader to the published results for POPF (Coles et al. 2010) and IPC2011 (Jimenez and Linares-Lopez 2011) for details of its performance on such domains, and comparison to other planners. Unfortunately we are unable to compare the performance of our planner on PDDL+ domains with that of any other planner. 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).

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	$= t_1 + 15$
	$signal_2$	$= signal'_1 + 0.5*(t_2\text{-}t_1)$
	$signal'_2$	$= signal_2$
	$battery_2$	$= battery'_0 - 1 * (t_2 - t_0) \wedge \geq 8$
	$battery'_2$	$= battery_2 \wedge \geq 8$
		$battery_2 \leq 10 \vee signal_2 \leq 5$
		$battery'_2 \leq 10 \vee signal'_2 \leq 5$
$\psi_{1-transfer}$	$t_{\psi 1}$	$\geq t_1 \wedge \leq t_2$
	$battery_{\psi 1}$	$= battery'_0 - 1*(t_{\psi 1} - t_0) \wedge \geq 8$
	$signal_{\psi 1}$	$= signal'_1 + 0.5*(t_{\psi 1}\text{-}t_1)$
		$(battery'_1 \leq 10 \wedge battery_{\psi 1} \leq 10 \vee signal'_1 \leq 5 \wedge signal_{\psi 1} \leq 5)$
		$\wedge \quad (battery_{\psi 1} \leq 10 \wedge battery_2 \leq 10 \vee signal_{\psi 1} \leq 5 \wedge signal_2 \leq 5)$

Table 2: Example MIP featuring a Disjunctive Invariant

As a baseline for comparison we therefore use POPF, reasoning with the 'efficient' version of the clip compilation (Section 3.1). POPF (and its siblings) are the only currently available systems for solving even the compiled problems. As no standard PDDL+ problem sets exist we created three of our own based on those in the existing literature[4]. Exogenous continuous numeric change is fundamental to all of these domains, and is the key to industrial interest in the application problems: demand in the transformer domain and shipping costs in the LSFRP. Table 3 shows performance on these problems handling processes and events natively (P/E) and using the Section 3 compilation (comp).

The first domain is a variant of the cooled satellite domain described in (Coles et al. 2012). An extension of the IPC2000 satellite domain, this allows active cooling of imaging sensors to reduce the required exposure time at the expense of increased power demands. Sunrise/sunset are processes, conditioned on elapsed time, that increase/decrease the (solar) power available to satellites. The #S/#G row in Table 3 shows the number of satellites and goals in each problem. The compilation scales very poorly in this domain, indeed the planner solves only 1 problem; the P/E configuration scales much better.

Our second domain is the transformer domain originally described in (Bell et al. 2009). This models a real problem in power networks where customer demand, changing exogenously across the day, causes the voltage at a substation to change. The goal in this problem is to reach a specified time

[4]PDDL descriptions of our domains and problems are available from: http://www.inf.kcl.ac.uk/staff/amanda/ProcessesAndEvents/

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: Time (in seconds) taken to solve problems in PDDL+ Domains, with native process and event handling (P/E) versus the compilation (comp). '-' indicates that the problem was unsolved, 'x' marks problems that do not exist.

and maintain the voltage within a specified range, by switching transformers and capacitors. The original encoding of the problem discretised the change over half-hour periods, using timed-initial literals and a compilation similar to the one above (but simpler, as it sacrifices accuracy to discretise change) to enforce the exogenous change. Our encoding has a more accurate continuous piecewise-linear model of this crucial feature: using processes to update the voltage linearly, one for each half-hour period. We model the voltage going too high (or low) using events, with the precondition $voltage > max$ (resp. $< min$), that delete a fact required by the goal. From the table we see that no problems are solved using the continuous compilation; while the P/E configuration scales well. Even-numbered problems model winter demand and odd problems summer demand. Problem n+2 has one additional half-hour period of demand change than problem n. Performance on the winter configuration does not scale quite as far as summer, as more switching 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 industrial interest in this domain is to minimise cost. Hotel cost is paid for each vessel 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, 'sail-on-service' actions are available on certain routes at certain times, and 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. This domain is interesting in that a simpler TIL compilation could not be used here because processes depend not on elapsed time, but actually on the activity the ship is performing. 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 of the suite of real-world-sized problems developed in conjunction 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 gives significant scalability improvements. As the compilation solved so few problems it is difficult to make conclusions about solution quality: on the 3 problems that were mutually solved, 2 had solutions of equal quality and in the other the P/E configuration found the better solution. In the future we plan to consider plan quality and extend our approach to 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 Comms* 22(1):37–57.

Boddy, M. S. 2003. Imperfect match: PDDL 2.1 and real applications. *JAIR* 20:133–137.

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

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

Cushing, W. 2012. When is temporal planning really temporal? PhD Thesis, Arizona State University.

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

Fox, M., and Long, D. 2006. Modelling mixed discrete continuous domains for planning. *JAIR* 27:235–297.

Fox, M.; Long, D.; and Halsey, K. 2004. An Investigation into the Expressive Power of PDDL2.1. In *Proc. ECAI*.

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

Jimenez, S., and Linares-Lopez, C. 2011. IPC 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 *Proc. AAAI*.

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

McDermott, D. 2003b. Reasoning about Autonomous Processes in an Estimated Regression Planner. In *Proc. ICAPS*.

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

Penna, G. D.; Intrigila, B.; Magazzeni, D.; and Mercorio, F. 2009. UPMurphi: a tool for universal planning on PDDL+ problems. In *Proc. 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. PhD Thesis, 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 *Proc. ICAPS*.