Extending the Use of Inference in Temporal Planning as Forwards Search

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

Abstract

PDDL2.1 supports modelling of complex temporal planning domains in which solutions must exploit concurrency. Few existing temporal planners can solve problems that require concurrency and those that do typically pay a performance price to deploy reasoning machinery that is not always required. In this paper we show how to improve the performance of forward-search planners that attempt to solve the full temporal planning problem, both by narrowing the use of the concurrency machinery to situations that demand it and also by improving the power of inference to prune redundant branches of the search space for common patterns of interaction in temporal domains that do require concurrency. Results illustrate the effectiveness of our ideas in improving the efficiency of a temporal planner that can solve problems with required concurrency, both in domains that exploit this ability and in those that do not.

1. Introduction

It has been observed (Cushing et al. 2007; Coles et al. 2008) that the most interesting temporal planning domains require actions to be performed concurrently to achieve a goal. The temporal domains used in the competitions are *simple* in the sense that no concurrency is required and sequenced solutions are adequate. When *required concurrency* is present, finding a solution requires interleaving of the starts and ends of actions, often in an intricate way.

PDDL2.1 (Fox and Long 2003) introduced an extension of the classical STRIPS action model to include durative actions. In contrast to the durative action model in TGP (Smith and Weld 1999), PDDL2.1 actions can have effects at both the start and end of their execution, along with conditions that must hold at the start, end, or during the execution of the action. Domains requiring concurrency can be expressed by, for example, defining durative actions that make resources available at their starts and remove them at their ends. If another action relies on the availability of a resource that is provided in this way then its execution must run in parallel with that of the action providing the resource.

When there is no required concurrency the simplest approach to temporal planning is to treat durative actions as units with delete effects at the beginning of their execution and add effects at the end. However, this *compressing* strategy fails in domains with required concurrency (Cushing et al. 2007). A more capable approach is to split durative actions into start and end actions and to perform search over invariant-respecting sequences of these actions, checking temporal constraints (using a Simple Temporal Network (STN) solver, for example) to confirm temporal consistency of the choices considered. This is how the CRIKEY planners (Coles et al. 2008) overcome the limitations of the compressing approach.

Whilst planning with start and end actions is a solution to required concurrency it increases the cost of planning in cases where concurrency is not required. In many domains there is a mixture of behaviour that must be parallelised and behaviour that can be sequenced. If the re-representation of the domain in terms of start and end actions is performed regardless of whether it is necessary, there can be significant costs in terms of performance. Even if the scheduler is efficient, the plan search space is exponentially larger than in the compressed case, as each durative action in the plan requires the construction of two action end points. Performance can be greatly improved by recognising actions that can be compressed, allowing a hybrid representation comprising some start-end separation and some compression.

In this paper, we present techniques that significantly improve the efficiency of temporal planning in domains with and without required concurrency. Firstly, we show how an automatic analysis can be used to identify a large class of common durative action structures that do not require concurrency and can therefore be handled more efficiently. We call these *compression-safe* actions. Secondly, we introduce methods for handling a range of interactions that arise in domains with required concurrency, supporting improved efficiency in planning in such domains. We present two classes of techniques in this case: one based upon the detection of 'one-shot' actions, and one based on interactions between invariants and end delete effects. Empirical data are presented demonstrating the power of our inference techniques in domains with and without required concurrency.

2. Background

A ground action A in PDDL2.1 (that is, an action schema whose parameters have been specified) is defined by a tuple $\langle pre_{\vdash}, \mathit{eff}_{\vdash}, pre_{\leftrightarrow}, pre_{\dashv}, \mathit{eff}_{\dashv}, mindur, maxdur \rangle$ where:

- $pre_{\vdash}, \mathit{eff}_{\vdash}$ denote the conditions that must hold at the start of the action, and the effects which occur immediately after A is applied. $\mathit{eff}_{\vdash}$ comprises propositional delete effects and add effects and numeric effects, denoted $\mathit{eff}^{-}_{\vdash}$ and $\mathit{eff}^{+}_{\vdash}$, and $\mathit{eff}^{n}_{\vdash}$ respectively.
- $pre_{\dashv}, \mathit{eff}_{\dashv}$ denote the conditions that must hold at the end of the action, and the effects which then occur.
- $pre_{\leftrightarrow}$ denote the invariant (`over all`) conditions that must hold throughout the action.
- $mindur, maxdur$ are functions which, given the values of the numeric variables when the action is started, return lower- and upper-bounds on the duration of the action.

Note that we are not treating conditional effects, disjunctive or negative preconditions (which can be compiled out) or complex duration constraints (such as disjunctive constraints or conditions that refer to the state at the end of execution).

A simple action compression strategy for planning with these actions is to construct, from each durative action A, a notionally atemporal action A' whose preconditions and effects correspond to the weakest preconditions and strongest effects of A. In the propositional case, these are:

$$\begin{aligned} pre(A') &= pre_{\vdash} \cup ((pre_{\leftrightarrow} \cup pre_{\dashv}) \setminus \mathit{eff}^{+}_{\vdash}) \\ \mathit{eff}^{+}(A') &= \mathit{eff}^{+}_{\vdash} \setminus \mathit{eff}^{-}_{\dashv} \cup \mathit{eff}^{+}_{\dashv} \\ \mathit{eff}^{-}(A') &= \mathit{eff}^{-}_{\vdash} \setminus \mathit{eff}^{+}_{\vdash} \cup \mathit{eff}^{-}_{\dashv} \setminus \mathit{eff}^{+}_{\dashv} \end{aligned}$$

A non-temporal planner can plan with these compressed actions and a scheduler can be used to label the actions with timestamps and durations *post hoc*. In scheduling, the effects are assumed to appear at the end of the action (after its appropriate duration) and its preconditions are assumed to be required before it begins and over its entire duration. This is the semantics of durative actions assumed in TGP (Smith and Weld 1999), but it is also a transformation used in planners such as LPG (Gerevini, Saetti, and Serina 2006) and MIPS (Edelkamp 2003). If an action A has effects at both the start and end, then A' is not an accurate reflection of its behaviour. At best, it is inefficient: initial add effects appear later than necessary and initial preconditions must be sustained longer than necessary. At worst, it is incomplete: if a fact p is added at the start and deleted at the end of A it will not appear in the effects of A' and, if p is required to find a solution, then no solution will exist in the compiled domain.

An alternative to action compression, introduced in LPGP (Long and Fox 2003), is to split actions A into two components: a start action $A_{\vdash}$ and an end action $A_{\dashv}$ with preconditions and effects $\langle pre_{\vdash}, \mathit{eff}_{\vdash}\rangle$ and $\langle pre_{\dashv}, \mathit{eff}_{\dashv}\rangle$ respectively. Invariants can also be represented by an action $A_{\leftrightarrow}$. Unlike the action compression case, it is not sufficient to use an unmodified planner with these actions and address the temporal aspects of the problem afterwards. Three additional constraints must be considered during planning:

1. if $A_{\vdash}$ has been applied, $A_{\dashv}$ must be applied at some point in the future;
2. between $A_{\vdash}$ and $A_{\dashv}$, the invariant of A must hold;
3. the timestamps of $A_{\vdash}$ and $A_{\dashv}$ must satisfy $mindur(A)$ and $maxdur(A)$.

The CRIKEY planners (Coles et al. 2008) use this split-action representation and perform forward-chaining search in a manner similar to that of FF (Hoffmann and Nebel 2001), with three modifications to address the constraints listed above. First, to restrict action applicability, each state maintains a *start event queue*: a list of actions that have started but not yet finished. Action ends are applicable only if there is a corresponding entry in the queue and no action can be applied whose effects invalidate an invariant of one of the actions in the queue (except the end of the only action with that invariant). Second, to handle the temporal problem constraints, a Simple Temporal Network (STN) is constructed and used at each state to check the temporal validity of action sequences. Third, the definition of a goal state is changed to reflect the fact that all actions must have finished executing: the start event queue must be empty.

Planning with split actions allows a planner to solve problems with required concurrency, which is a problem that is potentially much harder than classical planning (Rintanen 2007). In practice, the search space is twice as deep (each durative action becomes two end points), so there is a very real cost to planning with this representation. Planning with start- and end-actions also brings with it challenges in terms of effective search guidance, with existing planners relying on relaxation-heuristic approaches (Coles et al. 2008). We consider two questions:

1. How can *compression-safe* actions be recognised and exploited to narrow the gap between planning with action compression and planning with split actions?
2. How can orderings be inferred between the ends of actions which have started, but not yet finished, reducing search when planning with split actions?

2.1 Terminology: Variables in Temporal Planning

Considering temporal forward-chaining planning as an incremental variable–value assignment problem, we define three categories of variables:

- action variables: act_n denoting the action start or end choice at step n in the plan;
- timestamp variables: t_n, the time at which act_n is applied;
- pairing variables: $pair_n$, the index of the other end of act_n in the plan, or *undef* if this is not yet in the plan.

To represent the logical and numeric constraints on plans we use variables to record states:

- fact variables: P_n for each fact P, denoting whether P holds in the state following execution of act_{n-1} (or the initial state when $n = 0$);
- numeric variables: V_n for each numeric state variable V, denoting the value of V in the state following act_{n-1} (the initial state when $n = 0$).

The values of P_n, V_n determine which actions are in the domain of act_n, according to their preconditions. We abuse notation by using P_n and V_n to refer to the valuations over all facts and numeric variables in state n.

The temporal constraints on t_i are as follows:

$$\forall i = 0...n \cdot t_i + \epsilon <= t_{i+1} \quad (1)$$

$$(pair_j = i) \wedge act_i \text{ is a start action} \Rightarrow (t_j - t_i) \in [mindur(act_i), maxdur(act_i)] \quad (2)$$

Constraint 1 ensures that the actions are properly sequenced and separated (note that the use of ϵ could be made dependent on whether the actions actually interfere). Constraint 2 ensures that actions satisfy their duration requirements.

A sequence of actions $act_0..act_n$ achieves a goal formula *goal*, specified over state facts and numeric variables, if the usual precondition and effect requirements are satisfied and:

$$(P_n, V_n) \models goal \text{ and } \forall i = 0...n \cdot pair_i \neq \text{undef} \quad (3)$$

To support the construction of a plan by forward-search, each state in CRIKEY3 (Coles et al. 2008) contains:

- The action variables $act_0..act_n$;
- The pairing variables $pair_0..pair_n$;
- The most recent fact and numeric variables, P_n and V_n;
- A start event queue containing actions whose starts have appeared in the plan, but whose ends have not.

Search is performed by actions to the end of the sequence, updating the state variables accordingly, and using a STN to check if the timestamp variables $t_0..t_{n+1}$ can be consistently assigned values.

3. Recognising Compression-Safe Actions

We begin by identifying a class of durative actions that can be safely handled by a form of compression. The technique we propose here is not quite the same as pure compression, since we allow actions to have initial preconditions and effects. However, we show that for the class of actions we call *compression-safe*, there is an efficient way to both recognise and exploit them. As a typical example, consider the action 'drive-truck' from the Driverlog domain. This action performs what can be thought of as a temporally extended version of a classical STRIPS action, with delete effects taking immediate effect, add effects delayed until the end and prevail conditions invariant across the action. Thus, at the start, the truck leaves A and later, at the end, it arrives at B, while throughout the driver has to remain in the truck.

When the start of an action is added to a plan, its end must be applied at some future point, according to the duration constraints of the action. In general, in forward search, the end point of an action that has been started will have to be considered as a choice for application at each incremental extension of the plan until it is actually applied. The key to identifying compression-safe actions lies in identifying a class of actions for which this evaluation can be eliminated. This arises when the end of the action satisfies constraints that imply that its application can always be determined by the structure of the developing plan. This situation arises when the end of an action can have only positive impact on the plan structure, which motivates the following definition:

Definition 31 — Compression-safe Actions
A durative action, A, is *compression-safe* if it satisfies:

1. $pre(A_{\dashv}) \subseteq pre(A_{\leftrightarrow})$;
2. $\mathit{eff}^{-}(A_{\dashv}) = \emptyset$;
3. $\mathit{eff}^{n}(A_{\dashv}) = \emptyset$;
4. $\mathit{eff}^{n}(A_{\vdash})$ does not depend on the duration of A.

It is worth recalling that we are assuming actions have simple positive preconditions (negative preconditions can be compiled out, but this process will affect which actions are compression-safe). The definition ensures that the ends of compression-safe actions can always be applied in a state generated in the incremental extension of a plan, since this extension is required to maintain the invariants of open actions, which implies the precondition of the ends of compression-safe actions.

Compression-safe actions form a common idiom in temporal planning domains. It is typical for durative actions to consume or lock resources they require at the start of execution, releasing them by positive effects at the end. Note that this idiom extends to numeric effects, where production of resources, which can be seen as positive, is enacted at the end of a durative action. We exclude this case in our definition for reasons that we discuss below in section 3.3

The identification of compression-safe actions allows us to apply a simple technique to eliminate the need to evaluate the choice of applying the ends of such actions explicitly.

3.1 Compression-Safe Actions in Planning

Recall that in forward search planning in CRIKEY3, on the addition of a start action $C_{\vdash}$ to the plan an entry for C is added in the start event queue. The state is also updated to reflect the effects of execution of $C_{\vdash}$. At subsequent steps in the incremental extension of the plan, $C_{\dashv}$ will be considered as a choice for execution whenever its preconditions are satisfied. Until it is applied, the invariants of C will be maintained by rejecting any action choices that violate them. If C is compression-safe, $pre(C_{\dashv}) \subseteq pre(A_{\leftrightarrow})$, so $C_{\dashv}$ will always be applicable until it is applied.

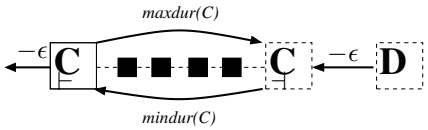
To exploit compression-safety, we modify the behaviour of the planner as follows: when the start of a compression-safe action, $C_{\vdash}$, is applied, after creating an entry for it in the start event queue (denoted e), we immediately consider its corresponding end $C_{\dashv}$, and update the state as follows:

- for each effect $p \in \mathit{eff}^{+}(C_{\dashv})$, p is added to the state fact set P, annotated with a reference to e to indicate that $C_{\dashv}$ must be applied to obtain p.
- for each invariant $inv \in pre(C_{\leftrightarrow})$, the fact that e requires inv as an invariant is annotated with the fact that the invariant can be lifted by applying $C_{\dashv}$.

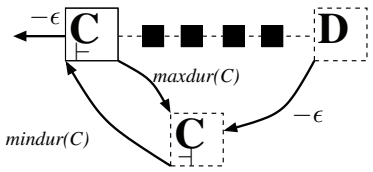
As planning continues we ignore $C_{\dashv}$ as a possible choice, but insert it according to the following rules. If a start or end action, D, is applied then:

- if the negative effects of D conflict only with start event queue invariants corresponding to compression-safe actions, then the ends of the conflicting compression-safe actions are inserted into the plan and D is applied after them.
- if D requires a precondition p annotated in the current state with a dependency on one or more event queue entries, then D is considered to be applicable, supported by a choice of one of the dependencies (typically there is only one) being selected to end prior to application of D.

In this way, the choice of where to insert the ends of compression-safe actions into the plan is avoided: each is



(a) Total Ordering with $C_\vdash$ Inserted



(b) Minimal Sufficient Ordering of $C_\vdash$

Figure 1: Ordering $C_\dashv$ in the Plan

inserted as it is needed, when the consequences of its completion is required. The consequent reduction in the number of choice points reduces the search space when planning with start- and end-actions: only one choice point is introduced for each durative action, rather than two.

In the heuristic evaluation of a state that includes annotated propositions, we simply ignore the annotations (which represents a further relaxation).

3.2 Efficient Exploitation of Compression-Safety

By recognising compression-safe actions, we have observed that the ends of such actions can be inserted as needed into the plan during forward search. There is, however, an important potential source of inefficiency arising as a consequence of the total ordering between states visited along the trajectory constructed by forward search. When $C_\dashv$, the end of a compression-safe action, is inserted into the plan, the ordering constraints identified in section 2.1imply that any action that has been added to the plan prior to $C_\dashv$ must be *scheduled* before it, and any action that is inserted after $C_\dashv$ must be *scheduled* after it. The former of these has an important consequence: the makespan of the actions applied between $C_\vdash$ and $C_\dashv$ must not exceed its duration. This situation is shown in Figure 1a. If the application of $C_\dashv$ is only ever considered when it is required to support a precondition or to avoid the invariant associated with C, then insertion of $C_\dashv$ can be delayed too late, making the STN unsolvable.

Fortunately, we can easily resolve this problem: since the maintenance of the invariant of C is sufficient to guarantee the executability of $C_\dashv$ and $C_\dashv$ has only positive effects, we can insert $C_\dashv$ *anywhere* between $C_\vdash$ and the current end of the plan, subject only to the temporal constraints on the duration of C (Figure 1b). Thus, when $C_\dashv$ is required to support execution of D for either of the reasons identified above, $C_\dashv$ is inserted into the plan with a temporal constraint placing it before D and after $C_\vdash$, according to the duration constraints on C. Similarly, to ensure that the effects of $C_\dashv$ are not exploited after C must have finished, or after another action could have deleted them, we add a further ordering constraints, such that for any action F in the plan between $C_\vdash$ and $C_\dashv$: $\mathit{eff}^-(F) \cap \mathit{eff}^+(C_\dashv) \neq \emptyset \Rightarrow t(F) < t(C_\dashv)$. Should the addition of one of these constraints lead to an inconsistent STN, during addition of an action F to the plan, then $C_\dashv$ is added to the plan prior to F and the temporal constraints adjusted accordingly.

3.3 Safe Exploitation of Compression-Safety

A natural concern is whether the method just described preserves soundness and does not impact on the completeness of the search performed by the planner. We claim the following result:

Theorem 1 *In a forward-search planner, exploitation of compression-safety as described in sections 3.1and 3.2is sound and does not compromise completeness.*

Proof: Soundness is confirmed by three observations: firstly, $C_\dashv$ is only applied in a state that precedes the first point at which the invariant of C is violated and C cannot itself violate the invariant of any other action. Since the preconditions of $C_\dashv$ are implied by the invariant of C, $C_\dashv$ must be applicable in such a state. Secondly, the temporal constraints on C are added to the STN and must be solvable in order to complete the plan. Finally, if $C_\dashv$ is added to support p we can confirm that p is protected from $C_\dashv$ until the point where it is required because all actions deleting p are constrained to precede $C_\dashv$. Therefore, any plan produced using the described techniques will be sound.

Note that we do not claim that the planner is necessarily complete, but rather that the techniques we describe above do not compromise completeness. To show this, we need only demonstrate that the only branches removed from the search space are branches that contain no solutions. The only branches that can be affected are those that begin with the execution of $C_\dashv$ after C has started, since it is only these branches that are no longer considered explicitly. Consider why application of $C_\dashv$ might lead to a solution: there are only three possibilities. Either some positive effect of $C_\dashv$ supports a subsequent action, or else an invariant of C must be removed to allow application of an action, or, finally, some action D must be applied such that $t(D) - t(C_\vdash) > maxdur(C)$. In the first two cases the corresponding branch remains open to search since the techniques described above allow $C_\dashv$ to be inserted into the plan to support either of these cases.

The last case is the most subtle. There are two possibilities: either D deletes an effect of $C_\dashv$, in which case the constraint described at the end of section 3.2leads to a contradiction and, as explained, $C_\dashv$ is inserted before D. If D does not delete an effect of $C_\dashv$ then it is added without constraint and the effects of $C_\dashv$ remain part of the state. There are now two further possibilities: either one of the earlier cases occurs and $C_\dashv$ is added to the plan, or else the plan is completed and $C_\dashv$ is added at the conclusion of the planning process. In the former case, the temporal constraints will place $C_\dashv$ *prior* to D, but the effects of $C_\dashv$ will remain active, since D did not delete them so it will remain valid to use them, so that this creates a branch equivalent to one in which $C_\dashv$ was added to the plan before D. In the latter case, the temporal constraints will simply insert $C_\dashv$ at the appropriate point in the plan, completing the plan.

All the branches in which application of $C_\dashv$ is useful have been shown to remain available (if they are not excluded

from the search space by other features of the planner), so the techniques described above do not compromise completeness. ♠

A second question is whether the definition of compression-safety is both necessary and sufficient to support the exploitation we have described. The fact that it is sufficient is demonstrated by the proof of completeness above. However, our definition of compression-safety is stronger than is required to be able to safely avoid considering ending a compression-safe action as a separate choice. In fact, it is sufficient that actions have only end effects that cannot interfere with the invariants or preconditions of any other actions in the plan between the start and end of the compression-safe action. To identify actions that satisfy such a condition in general is as hard as planning, since it is necessary to determine which actions might be applied during the interval. Thus, compression-safety is defined to compromise between efficient identification of candidate actions and the subsequent exploitation of those candidates.

Perhaps the most interesting extension of the current definition would be to include positive numeric effects: it seems reasonable to suppose that resource-producing end effects could be handled in the same way as positive logical effects. However, there is a complication: where positive logical effects can be attributed to just one achieving action when there is a choice, resource production might be split between some subset of producers. For example, if we have three producers, producing 1, 2 and 3 units of a resource respectively, then a precondition requiring 3 units of the resource could be supported by the first and second, the third alone or all three. The selection of supporters appears to grow exponentially as the number of producers grows. Nevertheless, the possibility to extend compression-safety to include some class of numeric effects is attractive and we are continuing to explore the ways in which this might be achieved.

4. One-Shot Inference

Although the identification and exploitation of compression-safe actions offers significant savings when actions do not participate in interactions that require concurrency, the whole purpose of splitting actions is to allow us to solve problems where concurrency is required. Therefore, we now consider some common structures in domains that have required concurrency in order to identify ways to improve the inference process during plan construction and eliminate redundant choices from the search space.

We first consider the 'one-shot' constraint family: an action A is one-shot if it has a precondition p, true in the initial state, deleted by A, and added by no other action. Once A has been applied, it removes its own precondition which cannot be restored. One-shot actions can be collated into 'one-shot action sets' if they are all one-shot actions over the same resource, p. Only one member of such a set can ever be applied, since once one of them has deleted P there is no way to restore it.

We now extend this definition to the temporal case. Any of the following situations allow us to recognise a durative action A as one-shot:

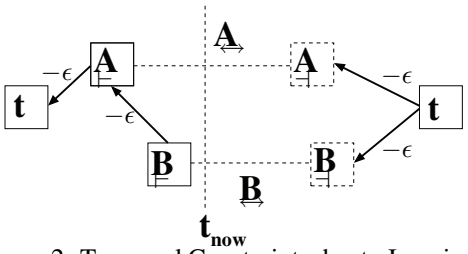


Figure 2: Temporal Constraints due to Invariants

1. $A_{\vdash}$ has a precondition p, deletes p, and p is not added by any action (including $A_{\dashv}$).
2. Either $A_{\leftrightarrow}$ or $A_{\dashv}$ has a precondition p, $A_{\dashv}$ deletes p, and p is not added by any action.
3. $A_{\vdash}$ has a precondition p, $A_{\dashv}$ deletes p, p is not added by any action, and A is mutex with itself.

Note that without the mutual-exclusion restriction in the last case, one could form the action sequence $A_{\vdash}, A_{\vdash}, A_{\dashv}, A_{\dashv}$ in a plan. To form one-shot action sets this mutex constraint needs to be extended: if an action, A, is one-shot under category 3, the other actions within the set must be pairwise mutex with A.

Using the terms defined in Section 2,a one-shot action set AS (containing non-temporal actions, or the starts of temporal actions) adds the constraint:

$$(act_i \in AS) \Rightarrow \bigcup_{n=(i+1)..\infty} act_n \notin AS$$

We can extend the definition of one-shot actions to include facts as follows:

Definition 41 — One-Shot Fact
A fact p is one-shot iff either:

- $p \in I$, no action adds p, and $p \in pre(A) \Rightarrow p \in del(A)$;
- all the actions adding p form a one-shot action group, G, no actions outside G add p, and $p \in pre(A) \Rightarrow p \in del(A)$.

This definition can be used to extend the definitions of one-shot actions and facts inductively: a set of actions that depend on and delete the same one-shot fact are also one-shot actions. Effects of these actions are then candidate one-shot facts and so on.

5. Orderings on Open Action Ends

As discussed in Section 2,to ensure legal solution plans are found as CRIKEY3 searches with split actions, $A_{\vdash}$ and $A_{\dashv}$, three conditions must hold: every action, A, started must subsequently be ended, its invariants must hold during its execution and the timestamps assigned to $A_{\vdash}$ and $A_{\dashv}$ are constrained by the duration constraints on A.

In the STN maintained by CRIKEY3, nodes are added to represent the ends of actions that are started, with constraints to ensure that they follow all actions in the plan so far. This process allows some sources of conflict to be detected early. For instance, in Figure 2, suppose A and B have both started, and by time t_{now} an action C has started and finished. If $mindur(C)$ exceeds $maxdur(A)$ or $maxdur(B)$, a negative cycle would be introduced into the STN because the fact

that $A_\dashv$ and $B_\dashv$ have yet to complete means the entire duration of C is (inconsistently) included within the execution of A and B.

Although this temporal reasoning is useful, it can be strengthened by observing the logical constraints imposed by the actions and inferring from them temporal constraints. For example, if we can show that the action that started at i must finish before that which started at j, we can add the constraint $t(i) + \epsilon \leq t(j)$ to the STN, strengthening the inference that is then possible in the STN. We now present two techniques that allow us to strengthen the STN with further inferences in similar ways.

5.1 Invariant–End Delete Conflict Ordering

Figure 2 shows the state where two actions have started in the order $A_\vdash, B_\vdash$. This implies that their end points will have to be added to the plan and that until they are the invariants of the actions hold. However, the ends of actions can have effects that force orderings between the ends of open actions because of interactions between the final delete effects and invariants of open actions as follows.

Definition 51 — Invariant–End Delete Ordering If a start-action $A_\vdash$ is applied in a state with open event list E, then its corresponding end, $A_\dashv$, must follow the ends of the events in the set: $\{e \in E \mid e.op_\leftrightarrow \cap \mathit{eff}^-(A_\dashv) \neq \emptyset\}$

In Figure 2, therefore, if $\mathit{eff}^-(B_\dashv) \cap A_\leftrightarrow \neq \emptyset$ then $B_\dashv$ must follow $A_\dashv$. This constraint can be recognised as a partial-order planning (POP) strategy to protect a causal link. The alternatives that are usually considered in a POP can be ignored because of the ordering constraints implied by the structure of the plan built by forward search.

5.2 End Effect–End Condition Ordering

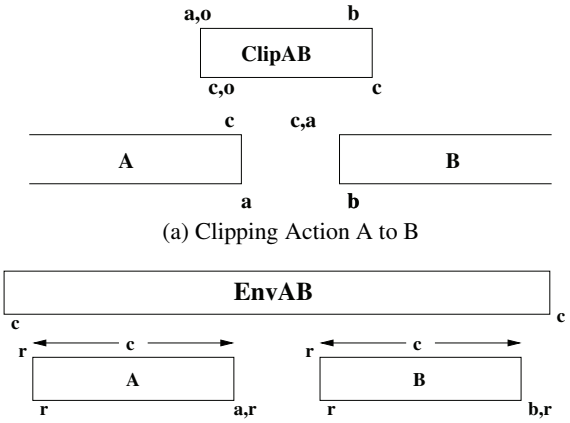
Ordering implied by interactions between end effects and invariants is forced, but when two action ends interact so that $B_\dashv$ adds a precondition p of $A_\dashv$, the ordering is not forced: some other action that adds p could be added prior to $A_\dashv$, and $B_\dashv$ then follow $A_\dashv$. Similarly, if $B_\dashv$ deletes p, some other action could be added to reestablish p prior to $A_\dashv$. However, in the case of one-shot actions and one-shot facts, end orderings can be established:

- if p is a one-shot fact and $B_\dashv$ adds p, then application of B renders all other actions adding p inapplicable, so $B_\dashv$ must precede $A_\dashv$;
- if B is a one-shot action where $B_\vdash$ adds p and $B_\dashv$ deletes it then $A_\dashv$ must precede $B_\dashv$
- if no action adds p and $B_\dashv$ deletes it, then $A_\dashv$ must precede $B_\dashv$.

5.3 Required Concurrency in Temporal Domains

We now consider two patterns that are common structures in temporal domains with required concurrency: clips and envelopes (Fox, Long, and Halsey 2004).

Analysis of Clips A clip, as illustrated in Figure 3a is an action enforcing a maximum separation between two sequential actions. For example, if a unit must be inspected within 10 minutes of closing it down, then the end of A would represent closing the unit down and B would be the inspection, clipped to the end of A by *clipAB* which would have a duration of 10 minutes. In general, *clipAB* enforces a maximum separation between the end of A and the start of B. The clip is constructed so that there is only one legal ordering for $clipAB_\vdash$, $A_\dashv$, $B_\vdash$ and $clipAB_\dashv$.



(a) Clipping Action A to B

(b) An Envelope containing A and B

Figure 3: Clipping and Envelope Actions

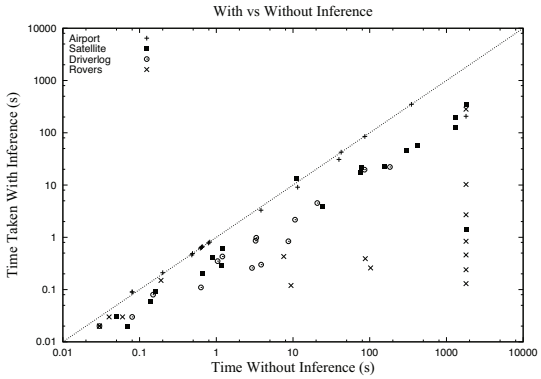
During forward search, when $A_\vdash$ and $clipAB_\vdash$ have been applied, a feasible schedule for this part of the plan is to start $clipAB_\vdash$ a small amount ϵ after $A_\vdash$. There will then appear to be a period of as much as $maxdur(A) - \epsilon$ time left until the end of A. If A encapsulates the availability of a resource, search can consider many possible plans exploiting this resource, deferring addition of $A_\dashv$ and $clipAB_\dashv$. However, given that $A_\dashv$ must precede $clipAB_\dashv$, the maximum amount of time left until the end of A is actually $maxdur(clipAB) - \epsilon$. Typically, this is much shorter, with clips often being used to enforce a short separation between contiguous chunks of activity.

Applying our inference techniques to this situation we observe that *clipAB* is one-shot and hence, $t(A_\dashv) < t(clipAB_\dashv)$ (Section 5.2 Therefore, the scope for branching and deferring the end actions is reduced and, in the case of a tight clip, search quickly determines that the only viable action sequence, once the clip has begun, is to interleave the end of A, B and *clipAB* as illustrated.

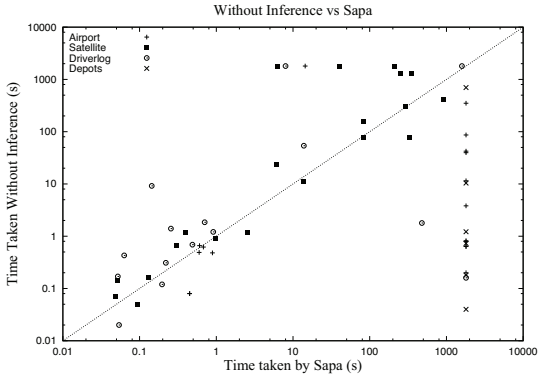
Analysis of Envelopes Envelopes, as illustrated in Figure 3b, can serve two purposes: they can force some action sequence to occur within a given duration, or they can be used to model that a resource is only available for a fixed window. As with clips, in forward search, inefficiencies arise when an envelope and its content actions have started but not yet finished. In Figure 3b, supposing the envelope *envAB* has started, A has started and finished and $B_\vdash$ has been applied. The open ends at this point are then $envAB_\dashv$ and $B_\dashv$. If $mindur(A)+mindur(B) > maxdur(envAB)$ then the sequence cannot lead to a solution plan but, while the open ends are deferred, search can branch over many possible action choices.

Applying our inference techniques gives:

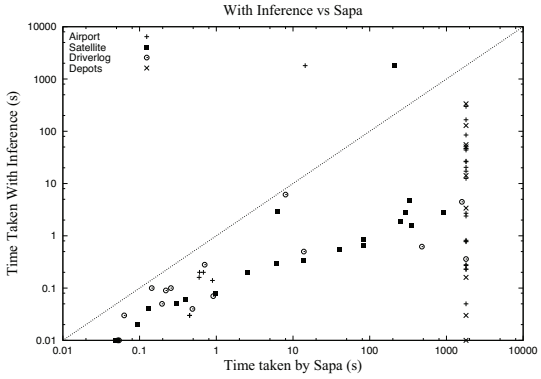
- $envAB_\dashv$ deletes an invariant of B, and hence $t(B_\dashv) <$



(a) Inference in IPC3/4 Domains



(b) Comparison to Sapa (No Inference)



(c) Comparison to Sapa (With Inference)

Figure 4: Results for CRIKEY3 with and without inference: the power of compression-safety.

$t(envAB_{\dashv})$ (Definition 51);

- $mindur(B) > maxdur(envAB) - mindur(A)$, and hence the STN will enforce $t(envAB_{\dashv}) < t(B_{\dashv})$.

Together these imply a contradiction, so at the point of adding $B_{\vdash}$ to the plan, the inconsistency can be inferred and further pointless search avoided.

6. Evaluation

We evaluate our inference techniques by considering the performance of CRIKEY3 in a range of benchmark domains. Measurements use a 3.4GHz Pentium IV machine with a limit of 1GB of memory and 30 minutes on each test.

First, we explore the performance of the compression-safe action inference and exploitation. The purpose of this inference is to attempt to mitigate the cost of supporting richer temporal reasoning needed to solve problems with required concurrency when facing domains in which this power is not actually needed. We consider the time taken to solve problems from 3rd and 4th International Planning Competitions, using the Simple Time formulations from the former and temporal (STRIPS) formulations from the latter. IPC benchmark domains contain only simple temporal interactions, making them an ideal test set to demonstrate the power of compression-safety. The results of this are shown in Figure 4. In order to ensure that the results are distinguishable, we have only plotted the results for a representative sample of domains. The remaining domains (Pipesworld, Rovers and Zeno Travel) were also tested providing a similar picture. In all benchmark domains tested we observe that the performance is improved by the inference techniques. We confirmed that the compression-safety technique is the one responsible for these improvements by disabling the other techniques. With the exception of the Rovers domain, which we discuss later, disabling all of the inferences other than compression-safety generates performance indistinguishable from that in Figure 4. These domains are temporally simple, with no clips or envelopes, and the actions are almost all compression-safe, so these results show that the exploitation of compression-safe actions in domains of this type leads to the elimination of the overhead involved in supporting more complex temporal reasoning.

Figures 4b and 4c show a comparison of CRIKEY3 with Sapa (Do and Kambhampati 2001), with and without the inference enabled, on the same benchmark domains. We select Sapa for comparison as it is also a forward search temporal planner, although it explores a reduced search space that prevents it from solving problems with required concurrency. Sapa and CRIKEY3 without inference have very similar performance, with each planner excelling in a subset of the problems. Figure 4c shows an important result: CRIKEY3 with inference (in particular, exploiting compression-safe actions) solves all but two problems faster than Sapa and solves many more problems than Sapa does. Exploiting compression-safety allows CRIKEY3 to overcome the penalty associated with using the full temporal flexibility in domains where it is unnecessary. This is possible because inference of the application of the ends of compression-safe actions eliminates reliance on search to find an ordering between all start and end actions: when appropriate, end actions are simply inserted into the plan where needed.

6.1 Domains with Further Temporal Interaction

To illustrate the power of our other techniques we must consider problems with required concurrency. There is one domain taken from IPC3 that does have some temporal interaction and can benefit from inferring end orderings. The temporal inferences are not specific to this domain but this is the only instance of such interaction that occurs in the benchmark domains. In more richly temporal problems,

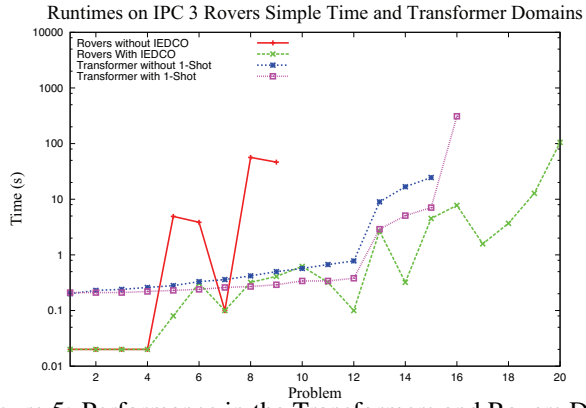


Figure 5: Performance in the Transformers and Rovers Domains, with and without End Action Orderings

end-action ordering is much more important.

In Rovers (Simple Time) the action `take_image` has an invariant, that the camera be calibrated, deleted upon completion of the action. The effect of this is to ensure that the camera must be recalibrated for each image to be taken. When planning with this domain, a planner can fall into the trap of starting two take image actions, using the same camera and a different target, but then be unable to apply the end of each action, as each deletes the invariant of the other. Indeed, this is precisely what happens in CRIKEY3, causing Enforced Hill Climbing (EHC) (Hoffmann and Nebel 2001) to fail and the planner to resort to best-first search. Using the same heuristic strategy, the planner explores the same branch again during best-first search, before eventually backtracking and finding a solution to the problem (if this is feasible within the time and memory limitations). However, invariant–end-delete ordering (Definition 51), allows CRIKEY3 to detect this inconsistency, and therefore not start two take image actions with the same camera at the same time: upon attempting to start a second `take_image` action the inferred ordering constraint introduces a cycle in the temporal constraints. As shown by the much improved scalability in Figure 5 (first two lines in the legend), employing this ordering allows problems to be solved much faster, and indeed for many more problems to be solved. The data shows performance with compression-safety in both cases: this remains useful for other actions in the domain.

In order to investigate the power of the inference in a temporally complex domain, we used a domain based on an application problem in voltage control. The details of the domain are not important, other than to observe that the encoding we used for these experiments was selected because it uses a significant set of clips to link together collections of actions that service particular elements of the voltage control system. The clips ensure that these actions are kept tightly grouped in time, with each action starting shortly after the previous one ends. In the problem instances we used for testing the number of clips varies from one in the smallest problem to 23 in the largest. Other aspects of the problem changed in this scaling, so the costs for solving the problems reflect not only the management of the clips, but also the time required to plan the actions that execute alongside the clipped sequences. In these experiments the clipped actions also have continuous linear effects active across their durations and so we used COLIN (Coles et al. 2009), a continuous-numeric variant of CRIKEY3, to obtain the results shown in Figure 5 (lines 3 and 4). As can be seen, we obtain a significant improvement in performance. For instance, on problem 15 (planning with 16 clips) the time taken to find a solution plan is reduced from 24.52 to 7.08 seconds. In both the Rovers domain and here in the voltage control domain, inference using end-actions has shown to improve the performance of CRIKEY3 (and its continuous-planning extension, COLIN).

7. Conclusions and Future Work

We have presented two useful classes of inference techniques to support temporal planning, one to identify situations in which simpler temporal reasoning is sufficient for handling actions and the other to prune branches in the search space using inferences based on interactions between action end points. A theoretical analysis of the latter reveals that the techniques cope well with clip and envelope structures, empirically confirmed by performance in the voltage control domain. In Planning Competition domains exploiting compression-safety is shown to be highly effective at reducing the burden of managing temporal planning. We now intend to extend the class of compression-safe actions to include 'safe' numeric effects, and to extend the range of inference techniques applicable in temporal and non-temporal planning settings.

References

Coles, A. I.; Fox, M.; Long, D.; and Smith, A. J. 2008. Planning with problems requiring temporal coordination. In *Proc. of AAAI 08*.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2009. Temporal planning in domains with linear processes. In *Proc. of IJCAI 09*.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. 2007. When is temporal planning *really* temporal planning? In *Proc. of IJCAI 07*, 1852–1859.

Do, M. B., and Kambhampati, S. 2001. Sapa: a Domain-Independent Heuristic Metric Temporal Planner. In *Proc. 6th European Conf. on Planning (ECP)*, 82–91.

Edelkamp, S. 2003. Taming numbers and durations in a model-checking integrated planning system. *J. of AI Res.* 20:195–238.

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

Fox, M.; Long, D.; and Halsey, K. 2004. An investigation into the expressive power of PDDL2.1. In *Proc. of ECAI'04*.

Gerevini, A.; Saetti, A.; and Serina, I. 2006. An approach to temporal planning and scheduling in domains with predictable exogenous events. *J. of AI Res.* 25:187–231.

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

Long, D., and Fox, M. 2003. Exploiting a Graphplan Framework in Temporal Planning. In *Proc. of ICAPS 03*, 52–61.

Rintanen, J. 2007. Complexity of concurrent temporal planning. In *Proc. of ICAPS 07*.

Smith, D., and Weld, D. 1999. Temporal Planning with Mutual Exclusion Reasoning. In *Proc. of IJCAI 99*, 326–337.