Proceedings of the Twenty-Sixth International Conference on
Automated Planning and Scheduling (ICAPS 2016)

Have I Been Here Before? State Memoization in Temporal Planning

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

Abstract

State memoization is critical to the good performance of heuristic forward search planners, which represent a significant proportion of the current state-of-the-art planning approaches. In non-temporal planning it is sufficient to discard any state that has been generated before, regardless of the path taken to reach that state, with the only side-constraint being plan cost. We begin this paper by demonstrating that the use of this technique in temporal planning can lead to loss of optimality with respect to metrics involving makespan and that in the case of more expressive domains can lead to loss of completeness. We identify the specific conditions under which this occurs: states where actions are currently executing. Following from this we introduce new memoization techniques for expressive temporal planning problems that are both completeness and optimality preserving, solving the challenging problem of determining when two states in temporal planning can be considered equivalent. Finally, we demonstrate that these have significant impact on improving the planning performance across a wide range of temporal planning benchmarks in the POPF planning framework.

1 Introduction

Forward search planners explore the planning state space by building a tree forwards from the initial state. In general, however, the underlying state space is a directed graph, so in generating a tree the planner will encounter states more than once. Expanding these duplicate states would dramatically increase the size of the search space, so most (if not all) forward search planners make use of state *memoization* – recording which states have already been seen. When new states are generated, these are kept only if they have not already been memoized.

This technique is a crucial element of the performance of planners, but receives little attention: in classical planning, memoization is *memoryless*, and two states can be considered to be equivalent if the same facts are true. The only residual consideration might be which was reached with lowest cost. While there is some work looking at how to efficiently store the set of memoized states (Schmidt and Zhou 2011), memoization within contemporary classical planners is rarely discussed in the literature, as it is more often thought to be an implementation detail.

As planners are asked to solve increasingly expressive problems, particularly with an explicit model of time, the challenge of deciding whether two states can be considered equivalent becomes much more significant. We can no longer compare states based only on facts and/or the values of numeric variables, as the temporal constraints of the plan that reaches a state have a direct effect on whether the goals can be reached. In effect, memoization is no longer memoryless – the path that reaches some combination of facts and variable values, now matters – and determining whether two plans are equivalent is therefore much more challenging.

In this paper, our contributions are twofold. We first explore in depth the issues of memoization in temporal planning, presenting cases where classical memoization is incomplete, and where it precludes finding optimal solutions. Second, we propose alternative completeness and optimality preserving approaches to state memoization for temporal planning. We present empirical results demonstrating the performance of a temporal planner using these approaches, and we discuss the trade-offs between pruning more liberally; and maintaining optimality and completeness.

2 Problem Definition

In this paper we consider PDDL2.1 (Fox and Long 2003) Temporal Planning Problems. Two classes of actions are available in this formalism: instantaneous, and durative.

Each instantaneous action A has precondition $\text{pre}(A)$ which must be true for A to be applied. If A is applied its effects are realized. $\text{eff}^+(A)$ and $\text{eff}^-(A)$ denote propositions added and deleted. $\text{eff}^{num}(A)$ denotes the numeric effects.

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. Finally, the action has a duration constraint: a conjunction of numeric constraints applied to a special variable dur_A denoting its duration. Here we assume that dur_A does not appear in numeric effects of actions.

Following (Long and Fox 2003a), 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 $\text{pre}_\vdash A$ 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.

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

- $F^+(p)$ ($F^-(p)$) 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.
- $FP^-(p)$, similarly, records negative preconditions on p.
- $V^{\mathit{eff}}(v)$ gives the index of the step in the plan that most recently had an effect upon variable v;
- $VP(v)$ is a set containing the indices of steps in the plan that have referred to the variable v since the last effect on v. A step depends on v if it either has a precondition on v; an effect needing an input value of v; or is the start of an action with a duration depending on v.

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, the 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 the next deletor of p will, ultimately, be ordered after it. Similarly, the precondition $\neg p$ is ordered after some $F^-(p)$ and before the next $F^+(p)$.
- Steps modifying v are totally ordered, and steps referring to v are fixed within this order (due to effects on v being ordered after the pre-existing $VP(v)$).
- 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 labelled 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 $[lb, ub]$.

Search immediately discards states with inconsistent STNs: those with negative-length cycles. The task of planning 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.

Our work explores pruning in temporal planning, based on memoization: identifying states that can be deemed equivalent to those already seen. This is related to the idea of identifying symmetries. Existing work has looked at eliminating symmetric states, e.g. (Fox and Long 1999; Pochter, Zohar, and Rosenschein 2011; Domshlak, Katz, and Shleyfman 2012); and identifying plan permutation symmetries e.g. (Long and Fox 2003b). The latter is most related to our approaches, that look for equivalences between plans, though the prior work is not within temporal planning.

Throughout this paper we discuss our new techniques within the framework of POPF. We rely on a single key assumption: that each new action added to the plan is only ever ordered *after* existing actions and never explicitly ordered before them: this assumption holds for any planner performing forward search. CRIKEY and COLIN can be thought of as restrictions of POPF where each step i is ordered ϵ after step i-1, regardless of whether the steps interact in any way. The same restriction applies in decision-epoch planners, such as TFD (Eyerich, Mattmüller, and Röger 2009) and Sapa (Do and Kambhampati 2003), but in addition these planners do not use an STN; instead, as a simplification, when an action is started, its timestamp is fixed, and its end added to the event queue to occur at some fixed future time. All arguments that refer to POPF apply to any of the above planners because POPF's plan representation is a generalisation of the plans that can be represented by other planners (the converse is not true). A totally ordered plan can be converted to a POPF plan representation by applying the steps in order using the POPF annotation update rules, thereby yielding an STN. Thus we present our techniques within the POPF framework allowing generalisation to all the above planners.

3 Memoization: Completeness & Optimality

In classical propositional planners, state memoization is very effective for avoiding redundant search. If two sequences of actions $P = [p_0..p_n]$ and $Q = [q_0..q_m]$ reach the same state – i.e. the same facts are true – then only one of these must be kept. Simply, all plan extensions that would reach the goal from P would do so from Q, and vice-versa.

When planning with numeric state variables, the the principle is the same, but static analysis can additionally reveal *dominance constraints* on state variables, which can be exploited to do better than asserting that two states are interestingly different if their numeric state variable values differ in any way (Hoffmann 2003; Kvarnström, Doherty, and Haslum 2000). If it can be proven that larger values of v are better (in terms of preconditions referring to v, as an input to effects on other variables, and according to the plan quality metric) then for two states P and Q identical modulo the value of v, and where $P.v > Q.v$, only P need be kept. Any plan extension from Q would work from P, and would reach a state with better or equal cost; but not vice-versa. In general, we can say that for states in which the same facts are true, and variables for which there is no clear dominance hold identical values, then we need only keep the *Pareto front* of these states – any such state that is Pareto-dominated by another, can be pruned.

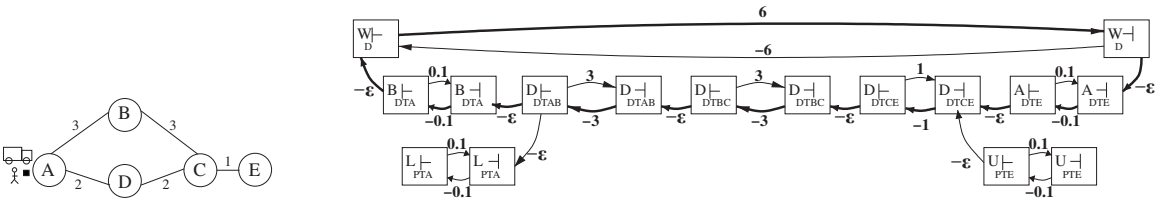


Figure 1: Left: Example Driverlog Problem. Right: STN for Solution Plan with Shifts; actions and parameters are abbreviated with their first letter, except disembark which is labelled A (alight). (For instance 'DDTCE' is `(drive driver truck C E)`.) Some STN edges that are subsumed transitively by others have been omitted for clarity.

In the remainder of this paper we refer to memoization of this nature, with Pareto dominance for numeric variables if present as *STRIPS memoization*, and use *STRIPS*(S) to refer the projection of the state S onto only its facts and numeric variables.

The introduction of durative actions changes the problem fundamentally. In the classical setting, if two permutations of actions reach the same state, only one permutation needs to be kept. In temporal domains, however, the presence of temporal constraints invalidates the assumption that all extensions of states S and S', where *STRIPS*(S)=*STRIPS*(S'), will reach the goals from both S and S'.

3.1 STRIPS Memoization is Sub-Optimal

To demonstrate sub-optimality we use a problem in the Driverlog domain, with the initial state shown in Figure 1 (left) (Long and Fox 2003c). The numbers on edges are the time taken to drive between the respective locations; board, load and unload all have duration 0.1 (recall that loading and unloading packages does not require the driver). The goal is that the package is at E. The planner may find the following plan (written in terms of snap-actions):

0:(board driver truck A)$_\vdash$	1:(board driver truck A)$_\dashv$
2:(load pkg truck A)$_\vdash$	3:(load pkg truck A)$_\dashv$
4:(drive driver truck A B)$_\vdash$	5:(drive driver truck A B)$_\dashv$
6:(drive driver truck B C)$_\vdash$	7:(drive driver truck B C)$_\dashv$
8:(drive driver truck C E)$_\vdash$	9:(drive driver truck C E)$_\dashv$
10:(unload pkg truck E)$_\vdash$	11:(unload pkg truck E)$_\dashv$

Suppose STRIPS memoization is employed in search, and having applied snap actions 0–7, the state S is reached and memoized. If search later considers an alternative plan, where the truck moves from A to D to C, instead of going via B, the state S' reached by this alternative plan will be pruned: *STRIPS*(S)=*STRIPS*(S'). However, it is on the path to a plan with a better makespan, in this case the optimal solution, which would now be pruned from the search space. Of course this is a simple example to illustrate the point but in general, if we wish to preserve optimality and we have two plans that reach states S and S' where *STRIPS*(S)=*STRIPS*(S'), we can only prune S' if we have already seen S *and* the plan via S will admit a solution of better or equal quality according to the metric.

Temporal Fast Downward (Eyerich, Mattmüller, and Röger 2009) addresses this optimality concern by comparing states by facts, makespan, and how long it is until the end of any open actions. This is appropriate when performing Decision Epoch Planning (Cushing et al. 2007); but as this is incomplete in PDDL2.1 domains with required concurrency, this is not a general-purpose solution. In general, a single makespan figure is not a sufficient criterion for determining whether one state is better than another; and there is no fixed schedule of when actions are due to end.

Of course not all plans that lead to the same STRIPS state are interestingly different. Suppose we add to our running Driverlog example a second package at A, to be delivered to location D. There is no useful distinction between plans that permute starting to load p1 and p2 onto the truck. This would trivially be captured by STRIPS memoization, as the plans reach the same facts. However, in the general case for temporal planning, we cannot prune permutations unless the plans also have equivalent temporal constraints. Currently the POPF family of planners has to keep both states. Memoisation in decision epoch planners prunes a state S' with timestamp $t(S')$, if a state S has been seen where *STRIPS*(S)=*STRIPS*(S'), $t(S') \geq t(S)$, and the event queue in S is the same as that in S'. In a decision-epoch planner, starting to load p1 before p2 or vice versa leads to states S, S' where *STRIPS*(S)=*STRIPS*(S'), but the times of the queued ends of actions are different by ϵ – so both states must be kept. This is a significant source of inefficiency, where our advanced memoisation techniques can improve efficiency for all of these approaches; we return to this later.

3.2 STRIPS Memoization is Incomplete

So far we have seen that optimality can be compromised when using STRIPS memoization in temporal planning; but worse, even completeness is not guaranteed. Suppose we modify our Driverlog instance in Figure 1 to model the shifts worked by drivers, similar to the Driverlog Shift domain (Coles et al. 2009b). We add an action `work` (applicable only once) of duration 6 that adds `(working driver)` at the start, and deletes it at the end. This fact then becomes an invariant condition of the `board` and `drive` actions.

An STN for a temporally invalid plan for this problem is shown in Figure 1 (Right). An STN is invalid *iff* it contains a negative cycle, which in this case (highlighted in bold) is due to attempting to schedule 7 time units of `drive` actions, within the 6 time units allowed by `work`. An alternative valid plan is to drive via D rather than C, reducing the total `drive` time to 5, thus eliminating the highlighted inconsistency.

Let us consider what happens during building this temporally invalid plan. We begin with the first 6 steps:

0:(work driver)$_\vdash$	
1:(board driver truck A)$_\dashv$	2:(board driver truck A)$_\dashv$
3:(load pkg truck A)$_\vdash$	4:(load pkg truck A)$_\dashv$
5:(drive driver truck A B)$_\vdash$	

At this point our intuition tells us that the planner has made a mistake that now means no extension of this plan can reach the goal. Unfortunately, the planner cannot detect this: in the general case it is quite possible that some plan exists starting with these actions, and proving otherwise has the same complexity as Plan Existence. The planner can therefore continue to extend this plan, applying the steps:

6:(drive driver truck A B)$_\dashv$	
7:(drive driver truck B C)$_\vdash$	8: (drive driver truck B C)$_\dashv$

...and then memoize the resulting state S: the truck is at C; containing the driver and package. This is an important point: the valid plan, driving via D, must pass through a state S' where *STRIPS*(S)=*STRIPS*(S'). Note that even adding facts to note which actions are open in a state (Long and Fox 2003a) would not make *STRIPS*(S)$\neq$*STRIPS*(S'). Thus, if the plan via S is generated before that via S', STRIPS memoization will prevent generation of the valid plan. However, again the planner is not able to detect this problem so will carry on searching. (Note that the plan is still temporally consistent, and would remain so even if work$_\dashv$ were applied as step 9.) The planner now applies the final actions:

9:(drive driver truck C E)$_\vdash$	10:(drive driver truck C E)$_\dashv$
11:(unload pkg truck E)$_\vdash$	12:(unload pkg truck E)$_\dashv$
13:(work driver)$_\dashv$	

3.3 STRIPS Memoization is Complete for States with No Open Actions

There are certain conditions in temporal planning where using STRIPS memoization preserves completeness but not optimality; specifically, in states with no open actions. As we have seen, incompleteness due to STRIPS memoization arises in a specific situation: two plans lead to S and S' where *STRIPS*(S)=*STRIPS*(S'); but only one of these can be extended to produce a temporally consistent plan. We therefore begin by considering the conditions under which a plan may be pruned due to temporal inconsistency.

First, observe that during state expansion in POPF, described in Section 2, all temporally invalid plans are immediately discarded; so if we are applying an action in a state, the incumbent STN must be temporally consistent. Second, new actions can only be demoted (ordered after) existing plan steps with which they have a conflict: ordering them before existing actions is not permitted. This is consistent with the forward nature of search. Adding a start snap action $B_\vdash$ to a plan $[a_0..a_n]$ can therefore only introduce STN constraints of the form $\langle a_{n+1}, a_i \in [a_0..a_n], \{0, -\epsilon\}\rangle$. To make the incumbent STN become inconsistent we need to create a negative cycle; since there were none beforehand, any new cycle must go via the new vertex $B_\vdash$. But, we cannot create a temporal inconsistency this way, because $B_\vdash$ has no incoming edges: no actions are yet ordered after $B_\vdash$, as it can never be ordered before existing steps; and no later steps have been applied after it yet.

Temporal inconsistencies can, however, occur when an end snap action $C_\dashv$ is applied: it can be constrained to be ordered after existing steps, but crucially, a maximum-duration constraint $\langle C_\vdash, C_\dashv, ub\rangle$ is added to the STN. This bounds the time available for activities ordered to occur between these points. Of course, because applying $C_\vdash$ necessarily implies a commitment to eventually applying $C_\dashv$, the plan effectively becomes temporally inconsistent at the point where an action has been applied that implies this maximum duration constraint will be violated. For instance, in the example in previous section, when (drive driver truck A B)$_\vdash$ was applied, there was no longer time to complete the requisite activities before (work)$_\dashv$. However, this issue is only detected, and the plan only pruned, when (work)$_\dashv$ is actually applied: until that point we do not know which actions it will need to be ordered after in order to satisfy its end preconditions.

Recall, for incompleteness to arise, there must be some plan extension that would lead to a temporally valid solution from S', but not from S; but *STRIPS*(S)=*STRIPS*(S'), and search encountered S first. Let π be the plan reaching some state S^π, nominally the current state, and $\langle A, T\rangle$ be the corresponding STN as generated by POPF for π. Let π' be any future extension of π comprising actions applied after we reach S^π, leading to a state S^P with STN $\langle A', T'\rangle$, and plan $P = (\pi : \pi')$. We define the subsets of T' containing only edges that start and end in π and π', respectively, as:

$$T'(\pi) = \{\langle i, j, c\rangle \in T' \mid i \in \pi \wedge j \in \pi\}$$
$$T'(\pi') = \{\langle i, j, c\rangle \in T' \mid i \in \pi' \wedge j \in \pi'\}$$

If S^P is temporally invalid either:

1. $T'(\pi)$ contains a negative-cost cycle;
2. $T'(\pi')$ contains a negative-cost cycle;
3. There is a negative-cost cycle in T' due to edges between steps in π and steps in π'.

In case 1 the state S^π would never be considered for extension or memoized, as it would have immediately been discarded. In case 2, because π' is itself inconsistent it cannot be a valid extension to *any* plan: π' is never a temporally valid extension of π from S^π, regardless of the actions in π.

This leaves us with case 3. For a cycle to occur between the vertices for π and π', there must be some edge from a vertex in $T'(\pi)$ to one in $T'(\pi')$; and another from $T'(\pi')$ to $T'(\pi)$. As noted above, applying the steps π' after π introduces ordering constraints – these are all backwards edges from $T'(\pi')$ to $T'(\pi)$, as threats are resolved and preconditions are met by ordering new actions *after* existing actions. The only forward edges that could go from $T'(\pi)$ to $T'(\pi')$ are the maximum duration constraints of actions. So, for there to be an edge from $T'(\pi)$ to $T'(\pi')$ there must be an action A where $A_\vdash$ is in π and $A_\dashv$ is in π'; that is, there was an open action in S^π.

This makes intuitive sense: if we have started an action but not yet finished it, we have a commitment to respect the duration constraint of that action, and could potentially make the plan temporally invalid if we do not do so. However, if all actions have already finished then applying an action cannot possibly break any of the existing constraints, as it will only be ordered after the existing actions.

At first glance it might seem that the potential impact of this observation is limited as for all durative actions we must

apply the start, and then apply the end at some later time, thereby passing through a state with an open action. However, POPF identifies compression safe actions: those whose end can be added to the plan immediately after its start without compromising completeness (Coles et al. 2009a).

We will now show that if we apply $A_\dashv$ immediately after $A_\vdash$ we cannot create negative cycles in the STN. Suppose $A_\dashv$ is ordered after some snap-action B. Because $A_\dashv$ can only be ordered after previous actions, and there was no opportunity to apply an action between $A_\vdash$ and $A_\dashv$, then either B=$A_\vdash$ or B was already in the plan *before* $A_\vdash$. In the former case, a negative cycle would imply that the action's duration constraints contradicted themselves: the lower bound is in excess of the upper bound. We can reasonably assume such an action would not even be started, as it is not applicable. Thus, we focus on the latter case. For there to be a negative cycle involving $A_\vdash$ and $A_\dashv$ there must be a path from $A_\dashv$ to $A_\vdash$ via B. However, there cannot be an edge from B to $A_\vdash$ because B was already in the plan before $A_\vdash$ was applied; so if $A_\vdash$ needed to be ordered with respect to B it would have to come after B. Thus, only edges from $A_\vdash$ to B could be present in the STN: there can be no edges from B to $A_\vdash$, so applying $[A_\vdash, A_\dashv]$ cannot cause a negative cycle.

4 Memoization Strategies

We now propose three new techniques all of which preserve completeness; and two of which preserve optimality.

4.1 Restrict STRIPS Memoization to States with No Open Actions

As discussed in Section 3, STRIPS memoization in temporal planning can be incomplete *iff* it is used to memoize states with open actions. A minor tweak is as follows: if a state has no open actions, apply STRIPS memoization; otherwise, keep it. The advantage of this approach is that as many domains are compression safe, most of the time STRIPS memoization will be used. The disadvantage though is that as soon as this is not the case, for instance in domains with required concurrency (e.g. envelope actions as in Driverlog Shift) most states are kept.

4.2 Memoization of Isomorphic Partial Orders

In the general case, permuting the order in which actions are applied will affect the ordering constraints between them; and this in turn affects whether it is possible to reach the goal. However, following the forward partial-order semantics of POPF, two plan steps a_i, a_{i+1} are only ordered with respect to each other if they interact in some way: an effect of one meets or threatens the precondition of the other.

As an example, consider a simple Driverlog problem where two packages *p1*, *p2* and a truck are at a location. At least two actions are applicable: start loading *p1* onto the truck; or start loading *p2*. These would each be ordered after the action that moved the truck to that location, but would not need to be mutually ordered. Thus, once both have been applied, in either order, the resulting partial order plans are isomorphic: the nodes in the graph are labelled with the same actions, and the edges carry the same temporal constraints. A cautious memoization strategy would still keep both plans, though, as in general, the steps *might* have been mutually ordered. But, if they are not, we need only consider one of any set of plans that have isomorphic partial orders in order to maintain completeness and optimality.

Detecting plans with partial orders isomorphic to those already seen corresponds to solving a very restricted form of coloured graph isomorphism. We can exploit extensively the fact that the graph corresponds to a partial-order plan found incrementally by forward state progression. First, we transform the STN (A, T) corresponding to a plan P into a coloured digraph with vertices V and unlabelled edges E. We define a morphism $\sigma(a_i)$ that maps each vertex $a_i \in A$ to some $v \in V$, where v is coloured according to the snap-action it represents, and the instance of that action in P:

- If a_i is the nth instance of an instantaneous action A, $\sigma(a_i)$ is coloured $\langle A, n\rangle$;
- If a_i is the nth instance of $A_\vdash$ $\sigma(a_i)$ is coloured $\langle A_\vdash, n\rangle$;
- If a_i is the nth instance of $A_\dashv$ $\sigma(a_i)$ is coloured $\langle A_\dashv, n\rangle$.

We also define σ^{-1} as the inverse of this, that maps a vertex $v \in V$ to some plan step $a_i \in P$.

Note that including n within the colour means our approach will not detect all isomorphisms. If the same snap-action is added to the plan twice, but there is no mutual ordering (even transitively), they could in principle be interchangeable, but n forces them not to be. For instantaneous actions that have effects, an instance of A will always be ordered after earlier instances of A as it affects the same facts and/or variables, so this is not a concern. For durative actions, though, we may obtain snap-actions with no effects, which in turn allow us to self-overlap an action, i.e. to start it several times before finishing it. In practice, such plans are uncommon, so this is not a considerable limitation, and our approach degrades gracefully: it is not complete (will not find all isomorphisms), but it is sound (guaranteeing we will never consider two non-isomorphic plans to be equivalent).

An edge $\langle\sigma(a_i), \sigma(a_j)\rangle \in E$ denotes that step a_i precedes a_j. Edges are derived from T as follows:

- If $\langle a_j, a_i, c \in \{0, -\epsilon\}\rangle \in T$, due to an interaction between steps a_i and a_j, there is an edge $\langle\sigma(a_i), \sigma(a_j)\rangle \in E$. Note the label is discarded – referring to Section 2, whether the gap is 0 or ϵ depends solely on the actions involved; this is subsumed by the colours of the vertices.
- If $\langle a_j, a_i, -lb\rangle \in T$ due to the lower-bound on the duration of the action that started at a_i and finished at a_j, then $\langle\sigma(a_i), \sigma(a_j)\rangle \in E$. lb is either constant or a function of the state in which the action was applied. If two partial-order plans are isomorphic, the state in which the action was applied is the same, so this edge necessarily has the same label; and thus the label can be ignored.
- Edges $\langle a_i, a_j, ub\rangle \in T$ are ignored. As in the lower-bound case, the upper-bound depends only on the state in which the action was applied, and an isomorphic plan would reach this same state. Further, the n values on vertices capture the pairing between starts and ends, implicitly capturing that there is *some* maximum constraint here.

Algorithm 1: Canonical Vertex Colour Order

Data: $\langle V, E\rangle$, a coloured graph
Result: cs, a canonical sequence of vertex colours
1 $cs \leftarrow []$;
2 **while** $V \neq \emptyset$ **do**
3 $\quad open \leftarrow \{v \in V \mid \forall v' \in V.\langle v', v\rangle \notin E\}$;
4 $\quad next \leftarrow (v \in open \mid \forall v' \in open, v' = v \vee v < v')$;
5 $\quad$ append $next$ to cs;
6 $\quad V \leftarrow V \setminus \{next\}$;
7 $\quad E \leftarrow \{\langle i, j\rangle \in E \mid i \neq next\}$;

For a graph $\langle V, E\rangle$, we can then derive a canonical form: a sequence of vertex colours, obtained by visiting the vertices in a specific order. The vertices are visited by a topological order traversal; where ties between which vertex to visit next are broken by using an ordering relationship based on their colour. To order vertices, each action A is given an arbitrary but unique identifier $id(A)$[1]. We then sort the space of possible colours in ascending order as follows:

- Colours $\langle A, n\rangle$, sorted lexicographically by $\langle id(A), n\rangle$ – i.e. for an action A, $(\langle A, j\rangle < \langle A, k\rangle)$ if $(j < k)$.
- Colours $\langle A_{\vdash}, n\rangle$, sorted lexicographically by $\langle id(A), n\rangle$;
- Colours $\langle A_{\dashv}, n\rangle$, sorted lexicographically by $\langle id(A), n\rangle$.

With this ordering, we then use the notation $v < v'$ to test whether some vertex's colour is less than another according to this order. This is used within the topological-order traversal shown in Algorithm 1 – at line 4, from the topologically open vertices (those with no incoming edges), the next vertex chosen is that which is less than all others. We can be confident this ordering is canonical, as it inherently preserves topology, and no two vertices have the same colour: where the same action appears more than once in a plan, the n values distinguish the respective vertices.

As an example of the output of this algorithm, we refer to the partial-order plan in Figure 2. Here, the plan $[AFCG]$ has been extended by applying B as step 4 of the plan. The coloured graph for this partial order will colour the vertices according to the actions they denote (A, F, C, G or B), and the edges will be inverted from those in the partial order: $A \rightarrow F$, $F \rightarrow C$, and so on. The canonical sequence, following Algorithm 1 is then $[AFBCG]$ – after visiting A and then F, there are two open vertices, C and B; and assuming alphabetical order, $B < C$, so B is chosen first.

There is one task remaining with the canonical sequence: using σ^{-1}, map it into a canonical permutation of steps from the original plan P. For a canonical sequence $cs = [v_0..v_n]$ we define $cp = [\sigma^{-1}(v_0)..\sigma^{-1}(v_n)]$. cp has a partial-order that is isomorphic to that of P, but the particular snap-action at a particular step index may have changed, due to the ordering constraint between vertex colours. Crucially, we can now define memoization based on not visiting two plans with the same partial order, very succinctly:

- For each plan P, compute the canonical plan cp.

[1]Our implementation assigns these based on grounding order.

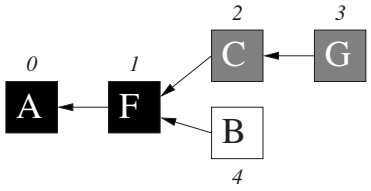


Figure 2: Applying B after $AFCG$

- If cp has not been seen before memoize it and keep P;
- If cp has been seen before, prune P.

Thus, with reference to Figure 2, if the plan $[AFCGB]$ was considered during search, and was the first where $cp = [AFBCG]$, it would be kept, and cp memoized. If the plans $[AFBCG]$ or $[AFCBG]$ (which have partial-orders isomorphic to Figure 2) were later considered they would be pruned, as their canonical orders are also $[AFBCG]$.

4.3 Memoization using Fact Availability Times

Eliminating plan permutation symmetries is effective at pruning states reached by a different permutation of the same actions, but does nothing for the case where different actions reach similar states that are either interchangeable or, provably, one is better than the other.

For a plan P=$[a_0..a_n]$ that reaches a state S with temporal constraints T, solving T using a shortest-path algorithm returns the earliest time at which each step can be applied. If there are no open actions, and the STN was consistent, these are acceptable timestamps for these steps in all plans reachable from S. This follows POPF's state progression semantics: the only new temporal constraints that appear when appending new steps to P are either between these new steps; or order new steps a *minimum* amount of time (0 or ϵ) after steps in P. A *maximum* amount of time between steps in P and new steps is never required, so in turn, delaying the steps in P is never required. Conversely, if there are open actions, ending an action limits the amount of time between its start and its end, potentially requiring the start to be delayed to satisfy the duration constraint; so we cannot yet fix the times at which steps occur. Delays may also be beneficial in problems with continuous or duration-dependent effects, where the metric is to maximize makespan, or for Timed Initial Literals (Hoffmann and Edelkamp 2005); but these are outside the scope of our work.

Given the timestamp $t(a_i)$ of each step a_i in a plan to reach a state S with no open actions, and the annotations in states in POPF, we can derive several timestamp values:

- $use(S, p)$=$\max\{t(F^+(p)), t(F^-(p))\}$: the fact p can be used after the time of the step that last added/deleted it;
- $change(S, p)$=$\max\{t(i)$+$d \mid \langle i, d\rangle \in FP^+(p)\}$: p can be deleted after the latest step of which it is a precondition;
- $use(S, \neg p)$=$\max\{t(F^+(p)), t(F^-(p))\}$;
- $change(S, \neg p)$=$\max\{t(i)$+$d \mid \langle i, d\rangle \in FP^-(p)\}$;
- $use(S, v)$=$t(V^{\mathit{eff}}(v))$: v can be used after its last modifier;
- $change(S, v)$=$\max\{t(i)$+$d \mid \langle i, d\rangle \in VP(v)\}$;

These values can be defined for every fact p and variable v in the planning task. With these we can now make meaningful comparisons between states S and S' where *STRIPS*(S)=*STRIPS*(S') and no actions are executing. Simply, if the state S *Pareto dominates* S' in terms of these values, S' can be discarded.

$$\begin{aligned} dom(S,S') \vdash \quad & \forall p(\quad use(S,p) \leq use(S',p) \\ & \qquad \wedge change(S,p) \leq change(S',p)) \\ & \wedge \forall p(\quad use(S,\neg p) \leq use(S',\neg p) \\ & \qquad \wedge change(S,\neg p) \leq change(S',\neg p)) \\ & \wedge \forall v(\quad use(S,v) \leq use(S',v) \\ & \qquad \wedge change(S,v) \leq change(S',v)) \end{aligned}$$

The rationale for this definition is intuitive: if the plan to S' was extended to reach the goal, then the same plan extension could be applied from S; and no action would be scheduled any later, because no *change* or *use* value in S exceeds its counterpart in S', thus optimality is preserved.

The caveat behind these exact definitions of *change* or *use* is that they can substantially underestimate the actual times at which actions that use or changes facts/variables could be applied. For instance, in Driverlog, if a truck moves from A to B, deleting `(at t1 A)` at time 0 and adding `(at t1 B)` at time 3, then $change(\texttt{at t1 A}) = 0$ and $change(\texttt{at t1 B}) = 3$. The former of these is a substantial underestimate, but reflects the fact that in principle, an instantaneous action with no preconditions and the single effect `(at t1 A)` could be applied ϵ after that time. In reality, though, such an action does not exist – moving the truck back to A would require deleting the fact that it was at B, which means coming after time $change(\texttt{at t1 B})$.

Worse, if the truck in one plan moves from A to B to C, and in another from A to D to C (see Figure 1, left) these two plans would have different *change* values for $change(\texttt{at t1 B})$ and $change(\texttt{at t1 D})$. This would break the Pareto dominance, and both plans would be kept; even though going via D is preferable. Again, in reality, changing the value of these intermediate facts would require deleting C, and hence coming after time $change(\texttt{at t1 C})$.

Generalizing this, we observe that the facts such as `(at t1 A)` and `(at t1 B)` are *mutually exclusive* given the actions in the domain, and the initial state: it is impossible to reach a state in which both are true; and hence, if one is true, it must be deleted before the other is made true. Using $mutex(p)$ to denote the set of facts mutually exclusive with p, we can define $use'(S,p)$ and $change'(S,p)$ as follows:

$$use'(S,p) = \max\{use(S,p), \max_{p' \in mutex(p)} change(S,p')\}$$

$$change'(S,p) = \max\{change(S,p), \max_{p' \in mutex(p)} change(S,p')\}$$

That is, if p is true, then we can use it after the last point it was added ($use(S,p)$); otherwise, we must first change (delete) the other facts in the mutex group before an action that added it could conceivably be applied. Similarly, if p is true, then we can change (delete) it after $change(S,p)$; but if it is false, then the step that changes (adds) it must follow the deletion of the other facts in the mutex group. Using these modified versions of $use(S,p)$ and $change(S,p)$ in dom gives us a stronger definition of Pareto dominance.

4.4 Applicability to Other Forwards Planners

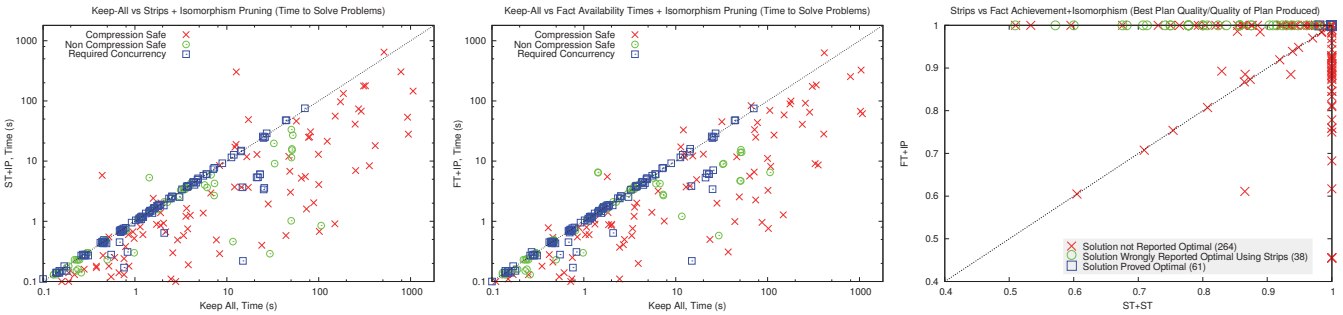
Application of our remaining techniques to other forward search planners that impose a total ordering can be achieved by running the plan to each state through POPF state progression (as noted at the end of Section 2). When this is done the techniques are directly applicable to all such planners, and the benefits for STN-based planners (e.g. COLIN and CRIKEY) are as described for POPF.

Decision-epoch planners already implicitly benefit from using STRIPS memoization in states *with no open actions*. For states S, S' with timestamps $t(S), t(S')$, and where *STRIPS*(S)=*STRIPS*(S'), S' will be pruned if $t(S') \geq t(S)$, and the event queues are the same (in this case both empty). But, suppose states have open actions. The plans $[A_\vdash, B_\vdash]$ and $[B_\vdash, A_\vdash]$ lead to different states, even if the facts and timestamps are the same, as the event queues differ: S has $A_\dashv$ queued at (t+dur_A) and $B_\vdash$ at (t+ϵ+dur_B) while S' has $A_\dashv$ at (t+ϵ+dur_A) and $B_\vdash$ at (t+dur_B). Our techniques improve the situation here in two regards. In domains where all actions are compression safe, if two event queues contain the same actions, these by definition do need not be ordered with respect to each other, so the two event queues can be considered equal. In other domains, isomorphism pruning can detect cases where no ordering constraints need to exist between A and B (i.e. they do not interact) allowing one of these two states to be pruned. Finally, as decision epoch planners also usually lift a partial order at the end of search, Section 4.3 allows tie-breaking between states that have equal timestamps during search; but where one will admit a better partial-order plan than the other, when the goals are reached and the partial-order plan is lifted.

5 Evaluation

In this section we compare the performance of our memoization strategies within the POPF framework. To allow us to focus on memoization all planner configurations use WA* search with W=5. All tests are run on 3.5GHz machines, restricted to 30 minutes of CPU time and 4GB of memory. We used all temporal benchmark domains from the International Planning Competition (IPC) series, and collated required concurrency domains from the temporal planning literature (since these are rare in IPCs). As memoization based on fact availability times relies on mutex groups, for which we use the Temporal Fast Downward SAS+ translator, we include only those domains that it supports. Finally, we created one new domain with required concurrency, 'crewplanning envelope'. The original IPC2008 temporal crewplanning domain does not correctly enforce the temporal constraints of the problem, where activities have deadlines on certain days. We added envelope actions to enforce these constraints and capture the maximum duration of days, capturing the interesting temporal features of the problem.

We present results for seven planner configurations in Table 1, defining each by what happens in states with and without open actions. We include two reference configurations: Keep All states, which necessarily preserves completeness and optimality; and STRIPS memoization applied to states with and without open actions, to indicate the price we are



(a) Keep All vs ST+Isomorphism Pruning (b) Keep All vs FT+Isomorphism Pruning (c) Quality in STRIPS+STRIPS vs FT+IP

Figure 3: Comparison of Approaches in terms of Time (left and centre) and Quality (right)

paying to maintain completeness and optimality. We split our domains into three categories. First, Compression Safe (CS), where all actions in the domain are compression safe, so no states have open actions, and STRIPS memoization is completeness (but not optimality) preserving. Second, Non-Compression Safe: these contain some actions that the planner is not able to identify as compression safe, though in some cases this is due to limitations of the analysis (e.g. end numeric effects are taken not to be compression safe), rather than indicating required concurrency (Cushing et al. 2007). Finally, domains with required concurrency, where actions must be executed simultaneously in any solution plan.

Table 1 presents *coverage* figures, i.e. the number of problems solved. The practical limit on whether a problem was solved was memory usage, rather than CPU time: across all unsolved problems, there were only seven cases where the planner timed out before it ran out of memory. Improved coverage is indicative of effective memoization, as keeping states in memory accommodates for the majority of the memory usage. In CS domains, where it is complete, STRIPS memoization solves the most problems: this is to be expected as it strictly dominates all other strategies in terms of nodes pruned; albeit at the cost of optimality. Comparing STRIPS to Keep All it is clear that effective memoization is crucial to the performance of forward-chaining planning. Note that in CS domains, there are no open actions, so the 'Open Actions' strategy (second row of the table) is never used: all STRIPS-based configurations perform identically.

For non-CS domains, and those with required concurrency, the table indicates in parentheses after the domain name the percentage of states generated by all configurations during search in which there were open actions; i.e. the fraction of states that were handled by the second memoization approach. In these domains, the incompleteness of STRIPS memoization becomes more apparent: in ST+ST, all problems in parcprinter and both variants of UMTS are reported as unsolvable, when in fact many of these are solved by other configurations. The 7 time outs were on these non-CS domains: the pruning was sometimes so over-zealous, the planner did not generate enough non-pruned states in 30 minutes, to use all the memory. However, in spite of this, it still finds solutions to many problems in domains where it is in theory incomplete, as the over-zealous pruning pays off in terms of scalability, by keeping far fewer states in memory.

The configuration pruning using solely Isomorphic Plan (IP) pruning solves a total of 25 more problems than keeping all states. This is very promising as IP is both complete and optimal in all settings, and in principle can be directly applied to domains that are more expressive (e.g. those with continuous numeric change and duration-dependent effects). The best results here, however, arise when using STRIPS pruning in states with no open-actions, and IP otherwise: making use of the most powerful approach that is complete for the class of states being respectively considered.

We would naturally expect the combinations using Fact Availability Time (FT) pruning to have slightly lower coverage than their STRIPS analogues, as STRIPS memoization prunes strictly more states. However, it is pleasing to note, given FT+IP preserves completeness *and optimality*, that it solves 100 more problems than Keep All; and is just 14 short of the best non-optimality preserving configuration, ST+IP. The graphs in Figures 3a and 3b also demonstrate a convincing decrease in time taken to solve mutually solved problems for ST+IP and FT+IP compared to the only prior completeness and optimality preserving approach, Keep All.

In order to see the benefits of FT pruning we must consider solution quality. To optimise quality we allowed WA* search to continue after a solution was found, pruning states where the reachable metric value is no better than that of the incumbent best solution. We estimated the reachable metric using the admissible makespan estimate derived from the Temporal Relaxed Planning Graph. In all temporal benchmarks considered the metric specified is 'minimize total-time', with the exception of ZenoTravel which requires minimisation of a weighted sum of total-time and fuel used.

Figure 3c compares the quality of plans produced by using ST+ST versus FT+IP, computing plan quality scores in the range $[0, 1]$ as follows: for each problem, find the best quality solution (by any configuration); then divide this by the quality of the plan found by a given configuration to obtain its score (bigger is better). In almost all mutually solved problems the best quality plan is found by one of these two configurations, but there is a split as to which performs better: ST+ST is better in 69 problems, FT+IP in 52. The two produce solutions of the same quality in 136 problems.

This illustrates an trade off in optimisation for satisfycing

States without Open Actions		KA	ST			IP	FT	
States with Open Actions	#P	KA	KA*	IP*	ST*+	IP	KA	IP
02-rovers-simple†	20	17	18	18	18	18	18	18
02-zeno-simple†	20	13	15	15	15	13	14	14
02-depots-simple	22	3	11	11	11	4	8	8
02-depots†	22	3	11	11	11	4	10	10
02-driverlog-simple†	20	12	15	15	15	12	15	15
02-driverlog†	20	11	15	15	15	14	16	16
02-satellite-simple	20	9	9	9	9	9	9	9
02-satellite†	20	6	8	8	8	7	9	9
04-pipes-notankage†	50	18	32	32	32	22	29	29
06-trucks-strips†	30	20	22	22	22	21	22	22
08-crewplanning	30	18	18	18	18	18	18	18
08-modeltrain-num	30	0	2	2	2	0	2	2
08-pegsol†	30	26	30	30	30	28	29	29
11-floortile†	20	0	9	9	9	0	8	8
11-parking	20	19	20	20	20	19	20	20
11-sokoban-strips	30	3	17	17	17	3	15	15
11-storage	20	0	3	3	3	0	2	2
CS Total	**424**	**178**	**255**	**255**	**255**	**192**	**244**	**244**
02-rovers (5%)	20	8	9	9	8	8	8	8
04-pipes-tankage (10%)†	50	8	10	11	13	8	9	10
08-elevators-num (26%)	30	4	7	7	19	5	7	9
08-transport-num (34%)	30	1	5	7	9	1	5	5
11-parcprinter (70%)	20	0	3	4	0	1	3	4
08-opnstack-num (86%)	30	29	30	30	30	30	30	30
04-umts (97%)	50	39	48	48	0	43	48	48
Non-CS Total	**280**	**108**	**131**	**136**	**105**	**117**	**130**	**135**
11-turnandopen (1%)	20	1	8	8	8	1	6	6
Driverlog Shift (81%)	20	9	9	10	13	10	9	10
08-crewplan-env (94%)	30	5	5	5	5	5	5	5
11-matchcellar (98%)	20	20	20	20	20	20	20	20
04-umts-tw-comp (98%)	50	42	42	43	0	43	42	43
P2P (Huang et al.) (99%)	13	13	13	13	13	13	13	13
11-tms (99%)	20	2	2	2	0	2	2	2
RQ Total	**173**	**92**	**99**	**101**	**59**	**94**	**97**	**99**
Total	**877**	**378**	**485**	**492**	**419**	**403**	**471**	**478**

Table 1: Coverage of each strategy on benchmark domains: Keep All (KA), Strips (ST), Isomorphism Pruning (IP), Fact Availability Times (FT). * Indicates strategies that are not optimality preserving, + those not completeness preserving. Underlined results for FT+IP and ST+ST indicate which of the two achieved the higher quality score (no underline implies the two were equal).

planning: as noted earlier, the zealous pruning of STRIPS memoization can allow it to solve more problems despite being incomplete; when optimising this translates into considering more possible plans within the time and memory limits imposed, and thus finding solutions other configurations will not reach. Indeed, in domains where ST+ST obtained a better total quality score (sum of quality scores across mutually solved problems) than FT+IP, underlined in Table 1, the coverage of ST+ST is higher too, indicating that it can find solutions more easily. It is also notable that the domains where ST+ST erroneously deemed at least one sub-optimal solution optimal (marked † in Table 1) are most of the domains in which FT+IP found better solutions. This suggests ST+ST has pruned some states on the path to good solutions.

In general we observe that in problems where it is difficult to find solutions at all, ST+ST does better in terms of quality, due to better scalability; whereas in those where the planner can find multiple solutions with more conservative pruning, FT+IP prevails. Of course, if guarantees of optimality are required then the option of using STRIPS memoization is removed entirely; FT+IP on the other hand could safely be used for memoization inside an optimal temporal planner.

6 Conclusions

In this paper, we discussed the limitations of prior approaches to state memoization when applied to temporal planning problems as expressed in PDDL2.1, in particular that STRIPS memoization does not preserve completeness or optimality. We presented alternative approaches that address these issues, and evaluated their performance. The results indicate these surpass the performance of STRIPS memoization in temporally expressive domains; whilst approaching its performance in temporally simple domains, where its incompleteness is not always a hindrance.

References

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2009a. Extending the use of inference in temporal planning as forwards search. In *Proc. ICAPS*.

Coles, A.; Fox, M.; Halsey, K.; Long, D.; and Smith, A. 2009b. Managing concurrency in temporal planning using planner-scheduler interaction. *AIJ* 173(1).

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proc. ICAPS*.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. 2007. When is temporal planning *really* temporal planning? In *Proc. IJCAI*.

Do, M. B., and Kambhampati, S. 2003. Sapa: Multi-objective Heuristic Metric Temporal Planner. *JAIR* 20.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced Symmetry Breaking in Cost-Optimal Planning as Forward Search. In *Proc. ICAPS*.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proc. ICAPS*.

Fox, M., and Long, D. 1999. The Detection and Exploitation of Symmetry in Planning Problems. In *Proc. IJCAI*.

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

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

Hoffmann, J. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *JAIR* 20.

Huang, R.; Chen, Y.; and Zhang, W. 2009. An Optimal Temporally Expressive Planner: Initial Results and Application to P2P Network Optimization. In *Proc. ICAPS*.

Kvarnström, J.; Doherty, P.; and Haslum, P. 2000. Extending talplanner with concurrency and resources. In *Proc. ECAI*.

Long, D., and Fox, M. 2003a. Exploiting a Graphplan Framework in Temporal Planning. In *Proc. ICAPS*.

Long, D., and Fox, M. 2003b. Plan permutation symmetries as a source of planner inefficiency. In *Proc. PlanSIG*.

Long, D., and Fox, M. 2003c. The 3rd International Planning Competition: Results and Analysis. *JAIR* 20.

Pochter, N.; Zohar, A.; and Rosenschein, J. 2011. Exploiting problem symmetries in state-based planners. In *Proc. AAAI*.

Schmidt, T., and Zhou, R. 2011. Representing Pattern Databases with Succinct Data Structures. In *Proc. SOCS*.