Forward-Chaining Partial-Order Planning

Amanda Coles and **Andrew Coles** and **Maria Fox** and **Derek Long**
Department of Computer and Information Sciences,
University of Strathclyde, Glasgow, G1 1XH, UK
email: firstname.lastname@cis.strath.ac.uk

Abstract

Over the last few years there has been a revival of interest in the idea of least-commitment planning with a number of researchers returning to the partial-order planning approaches of UCPOP and VHPOP. In this paper we explore the potential of a forward-chaining state-based search strategy to support partial-order planning in the solution of temporal-numeric problems. Our planner, POPF, is built on the foundations of grounded forward search, in combination with linear programming to handle continuous linear numeric change. To achieve a partial ordering we delay commitment to ordering decisions, timestamps and the values of numeric parameters, managing sets of constraints as actions are started and ended. In the context of a partially ordered collection of actions, constructing the linear program is complicated and we propose an efficient method for achieving this. Our late-commitment approach achieves flexibility, while benefiting from the informative search control of forward planning, and allows temporal and metric decisions to be made — as is most efficient — by the LP solver rather than by the discrete reasoning of the planner. We compare POPF with the approach of constructing a sequenced plan and then lifting a partial order from it, showing that our approach can offer improvements in terms of makespan, and time to find a solution, in several benchmark domains.

1. Introduction

Partial-order planning was, until the late 1990s, the most widely researched and most popular approach to planning. A central reason for this is that the least-commitment philosophy, in which decisions about action orderings and parameter bindings is postponed until a decision is forced, is an intuitively attractive one. Nevertheless, the last decade has shown the power of early commitment under informed heuristic guidance combined with comparatively much lighter-weight search machinery able to quickly generate search states and backtrack across alternatives. The general move towards state space search ignores another important benefit of partial-order plans: the flexibility in execution that they offer.

In this paper we explore the extent to which we can preserve the benefits of partial-order plan construction within the forward search framework. This is motivated by the observation that the forward search approach can be seen as committing to a sequence of choices of actions, but not necessarily to the order of their application. By reducing the ordering constraints that are imposed during the construction of the sequence of action choices we retain elements of the least-commitment approach and are able to produce partially-ordered plans with the robustness and flexibility that they can offer.

In this paper we explore the following questions:

- Can forward search planning be modified to support efficient construction of partially-ordered plans?
- Is it more effective, in terms of time or plan quality, to construct partial-order plans directly, or to lift partial-orders from totally ordered plans?

We explore these questions in the context of temporal planning with numbers and continuous effects. Temporal planning, in particular, includes problems where partial-order planning appears to be particularly important, but the interactions between partial-order structures and numeric effects increases the challenges in achieving effective handling of the necessary reasoning.

In the remainder of the paper we consider the background to the problem and then present our approach to solving it. We describe our implementation of the solution and present results comparing its performance in various domains with a partial-order lifting approach applied to the plans produced by a straightforward forward state-space search approach.

2. Background

Partial-order planning has become unfashionable, largely due to the remarkable performance gains we have witnessed in forward state-space search planners over the past decade. Despite the power of forward search, partial-order planning appears to offer a more flexible approach to temporal planning, where the precise embedding of actions in time can be delayed until constraints emerge. Temporal planning has become a far more central concern in the planning community and this appears to offer an opportunity to exploit partial-order planning benefits. However, planning with numbers, which is also an important concern, appears to be made more difficult in the partial-order framework, where the values of metric variables can become difficult to reason with under the effects of partially-ordered actions.

VHPOP (Younes and Simmons 2003) is the only recent partial-order planner to exploit the approach for temporal

planning. Zeno (Penberthy and Weld 1994) was a more ambitious planner designed to solve problems involving continuous processes as well as time, built on a UCPOP foundation (Penberthy and Weld 1992). The former used a grounded representation and was unable to handle numbers, but neither of the planners could be considered competitive or scalable (VHPOP is the more competitive, but remains far behind current leading technology).

In this work we use PDDL2.1 temporal models, in which durative actions can be considered as instantaneous actions linked by a duration and, possibly, invariant conditions. We consider temporal–numeric planning within a forward state-space search framework. In order to handle the combination of temporal, numeric and continuous effects we build on the COLIN (Coles et al. 2009a) planner, which is the only PDDL planner generally available that can handle this combination. The remainder of this paper relies on some background knowledge of the techniques employed in COLIN and its purely temporal-planning predecessor, CRIKEY 3. The important elements of this are included, for convenience, below. In the general case, states in a temporal-metric planning problem can be characterised by a tuple $\langle F, V, Q, P, C\rangle$, where:

- F is the set of atomic propositions that hold in the state.
- V is the vector of values of the task numeric variables. In COLIN, where actions can have linear continuous numeric effects, V is represented by V^{min} and V^{max}, the lower- and upper-bounds on each V respectively, in the current state (depending on how long the state persists).
- Q, the event queue, is a list of actions whose execution has begun but not yet finished. For each $(a, s) \in Q$, a identifies a ground action, and s the step at which it began.
- P is the plan to reach the current state.
- C is a list of temporal constraints over the steps in P, each of the form $lb \leq (t(j) - t(i)) \leq ub$ for a pair i, j of step indices from the plan.

Adding actions to a plan involves appending the corresponding start and end points of the action to the plan, not necessarily in successive steps. When an action is added to the plan, the state is updated. As in classical planning, applying an action (end point) a updates F and V according to its effects, with the additional constraint that a can only be applied if its effects do no conflict with the invariants of any action in Q. To respect the temporal structure of the problem, C is updated as each step is added to the plan. There are alternative approaches to this. For example, in Sapa (Do and Kambhampati 2001), each new start action is added immediately after the preceding action (allowing a separation of ϵ to avoid interactions), while end actions can be added by advancing time to the next event in Q and then enacting that event. Although Sapa does not use an explicit representation of C, this is equivalent to adding constraints to C: $t(i) - t(i-1) = \epsilon$ in the first case, where i is the index of the newly added action and $t(i)$ is its time of execution, and $t(i) - t(s) = dur(a)$ in the second case, where the ith action is the end of durative action a that started with start action s. CRIKEY 3 (Coles et al. 2008) uses a different approach, representing the temporal constraints explicitly. Adding a to the plan at index i changes C as follows:

- if step i is the start or end of an action, the sequence constraint $\epsilon \leq t(i) - t(i-1)$ is added to C.
- if step i ends the action (a, s) from Q, the constraint $mindur(a) \leq t(i) - t(s) \leq mindur(a)$ is added to C (where $mindur$ and $maxdur$ are the bounds on the duration of a at step s.)

The more complex constraints used in CRIKEY 3 generate a Simple Temporal Problem (STP) (Dechter, Meiri, and Pearl 1991) which can be used to check the temporal soundness of decisions as they are made. This additional complexity is necessary, however, in order to allow temporal planners to reason with problems that exhibit required concurrency or other more complex temporal constraints.

To perform forward search planning in domains with linear continuous numeric effects, COLIN uses an LP solver to solve an augmented instance of the STP. First, additional variables are added to record the values of the task numeric variables along the trajectory of the plan. For each step i, the variables $v_i \in V(i)$ record the values of each of V prior to i. Likewise, $v'_i \in V'(i)$ record those immediately following i. Preconditions and effects of actions can then be represented as constraints over these values:

- Numeric preconditions required at point i (the start or end of an action) are added as constraints over $V(i)$.
- Numeric invariants starting at step i and finishing at step j are added as constraints over $V'_i..V'_{j-1}$ and $V_{i+1}..V'_j$, i.e. on points within the open interval between the start and end of the action.
- Instantaneous numeric effects are added as a constraint relating V_i to V'_i. For example, $v'_i = v_i + w_i$ records that step i increases the value of v by the value of w.
- Continuous numeric effects are added as constraints over both timestamps and state trajectory variables. For example, $v_{i+1} = v_i + 2(t(i+1) - t(i))$ records that a continuous effect is changing v at a rate of 2 between steps i and $i+1$.

By combining the STP with the metric constraints, the LP can be used to check both the temporal consistency of actions (as before), and also check that any interactions between time and numbers can be resolved (for instance, by delaying the point at which a precondition is needed until sufficient continuous change has occurred.)

3. Limitations of Forward State-Space Search

Forward state-space search has proved to be an effective way to find feasible plans. A key strength, when compared to partial-order approaches, is that it avoids the need for explicit search to resolve threats by imposing a total-order on actions: each new action added to the plan comes after all those in the plan so far. The total ordering of actions ensures that each new action cannot threaten earlier preconditions or effects (it is automatically promoted), and no later action can threaten its preconditions or effects (as that will, itself, be promoted). Forward search planning, in the temporal case, additionally requires that no action appended to the plan can

threaten the invariants of actions that have started but not yet finished: threatening actions cannot be added until later, and hence are necessarily ordered to occur after the actions ending the invariants.

Whilst a total-order eliminates threats, it comes at the cost of early commitment. Suppose one were to add a sequence of non-durative actions $[A, B, C]$ to a plan, where A and B do not interfere with each other, and C depends only on B. In a forward search planner, these actions would be given sequential timestamps, say $[0, 1, 2]$, even though A and B could equally well go in the opposite order (i.e. $[1, 0, 2]$). Furthermore, if we were to add another action to the plan that must follow B but has no impact on A or C, the timestamp assigned to it would be 3, even though 1 would be acceptable. In other words, the price paid for threat-resolution using a total-order is that actions never occur in parallel and their ordering is determined early, in order to to avoid mutual interference — even when there is none. Fortunately, forward search is also able to support very rapid search tree generation and backtracking, allowing it to recover from poor choices of ordering.

In the temporal-case, where planning considers both the starts and ends of actions, not only does a total-order lead to poor-quality plans, it makes search far more difficult. To see why this is, consider the example of two durative actions, A and B, with start points $A_\vdash$ and $B_\vdash$ and ends $A_\dashv$ and $B_\dashv$. Suppose that B is longer than A, and the actions do not interfere with one another at the start, but due to their end conditions and effects, $B_\dashv$ must precede $A_\dashv$. If the planner chooses to construct the plan by adding the actions: $A_\vdash$, ..., $B_\vdash$, ..., $B_\dashv$, in that order, (where the ellipses indicate other actions in the plan, none of which affect A or B) then the temporal constraints will turn out to be unsatisfiable and require the planner to backtrack through all intermediate decisions until it reorders $B_\vdash$ before $A_\vdash$ — even though the ordering on the starts of the actions is logically uninteresting, as the two do not interact.

Some of the problems that arise because of early commitment in a temporal planning context can be preempted, as discussed in (Coles et al. 2009b), but there remain others that follow from the early commitment approach. One of the most serious problems for forward search temporal planning is in solving problems with deadlines, where deadlines can arise as the consequence of timed initial literals (creating absolute deadlines) or within the interval of one or more durative actions that provide some resource throughout their execution (creating deadlines relative to their starting points). Deadlines cause problems for forward search because the point of failure in constructing a plan against a deadline is at the deadline itself. However, the *reason* for the failure can often be a poor choice of action ordering far earlier in the plan, where the choice was made arbitrarily as an early and unnecessary commitment. In these cases, forward search is typically faced with significant backtracking as the planner attempts to explore plan permutations without effective guidance. As we will go on to demonstrate, reasoning directly with partial-orders makes our planner far more effective when planning against deadlines.

These problems motivate the exploration of an approach that exploits less commitment in the ordering of actions.

4. Modifying Forward Search to Reduce Commitment

We propose to reduce the commitment to ordering choices during forward search by adding constraints to the temporal orderings of actions only as they are required to ensure preconditions are met. The approach represents a compromise between the total-ordering commitment of standard forward search and the least commitment approach in partial-order planning, since we commit to ordering choices that ensure consistency of the plan, even though in some cases disjunctions of constraints would be sufficient. Whilst our approach is general and applicable to non-temporal and temporal planning, we restrict our attention to the temporal case, first considering a purely propositional representation, and then extending our approach to include both instantaneous and (linear) continuous numeric change.

4.1 Propositional Temporal Planning

In order to extend our forward search to support partial-order planning we extend the representation of the state described in Section 2. We add further elements to the tuple as follows:

- F^+ (F^-), where $F^+(p)$ ($F^-(p)$) gives the index of the step a by which fact p was most recently added (deleted), respectively.
- FP, where $FP(p)$ is a set of pairs $\langle i, d\rangle \in (\mathbb{N}_0 \times \{0, \epsilon\})$:
  - $\langle i, 0\rangle \in FP(p)$ records that step i is at the end of an open interval during which p is required to hold. In PDDL these arise due to invariants on actions, in which case i is the end step of an action of which p is an `over all` condition.
  - $\langle i, \epsilon\rangle \in FP(p)$ records that step i is the start of an interval (half-closed on the left for consistency with PDDL semantics) where p is required to hold. In PDDL these correspond to `at start` or `at end` conditions relevant to the step i.

The process of updating the state on application of start or end actions is shown in Algorithms 1 and 2. On applying a start action, $A_\vdash$, at step i in the plan, the following constraints are added to the partial-oder:

- for each $p \in pre(A_\vdash)$, the recorded achiever of p is demoted to come before step i (line 6). Note that this implies commitment to an achiever and ignores the possibility of exploiting *white knights* (Chapman 1987).
- for each negative effect p of $A_\vdash$, p is removed from the state, and step i is promoted to occur after any action that requires p (line 10). This is a standard promotion declobbering strategy. The alternative demotion choice (placing step i before the achiever for p) is not considered, though this does not sacrifice completeness, as search could backtrack and find a plan in which $A_\vdash$ comes earlier.
- for each positive effect p of $A_\vdash$, p is added to the state, and step i is recorded as the achiever of p (line 10). Here the order of selection of actions leads to early commitment to

Algorithm 1: Starting Actions

Data: S, a state; A, an action to start
Result: S'

$S' \leftarrow S$;
$sstep \leftarrow$ next index in P;
$estep \leftarrow$ next index in P;
$C' \leftarrow C+$ the duration constraint of A between $sstep$ and $estep$;
for $p \in pre(A_{\vdash})$ **do**
  if *defined* $S'.F^+(p)$ **then** add $t(sstep) \geq t(S'.F^+(p)) + \epsilon$ to $S'.C$;
  add $\langle sstep, \epsilon \rangle$ to $S'.FP(p)$;
for $p \in eff^-(A_{\vdash})$ **do**
  if *defined* $S'.F^+(p)$ **then** add $t(sstep) \geq t(S'.F^+(p)) + \epsilon$ to $S'.C$;
  for $\langle i, d \rangle \in S'.FP(p)$ **do** add $t(sstep) \geq t(i) + d$ to $S'.C$;
  remove p from $S'.F$;
  add $t(sstep) \geq S'.F^-[p] + \epsilon$ to $S'.C$;
  $S'.F^-(p) \leftarrow sstep$;
for $p \in eff^+(A_{\vdash})$ **do**
  $d \leftarrow S'.F^-(p)$;
  if d *is defined* $\wedge\ d \neq sstep$ **then** add $t(sstep) \geq t(d) + \epsilon$ to $S'.C$;
  add p to $S'.F$;
  add $t(sstep) \geq S'.F^+[p] + \epsilon$ to $S'.C$;
  $S'.F^+[p] \leftarrow sstep$;
for $I \in pre(A_{\leftrightarrow})$ **do**
  $a \leftarrow S'.F^+(p)$;
  if $a > 0 \wedge a \neq sstep$ **then** add $t(sstep) \geq t(a)$ to $S'.C$;
  add $\langle estep, 0 \rangle$ to $S'.FP(p)$;
add $(A, sstep, estep)$ to $S'.Q$;
return S'

possible orderings: alternative achievers are displaced by the addition of $A_{\vdash}$.

- for each invariant $p \in pre(A_{\leftrightarrow})$, if $A_{\vdash}$ did not achieve p, the recorded achiever is demoted to come before step i.

Applying an end action is similar, although without the need to consider any `over all` conditions.

Figure 1 shows an example in which the current state, S, has evolved from the application of a series of actions, represented by the numbered nodes — the numbers indicate the order in which the actions were added. The dotted boxes holding the ends of actions A and B indicate that the state records actions that remain in progress. The dotted ellipses show the points at which states are created during the forward search, each following the addition of one of the actions. Action 1 is recorded as the most recent achiever for F, while action 2 is the most recent deleter for G. When step 4 is applied, the only necessary constraint is that step 1 should precede it, indicated by the connecting arrow.

Algorithm 2: Ending Actions

Data: S, a state; A, $sstep$, $estep$, an entry from Q
Result: S'

$S' \leftarrow S$;
for $p \in pre(A_{\dashv})$ **do**
  if *defined* $S'.F^+(p)$ **then** add $t(estep) \geq t(S'.F^+(p)) + \epsilon$ to $S'.C$;
  add $\langle estep, \epsilon \rangle$ to $S'.FP(p)$;
for $p \in eff^-(A_{\dashv})$ **do**
  if *defined* $S'.F^+(p)$ **then** add $t(estep) \geq t(S'.F^+(p)) + \epsilon$ to $S'.C$;
  for $\langle i, d \rangle \in S'.FP(p)$ **do** add $t(estep) \geq t(i) + d$ to $S'.C$;
  remove p from $S'.F$;
  add $t(estep) \geq S'.F^-[p] + \epsilon$ to $S'.C$;
  $S'.F^-(p) \leftarrow estep$;
for $p \in eff^+(A_{\dashv})$ **do**
  $d \leftarrow S'.F^-(p)$;
  if d *is defined* $\wedge\ d \neq estep$ **then** add $t(estep) \geq t(d) + \epsilon$ to $S'.C$;
  add p to $S'.F$;
  add $t(estep) \geq S'.F^+[p] + \epsilon$ to $S'.C$;
  $S'.F^+[p] \leftarrow sstep$;
remove $(A, sstep, estep)$ from $S'.Q$;
return S'

4.2 Numeric Temporal Planning

Planning systems have become increasingly capable over the past decade and the extension to handle explicit time and numbers are important developments that extend the applicability of the technology. For this reason, it is essential to consider the impact of modifications to planning algorithms on their ability to continue to handle these extensions. In the propositional case, the state tuple is extended to record achievers and deleters for propositions. To extend this idea to numbers, similar annotations are recorded for numeric state variables. For each $v \in V$:

- $V^{\mathit{eff}}(v)$ records the index of the most recent step having an instantaneous effect on v.
- $V^{cts}(v)$ records a set of pairs of start and end step indices, where $(i, j) \in V^{cts}(v)$ indicates that the action that began at i and will finish at j (where step j is still in the event queue) has a continuous numeric effect upon v.
- $VP(v)$ records a set of step indices, where $i \in VP(v)$ when step i depends on the value of v. This arises in three cases:
  - step i has a precondition involving v;
  - step i has an effect whose outcome depends on the prior value of v (e.g. it increases v by some amount, or changes another variable according to some function involving v);
  - step i is the start of an action whose duration depends on v.

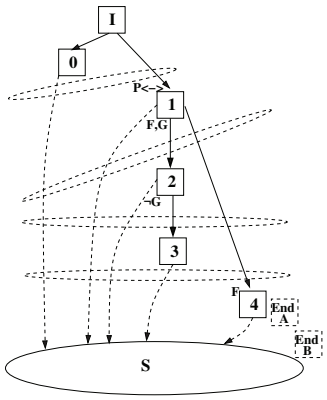


Figure 1: Determination of which facts are true in a state.

- $VI(v)$ records pairs of indices so that $(i,j) \in VI(v)$ when the action that begins at step i and will end at step j, where j is currently in the event queue, has an `over all` condition dependent on v.

Algorithms 1 and 2 can then be extended to handle the constraints implied by the management of numeric effects and preconditions. When adding a step A at index i:

1. If the effect of A depends on the value of v:
   - add $t(i) \geq t(V^{\mathit{eff}}(v)) + \epsilon$ to $S'.C$ to promote A after the most recent action to affect v;
   - $\forall (s,e) \in S.V^{cts}(v)$, add $t(s) + \epsilon \leq t(i)$ and $t(i) + \epsilon \leq t(e)$ to $S'.C$, to place the dependent effect inside currently active process effects (see further comment below).
   - add i to $S'.VP(v)$.
2. If A has an instantaneous numeric effect on v:
   - add $t(i) \geq t(V^{\mathit{eff}}(v)) + \epsilon$ to $S'.C$ to order updates on v;
   - $\forall j \in S.VP(v)$, add $t(j) + \epsilon \leq t(i)$ to $S'.C$ to avoid harmful interactions between the effect of A and actions that depend on it;
   - $\forall (s,e) \in (S.V^{cts}(v) \cup S.VI(v))$, add $t(s) + \epsilon \leq t(i)$ and $t(i) + \epsilon \leq t(e)$ to $S'.C$, placing the step inside the range of active continuous effects;
   - $S'.V^{\mathit{eff}}(v) \leftarrow i$.
   - update $S'.V^{min}(v)$, $S'.V^{max}(v)$ according to the effect.
3. If A starts an action (finishing at j) with an `over all` condition on v:
   - $\forall (s,e) \in S.V^{cts}(v)$, add $t(s) + \epsilon \leq t(i)$ and $t(i) + \epsilon \leq t(e)$ to $S'.C$, placing the step inside the range of active continuous effects;
   - if A does not have an update effect on v, add $t(i) \geq t(S.V^{\mathit{eff}}(v)) + \epsilon$ to $S'.C$ to promote the invariant past the most recent effect on v;
   - add (i,j) to $S'.VI(v)$.
4. If A starts an action (finishing at j) with a continuous effect on v:
   - if A does not also have an instantaneous update effect on v, add $t(i) \geq t(V^{\mathit{eff}}(v)) + \epsilon$ to $S'.C$ to sequence updates on v;
   - $\forall j \in S.VP(v)$, add $t(j) + \epsilon \leq t(i)$ to $S'.C$;
   - $\forall (s,e) \in VI(v)$, add $t(s) + \epsilon \leq t(i)$ and $t(i) + \epsilon \leq t(e)$ to $S'.C$, placing the step inside the range of active invariant conditions;
   - add (i,j) to $S'.V^{cts}$.
   - $S'.V^{\mathit{eff}}(v) \leftarrow i$.
5. If A ends an action that began at k, with a continuous effect on v:
   - $\forall (s,e) \in VI(v)$, add $t(s) + \epsilon \leq t(i)$ and $t(i) + \epsilon \leq t(e)$ to $S'.C$;
   - remove (k,i) from $S'.V^{cts}(v)$;
   - $\forall (s,e) \in S'.V^{cts}(v))$, add $t(s) + \epsilon \leq t(i)$ and $t(i) + \epsilon \leq t(e)$ to $S'.C$, placing the action inside the range of active continuous effects;
   - $S'.V^{\mathit{eff}}(v) \leftarrow k$.
6. If A ends an action that had an `over all` condition on v:
   - add i to $S'.VP(v)$;
   - remove (k,i) from $S'.VI(v)$.

The key outcome of these ordering constraints is that we impose a total ordering on steps changing the value of a variable v, corresponding to the order in which the steps are added to the plan. In doing so, at any point we can determine the value of v by ordering steps predicated on v but not changing it to occur after the last step changing v and before the next. A further point to observe is that we force conditions that depend on active process effects to lie within those processes (and, similarly, within the range of active invariants). This is a further commitment, since the step might turn out to be better ordered after one or more of the end points of active processes or invariants. The ordering choices in this case are left to backtracking search to identify. The consequence of all these constraint additions, including those placing steps within the durations of relevant active processes or invariants, is that our implementation commits to choices that allow us to maintain an unambiguous value for each of the metric fluents while avoiding commitments about orderings relative to actions that have no relevant interaction with the inserted step. This is a compromise between least commitment and total commitment.

4.3 Checking Action Choice Consistency

Having constructed the ordering constraints, $S'.C$, it is necessary to check that they are consistent. In the absence of continuous numeric effects, $S'.C$ can be represented as a Simple Temporal Network. It is straightforward to then check the consistency of $S'.C$ and, if valid, obtain a timestamp for each step in the plan. In the presence of linear continuous numeric effects, we use an adaptation of the temporal–numeric consistency approach of COLIN. As discussed in Section 4.2, the constraints required to manage numeric effects ensure that we still have a total ordering on the numeric effects interacting with a given $v \in V$. Similarly, the constraints on actions with preconditions on v fix

their location in the plan to occur after the effect prior to their addition, and before the next.

The construction of the LP in COLIN relies on three consequences of the total ordering of steps:

1. It can be assumed that the order of steps in the plan is the same as the order that they were added to it.
2. The values of the state variables at the start of each step can be defined in terms of the values after the previous step, according to the impact of any active continuous effects: the gradient on each v, δv is known (the combined process effects of all active continuous effects on the variable), and hence $v_i = v'_i + \delta v(t(i+1) - t(i))$.
3. By adding a dummy step now to the end of the plan, and maximizing and minimizing the values of each $v_{now} \in V_{now}$, one can obtain upper- and lower-bounds on each $v \in V$ after the steps added to the plan so far.

Having abandoned the total ordering on the steps in the plan, we must consider the consequences of losing these three properties.

1) Determining Step Order. The constraints we add to the plan ensure that any ordering of the actions that is consistent with the partial-order is a valid total-order. Therefore, we can select an arbitrary topological sort of the partial-order as a step order in which to process the plan.

2) Determining the Values of the Fluent at Each Step. Whilst building the LP, we record three values for each $v \in V$:

- v^{val}: the LP variable containing the value v'_i after the last step m to have an effect upon v;
- v^t: the timestamp variable of the last step m to have an effect upon v;
- δv: the current total gradient of the active linear continuous change upon v (as in COLIN).

When visiting step i, for each variable $v \in V$ we can determine v_i before i as:

$$v_i = v^{val} + \delta v(t(i) - v^t)$$

That is, rather than calculating the value of v based on the value after the previous step in the plan, it is calculated using the value after the last step to modify v, and the time since that step.

On execution of each step i, the values associated with each metric fluent $v \in V$ are updated as follows:

- If step i has an instantaneous effect on v, its effect is handled as in COLIN, by creating a constraint relating $v'(i)$ to $v(i)$) and by setting v^{val} to $v'(i)$, and v^t to $t(i)$.
- If step i is the start of an action with a continuous effect changing v at the rate of c per time unit, c is added to δv and $v^{val} \leftarrow v'(i)$, $v^t \leftarrow t(i)$.
- If step i is the end of an action with a continuous effect on v with rate c, c is subtracted from δv and $v^{val} \leftarrow v'(i)$, $v^t \leftarrow t(i)$.

3) Determining the Minimum and Maximum Values of $v \in V$. In COLIN, before each iteration of the action selection process, a single dummy timestamp variable t_{now} is added to the LP, along with a vector of values representing the metric fluents, V_{now}. These values are used to represent the timestamp and metric fluent values in the current state and bounds on them are determined using the LP according to constraints of the form:

$$v_{now} = v'_n + \delta v(t_{now} - t(n))$$

In the partial-order version of our planner, the value of a variable v is no longer necessarily linked to the last step of the plan, so instead we create one dummy timestamp variable $t_{now,v}$ for *each* $v \in V$. We then constrain each v_{now} as follows:

$$v_{now} = v^{val} + \delta v(t_{now,v} - v^t)$$

5. Modifying the Heuristic

Section 4 describes the way in which the state representation can be extended and correctly maintained in order to achieve a partial-order plan construction within a forward planning framework, working with time, numbers and continuous effects. We now turn our attention to heuristic search guidance in the search space of these states.

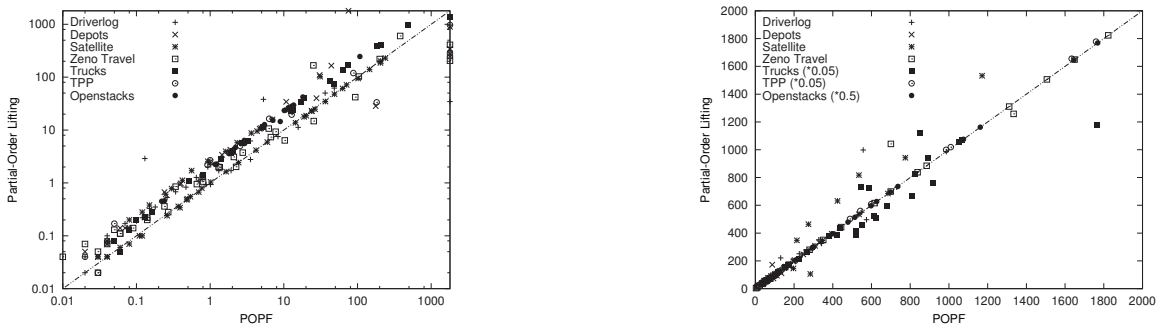
One alternative is straightforward: we can discard the additional information recorded in the state, and evaluate states using the existing temporal relaxed planning graph (TRPG) heuristic in COLIN. Doing this discards important information, however: all facts in the current state are added to fact layer $t = 0$ of the relaxed planning graph, irrespective of when they were achieved. There is therefore no pressure to choose actions in the relaxed plan that could go earlier rather than later in the partial-order. In the construction of the states we accumulate information about the earliest times at which each fact could be achieved and this can be used to improve the heuristic estimates if we use the relaxed plans to estimate makespan.

To begin with, when step i is added to the plan, we use the LP to calculate the earliest timestamp at which it could be executed, $t_{min}(i)$. In the TRPG each layer is associated with the earliest time it can represent. We observe that the earliest time a fact p is available will be $ft(p) = max\{t_{min}(F^+(p)), t_{min}(F^-(p)) + \epsilon\}$, because either it will be achieved by its last achiever as early as that action could be applied, or else it must be reachieved after its last deleter, again at its earliest application. Thus p is not added to the TRPG until a fact layer occurring at $ft(p)$. Similarly, for each numeric precondition specified over variables $vars$, we delay the layer at which it is considered satisfied to:

$$ft(vars) = \max_{v \in vars} (t_{min}(V^{\mathit{eff}}(v)) + \epsilon)$$

These modifications prevent actions from being selected earlier that their preconditions can be achieved. Furthermore, the constraints that affect the scheduling of actions also imply that any action that adds p will be scheduled after existing actions affecting p, so after $ft(p)$, and any action that deletes p must also come after actions requiring p. Thus, actions that delete p will necessarily be delayed until:

$$fd(p) = \max[ft(p), \max_{(i,d) \in FP(p)} (t_{min}(i) + d)]$$



(a) Time (b) Makespan

Figure 2: Comparing POPF to Standard Total-Order Search followed by Partial-Order Lifting

By similar reasoning, a numeric effect, ne, updating variable v, must be scheduled after the last actions affecting any of the variables appearing in ne, $vars$, and also after the last point at which v is required:

$$ft(ne) = \max[ft(vars), \max_{i \in VP(v)}(t_{min}(i) + \epsilon)]$$

In our evaluation we compare results using both the original heuristic and the modified heuristic. A side-effect of the use of a relaxed plan heuristic is the opportunity to extract *helpful actions* (Hoffmann and Nebel 2001). When using our partial-order plan structure we define helpful actions to be all actions in the relaxed plan whose preconditions are satisfied in the current state (rather than in the first layer of the graph, since the TRPG is built outwards from time $t = 0$).

6. Evaluation

We now present results comparing our partial-order forward planner with the alternative approach of lifting a partial-order from a totally ordered plan produced by COLIN. Lifting an optimal partial-order is NP-hard, but finding a good partial-order is inexpensive in general using a straightforward approach (Veloso, Pérez, and Carbonell 1990). All tests used a 3.4GHz Pentium IV machine, limited to 30 minutes and 1.5GB of memory. We tested on a wide range of temporal domains from recent planning competitions, but space limitations prevent us from showing results for all domains. We plot a representative sample of results and comment on results from other domains in the text. In order to maximize the amount of data we can plot, results for IPC 3 domains include both the simple time and time variants of the respective domain (and for satellite, also complex time). In all sets of results we plot the time taken for our approach using partial orders (POPF), against the time taken using the standard total-ordered search but then lifting a partial-order (PO-LIFT). Points above the line $y = x$ show better performance (shorter time/shorter makespan) for POPF.

Figure 2a, shows the performance of our approach *using the original heuristic*. This directly compares the costs of reasoning with partial-orders during search with and lifting a partial-order from a totally ordered plan (including the time to construct the plan). The results show that reasoning directly with partial-orders turns out to be consistently marginally more efficient. This result surprised us and we are still attempting to understand it better. Our current hypothesis is that plan permutation symmetries might be reduced through the direct reasoning with partial-orders.

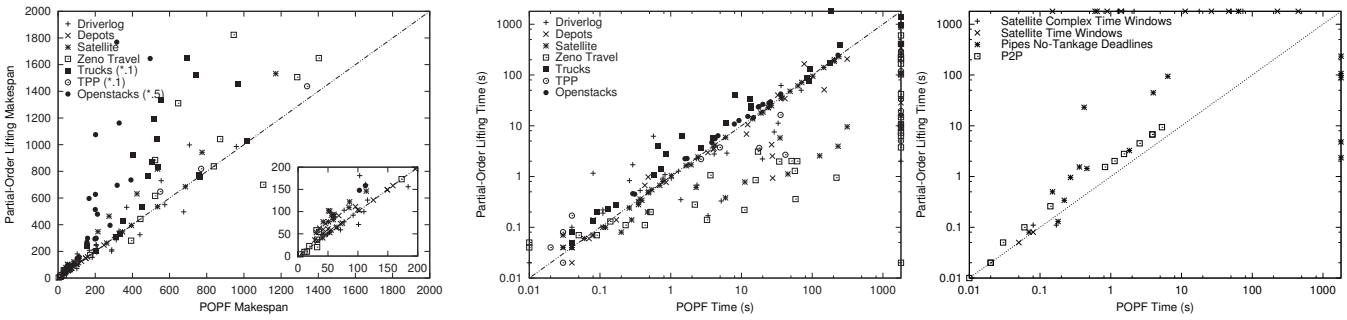
Figure 2b compares the makespans for plans produced by each approach. The partial-order planner achieves essentially identical performance to the lifting approach in most cases. These results serve to emphasise the importance of coupling the construction of a partial-order with a heuristic tuned to exploit the power this offers. With the standard heuristic the planner fails to perceive the opportunities to construct plans with shorter makespan.

In contrast, Figure 3a shows the makespan of plans generated using the POPF approach combined with the heuristic outlined in Section 5. In this case it is clear that plans of shorter makespan are produced, often by significant margins. The heuristic supports the planner in finding choices of action that make plans shorter[1].

There are some interesting anomalous points on the graph where the POPF planner produces plans of longer makespan. There are only two domains in which this occurs: Driverlog and Zeno travel. In Driverlog this happens in problems that have goals for both trucks and drivers. The shortest-makespan way to achieve a driver goal, within the relaxed planning graph, is to for a driver to drive a truck to the driver's destination, so the heuristic will prefer this decision. However, if the truck also has a goal destination, some driver must leave his goal to drive the truck to its goal location. Then, to reach the goal state, the driver must returns again on foot. This is an example in which the short-makespan heuristic ends up producing poor decisions. A similar situation arises in Zeno Travel.

Figure 3b shows that the new heuristic is more expensive to compute. The heuristic guides the search towards plans with shorter makespan rather than towards plans that contain fewer steps and this can lead to larger numbers of steps and slower plan construction. This increased computation

[1]The Wilcoxon ranked sign test confirms that the POPF planner produces shorter plans with P=95%.



(a) Makespan (inset expands lower left) (b) Time (c) Time with Deadlines

Figure 3: Results for the Planner making use of the new Heuristic

time also results in slightly poorer coverage. We demonstrated earlier, however, that if a fast solution is desired, POPF search with the original heuristic is the most efficient. Planning with partial-orders therefore offers us a choice between heuristics that favour performance or plan quality.

Our final set of results (Figure 3c) considers using our new heuristic on problems with deadlines[2], and a P2P domain (Huang, Chen, and Zhang 2009), where good solutions rely on downloading files within planned envelopes of availability. In all these problems, reasoning about the makespan of the plan head is critical in finding solutions to the problems. These domains, containing either timed initial literals (TILs) or envelopes, POPF with the new heuristic finds plans more quickly in general and provides much better coverage (in the Satellite variants, for instance, coverage is increased five-fold). In reasoning with the makespan of the plan head, POPF is able to ensure that the maximum amount of activity can occur before the deadline, and improve the opportunity to solve the problem. On the other hand, standard search often commits early to sequential decisions that mean the deadlines cannot be met, and spends a great deal of time fruitlessly searching unforeseen deadends. Here, reasoning about partial orders in the plan is critical to solving problems efficiently, encouraging parallel activity rather than only progress to the goal. Makespan data is not shown for this set of results, as the deadlines restrict the makespans of valid plans, so if the problems are solved, plans of similar makespan are produced (in the case of the P2P domain, makespans equal those known to be optimal.)

7. Conclusion

Partial-order planning is an intuitively attractive strategy, but has proved difficult to achieve efficiently. We have shown that it is possible to achieve some of the advantages of partial-order planning in a forward planning framework using an expressive planning language. The complexity of partial-order reasoning can be managed by finding a compromise between least-commitment and total commitment. Doing so can yield significant advantages in plan quality, using a more informed heuristic for guiding the search within this partial-order structure.

There remain several opportunities to extend the ideas we have described. We are interested in recognising automatically the situations in which the extended heuristic will offer benefit. The integration of linear program into the combinatorial decision-making in planning is a fruitful route to extension of capabilities of planners and there remain many further directions in which to exploit this line of research.

[2]We use all IPC domains with deadlines but omit Airport, as a bug in the ADL-to-STRIPS compilation means the TILs do not actually interact with the problem.

References

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 29:333–377.

Coles, A. I.; Fox, M.; Long, D.; and Smith, A. J. 2008. Planning with Problems Requiring Temporal Coordination. In *Proc. AAAI*.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2009a. Temporal Planning in Domains with Linear Processes. In *Proc. IJCAI*.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2009b. Extending the Use of Inference in Temporal Planning as Forwards Search. In *Proc. ICAPS*.

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

Do, M. B., and Kambhampati, S. 2001. Sapa: a Domain-Independent Heuristic Metric Temporal Planner. In *Proc. European Conference on Planning (ECP)*, 82–91.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

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

Penberthy, J., and Weld, D. 1992. UCPOP: a sound, complete, partial-order planner for ADL. In *Proc. Int. Conf. On Principles of Knowledge Representation and Reasoning*, 103–114.

Penberthy, J., and Weld, D. 1994. Temporal planning with continuous change. In *Proc. AAAI*.

Veloso, M.; Pérez, M.; and Carbonell, J. 1990. Nonlinear planning with parallel resource allocation. In *Proc. of the DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, 207–212.

Younes, H. L. S., and Simmons, R. G. 2003. VHPOP: Versatile Heuristic Partial Order Planner. *JAIR* 20:405–430.