Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008)

Planning with Problems Requiring Temporal Coordination

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

Abstract

We present the first planner capable of reasoning with both the full semantics of PDDL2.1 (level 3) temporal planning *and* with numeric resources. Our planner, CRIKEY3, employs heuristic forward search, using the start-and-end semantics of PDDL2.1 to manage temporal actions. The planning phase is interleaved with a scheduling phase, using a Simple Temporal Network, in order to ensure that temporal constraints are met. To guide search, we introduce a new temporal variant of the Relaxed Planning Graph heuristic that is capable of reasoning with the features of this class of domains, along with the Timed Initial Literals of PDDL2.2. CRIKEY3 extends the state-of-the-art in handling the full temporal expressive power of PDDL2.1, including numeric temporal domains.

1 Introduction

A rich model of durative actions was introduced in PDDL2.1 (Fox & Long 2003). Actions can have effects at both their start and end points and have conditions that must hold either at the start, at the end or throughout execution. However, most state-of-the-art temporal planners (Gerevini, Saetti, & Serina 2006; Do & Kambhampati 2001; Chen, Wah, & Hsu 2006) work with a "compressed" action representation, translating PDDL temporal actions into a single temporally-extended STRIPS action. Under these semantics, introduced in TGP (Smith & Weld 1999), all effects happen instantaneously at the *end* of the action, and all preconditions must hold throughout its execution. In fact, TGP semantics also assert that propositions whose values change at the end of the action are considered to have undefined truth values during the execution of the action, so no concurrent activity may depend on these propositions. Compressed action representations do not usually respect this refinement of the semantics. When relying on these simplifying assumptions, planners cannot reason with problems that require concurrency for a solution.

In (Cushing *et al.* 2007), the term 'required concurrency' is defined to refer to problems that can only be solved using concurrent activity. That is, under PDDL2.1 semantics, in any valid solution it is necessary for the start or end of at least one action to happen between the start and end of another. 'Coordination', defined earlier in (Halsey, Long, & Fox 2004) and used in this work, refers to problems whose solutions can benefit from concurrent activity, subsuming required concurrency, but also including problems where the intention is to minimise makespan.

A number of existing planners can reason with domains where coordination is required. One example is LPGP (Long & Fox 2003), a GraphPlan-based temporal planner. VHPOP (Younes & Simmons 2003) and, more recently, CRIKEY (Coles *et al.* 2008), can also handle coordination. However, all of these planners are restricted in various ways and do not support the full semantics of PDDL2.1 or, in some cases, the use of numeric resources. Many real problems require resources to be replenished in order to achieve goals, allowing 'time' to be traded for a numeric resource. The actions for this might represent charging a battery on a rover or a resource production activity. Supporting the reasoning that this demands of a planner is a major challenge, not fully addressed by any previous planner.

In this paper we present CRIKEY3, a planner capable of reasoning with temporal and numeric domains, fully respecting the semantics of PDDL2.1. We first discuss some of the implications of this task. We then briefly introduce the planner CRIKEY, used as the basis for our planner, and follow this with details of the modifications we make to it. We also detail another contribution: a fully temporal and metric relaxed planning graph for PDDL2.1, supporting Timed Initial Literals (TILs) (first introduced in PDDL2.2). Finally, we present results to demonstrate the expressivity of our new planner and discuss directions for further work.

2 Temporal Numeric Interactions

The interaction between temporal and numeric planning is particularly interesting: as noted above, many real problems involve trading time for resources. In coordination problems, time itself can be an important resource. For example, if one action supplies a resource only over its duration, then the management of the activities that require this resource will demand concurrent execution of the resource provider and the resource consumers.

A simple example that we will use in this paper is the following. Suppose that we have two actions OPERATE_MINE and MINE_FOR_COAL. The OPERATE_MINE action causes a coal mine to be operational at the start of its execution, but closes the mine at its end. The MINE_FOR_COAL action

requires the coal mine to be operational throughout its execution and at its end produces a unit of coal. To produce any coal will require coordination: the concurrent execution of the OPERATE_MINE and MINE_FOR_COAL actions. Using the TGP compilation for a propositional version of this domain results in the fact that the mine is operational during the execution of the OPERATE_MINE action being lost. When compressing the action into a single STRIPS action, the start effect `(operational ?m)` is cancelled out by the end effect `(not (operational ?m))`, erasing the effect altogether.

A more interesting problem arises when the problem is encoded using numeric resources. In that case, we can pose the goal that several, ten say, units of coal be produced. If the duration of the OPERATE_MINE action is less than ten times that of the MINE_FOR_COAL action, then self-overlapping actions are *necessary* in order to satisfy the goal. In the context of propositional planning, Rintanen has shown (Rintanen 2007) that the class of planning problems in which multiple instances of the same ground action can execute concurrently is EXPSPACE-hard. In numeric problems self-overlapping actions have a far more important role to play, as the preceding example illustrates. CRIKEY3 is the first planner capable of reasoning about problems that require this kind of sophisticated coordination.

3 CRIKEY

The work we present here is based on CRIKEY (Coles *et al.* 2008; Halsey, Long, & Fox 2004). CRIKEY is a temporal planner capable of reasoning with coordinated actions. It employs the technique first used in LPGP (Long & Fox 2003) of splitting durative actions into start and end actions, referred to as *snap actions*. With these it performs forward-search in state-space using the Relaxed Planning Graph heuristic in a similar manner to FF (Hoffmann & Nebel 2001). Although splitting the actions into snap-actions causes a constant-factor increase in the search space, the alternative encoding used in the majority of other temporal planners is fundamentally limited and is neither sound nor complete with respect to PDDL2.1 (Coles *et al.* 2008).

To manage the temporal constraints between snap-actions, CRIKEY relies on an extension of the state to contain *envelopes*. Envelopes serve to capture the active temporal constraints imposed by the snap-action choices. For full details, we refer the reader to (Coles *et al.* 2008). However, for the purposes of this paper, it is sufficient to think of envelopes as follows. Suppose that each durative action, A, is split into start and end snap-actions denoted $A_\vdash$ and $A_\dashv$, respectively. Whenever a start action $A_\vdash$ is applied, a new envelope is created. It is closed when the corresponding end action, $A_\dashv$, is applied. The envelope spans from $A_\vdash$ to $A_\dashv$, and contains an associated set of *content* snap-actions. A snap-action is added to the contents of an envelope if it *interacts* with any action in the envelope, including the two envelope end points, where interaction occurs when a precondition of one action is deleted or achieved by the other. To check for temporal consistency, each envelope has an associated Simple Temporal Network (STN), populated with the content nodes and the two end points, and with edge weights recording temporal relationships (a nominal separation, ϵ, for simple precedence constraints between two snap-actions and the duration of the action A between any pair $A_\vdash$, $A_\dashv$). By maintaining a mini-STN, we can focus on the temporally relevant portions of the global schedule implied by the action choices made thus far.

4 Searching for Temporal Plans

In CRIKEY3, a number of modifications are made to the search space used in CRIKEY. In CRIKEY, each state S in the search space is a pair $\langle F, \xi \rangle$, where F is a state (consisting of propositional facts and fluent values) and ξ is a set of envelopes. In CRIKEY3, a state is $S = \langle F, E, T \rangle$, where F is a state, E is an ordered list of start events, recording actions that have started but not yet finished, and T is a collection of temporal constraints over the actions in the plan to reach F. Each entry $e \in E$ is a tuple $\langle op, i, dmin, dmax \rangle$ where:

- op is the identifier of a start snap-action;
- i is the index of the snap-action in the plan to reach S;
- $dmin, dmax$ are the minimum and maximum duration, determined in the state in which the action first executed.

This modified node definition requires corresponding modification to the successor function used in search. As in CRIKEY, a snap-action is deemed to be *logically applicable* in a state S if two conditions hold: F must satisfy its preconditions and it must not delete any active invariants. In CRIKEY, invariants are associated with each envelope in ξ. In CRIKEY3, the invariants are obtained from the durative action associated with the start action, op, in each $e \in E$.

Applying a snap-action to a state (as step i of a plan) leads to a successor $\langle F', E', T' \rangle$. Whether a start or end snap-action is applied, F is updated to reflect the effects of the snap-action to give F'. If a start-action $A_\vdash$ is applied, a new entry $\langle A_\vdash, i, min, max \rangle$ is added to the end of E to give E', with min and max corresponding to the minimum and maximum durations of A[1]. If an end snap-action $A_\dashv$ is applied the start entry $\{e \in E \mid e.op = A\}$ with which to pair it must be chosen. Each possible choice leads to successors with different pairs E', T' obtained when $A_\dashv$ is paired with an entry $e \in E$, $E' = E \setminus e$, and:

$T' = T \cup \{e.min \leq t(i) - t(e.i) \leq e.max\}$.

Where CRIKEY relied on compilation to handle PDDL2.2 Timed Initial Literals (TILs), CRIKEY3 is able to handle them more directly. We use dummy 'TIL' actions that combine the effects of the TILs at each time point, and these can be added to the plan if they do not delete the invariants of any open action. As a special case, TIL actions do not create an entry in E: only the facts in F are amended. They do, however, produce an updated set of temporal constraints: $T' = T \cup \{ts \leq t(i) - t(a_0) \leq ts\}$ where a_0 is a dummy event denoting the start of the plan, and ts is the timestamp at which the TILs are prescribed to happen.

The behaviour we have described ensures that plans produced by CRIKEY3 are logically sound: the check for logical applicability, coupled with the maintenance of E throughout

[1] In many domains these will be the same, but PDDL2.1 allows durational inequalities to be specified.

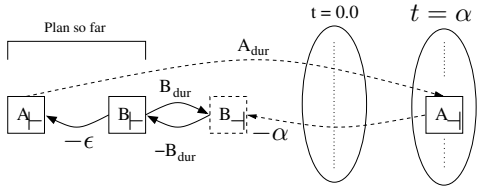

Figure 1: Extended state pruning. Here, the planner is considering adding the snap-action $B_\dashv$ to the plan.

search, ensures no preconditions, either propositional or numeric, can be unsatisfied. It remains to be demonstrated how we ensure that plans are *temporally* sound.

4.1 Temporal Applicability

From a state $S = \langle F, E, T\rangle$, reached by applying i actions, it is possible to induce a Simple Temporal Network (STN) for the snap-actions in the plan so far and the temporal relationships between them. The nodes in the STN each correspond to single steps, $a_1, ..., a_i$, in the plan and a_0, denoting the start of the plan. The temporal constraints in the STN are:

$$T \cup \{\epsilon < t(a_{j+1}) - t(a_j) \mid j = 0...i-1\} \cup \{t(a_0) \geq 0\} \quad (1)$$

These constraints enforce a total order on the snap-action steps by requiring a minimal separation ϵ between each successive pair of actions in the plan and that the start of the plan can be no earlier than time zero.

The consistency of the resulting STN can then be checked by using a shortest-path algorithm to check for cycles. By checking the temporal consistency of the state, S', produced by application of an action to a consistent state, S, it is possible to ensure that the final plan is temporally valid.

5 Exclusion of Inevitable Cycles

Repeatedly applying a shortest path algorithm to detect cycles within the STN induced by action choices is expensive. In an interesting subset of cases, detecting cycles within the induced STN can be achieved in advance. For instance, consider the following action sequence, illustrated on the left of Figure 1: $A_\vdash, B_\vdash, B_\dashv$. Such an action sequence, in isolation, will always be temporally valid. However, under the semantics of PDDL2.1, no actions may remain executing in a goal state, so any valid solution plan found from here will necessarily include $A_\dashv$, introducing a temporal constraint between $A_\vdash$ and $A_\dashv$. If the minimum duration of B exceeds the maximum duration of A, it is clear that a cycle will be created in the STN as soon as $A_\dashv$ is applied. In the worst case, a great deal of search could be performed before the addition of $A_\dashv$ causes the inevitable cycle to be discovered.

To detect these cases, without needing to build an STN, we augment each start event queue entry $e \in E$ with two further values $tmin$ and $tmax$. These are, respectively, the minimum and maximum time that could have elapsed since the corresponding event, and are updated when a successor state is generated. If a start action is applied then time is only forced to move on by ϵ, according to the total-ordering imposed on the snap-action choices in expression 1. Therefore, in this case, each of the $tmin$ and $tmax$ values is incremented by ϵ. If an end action is applied, associated with start queue event entry e, then:

- for events b that precede e, $tmin$ and $tmax$ are incremented by $\max[e.dmin - e.tmax, \epsilon]$
- for events b that follow e, $tmin$ is incremented by ϵ and $tmax$ by $\max[e.dmin - e.tmax, \epsilon]$.

Then, in any state reached during search, if for any start event queue entry $tmin \geq dmax$, a cycle will inevitably be introduced into the STN upon the event being closed. In the earlier example, applying the sequence of three actions would lead to the $tmin$ value for $A_\vdash$ being set to $duration(B) + \epsilon$. If this value exceeds the duration of A, then the state will (correctly) be pruned because a cycle in a future STN is inevitable.

6 Temporal Relaxed Planning for PDDL

Having split PDDL2.1 durative actions into start and end snap-actions, it is necessary to make modifications to the relaxed planning graph (RPG) to ensure that it continues to give meaningful heuristic guidance. This section begins with an overview of existing adaptations of the RPG for temporal planning, and follows with a discussion of our approach and the benefits it provides.

6.1 RPGs for Temporal Planning

Three desirable qualities of a relaxed planning graph to guide temporal planning are that it should:

1. Respect the PDDL2.1 start–end semantics.
2. Respect relationships between start and end snap-actions: $A_\dashv$ can only be applied if $A_\vdash$ has been applied before it.
3. Account for action durations: end effects of actions are only available sufficiently far after they have started.

The first of these is necessary for the preservation of planner completeness: the relaxed plan heuristic should never incorrectly indicate that a dead end has been reached. This is a problem that affects the temporal relaxed planning graph (TRPG) employed in Sapa (Do & Kambhampati 2001). Here, each action is compressed into a temporally-extended action obeying the TGP semantics (see Section 1) before discarding delete effects and building a TGP-style planning graph (Smith & Weld 1999). This approach satisfies the second requirement (compressing actions enforces a correct ordering on start and end points) and the third requirement (the actions are labelled with durations). However, the first requirement is violated: the heuristic based on this TRPG finds false dead-ends.

CRIKEY uses a better TRPG for problems involving coordination, satisfying the first two requirements. For the first, relaxed plans are extracted from a classical RPG, built using snap-actions, hence providing relaxed solutions to coordination problems. For the second, end snap-actions are augmented with an additional dummy precondition achieved by their corresponding start. Thus, choosing to use the end of an action requires its start to be chosen at an earlier layer. During search, each state is also augmented with these dummy preconditions, so an end action can be added to the relaxed plan only if its start is applied at some earlier layer, or if the action is already executing.

Algorithm 1: Building the Temporal RPG

Data: $S = \langle F, E, T \rangle$ - state to be evaluated
Result: $R = \langle f_{0..n}, a_{0..n} \rangle$, a relaxed planning graph

1 $f_0 \leftarrow F$;
2 $t \leftarrow 0$;
3 **foreach** $A_\dashv$ **do**
4 **if** $\{e \in E \mid e.op = A\} = \emptyset$ **then**
5 $earliest(A) \leftarrow \infty$;
6 **else** $earliest(A) \leftarrow 0$;
7 **while** $t < \infty$ **do**
8 $f_{t+\epsilon} \leftarrow f_t$;
9 $a_t \leftarrow \{A_\dashv \mid pre(A_\dashv) \subseteq f_t \wedge earliest(A) \leq t\}$;
10 **foreach** *new* $A_\dashv \in a_t$ **do**
11 $f_{t+\epsilon} \leftarrow f_{t+\epsilon} \cup eff^+(A_\dashv)$;
12 $a_t \leftarrow a_t \cup \{A_\vdash \mid pre(A_\vdash) \subseteq f_t\}$;
13 **foreach** *new* $A_\vdash \in a_t$ **do**
14 $f_{t+\epsilon} \leftarrow f_{t+\epsilon} \cup eff^+(A_\vdash)$;
15 $earliest(A) = \min[earliest(A), t + lb(A)]$;
16 **if** $f_t \subset f_{t+\epsilon}$ **then** $t \leftarrow t + \epsilon$; **else**
17 $ep = \{earliest(A) > t \mid pre(A_\dashv) \subseteq f_t\}$;
18 **if** $ep \neq \emptyset$ **then** $t \leftarrow \min[endpoints]$;
19 **else** break;
20 **return** $R = \langle f_{0..n}, a_{0..n} \rangle$

6.2 A Novel Temporal RPG

We now present a new approach that meets all three requirements, building on the strengths of both the Sapa and CRIKEY TRPGs, but without their respective weaknesses.

Algorithm 1 shows our approach[2]. As in CRIKEY, we build an RPG using snap-actions, forwards from a state S, and maintaining start-action–end-action relationships. However, we also maintain temporal information, using time-stamped layers (beginning at time 0 for the first fact layer, i.e. the state to be evaluated). The temporal constraints are updated in three key places:

- At line 3 we initialise the earliest possible layer $earliest(A)$ for each end snap-action $A_\dashv$: 0 if the action is already under execution, ∞ otherwise.
- At line 15 $earliest(A)$ is updated if a start $A_\vdash$ newly appears in an action layer at time t. Here, $lb(A)$ denotes a lower bound on the duration of A. If this is not state dependent we use its actual value, otherwise we use a small value, ϵ, to push $A_\dashv$ into the next action layer.
- At line 9, on the basis of these values, we only add end actions to action layers if sufficient time has elapsed.

This process is similar to the construction of the time-stamped planning graph in TPSYS (Garrido, Fox, & Long 2002). To support TILs, the effects of the compilation from TILs to PDDL2.1 are simulated: the next dummy TIL action to be applied is available at time 0, and subsequent TILs are available after that at appropriate temporal intervals.

[2]In the interests of clarity, the handling of fluent values is omitted. These are handled just as in METRIC-FF (Hoffmann 2002).

On termination, a feasible relaxed plan can be extracted if two conditions hold (otherwise a dead-end has been found). First, the goals have to be satisfied in the final fact layer of the RPG. Second, the end actions (for any currently executing action) $re = \{A_\dashv \mid \exists e \in E \cdot e.op = A\}$ must each be present in an action layer. If these two conditions hold, the relaxed plan extraction algorithm from METRIC-FF (Hoffmann 2002) can be used with two modifications: the end actions re *must* be chosen from their earliest action layers, irrespective of whether their effects are needed and, if an end action $A_\dashv$ is chosen to achieve a fact and $A_\dashv \notin re$, then $A_\vdash$ must also be chosen in an earlier action layer.

6.3 Adding additional constraints

Algorithm 1 makes the optimistic assumption that the end of any open action can be applied as soon as its preconditions are satisfied. The rationale behind this is that the relaxed plan is built forwards from the point at which the last action in the plan was applied and, optimistically, that point could be pushed arbitrarily forwards in time to the point where all open actions have elapsed for long enough to be closed without delay. However, if a start action $A_\vdash$ has just been applied, then the corresponding end $A_\dashv$ cannot appear in the RPG until sufficient time has elapsed: moving $A_\vdash$ later in time would by no means reduce this. This constraint can be added to the RPG construction, delaying $A_\dashv$ to the appropriate action layer. We can also add the constraint that the invariants of A must be respected until this layer: since any action in the relaxed plan must follow $A_\vdash$, any invariants of A cannot be violated until after $A_\dashv$.

This principle of delaying the end effects of open actions can be extended to actions other than that just applied. Consider the action sequence $A_\vdash, B_\vdash, C_\vdash$. We know that the corresponding ends to all of these must be ordered after $C_\vdash$ and that, if the duration of B exceeds that of A, then $A_\dashv$ must occur before $B_\dashv$ (otherwise a trivial cycle is introduced). Hence, the maximum possible remaining time of the execution of A ($e.dmax - e.tmin$ for the start event queue entry corresponding to A) gives an upper bound on how far into the future $C_\vdash$ can be pushed with respect to $B_\vdash$. This then allows $B_\dashv$ to be delayed until the action layer with timestamp $e'.dmin - (e.dmax - e.tmin)$ (where e' is the start event queue entry corresponding to B) and the invariants of B must be respected over this duration.

These modifications allow the timestamps for the ends of actions to be delayed in the RPG, whilst maintaining admissibility. Previously, ends could appear at 0: now, they appear at their earliest feasible point.

6.4 Adding deadlines

A final modification we make to the TRPG is to include the deadlines implied by TILs. The idiom for encoding a deadline in this manner is characterised by a propositional fact p which is true in the initial state, is not added or deleted by any action, but is deleted by a TIL at time d and not added by any subsequent TIL. This enforces that d is the latest point at which p can be used as the precondition of an action.

During search, we can determine the earliest time point, t, from which the planning graph is being built by using the

same arithmetic described in Section 5: t is the minimum possible amount of time elapsed from an implicit start action fixed to be at the start of the event queue. With this, we can calculate the RPG layer at which each deadline should occur: a deadline at time d appears at layer $d - t$. After this point, during planning graph construction we can forbid the addition of any snap-action which requires p. As the time stamps for actions in the TRPG are admissible, and $d - t$ is an upper-bound on the deadline time, any action which has not appeared before that point could not possibly occur in any non-relaxed plan. If a relaxed plan cannot be found, the state is shown to be a dead end.

7 Further State Pruning using the TRPG

In Section 5 we showed that CRIKEY3 can find cycles in the STN when a start snap-action $A_\vdash$ in its plan has not yet been paired with a corresponding end $A_\dashv$. In general, when a snap-action $A_\vdash$ is applied, there are no guarantees that the preconditions of its end action $A_\dashv$ are satisfied, nor are they forced to remain true: they are only required to hold at the point when $A_\dashv$ is applied. If they are not achieved by this time, we reach a dead-end. The inevitable cycle detection will detect a useful subset of these, avoiding the need to heuristically evaluate the state or construct an STN, and using the TRPG information we can extend this further.

The right of Figure 1 continues the example covered earlier in the paper: once again, we are considering the application of $B_\dashv$. Assuming the cycle detection of Section 5 is satisfied, for heuristic purposes a relaxed plan will then be constructed to the goal state. When extracting the relaxed plan, an end action must be present for each action which is still executing: in this case, $A_\dashv$ must occur in the relaxed plan. In our example here, $A_\dashv$ becomes applicable at time $t = \alpha$ following the application of $B_\dashv$, in action layer a_α.

As we maintain admissibility in the TRPG, we have lower bounds on the times at which the end snap-actions of executing actions become applicable. These bounds can be incorporated into the STN and used to extend the consistency checking proposed in Section 4.1. We start with the nodes as before: a dummy node for time zero, and nodes $[0, d)$ for a plan of length d. For each entry in $\langle op, i, min, max, tick\rangle \in E$, corresponding to the addition of $op_\vdash$ to the plan, we add a node to the STN for $op_\dashv$, and two constraints involving this new node. First, $(t(op_\dashv) - t(i) \leq e.max)$: the time between the start and end of an action cannot exceed its maximum duration. Second, $\alpha \leq t(op_\dashv) - t(d)$: the end action $op_\dashv$ taken from the relaxed plan can occur no earlier than α after the last step, d, the last action choice made thus far. These additional constraints allow more dead-ends to be forecasted during search, avoiding the fruitless search and backtracking which would otherwise arise, and hence allow the heuristic computation to provide further search guidance.

8 Evaluation

We now evaluate CRIKEY3, using theoretical and empirical techniques. As the theoretical differences between CRIKEY and CRIKEY3 have been covered, we shall consider other related planners. First, CRIKEY3 bears some similarities

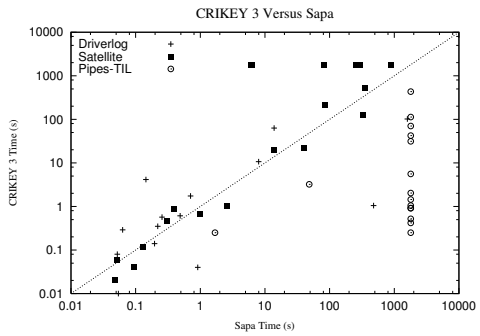

Figure 2: Comparison between Sapa and CRIKEY3

to Sapa and TEMPO (Cushing *et al.* 2007). Both Sapa and CRIKEY3 perform state-space search, augment states with temporal information, and use a TRPG-based heuristics. However, as observed in (Cushing *et al.* 2007), Sapa makes compromising assumptions that render it incomplete for problems requiring coordination. TEMPO is a theoretical planner, so comparisons in terms of heuristics or search algorithms cannot be made. We can remark, however, that by fixing the duration of an action when it is applied, TEMPO is incomplete under PDDL2.1 level 3 as it cannot handle durational inequalities; the mechanism by which these are handled in CRIKEY3 is covered in Section 4.

Our empirical analysis is divided into three sections. First, we compare to Sapa. As discussed, the two are theoretically close: in one sense, CRIKEY3 is a development of Sapa with the modifications to support coordination (such as a revised TRPG, and the use of STNs). Hence, these empirical comparisons give an indication of the price paid for supporting coordination, using the principles proposed in this paper. We then consider temporal domains with self-overlapping actions, but no metric fluents, comparing CRIKEY3 and VHPOP. Finally, we show results for CRIKEY3 on a temporal-metric domain with self-overlapping actions.

Figure 2 shows a comparison of the results of running CRIKEY3 and Sapa. As Sapa is not as expressive as CRIKEY3, two suitable benchmark domains were chosen: the IPC3 Driverlog-Time problems and the IPC4 Satellite-Time problems. By ignoring the potential for both coordination and self-overlapping actions, Sapa prunes branches from the search space that could render its search incomplete, whereas CRIKEY3 explores a search space that includes these branches. The results of this experiment are shown in Figure 2 and show that, in terms of time taken to solve problems, CRIKEY3 and Sapa are evenly matched: a point below the dotted line indicates that CRIKEY3 was faster, a point above the line that Sapa was faster. In terms of coverage, in Driverlog CRIKEY3 and Sapa each solved 15 problems and in Satellite, 14 and 20 respectively, within the allowance of 1800 seconds (points showing 1800 seconds all represent unsolved instances for the corresponding planner).

To compare performance with TILs we used the 'PipesWorld-NoTankage Temporal Deadlines' domain from IPC4; and, since Sapa cannot reason with TILs, we compare with TIL-SAPA (Kavuluri & Senthil 2004). The advantage of being able to coordinate actions and consider the impli-

cations of TILs in CRIKEY3 become clear: Figure 2 shows that CRIKEY3 solves 16 problems; TIL-SAPA solves only 2. Overall, this shows that CRIKEY3 extends the expressive power of Sapa and TIL-SAPA without excessive cost.

VHPOP and CRIKEY represent the current state-of-the-art in planning with coordination: Sapa cannot solve such problems; and TEMPO is, as yet, unavailable as an implementation. VHPOP, CRIKEY and CRIKEY3 can all solve the propositional version of the coal mining problem (Section 2) in less than a second. However, self-overlapping actions are not required to solve this problem and when we consider a temporal propositional domain which does require this a different picture emerges. The new problem we consider is shown in Figure 3: the goal is to achieve both $G1$ and $G2$ from an empty initial state.

CRIKEY3 solves this problem correctly in 0.01 seconds. CRIKEY is unable to solve this problem as it does not reason with self-overlapping actions at all. VHPOP might be expected to solve this problem, but it produces an invalid plan, missing the necessary second execution of action ONE in order to ensure that the end precondition, B, of the original instance of action ONE is satisfied. The failure of VHPOP to ensure that the end preconditions of actions are satisfied means that it is not able to solve problems with *weakly* conditional effects and therefore cannot solve problems with self-overlapping actions. This shows that CRIKEY3 is the only planner that can handle self-overlapping actions even in purely propositional domains.

Finally, we consider a more interesting type of problem with self-overlapping actions: a numeric temporal problem. No benchmark domains contain this feature so we use the numeric coal domain as described in Section 2. This domain requires both coordination and the self overlapping of the MINE_COAL actions: the OPERATE_MINE action is not long enough for the MINE_COAL actions to be sequentialised. No other planner can solve problems in this domain, so we have investigated scalability of CRIKEY3 on instances of growing size, as shown in Figure 4. Each problem n requires that $5n$ units of coal be produced and, as can be seen on the graph, CRIKEY3 scales sub-exponentially.

9 Conclusion

In this paper we have introduced a new planner, CRIKEY3, the first capable of reasoning with the full temporal coordination problem in numeric temporal planning domains, including deadlines. In particular, we handle the case where the same action instance can overlap itself multiple times and cases where coordination is necessary. Our results show that, despite reasoning with a more complex problem, CRIKEY3 remains competitive. Further, we have demonstrated the expressivity of CRIKEY3 showing its performance and scalability on domains with self-overlapping actions with and without numeric fluents, a class of domains which no other available planner is able to solve.

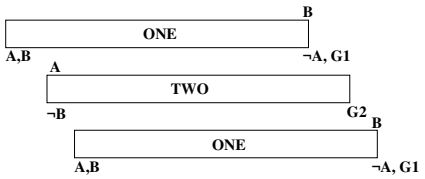

Figure 3: Propositional problem requiring coordination

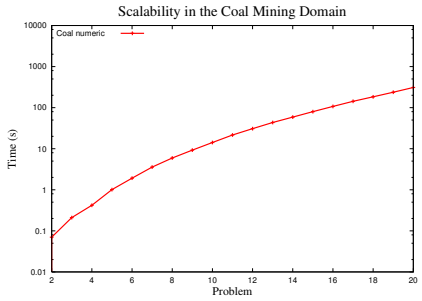

Figure 4: CRIKEY3 scaling behaviour.

Acknowledgments

We would like to thank Mausam and Will Cushing for interesting discussions about concurrency in temporal planning.

References

Chen, Y.; Wah, B.; and Hsu, C. 2006. Temporal Planning using Subgoal Partitioning and Resolution in SGPlan. *J. of AI Research* 26:323–369.

Coles, A. I.; Fox, M.; Halsey, K.; Long, D.; and Smith, A. J. 2008. Managing Coordination in Temporal Planning using Planner-Scheduler Interaction. *To appear, Artificial Intelligence*.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. 2007. When is temporal planning *really* temporal planning? In *Proc. of Int. Joint Conf. on AI (IJCAI)*, 1852–1859.

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

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

Garrido, A.; Fox, M.; and Long, D. 2002. A Temporal Planning System for Durative Actions of PDDL2.1. In *Proc. 15th European Conf. on AI, (ECAI)*, 586–590.

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

Halsey, K.; Long, D.; and Fox, M. 2004. CRIKEY: A Temporal Planner Looking at the Integration of Scheduling and Planning. In *Proc. Workshop on Integrating Planning into Scheduling (WIPIS)*.

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

Hoffmann, J. 2002. Extending FF to numerical state variables. In *Proc. 15th European Conf. on AI (ECAI-02)*, 571–575.

Kavuluri, B., and Senthil, U. 2004. Timed Initial Literals Using Sapa. IPC4 Booklet, ICAPS 2004. Extended Abstract.

Long, D., and Fox, M. 2003. Exploiting a Graphplan Framework in Temporal Planning. In *Proc. 13th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 52–61.

Rintanen, J. 2007. Complexity of Concurrent Temporal Planning. In *Proc. 17th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 280–287.

Smith, D., and Weld, D. S. 1999. Temporal Planning with Mutual Exclusion Reasoning. In *Proc. 16th Int. Joint Conf. on AI (IJCAI)*, 326–337.

Younes, H. L. S., and Simmons, R. G. 2003. VHPOP: Versatile heuristic partial order planner. *J. of AI Research* 20:405–430.