A Temporal Relaxed Planning Graph Heuristic for Planning With Envelopes

Amanda Coles, Andrew Coles
Department of Informatics,
King's College London, UK
email: {amanda,andrew}.coles@kcl.ac.uk

Abstract

When planning in temporal domains with required concurrency, *envelopes* arise where one or more actions need to occur within the execution of another. Starting an envelope action gives rise to an implicit relative deadline: all of the actions that need to occur within the envelope must complete before it ends. Finding effective heuristic guidance in these domains is challenging: the heuristic must not only consider how to reach the goals, but identify when it is not possible to achieve these implicit deadlines to avoid fruitless search. In this paper, we present an adaptation of a Temporal Relaxed Planning Graph heuristic, that accounts for dependencies between facts and actions in the relaxed planning graph; and the envelopes that are open in the state being evaluated. Results show that our new heuristic significantly improves the performance of a temporal planner on benchmark domains with required concurrency.

1 Introduction

Reasoning effectively about temporal constraints is essential in solving a wide range of realistic planning problems. Over the last decade *required concurrency* has emerged as one of the main challenges for effective temporal planning. In problems with required concurrency, it is necessary to start the execution of one or more actions during the execution of another, termed an *envelope* action, in order to solve the problem. This is a phenomenon that occurs in many problems when modeling the temporal constraints that arise. Sometimes these model the availability of a resource for a finite time, for example an envelope action modeling the shift of a worker, in which all activities involving that worker must take place; and other times they model a temporal constraint, for example in the transportation of perishable goods the goods can only be outside of chilled conditions for a finite amount of time, so all actions that involve the goods not being chilled must fit inside a finite envelope.

The problem of planning with envelopes (Fox, Long, and Halsey 2004) was identified soon after the introduction of the temporal variant of the planning domain definition language PDDL2.1 (Fox and Long 2003) and subsequently expanded on and posed as a major challenge for the community by Cushing et al. (2007). There has been much interest in solving problems with required concurrency, resulting in a number planning approaches capable of reasoning with these problems: forward search approaches, such as CRIKEY (Coles et al. 2009; 2008) and POPF (Coles et al. 2010), as well as SAT-based (Rankooh and Ghassem-Sani 2015) and local-search approaches (Gerevini, Saetti, and Serina 2010). Despite recent progress, problems with required concurrency remain much more challenging than those without, due to the additional temporal reasoning that is required (Rintanen 2007).

One major challenge in planning effectively with required concurrency is that of heuristic guidance, specifically allowing the planner to quickly recognise when the temporal constraints imposed by an open envelope can no longer be met, due to the actions applied within it. Without such guidance planners are blind to the fact that some states are dead-ends, and can continue searching forwards from them.

The challenge of reasoning about deadlines and windows of opportunity that are fixed with respect to absolute time, as modeled using Timed Initial Literals (TILs) (Hoffmann and Edelkamp 2005), has seen some attention – a number of planners that can reason with these (Gerevini, Saetti, and Serina 2006; Kavuluri and Senthil 2004) were developed for the 2004 International Planning Competition. Later work has considered heuristics for detecting temporal dead-ends in this setting (Marzal, Sebastia, and Onaindia 2014; Coles et al. 2008; 2010) as well as detecting recurrent TIL time windows (Tierney et al. 2012).

Here we are interested in the less-explored problem of providing effective heuristic guidance for planning with envelopes; that is, deadlines that arise not at a fixed time as a property of the problem, but rather as a result of the planner choosing to apply an envelope action: starting a worker's shift implies a *relative deadline* on the time by which all the activities requiring that worker must be completed. Such domains pose a particular challenge for commonly used *delete relaxation* heuristics (such as (Hoffmann and Nebel 2001)), as relaxing (ignoring) delete effects renders the heuristic blind to the implied deadlines.

We propose techniques that extend a Temporal Relaxed Planning Graph (TRPG) heuristic with information about the relative deadlines in the state being evaluated, due to *open actions* that have started but not yet finished. This forbids the heuristic from relying on actions that necessar-

Copyright © 2017, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

ily break these deadlines, and allows the detection of dead ends in the case where the resulting relaxed problem is thus unsolvable. We show that our new heuristic markedly improves the performance of a state-of-the-art temporal planner, in terms of time taken to solve problems and number of problems solved, across a range of benchmark domains that exhibit required concurrency.

2 Problem Definition

In this paper we consider propositional temporal planning problems expressed in PDDL2.1 (Fox and Long 2003). We first summarise the language semantics, then highlight the particular aspects we are concerned with in this paper, and describe the search framework in which we operate.

2.1 PDDL 2.1 Semantics

PDDL2.1 can express both non-temporal and temporal planning problems, through the use of two categories of actions: instantaneous, and durative.

Each instantaneous action A has a precondition $pre(A)$ that must be true for A to be applied. If A is applied its effects are realised: $\mathit{eff}^+(A)$ and $\mathit{eff}^-(A)$ denote propositions added and deleted.

Each durative action A has three sets of preconditions: $pre_{\vdash}(A)$, $pre_{\leftrightarrow}(A)$, $pre_{\dashv}(A)$. These represent the conditions that must hold at its start, throughout its execution (invariants), and at its end, respectively. Effects can occur at the start or end of A: $\mathit{eff}^+_{\vdash}(A)$ ($\mathit{eff}^-_{\vdash}(A)$) denote propositions added (resp. deleted) at the start. Similarly, $\mathit{eff}^+_{\dashv}(A)$, $\mathit{eff}^-_{\dashv}(A)$ record effects at the end. Finally, the action has maximum and minimum duration constraints, $max\_dur(A)$ and $min\_dur(A)$.

A plan can be thought of as a sequence of time-stamped *happenings*: points at which one or more instantaneous actions occur; and durative actions either start or end. By stepping through the happenings in chronological order, it is possible to establish the times at which facts become true, or false. We refer to each happening in the plan as a step, and assign each step a unique index i.

For an instantaneous action A to be applied at time t, $pre(A)$ must be true at t. To ensure the truth values of the facts in the precondition are not simultaneously changed, and used to meet the precondition:

- A minimum amount of time denoted by the value ϵ must elapse between the happening after which $pre(A)$ became true, and the point at which A is applied;
- A minimum amount of time, ϵ, must then pass before $pre(A)$ can be made false. The only exception to this is that A may immediately violate its own preconditions.

These provisos enforce all necessary orderings between actions, but allow actions whose preconditions and effects do not interfere to be executed simultaneously.

Following (Long and Fox 2003), 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). Action $A_{\vdash}$ has precondition $pre_{\vdash}(A)$ and effects $\mathit{eff}^+_{\vdash}(A)$ and $\mathit{eff}^-_{\vdash}(A)$. $A_{\dashv}$ is the analogous action for the end of A.

In terms of their preconditions and effects, these snap actions are subject to the same application rules as instantaneous actions. The invariant and duration constraints additionally require that:

- For an action starting at t and finishing at t', the invariant conditions $pre_{\leftrightarrow}A$ must hold in the interval (t, t'). This is equivalent to a dummy invariant-checking action $A_{\leftrightarrow}$ with $pre(A_{\leftrightarrow}) = pre_{\leftrightarrow}A$ being applicable at all points within this interval. As the interval is open at both ends, $pre_{\leftrightarrow}A$ may refer to facts affected by actions occurring at time t: $A_{\vdash}$, or any other action at that time, can cause $pre_{\leftrightarrow}A$ to become true. Similarly, effects at time t' are permitted to violate $pre_{\leftrightarrow}A$.
- The duration of the action, here $(t' - t)$, must obey the duration constraints $max\_dur(A)$ and $min\_dur(A)$ of the action. As a consequence of this, there must be a one-to-one correspondence between the starts and ends of A: it is not possible to start an action without later ending it; nor is it possible to end an action without having started it.

The task of planning is to find a time-stamped series of happenings that, when applied starting from the initial state, reach a state that admits the goals of the problem; in which no actions are executing; and for which all preconditions, invariants and temporal constraints are satisfied.

2.2 Temporal Envelopes

In non-temporal planning, it suffices that each step is ordered at some time after its preconditions are achieved; and some time before they are violated. The plan steps $[a_0, a_1, ..., a_n]$ can be given any sequence of timestamps $t_0 < t_1 < ... < t_n$. These could be incrementally higher multiples of ϵ, as this suffices to meet the minimum separation that must occur due to the ordering constraints between effects and preconditions. Or, they could be delayed arbitrarily, as there is no *maximum* time separation between steps.

In temporal planning, the presence of action durations introduces maximum time separation between start and end points. In effect, starting an action A creates a local deadline that its end must occur no later than would be permitted by $max\_dur(A)$. If it is possible to solve the problem without ever concurrently executing any two actions the goal can be reached by simply alternating start and end snap-actions; with each start placed after the preceding end, and each end placed the relevant duration after the preceding start.

In problems with *required concurrency* (Cushing et al. 2007) this is not the case: due to the interactions between preconditions and effects of actions, every solution requires that actions are executed concurrently. Cushing et al. define a number of cases in which this arises; in this paper we are concerned with envelopes (Coles et al. 2009). Example envelopes are depicted in Figure 1. In the simplest case (a), an action A forms a window of opportunity in which some fact p is available: it is added by $A_{\vdash}$ and deleted by $A_{\dashv}$. Thus, the action B, requiring p as an invariant, must occur wholly within the execution of A. In case (b), $A_{\vdash}$ must precede $B_{\vdash}$ to add the fact p; and then, $B_{\dashv}$ must precede $A_{\dashv}$ in order to add the fact q. The result, again, is strict concurrency between A and B, but this time to meet the end conditions of A

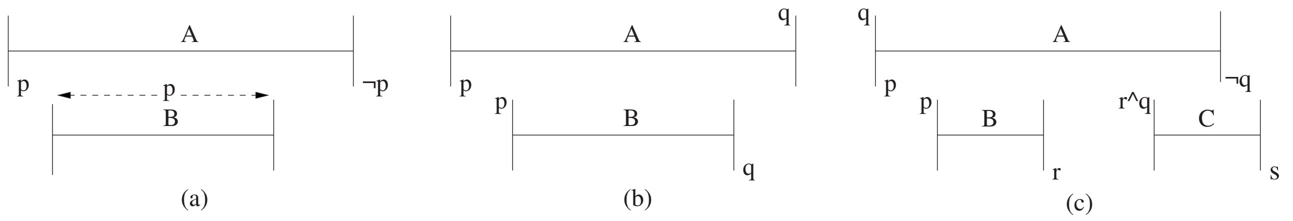


Figure 1: Examples of envelopes

rather than to precede a negative effect. More complex examples can be constructed, such as case (c), where A deletes q at the end, so must contain $C_{\vdash}$; but the preconditions of $C_{\vdash}$ are met by first applying B, which itself must follow $A_{\vdash}$.

Case (b) corresponds to a *causal loop* (Bit-Monnot, Smith, and Do 2016), the other cases represent some of many additional variants that our work captures. We capture more cases, by selectively considering delete effects: if an action A is started, we place actions after $A_{\vdash}$ not only to use its effects, but also to avoid conflicting with $pre_{\vdash}(A)$. We also place actions before $A_{\dashv}$ in order to avoid its negative effects. If this leads to snap-actions being scheduled *within* A, we have an envelope: $A_{\vdash}$ imposes a local deadline, determined by $max\_dur(A)$ by which these actions must occur, in order to precede $A_{\dashv}$. In contrast, Bit-Monnot et. al capture some different cases by delaying (or removing) $A_{\dashv}$ where the end conditions of A ($pre_{\vdash}(A)$) are reached later (resp. unreachable) in the relaxed reachability analysis.

Other prior work has considered recognising envelopes according to specific patterns (Coles et al. 2009; Cushing 2012). Cushing postulates that these can be used to prune search but highlights that automating the proofs of these is a significant open challenge. The strength of our approach is that many of these patterns are captured automatically via the STN without the need for any such proofs.

2.3 Temporal State Progression

We adopt the state progression semantics of POPF (Coles et al. 2010). Successive application of plan steps yields a partial-order plan, with ordering constraints between steps based on the facts they refer to. To facilitate this, each fact p, in each state, is annotated with the following information:

- $F^{+}(p)$ ($F^{-}(p)$) is the index of the plan step that most recently added (deleted) p;
- $FP^{+}(p)$ is a set of pairs, each $\langle i, d\rangle$, used to record steps with a precondition p. i denotes the index of a plan step, and $d \in \{0, \epsilon\}$. If d=0, then p can be deleted at or after step i: this corresponds to the end of an invariant condition. If d=ϵ, then p can be deleted ϵ after i or later.

Application of actions to states produces ordering constraints based on the annotations and updates their values.

- Steps adding p are ordered ϵ after $F^{-}(p)$; those deleting p, after $F^{+}(p)$. Hence, effects on a fact are totally ordered. Preconditions are fixed within this ordering: a step with precondition p is ordered after $F^{+}(p)$; and recording it in $FP^{+}(p)$ ensures future deletors of p are ordered after it.
- If step j ends an action A that began at step i, the interval $[i, j]$ must respect the duration constraints of A.

A partial-order plan in this form maps to a Simple Temporal Network (STN) – a labeled directed graph $\langle A, T\rangle$ where:

- The vertices $A = [a_0..a_n]$ are the steps of the plan;
- Each edge $\langle a_j, a_i, c\rangle \in T$ corresponds to either:
  - An edge $\langle a_j, a_i, c \in \{0, -\epsilon\}\rangle$ representing an ordering constraint that j must be 0 or ϵ time units after i due to the aforementioned partial ordering constraints.
  - One of a pair of edges $\langle a_j, a_i, -lb\rangle$,$\langle a_i, a_j, ub\rangle$, encoding that the duration of an action that started at i and finished at j must lie in the range $[lb, ub]$.

Solving this STN with a shortest-path algorithm identifies possible timestamps for the snap-actions. In the case of a consistent plan, this corresponds to the earliest point at which each step can be applied while respecting the ordering constraints. In the case of an inconsistent plan, a negative-length cycle is found – a snap-action must occur before itself – in which case the state is discarded. The task of planning in POPF is to find a sequence of steps that transforms the initial state into a goal state such that all preconditions/invariants are met; the STN is consistent; and there are no *open actions*: actions that have started but not yet finished.

In addition to the steps applied in the plan, *future nodes* representing the end snap actions of actions that have started but not yet finished, are added to the simple temporal network. These nodes represent the commitment we have made by starting an action that we must finish it; and must also respect its duration constraint. If a future end snap-action will delete p, then any snap-actions with a condition p are ordered before it. These ordering constraints would inevitably be added to the plan when the end snap-action was applied, but adding them early avoids search that is bound to fail – if the STN is inconsistent, then it has been shown the future end snap-action, which must necessarily occur in order to complete the plan, can never be applied.

While the heuristic described in this work is described within the context of POPF's state progression semantics, it is also compatible with other planning approaches. Decision-epoch planners, such as TFD (Eyerich, Mattmüller, and Röger 2009) and Sapa (Do and Kambhampati 2003) search over states that have a timestamp, and a queue of actions that have started but not yet finished. Starting an action fixes it to be applied at the timestamp in the state. In problems with required concurrency, however, this can preclude solving problems. For instance, if an action A provides a short envelope in which some other snap-action $B_{\dashv}$ must be applied, $A_{\vdash}$ needs to be delayed to occur just before $B_{\dashv}$, rather than occurring at the current timestamp.

Decision-epoch planning can solve some problems with envelopes, specifically those where there is no requirement for an action to be delayed to occur later than the current timestamp, so our new heuristic could be used in this setting. The use of an STN and partial-ordering constraints can be seen as providing a mechanism to facilitate these timestamp delays in the case where they are needed and other planners that use STN based reasoning and TRPG heuristics, for example (Coles et al. 2008; Gerevini, Saetti, and Serina 2010), could also make use of our approach.

3 Relaxed Planning with Envelopes

Heuristics are used to guide search in planning by providing goal-distance estimates and identifying dead ends. Commonly used is a TRPG heuristic: a Temporal variant of the Relaxed Planning Graph heuristic (Hoffmann and Nebel 2001). This has been explored in prior work, for example in Sapa (Do and Kambhampati 2003) and POPF (Coles et al. 2010). In the description of our approach we will focus on extending POPF due to its generality, existing support for required concurrency and the increased temporal flexibility for meeting deadlines that results from its partial order representation. However, as discussed our heuristic could be used within other temporal planning frameworks.

3.1 Temporal RPG Outline

The outline of POPF's TRPG heuristic is given in Algorithm 1[1]. As in the non-temporal case, the RPG consists of *action* and *fact* layers; but now, these are timestamped to denote the times at which facts were deemed to have been reached, and actions were deemed to be applicable. These are initialised at line 1 using a helper function shown in Algorithm 2. Facts in the state are queued to appear at the minimum time of the step that added the fact, by referring to the STN for S. If there are actions executing in S, and the ends of these do not have preconditions, these are queued to appear at the earliest point the action could end, again according to the STN. (If a decision-epoch planner, or a total-order planner such as CRIKEY3, is used rather than POPF, an alternative helper function (Algorithm 3) fulfills a similar role, but refers to the timestamp of the state and the timestamps of the queued action ends.)

The main graph expansion loop begins at line 4. It looks at the timestamps of the pending action and fact layers, and chooses the action or fact layer that occurs earliest.

For action layers: For all snap-actions, their add effects are added to the fact layer at that time (lines 9, 12, 21). For start snap actions, additionally, if the action's invariant conditions are satisfied (determined by calling Algorithm 4), the dummy invariant action is queued in al to appear immediately (line 14). When a dummy invariant action itself appears, if the action's end snap-action's conditions are satisfied, the end is queued to appear, delayed by the minimum plausible duration of the action (line 19). The delays chosen here reflect the semantics of PDDL described in Section 2:

Algorithm 1: Temporal RPG Expansion

```
   Data: $S$, a state being evaluated
   Result: A relaxed planning graph
1  $(fl, al) \leftarrow initialise\_layers(S)$;
2  $added \leftarrow \{(f, \emptyset) \mid$ each fact $f\}$;
3  $tstart, tinv \leftarrow \{(a, \emptyset) \mid$ each durative action $a\}$;
4  while $fl \neq [] \vee al \neq []$ do
5      $flt \leftarrow \min_{(t,l) \in fl} t$;  $alt \leftarrow \min_{(t,l) \in al} t$;
6      if $alt < flt$ then
7          $l \leftarrow al$.pop_front();
8          for instantaneous $a \in l$ do
9              for $f \in eff^+(a)$ do add $f$ to $fl[alt]$;
10         for start $a_\vdash \in l$ do
11             $tstart[a] \leftarrow tstart[a] \cup \{alt\}$;
12             for $f \in eff^+(a_\vdash)$ do add $f$ to $fl[alt]$;
13             $t' \leftarrow now\_true(a_\leftrightarrow, alt, added)$;
14             if $t' \neq \infty$ then add $a_\leftrightarrow$ to $al[t']$;
15         for invariant $a_\leftrightarrow \in l$ do
16             $tinv[a] \leftarrow tinv[a] \cup \{alt\}$;
17             $t' \leftarrow now\_true(a_\dashv, alt, added)$;
18             if $t' \neq \infty$ then
19                 add $a_\dashv$ to $al[t' + min\_dur(a)]$;
20         for end $a_\dashv \in l$ do
21             for $f \in eff^+(a_\dashv)$ do add $f$ to $fl[alt]$;
22     else
23         $l \leftarrow fl$.pop_front();
24         for fact $f \in l$ do
25             if $added[f] \neq \emptyset$ then continue;
26             $added[f] \leftarrow added[f] \cup \{flt\}$;
27             for $\{instantaneous\ a \mid f \in pre(a)\}$ do
28                 $t' \leftarrow now\_true(a, flt, added)$;
29                 if $t' \neq \infty$ then add $a$ to $al[t' + \epsilon]$;
30             for $\{a_\vdash \mid f \in pre(a_\vdash)\}$ do
31                 $t' \leftarrow now\_true(a_\vdash, flt, added)$;
32                 if $t' \neq \infty$ then add $a_\vdash$ to $al[t' + \epsilon]$;
33             for $\{a_\leftrightarrow \mid f \in pre(a_\leftrightarrow)\}$ do
34                 $t' \leftarrow now\_true(a_\leftrightarrow, flt, added)$;
35                 if $t' \neq \infty \wedge tstart(a) \neq \emptyset$ then add $a_\leftrightarrow$
                   to $al[\max(t', \min(tstart(a)))]$;
36             for $\{a_\dashv \mid f \in pre(a_\dashv)\}$ do
37                 $t' \leftarrow now\_true(a_\dashv, flt, added)$;
38                 if $t' \neq \infty$ then
39                     if $tinv(a) \neq \emptyset$ then
40                         $t'' \leftarrow \max(t' +$
                           $\epsilon, \min(tinv(a)) + min\_dur(a))$;
41                         add $a_\dashv$ to $al[t'']$;
42                     if $a$ is executing in $S$ then
43                         $t'' \leftarrow$ min time of $a_\dashv$ in STN;
44                         add $a_\dashv$ to $al[\max(t' + \epsilon, t'')]$;
```

[1]This is a marginal adaptation of the original that does not require time (ϵ) to pass between a fact being added, and an invariant consequently becoming true.

Algorithm 2: Initialise Layers (POPF)

```
Data: $S$, a state being evaluated
Result: Priority queues of fact and action layers, ($fl$, $al$)
1  $fl$, $al \leftarrow []$;
2  for $f \in S$ do
3      $t \leftarrow$ min time of step $FP^{+}(p)$ in STN for $S$;
4      Add $f$ to $fl[t]$;
5  for executing action $a \in S$ do
6      if $a_{\dashv}$ has no preconditions then
7          $t \leftarrow$ min time of $a_{\dashv}$ in STN for $S$;
8          Add $a_{\dashv}$ to $al[t]$;
9  for instantaneous or start action $a$ with no
   preconditions do  Add $a$ to $al[0]$;
10 return ($fl$, $al$)
```

Algorithm 3: Initialise Layers (Decision Epoch)

```
Data: $S$, a state being evaluated
Result: Priority queues of fact and action layers, ($fl$, $al$)
1  $fl$, $al \leftarrow []$;
2  $t \leftarrow$ timestamp in state $S$;
3  for $f \in S$ do  Add $f$ to $fl[t]$;
4  for executing action $a \in S$ do
5      if $a_{\dashv}$ has no preconditions then
6          $t' \leftarrow$ end time of $a$ according to $S$;
7          Add $a_{\dashv}$ to $al[t']$;
8  for instantaneous or start action $a$ with no
   preconditions do  Add $a$ to $al[t]$;
9  return ($fl$, $al$)
```

the invariants can begin immediately after the start, with no need for an ϵ gap; and the action can end $min\_dur(a)$ later.

For fact layers: If a fact appearing has caused the preconditions of an instantaneous or start action to become true (lines 29, 32), the action is queued to appear ϵ after the fact layer. For dummy invariant-checking actions, there is the additional consideration that the start of the action must already have appeared (line 35); if this is the case, the action is queued to appear either at the earliest time the start appeared, or the time of the fact layer, whichever is latest – no ϵ gap is needed for invariants. An end snap-action $a_{\dashv}$, whose preconditions are satisfied can be added in one of two cases:

- The invariant action has appeared. In this case, the end is queued at either $min\_dur(a)$ after the earliest time the invariant appeared, or ϵ after the fact layer, whichever is latest. This ensures the separation between these points respects the minimum duration, and in the case where the end snap-action has preconditions, these are satisfied.
- The action is executing in the state S being evaluated. In this case, there is no need to start it again the TRPG before it could end; and the invariant must already hold in S. Thus, it suffices to queue the end-snap action at a sufficiently late time: either the minimum time of its future node in the STN for S (see Section 2.3), or ϵ after the fact layer, whichever is latest.

Algorithm 4: Prior version of now_true

```
Data: $a$ – a snap action; $t$ – a timestamp; $added$ – the
      time at which each fact appeared in the RPG
Result: The time at which $pre(a)$ was satisfied, or $\infty$
1  for $f \in pre(a)$ do if $added[f] = \emptyset$ then return $\infty$;
2  return $t$;
```

Once the TRPG has been built, if the goals and the ends of executing actions have been reached, a relaxed plan can be extracted almost as in the non-temporal case: working backwards through the fact layers, starting with the goals at the layers in which they appear; choosing an action from the preceding action layer that adds the fact; and adding its preconditions as subgoals to be achieved at earlier layers. The only minor edits are that for each action executing in S, its end (and end preconditions) must be added; and that if $a_{\dashv}$ is added to support a fact, and a is not executing in S, $a_{\leftrightarrow}$ and $a_{\vdash}$ (and their respective preconditions) must also be added. In the extension to the TRPG which we will now describe, we do not change solution extraction further.

3.2 Envelope Layer Labels

The layers in the TRPG described above are indexed by timestamp – an admissible an estimate of the time a given fact or action could appear. As this is a delete-relaxation heuristic, negative effects are ignored, and it is assumed that if a fact is in S or is added, it persists throughout the TRPG.

In the case of Timed Initial Literals (Hoffmann and Edelkamp 2005) that impose deadlines (a fact is true initially, never modified and irreversibly deleted at some time) these timestamps can be used to prune actions from the TRPG (Coles et al. 2008; Tierney et al. 2012). Simply, if a fact f is deleted at time t, snap-actions referring to f that have not appeared before $t - \epsilon$ can never appear; and steps $a_{\leftrightarrow}$ requiring f, cannot appear after $t - min\_dur(a)$.

In the general case, such an approach cannot be taken to the delete effects attached to the ends of actions that are executing in S. If $a_{\dashv}$ deletes f, and its minimum timestamp in the STN is t, then this is only its *minimum* timestamp – not its maximum – so f is not obliged to disappear at this time. This maximum is often unbounded, as the start of the action, and hence its end, could be delayed. Moreover, if f is added by another action the TRPG, then this consideration is moot: this other achiever could be used to support preconditions on f, even if f was deleted by $a_{\dashv}$.

In the case of envelopes, we can do better. With reference to Figure 1(a), suppose we are in the state where $A_{\vdash}$ has been applied. If $B_{\vdash}$ is then applied, which uses p, it would need to be ordered after $A_{\vdash}$; and before the future end node $A_{\dashv}$. This imposes a *relative* deadline: the snap-actions for B must occur within the maximum time between $A_{\vdash}$ and $A_{\dashv}$. Delaying $A_{\dashv}$ can increase the *absolute* time at which p is deleted, but cannot obviate this relative constraint; it would merely increase the timestamps of $A_{\vdash}$,$B_{\vdash}$ and $B_{\dashv}$, too.

To capitalise on this, we modify the TRPG so that it tracks these relative dependencies. Instead of being indexed by timestamps, layers are indexed by labels:

Algorithm 5: Label Layer for Step

```
Data: $S$, a state being evaluated; $i$, a step index
Result: A layer label for step $i$
1  $t \leftarrow$ min time of step $i$ in STN for $S$;
2  $ft \leftarrow \emptyset$; $ae \leftarrow \emptyset$;
3  for each action $a$ executing in $S$ do
4      $d \leftarrow$ max time from step $i$ to end of $a$ in the STN;
5      for $f \in eff^-(a_\dashv)$ do
6          if $f \in ft$ then $ft[f] \leftarrow \min(d, ft[f])$;
7          else $ft[f] \leftarrow d$;
8      if $a_\dashv \in ae$ then $ae[a_\dashv] \leftarrow \max(d, ae[a_\dashv])$;
9      else $ae[a_\dashv] \leftarrow d$;
10 return $\langle t, ft, al\rangle$
```

Definition 3.1 — Layer label
A layer label is a tuple $\langle t, ft, ae\rangle$ where:

- t is a timestamp (as before);
- ft is a map from facts, to how long is is before that fact will be deleted. $ft[f] = \infty$ if no such deadline applies.
- ae is a map from the ends of open actions, to how long it is before they must have occurred. $ae[a_\dashv] = \infty$ if no such deadline applies.

These labels are updated and maintained as the TRPG is expanded. The effects of an action in layer alt appear in fact layer alt; but now alt is a layer label, rather than a timestamp, allowing effects to inherit deadlines from the actions that added them. The following modifications support this:

Initialising the TRPG: As noted earlier, the first step is to use Algorithm 2 to queue the facts in S and precondition-free ends of actions. This is modified so that at lines 3 and 7, instead of taking a timestamp t of some step i from the STN for S, Algorithm 5 is used to build the layer label for step i. In addition to the timestamp, this looks for relative deadlines imposed by actions executing in S. For a step i inside the envelope of an action a, the maximum time d between i and the end of a (line 4) is finite. Thus, the effect of lines 5 to 7 is to record relative deadlines due to negative end effects of a. In the case where i is ordered within multiple envelopes on a single fact f – due to multiple executing actions deleting f at the end – the tightest relative deadline is taken.

Lines 8 to 9 use the maximum time d for another purpose. Starting an action implies a deadline on applying its end, hence achieving its end preconditions. For example, in Figure 1(b), starting A implies a deadline on achieving its end precondition q. If another action is ordered after $A_\vdash$ then there is also a (shorter) deadline on applying $A_\dashv$ from this step. For example, if step i is $B_\vdash$ (which is ordered after $A_\vdash$ to use its effect p) the relative deadline d for reaching $A_\dashv$ from step i will be $(max\_dur(A) - \epsilon)$.[2]

It is worth noting here that the relative deadlines due to envelopes are inferred automatically from the STN whenever they arise. There is no need for a fragile analysis step to identify specific design patterns that imply potential envelopes: we are not limited to the cases shown in Figure 1.

[2]Line 8 is only relevant to states where two or more instances of the same action are executing concurrently. If this occurs, we optimistically take the maximum time left to either of these.

Algorithm 6: Extended version of now_true

```
Data: $a$ – a snap action; $ll$ – a layer label; $added$ – layer
      label(s) at which each fact appeared in the RPG
Result: Layer label when $pre$ is satisfied or $\langle\infty,\emptyset,\emptyset\rangle$
1  $\langle t, ft, ae\rangle \leftarrow ll$;
2  for $f \in pre(a)$ do
3      if $added[f] = \emptyset$ then return $\langle\infty,\emptyset,\emptyset\rangle$;
4      $ae' \leftarrow \{(a_\dashv, \max_{ll' \in added[f]} ll'.ae[a_\dashv]) \mid \text{all } a_\dashv\}$;
5      $ft' \leftarrow \{(f', \max_{ll' \in added[f]} ll'.ft[f']) \mid \text{all } f'\}$;
6      $ae \leftarrow \{(a_\dashv, \min(ae[a_\dashv], ae'[a_\dashv]) \mid \text{all } a_\dashv\}$;
7      $ft \leftarrow \{(f', \min(ft[f'], ft'[f'])) \mid \text{all } f'\}$;
8  for $f \in pre(a)$ do
9      if an action adds $f$ in the RPG then $ft[f] \leftarrow \infty$;
10     if $a$ is an invariant action then
11         if $ft[f] < min\_dur(a)$ then return $\langle\infty,\emptyset,\emptyset\rangle$;
12     else if $ft[f] < \epsilon$ then return $\langle\infty,\emptyset,\emptyset\rangle$;
13 return $\langle t, ft, ae\rangle$
```

Ordering layers: Previously, the priority queues of fact and action layers were ordered by timestamp. They are now ordered by label: in ascending order of timestamp; breaking ties using an arbitrary but fixed (lexicographic) ordering according to ft and ae. This is used to order the queues, and in Algorithm 1 (line 6) to determine the next layer to visit.

Keeping multiple *added entries:* Line 25 is updated to keep any achiever of f that is not Pareto-dominated by an existing member of $added[f]$. That is, there is no existing member with a smaller or equal timestamp, and longer or equal time left for each relative deadline.

An updated *now_true:* Previously, when queuing an action, it sufficed that its preconditions had appeared. Now, we calculate its layer label according to Algorithm 6. For each precondition f, we keep the most optimistic time left for each relative deadline across all adders of f (lines 4–5). The rationale for this is that during RPG expansion, we do not commit to which achiever *must* be used for f, as this is a disjunctive choice; but, as *some* achiever would be needed, a relaxation across all options is used. To combine these into a label for the precondition as a whole (lines 6–7), it takes the pessimistic combination of these – the minimum time left – as we certainly need one adder for each fact (this follows the propagation of maximum separation constraints in Dechter, Meiri, and Pearl (1991)). As the deadlines in ft arise due to delete effects at the end of open actions we can remove them as soon as an alternative adder of f is found, line 9 clears the relative deadline on f in this case.

Having determined the label applied to the preconditions of a, the relative deadlines are checked to determine whether a can be added to the RPG. For invariant-checking actions using f, the relative deadline on f must be at least the minimum duration of a: f cannot be deleted while a is executing. For other actions, the time left must be at least ϵ: f cannot be deleted until strictly after the snap action. If the deadlines cannot be met, a layer with timestamp ∞ is returned, indicating a cannot be applied here. This may prevent a from ever appearing, allowing the heuristic to detect dead ends if a is required for the relaxed plan, but cannot be applied due to relative deadlines. If the deadlines can be met, the appropriate layer label has been found and a is added to the RPG with an incremented version of this layer label.

Operations on layer labels: In Algorithm 1, at lines 29, 32, 40, and 44, actions are delayed to a future layer, by incrementing a timestamp t by some amount d. We can define adding an amount of time d to a layer label, by allowing d time units to elapse, and subtracting d from the time remaining for each relative deadline, that is:

$$\begin{aligned}\langle t, ft, ae\rangle + d = \quad & \langle t + d, \\ & \{(f, r - d) \mid (f, r) \in ft\}, \\ & \{(a_\dashv, r - d) \mid (a_\dashv, r) \in ae\}\rangle\end{aligned}$$

Lines 35 and 40 take the min (best case) and max (worst case) of two or more layer labels. min is defined following lines 4 and 5 of Algorithm 6: keeping the longest remaining time for each fact and action deadline, seen across all labels; and the minimum timestamp. max follows lines 6 and 7, keeping the least remaining time, and maximum timestamp.

Ending executing actions: At line 42, Algorithm 1 inserts the ends of actions that are executing in S. The condition here is amended to read simply:

if a is executing in $S \wedge t'.ae[a_\dashv] \geq \epsilon$ **then**

That is, now $pre(a_\dashv)$ is satisfied, there must be at least ϵ left before $a_\dashv$ must have occurred, according to the layer label for its preconditions. If this is the case, Algorithm 5 is used at line 43 to find the layer label for $a_\dashv$, and it is then queued. If not, $pre(a_\dashv)$ presently relies on a causal chain of actions that must begin after the start of $a_\vdash$, but exceeds its duration. Again if $a_\dashv$ does not appear now it may be applicable at a later layer when some other way to achieve its preconditions (shorter, or not ordered after $a_\vdash$) is found. Otherwise, if $a_\dashv$ never appears, the heuristic has found a dead end: we cannot achieve the preconditions of $a_\dashv$ in time.

4 Evaluation

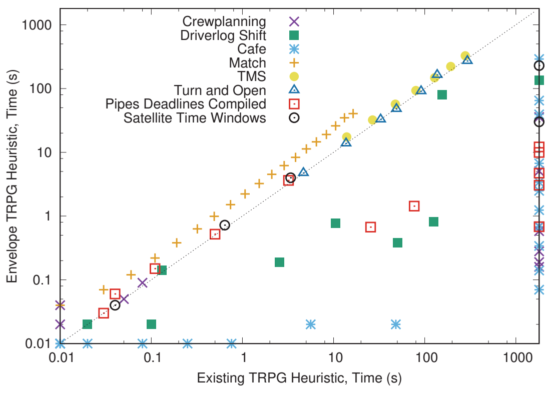
In this section we evaluate our new heuristic within the POPF framework, compared to the prior TRPG implementation as a baseline, in both cases using WA* search with W=5. For a detailed comparison of the performance of POPF to other planners we refer the reader to the results of the temporal track of the IPC 2011 planning competition (Linares López, Celorrio, and Olaya 2015). All tests are run on 3.5GHz Core i5 machines, restricted to 30 minutes of CPU time and 4GB of memory, though all tests that failed did so because of running out of memory rather than time.

We consider all benchmark domains from the literature and International Planning Competitions (IPCs) that make use of envelopes. Despite being useful for modeling various facets of planning problems, there are comparatively few such domains – at least when compared to propositional domains – due to the small number of planners that support required concurrency, and are thus able to solve them.

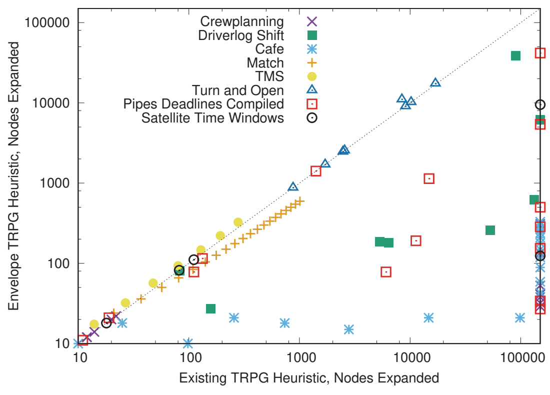
- Crewplanning (IPC 2008): The activities of the crew on a space station must all fit within the available days (modeled using envelopes). Some activities (e.g. crew exercise, sleep, eating) must take place every day; the results of science activities must be reported on certain days.
- Driverlog Shift (Coles et al. 2009): Drivers work fixed length shifts (modeled using envelopes), all activity involving a driver must take place during the driver's shift.
- Cafe (Coles et al. 2009): Each food order must be served within a fixed time of starting to prepare the order (to prevent food going cold). There are finite resources (chefs/equipment) in the kitchen for food preparation.
- Match (IPC 2011): this is the classic match domain in which a fuse must be mended in a dark cellar. The action 'light-match' is an envelope that provides the light required to do this during its execution.
- Temporal Machine Shop, TMS (IPC 2011): various parts must be made, combined and fired in a kiln at each stage. Firing the kiln is an envelope action, although there is no contention for using it as the kiln has infinite capacity.
- Turn and Open (IPC 2011): A variant of Gripper where the robot must turn a handle (an envelope action) to push a door open (inside the envelope) to walk between rooms.
- Pipes No Tankage Deadlines Compiled (IPC 2004): this is a compilation that uses envelopes instead of TILs to represent deadlines on delivery of oil via pipelines.
- Satellite Complex Time Windows Compiled (IPC 2004): this is a compilation where envelopes represent windows of opportunity for satellites to send image data to Earth.

The results, shown in Figure 2 show a noticeable overall performance improvement using the new heuristic. Figure 2a compares the time taken with the prior TRPG heuristic (x-axis) and with the new heuristic (y-axis). Points below the line y=x are where the planner solves problems more quickly using the new heuristic. In general the planner can solve problems much faster, but even more notably, the points on the right hand side (x=1800) represent problems that were solved with the new heuristic but not without. Indeed, using the new heuristic allows the planner to solve an additional 26 problems, across a range of domains. For reference, in terms of solution quality, the planners produced plans of equal makespan in all problems, except for two (in Turn and Open), where the planner found a shorter solution with the new heuristic. We are not, therefore paying a price in terms of quality for solving problems more quickly. Note, though, the makespan of the solution is determined to a large extent by the length of the longest envelope, which is a predefined feature of the problem, so it is difficult to meaningfully optimise makespan in many of these problems.

A closer look at the results on a per-domain basis leads us to some interesting insights. Broadly the evaluation domains are split into three categories: (1) those with envelopes that model some contended resource available only for a period of time, beginning when the planner chooses to start the envelope; (2) those with envelopes that model the availability of some resource, but there is no contention (it can be used in parallel, or all actions that could be applied trivially fit



(a) Time Taken to Solve Problems



(b) Nodes Expanded

Figure 2: Performance With and Without the New Heuristic

within it); (3) those with envelopes performing a compilation of temporal constraints (e.g. deadlines) that make facts available at fixed times (the TIL compilation domains).

Category (1) which includes the first four domains in the list above, are those in which we would expect the approach to most improve performance. Indeed, we see this reflected in the results, with the new heuristic allowing more problems to be solved in 3 out of the 4 domains. In the final domain, Match, it is interesting to note that the planner is actually slightly slower using the new heuristic. This is not due to a lack of informed guidance, indeed, as can be observed in Figure 2b the planner is actually expanding fewer nodes to find a solution. What we are observing here are the slight overheads in computing the more informed heuristic: although the heuristic detects that more 'light match' actions are required to mend all the fuses, leading to fewer nodes being evaluated, the increased per-state heuristic cost means this does not quite pay off in terms of overall runtime.

This observation leads us to our second category of domains, including TMS and Turn and Open. Both of these involve envelopes, but the available resources are not contended. In the contended domains, such as Driverlog Shift, all the activities of a given driver must be completed during their shift, and there is limited parallelism – a driver cannot drive two trucks at once. Or, in Cafe, the order must be completed within the specified delivery window, again with limited parallelism within the envelope. In TMS though, the kiln once firing can bake an arbitrary number of items in parallel so there is no contention in the envelope that limits the number of activities that can use the available resource in parallel. In Turn and Open, the required concurrency is simple and uncontested: the robot must turn the door handle and within that open the door. There is trivially sufficient time for this to happen in the envelope, and there are no other actions requiring that door handle to be turned that would be beneficial to also schedule within the envelope.

In these two domains we therefore see no improvement from the new heuristic because there are no issues with actions being unable to fit inside envelopes; the number of nodes expanded remains the same. We can, however, see that there are minimal overheads in using the new heuristic in these domains as the time taken to solve problems with and without it, shown in Figure 2a, are very similar. Indeed across all domains we never saw an instance where the problem was solved with the old heuristic, but is no longer solved with the new one, indicating that the overheads do not cause us to fail to solve problems when we otherwise would, even if the heuristic does not offer better guidance.

Our final group of domains are those in which Timed Initial Literals (TILs) have been compiled away and replaced with envelopes. TILs are facts that are added or deleted at a specific time and are specified in the planning problem. Such facts can instead be represented by envelopes which, for example, add the fact at the start, and delete it at the end. A short dummy envelope action can be used at the start of the plan, to ensure that all these envelopes start at the same time, and before any other actions. To model the fixed time windows in Satellite for sending images, further spacer actions must be added to 'pad' the time before the window opens. This compilation in general leads to challenging problems as it increases the number of actions in the domain, as well as the solution length; however, these existing benchmarks do show us an interesting use of envelopes to model deadlines, albeit those that are fixed in time, rather than the movable envelopes our heuristic was designed for.

Despite the challenging nature of these problems, our new heuristic allows the planner to solve an additional 7 problems in Pipes No Tankage Deadlines, almost twice as many as are solved with the old heuristic. While the extra actions introduced by the compilation clearly make the problem more difficult for the planner, the new heuristic is better at detecting when the planner can no longer meet the deadline before the envelope is closed: this leads to more states being recognised as dead-ends, giving solutions in less time and ultimately solving more problems. The Satellite domain is particularly challenging for temporally expressive planners: having open actions (for the envelopes) means that memoisation has to be very conservative (Coles and Coles. 2016) and there are many actions in Satellite that offer the opportunity to extend a plan without making progress (switch on/off, calibrate (repeatedly) and turn-to). In this domain the heuristic is able to recognise envelopes with deadlines and is able to make some improvement solving mutually solved problems faster, and solving a further two problems.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement No. 730086 (ERGO); and from EPSRC grant EP/P008410/1 (AI Planning with Continuous Non-Linear Change).

References

Bit-Monnot, A.; Smith, D. E.; and Do, M. 2016. Delete-free reachability analysis for temporal and hierarchical planning. In *Proceedings of the HSDIP Workshop at the International Conference on Automated Planning and Scheduling (ICAPS)*.

Coles, A. J., and Coles., A. I. 2016. Have I Been Here Before? State Memoisation in Temporal Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Coles, A. I.; Fox, M.; Long, D.; and Smith, A. J. 2008. Planning with problems requiring temporal coordination. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Coles, A.; Fox, M.; Halsey, K.; Long, D.; and Smith, A. 2009. Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence* 173.

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)*.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. 2007. When is temporal planning *really* temporal planning? In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Cushing, W. A. 2012. *When is Temporal Planning Really Temporal?* Ph.D. Dissertation, Arizona State University (p. 288).

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal Constraint Networks. *Artificial Intelligence* 49:61–95.

Do, M. B., and Kambhampati, S. 2003. Sapa: Multi-objective Heuristic Metric Temporal Planner. *Journal of Artificial Intelligence Research* 20.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

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

Fox, M.; Long, D.; and Halsey, K. 2004. An investigation into the expressive power of PDDL2.1. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*.

Gerevini, A.; Saetti, A.; and Serina, I. 2006. An Approach to Temporal Planning and Scheduling in Domains with Predictable Exogenous Events. *Journal of Artificial Intelligence Research* 25.

Gerevini, A. E.; Saetti, A.; and Serina, I. 2010. Temporal planning with problems requiring concurrency through action graphs and local search. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

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

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14.

Kavuluri, B., and Senthil, U. 2004. Timed Initial Literals Using Sapa. In *IPC 4 Booklet at the International Conference on Automated Planning and Scheduling (ICAPS)*.

Linares López, C.; Celorrio, S. J.; and Olaya, A. G. 2015. The deterministic part of the seventh international planning competition. *Artificial Intelligence* 223.

Long, D., and Fox, M. 2003. Exploiting a Graphplan Framework in Temporal Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Marzal, E.; Sebastia, L.; and Onaindia, E. 2014. On the use of temporal landmarks for planning with deadlines. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Rankooh, M. F., and Ghassem-Sani, G. 2015. ITSAT: An Efficient SAT-Based Temporal Planner. *Journal of Artificial Intelligence Research* 53.

Rintanen, J. 2007. Complexity of Concurrent Temporal Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

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 International Conference on Automated Planning and Scheduling (ICAPS)*.