Temporal Planning in Domains with Linear Processes

Amanda Coles, Andrew Coles, 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

We consider the problem of planning in domains with continuous linear numeric change. Such change cannot always be adequately modelled by discretisation and is a key facet of many interesting problems. We show how a forward-chaining temporal planner can be extended to reason with actions with continuous linear effects. We extend a temporal planner to handle numeric values using linear programming. We show how linear continuous change can be integrated into the same linear program and we discuss how a temporal-numeric heuristic can be used to provide the search guidance necessary to underpin continuous planning. We present results to show that the approach can effectively handle duration-dependent change and numeric variables subject to continuous linear change.

1 Introduction

Recently, considerable progress has been made in planning with time and metric fluents. However, there remains relatively little work exploring the combination of the two, where metric values change in time-dependent ways. PDDL2.1 [Fox and Long, 2003] and PDDL+ [Fox and Long, 2006] offer two ways to model time-dependent change: as discrete time-dependent effects of durative actions or as continuous process-dependent change. We present an approach to planning with linear continuous processes and with discrete time-dependent effects using a combination of forward state-space heuristic search planning and linear programming. The planner we describe is the only general planner able to solve the Airplane Landing problem [Dierks, 2005], posed as a challenge by Kim Larsen in his invited ICAPS lecture in 2005.

We briefly consider motivation for the problem and related work. We proceed by describing the temporal planning strategy that forms the basis of our planner, COLIN (COntinuous LINear process planner). We then describe the role of linear programming in our approach and describe its integration into our planner. Finally, we provide some results demonstrating the effectiveness of our approach.

2 Background

PDDL2.1 [Fox and Long, 2003] is a standard language for encoding planning domains with both time and numeric quantities. It raises the challenge of reasoning with time-dependent change, both discrete and continuous. Many problems involve reasoning with quantities that are subject to processes, such as energy management, fuel consumption and replenishment, use of storage tanks in chemical plants and so on. In some cases, these processes can be abstracted into discrete changes, but the sizes of those discrete changes depend on the time over which the corresponding processes can run. It is not always appropriate to discretise time because the granularity of the discretisation required to expose the necessary choices to a planner might be so fine-grained as to make the models infeasibly large and it might be difficult to identify an appropriate granularity without actually solving the planning problem in the first place. Equally, it is not always appropriate to discretise processes themselves, since it is sometimes necessary to manage interactions with the processes while they are running, accessing the values of metric fluents at intermediate stages of active processes.

Other researchers have considered the problem of planning in mixed discrete-continuous domains. Early work exploring planning with continuous processes includes the Zeno system of Penberthy and Weld [1994], in which processes are described using differential equations. Zeno is a partial-order planning system and, while impressively conceived, is unable to solve large problems and cannot handle overlapping processes affecting the same variable. Shin and Davis [2005] developed TM-LPSAT, using a SAT model of the discrete parts of a planning problem combined with an LP-solver to manage (linear) processes modelled in PDDL2.1. Linear constraints on the continuous dynamics are captured as logical variables. Once a satisfying assignment for the discrete problem is found, the linear constraints corresponding to the active variables are assembled into a linear program which is solved separately. If it cannot be solved, the system backtracks over satisfying assignments. The planner has no heuristic guidance and solves only relatively small problems.

Brian Williams and his colleagues have explored planning for hybrid systems [Léauté and Williams, 2005; Li and Williams, 2008]. This work has focussed on model-based control and uses techniques based on constraint reasoning. The continuous dynamics of systems are modelled as *flow tubes* that capture the envelopes of the continuous behaviours. The dimensions of these tubes are a function of time (typ-

```
(   :durative-action generate
    :parameters (?g - generator)
    :duration (= ?duration 100)
    :condition (over all (> (fuel-level ?g) 0))
    :effect
       (and   (decrease (fuel-level ?g) (* #t 1))
              (at end (generator-completed))))
```

Figure 1: Generate Action from the Generator Domain

ically expanding as they are allowed to extend) so that fitting together successive continuous behaviours involves connecting the start of one tube (the precondition surface) to the cross-section of the preceding tube at a time when there is a non-empty intersection between the spaces. Like TM-LPSAT, the planner, Kongming, relies on an underlying iterative search in increasing length action sequences, without heuristic guidance, and therefore suffers from the same scaling limitations as Graphplan.

McDermott's OPTOP system [McDermott, 2003] also plans with linear continuous change, but cannot handle concurrent continuous change to the same variable.

3 The Problem

In this paper we focus on solving PDDL2.1 planning problems using durative actions to model temporal structure. We extend earlier treatment of durative actions, such as [Gerevini *et al.*, 2006; Coles *et al.*, 2008; Do and Kambhampati, 2001; Edelkamp, 2003], in two ways: we allow actions of variable duration to have effects that depend on their durations and we allow linear continuous effects. The form of these is described in [Fox and Long, 2003], but we present a simple example here, based on a domain similar to the Generator domain [Howey and Long, 2003] (simplified so that numeric change is linear). In this domain, a generator must run for a fixed length of time (100 minutes) and, since it powers a critical system, it must run continuously. The generator consumes fuel from an initially full tank, with capacity 90 units, at a rate of 1 unit per minute. The tank can be refuelled while the generator runs, but its capacity must not be exceeded. The refuelling action increases the fuel in the tank at a rate of 2 units per minute for 10 minutes.

The action to run the generator is shown in Figure 1. The condition on executing this action is that the fuel level in the generator must never fall to zero. The continuous numeric effect, the first on the effects list, is the interesting feature of this action. In PDDL2.1 continuous numeric change is specified as the rate of change of a fluent, in this case `fuel-level`, with respect to time expressed by `#t`. `#t` is a special term denoting the time elapsed since the start of the process. Thus, the effect states that t time units after the action has been executed, fuel consumption is $t * 1$, or t.

Duration-dependent effects of durative actions have been used in prior benchmark domains (such as Rovers Time in IPC3). Durative actions with such effects can be split into two cases: a simple case, where the duration of the action is determined by the values of metric fluents in the state in which the action starts, which in turn determines the effect of the action, and a more complex case in which the duration of the action is only constrained in some interval. The latter case gives the planner freedom to choose the duration of the action within an interval at the time of application, creating a more complex branching choice at this point than in the simple case. Until now, no planners have dealt with this case.

Reasoning with continuous resources clearly depends on reasoning about both numeric resources and time. We will begin by describing an existing approach to reasoning with this combination, restricted to discrete numeric change, before going on to present an approach that also handles linearly changing continuous numeric effects.

4 CRIKEY3

To handle continuous numeric effects it is crucial to reason properly with time. In PDDL2.1, durative actions are modelled in terms of pre- and post-conditions at their start and end points, together with invariant conditions that must hold throughout their execution and a duration constraint that governs how long the action executes. In simple temporal domains, durations are fixed, while in more complex domains they can be dependent on the context in which the action is executed. PDDL2.1 also allows the specification of bounds within which action durations can be selected by the planner.

Many state-of-the-art planners simplify temporal planning into non-temporal planning by compiling each temporal action into a single instantaneous action representing the net effect of the execution of the whole action [Cushing *et al.*, 2007]. The resulting semantics are close to those of actions in TGP [Smith and Weld, 1999]. Such a compilation works for some simple problems without interesting temporal features, but when more complex problems are specified, using the full power of PDDL2.1, this approach is both unsound and incomplete. Throughout the remainder of this discussion we refer to this compilation approach as *compressing*.

In a solution to the simplified generator problem, the `refill` action must occur *during* the execution of the `generate` action. The constraint that the tank must not overflow prevents the `refill` action from being applied before the `generate` action, while the constraint that the generator must not run dry requires that the `refill` action be applied before the `generate` action ends.

In problems such as this compressing is not effective. A different approach is needed that takes note of the temporal structure of actions. This observation has led to the development of planners that avoid compressing and reason with richer temporal interactions. Such planners include Sapa [Do and Kambhampati, 2001], TPSYS [Garrido *et al.*, 2001], LPGP [Long and Fox, 2003], TM-LPSAT [Shin and Davis, 2005], CRIKEY3 [Coles *et al.*, 2008] and Kongming [Li and Williams, 2008]. CRIKEY3 is a forward chaining temporal planner that is able to reason with problems where concurrent actions are necessary in order to find a solution. It is the temporal planning strategy upon which we base the continuous planning framework.

CRIKEY3 rewrites temporal actions into two separate instantaneous actions: one representing the start of the action and the other the end. To ensure a one-to-one correspondence between starts and ends of actions in any plan, additional dummy facts are used. This rewriting is similar to that used

in LPGP. The starts and ends of actions can be interleaved by the planner during search. Ignoring the metric constraints for the moment, a solution can be constructed to the rewritten generator problem as follows: `0: generate_start; 1: fill_start; 2: fill_end; 3: generate_end`.

To ensure that planning with start and end actions remains sound requires two further elements. First, starting an action establishes invariants that must be maintained until the action is ended. To ensure this, the state representation is extended to record the invariants of any actions that have started, but not yet finished. When considering which actions have satisfied preconditions, actions that violate any of these active invariants are excluded. Second, when searching for a solution plan, we must also consider the duration constraints between the start and end points of actions. There are two sources of temporal constraints: successive steps in the plan are sequenced one after the other and the duration constraint of an action must hold between its start and its end. To capture these constraints, CRIKEY3 uses a Simple Temporal Network (STN). To encode the steps $1..i$ within the plan, one vertex is added to the plan for each, denoted $step_1..step_i$. Successive pairs of plan steps are constrained thus: $\epsilon \leq step_{i+1} - step_i$. Separation by a small non-zero amount, ϵ, is sufficient to avoid violation of mutex constraints between successive actions. A more sophisticated analysis to determine where this separation is necessary might be possible, but we have not yet explored this further. Then, for each linked pair of start and end actions $\langle A_{start}, A_{end}\rangle$, at steps i and j, a duration constraint is added: $min \leq step_j - step_i \leq max$, for appropriate minimum and maximum duration values for A.

These constraints together form an STN. If the STN is inconsistent the plan cannot be scheduled. By constructing an STN at each state in the search space, CRIKEY3 ensures the action choices are both logically and temporally sound.

To navigate the space of states we have described, CRIKEY3 employs a temporal relaxed planning graph (TRPG) heuristic. The first fact layer in the TRPG corresponds to the state being evaluated. Following this are time-stamped action and fact layers, with action layers containing both start and end actions and fact layers their (positive) effects. The durations of actions are used to offset start and end points between fact layers in order to capture the temporal relationships between the starts and ends of actions. If a start action A_{start} appears at layer t in the planning graph, its end is delayed until layer $t + dur_{min}(A)$. This is the earliest point at which the end could be applied, given that it has to follow the start. For each action that started before the current state but has not yet finished, the TRPG maintains an upper bound on the elapsed time since the start. The corresponding end action is delayed until the layer timestamped with the minimum duration of the action minus that bound. Both logical and temporal constraints are therefore considered within the heuristic and, as discussed in [Coles *et al.*, 2008], the search guidance is effective in domains with required concurrency and deadlines. Unlike the planning graph heuristic used in Sapa [Do and Kambhampati, 2001], the TRPG constructed in this way is able to correctly observe the PDDL start–end semantics: Sapa's planning graph with compiled actions results in false dead-ends. The TRPG also contains bounds estimates

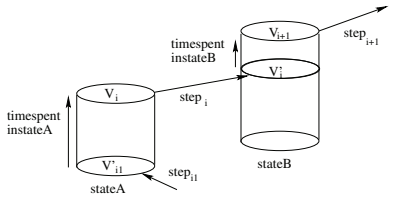

Figure 2: The relationship between numeric variables, actions and time, encoded in the LP.

for numeric variables, maintained and used in the same way as Metric-FF [Hoffmann, 2003], allowing CRIKEY3 to tackle problems with discrete numeric effects. CRIKEY3 does not handle continuous change.

5 Planning with Linear Continuous Change

The key differences between COLIN and CRIKEY3 that allow COLIN to reason with linear continuous change are in the representation of temporal constraints, the rules for state progression and the construction of the TRPG. These elements are detailed in the three following subsections. Briefly, the representation of temporal constraints uses a LP instead of a STN, allowing constraints on linear processes to be combined with temporal constraints. State progression is modified to use the same LP to confirm that processes interact correctly with action preconditions. The TRPG is modified to use the LP to model metric variables subject to continuous change. We now show how the LP is constructed and used.

5.1 LP Action Scheduling

In CRIKEY3 an STN was used to schedule the chosen sequence of start and end actions. With the presence of continuous numeric change, this is no longer adequate. Preconditions can be specified over numeric variables that are subject to continuous change so resolving temporal interactions cannot be separated from reasoning about numeric values. In the case of the generator problem, the solution is to refill the tank *while* the generator is running. If the interaction between time and numeric values is ignored, it seems that the fill action can immediately follow the action to start the generator. In fact, sufficient time must elapse to consume enough fuel for the tank to be refilled without overflowing.

In COLIN the plan is formulated as a linear program containing both the temporal constraints and the numeric constraints imposed by the actions. As well as defining timestamp variables for the start and end points of each action $step_i$, we define a collection of variables, V_i, that denote the values of the numeric state variables at $step_i$. We also define a collection of variables, V_i', denoting the values of the numeric variables immediately following the execution of the action at $step_i$. These sets of variables allow numeric preconditions and effects to be encoded in the LP.

- Start and end preconditions are constraints over V_i that must hold at the point the action is applied.
- An action, A, starting at step i and ending at step j has its invariants added as constraints over each of $V_i'..V_{j-1}'$ and $V_{i+1}..V_j$.

Variable	Constraints
$step_0$	N/A
v_0	=90
v_0'	$= v_0, > 0$
$step_1$	$\geq step_0 + \epsilon$
v_1	$= v_0' - 1.(step_1 - step_0), > 0$
v_1'	$= v_1, > 0, \leq 90$
$step_2$	$= step_1 + 10$
v_2	$= v_1' + 1.(step_2 - step_1), > 0, \leq 90$
v_2'	$= v_2, > 0$
$step_3$	$\geq step_2 + \epsilon$, and $= step_0 + 100$
v_3	$= v_2' - 1.(step_3 - step_2), > 0$
v_3'	$= v_3$

Table 1: Variables and Constraints for the Generator Problem

- Start and end effects on a variable v are written as constraints defining v_i' in terms of variables in V_i. For example, `(increase v (+ w x))` becomes: $v_i' = v_i + w_i + x_i$.

Following PDDL2.1 and PDDL+ semantics, continuous change always occurs within a state, progressing up the time dimension as shown in figure 2. Therefore, continuous change within a state can be represented using constraints that dictate the relationships between variables V_i' and V_{i+1}. The LP is constructed by iterating through the steps of the plan, tracking the accumulated rate of continuous change, δv_i, on each variable v at each step i:

- Initially, for each variable v, $\delta v_0 = 0$: no continuous change is active.
- If step i is the start of an action that changes v with rate m, $\delta v_{i+1} = \delta v_i + m$.
- If step i is the end of an action that changes v with rate m, $\delta v_{i+1} = \delta v_i - m$.
- Otherwise, $\delta v_{i+1} = \delta v_i$.

The values of each $v_{i+1} \in V_{i+1}$ can then be written as:

$$v_{i+1} = v_i' + \delta v_{i+1}.(step_{i+1} - step_i) \quad (1)$$

The resulting LP encodes the changes occurring to numeric values during plan execution, along with the conditions specified over them both at time points and as invariants. By setting the objective function to minimise the timestamp variable of the final step, attempting to solve the LP will either schedule the plan or discover that it is invalid.

The LP for the four step generator plan contains the variables and constraints shown in Table 1. The corresponding sequence of state transitions, coupled with the temporal structure and the timing of the variable values, is shown in figure 3. The constraint that $v > 0$ is added to all relevant points during `generate` and forces `fill_start` to occur sufficiently early. The constraint that $v \leq 90$ forces `fill_start` to occur sufficiently late. The solution assignments to $step_0$ and $step_1$ then give the timestamps of `generate_start` and `fill_start` and hence a schedule for the plan.

5.2 Using an LP to Determine State Validity

We have so far considered the scheduling of a plan containing actions with linear continuous effects. Search for a plan poses an interesting challenge. In a state reached after the execution of some actions, the values of numeric state variables depend on how much time elapses before the next action is applied. For example, once the generator is running, there is no fixed value for the level of fuel in the tank. In effect, unlike the discrete case, the numeric state variables no longer hold fixed values. The significance of this becomes clear when considering which actions to apply to expand a state. In CRIKEY3, numeric and propositional facts are known, and this information is used to determine whether actions are *logically applicable*. The STN is used to determine whether the action is also *temporally applicable*. With the scheduling now being performed using an LP, considering both time and numbers, aspects of numeric applicability are not fully known until attempting to schedule the plan. We can safely assume propositional applicability, but the scheduler now determines *numeric–temporal applicability*. This creates a problem because the test for numeric applicability is delayed until the the LP is checked. Many more actions appear applicable when logical preconditions are tested than will survive the (relatively expensive) numeric–temporal applicability test.

To solve this problem, we extend the state definition used in CRIKEY3 to record an upper and lower bound for each numeric variable in $\bigcup_i (V_i \cup V_i')$. To determine these bounds, we extend the formulation of the LP by the addition of a new collection of variables and constraints.

In the current state, which follows the application of the action at $step_i$, we define a set of variables V_{now} and a timestamp $step_{now}$. This timestamp represents the time in the current state and will become the time at which the action (if any) is applied at $step_{i+1}$. V_{now} is defined in terms of V_i' according to Equation 1 and $step_{now} \geq step_i + \epsilon$. For each action that has started but not yet finished, a maximum duration constraint is added between the start of the action and $step_{now}$: the time elapsed between the start of the action and now necessarily cannot exceed the duration of the action.

Having extended the LP to contain the now variables it can be used to find the upper and lower bounds on each numeric variable in the state. To achieve this, we change the objective function of the LP to maximise and minimise the values of each of the V_{now} variables in turn, in order to compute their upper and lower bounds. For efficiency, if the value of a variable is not continuously changing, its value can be established by applying the numeric effects of the actions in the plan starting at the initial state.

Reasoning about bounds on resources under the influence of linear processes is also condidered by [Frank and Morris, 2007]. They treat the harder problem of computing these bounds in partially ordered STNs but they do not discuss the integration of their solution with a planner. The power of their approach is not needed in COLIN because of the total ordering of the start and end points of the actions.

An LP is solved at every node in the search space, so its solution must be as efficient as possible. Variable bounds in the LP can be improved by storing, in each state, a lower bound on the timestamp of each action in the plan. When a state S is expanded to reach a state S' by applying an action a, the LP scheduler is used with the objective function to minimise the timestamp of a. Assuming the plan can indeed be scheduled, the value of the objective function is a lower bound on the

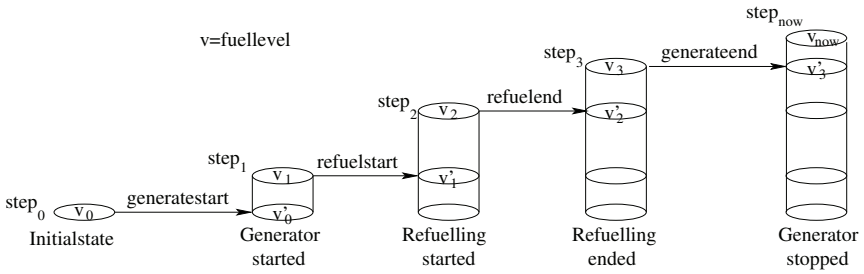

Figure 3: The state transitions and time dimension, showing variable values, for the generator problem

timestamp of a in S' and in all states reachable from S'. If these lower bounds are stored in a state, together with their associated actions, they can be used within the scheduler when evaluating successor states.

It is also useful to exploit the fact that $step_{now}$ will become the time of $step_{i+1}$, so that bounds on V_{now} become bounds on V_{i+1} when S is expanded.

5.3 Extending the Temporal RPG Heuristic

Finally, we describe how the temporal RPG heuristic of CRIKEY3 can be extended to provide guidance in problems with continuous numeric effects. Modifications are made in three places. First, since the current state no longer has fixed values for variables subject to continuous change, we use the bounds for the V_{now} variables as bounds on metric variables in the first layer of the temporal RPG.

The second modification concerns how the linear continuous effects are encoded. In CRIKEY3 change is instantaneous, occurring either at the start or the end of an action. In COLIN we make the optimistic assumption that the continuous effects of an action are available as soon as the action has started. For example, the net effect of the `refill` action is to increase the fuel level by 20. In the extended TRPG the increase occurs as a start effect of the `refill` action. If the start of the action appears in the planning graph, it has this start effect added to it. If an action has already started then the remainder of its effect, not already accounted for in the time that has elapsed since the action started, is used to compute the upper and lower bounds for the variables it affects.

Finally, we recognise certain implicit conditions introduced by continuous resource consumption and attach these to the end points of the relevant actions. Suppose an action, A, consumes a resource, v, during its execution and therefore requires $v > 0$ throughout. If the duration of A is d and the rate of consumption of v is m, then by the end of A the amount of v produced must be at least $d.m - ub(v)$, where $ub(v)$ is the initial upper bound on the level of v. This condition is added as an end precondition of A. When an action that consumes a resource has already started, we make the optimistic assumption that as much of its consumption as possible has been satisfied before the start of the current TRPG construction. This is implemented by adding a constraint to the end of the consuming action, A, that $v \geq (d - exec(A)).m - ub(v)$, where $exec(A)$ is the maximum time since the start of A.

6 Evaluation

COLIN can handle two features of PDDL that extend the state-of-the-art: general linear duration-dependent effects and the continuous linear change expressed through `#t`. Although other planners can solve some problems containing duration-dependent effects they are restricted to specific cases, as discussed below. Because COLIN is based on CRIKEY3 it can manage required concurrency, including concurrent actions with continuous effects. This is beyond the capabilities of other PDDL planners that are available for testing.

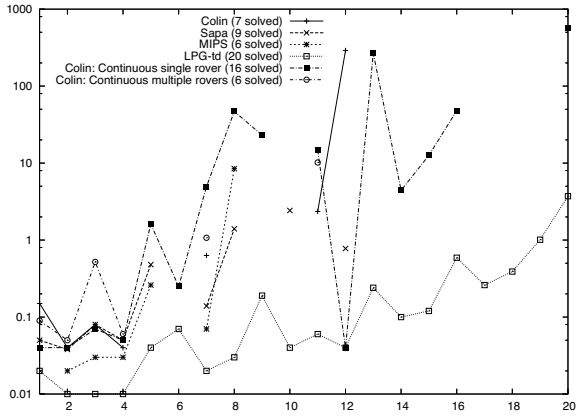

Figure 4: Time (in seconds) taken by several planners to solve Rovers Time problems and by COLIN to solve single and multiple rovers variants of Continuous Rovers problems

We first show that COLIN is competitive with other state-of-the-art planners to show that the cost of sophisticated temporal-numeric reasoning in COLIN is acceptable. We then demonstrate the performance of COLIN on domains with continuous numeric change and duration-dependent effects.

We compared the performances of COLIN and CRIKEY3 on a range of temporal problems to determine the overhead paid by COLIN. We ran both planners on a collection of 115 benchmark problems taken from Airport, Mine, Satellite, Pipes and Driverlog. CRIKEY3 solved 86 problems and COLIN solved 91, while 84 of the problems were solved by both planners. CRIKEY3 took a total of 4424.35 seconds to solve these 84 problems and COLIN took a total of 4857.32 seconds. At worst, this implies that COLIN pays a 10% overhead, but closer inspection of the data reveals that most of the overhead is paid in a small subset of the problems.

Figure 4 shows the performance of COLIN on the IPC3 Rovers Time domain and problem set, compared with other planners that can solve these problems. In this problem set the recharge action has a duration that is constrained by the remaining capacity of the rover battery and its recharge rate. During the recharge action power can also be consumed, so

the effect of recharging is determined by its duration, rather than simply by the capacity of the battery. MIPS, Sapa and LPG-td can all solve problems in this domain even though none of them can reason about continuous time. All of these planners use the compressing compilation which explicitly prevents them from reasoning about concurrent continuous activity. LPG-td can only cope with expressions of the form `(* ?duration rate)`. Sapa and MIPS [Edelkamp, 2003] can deal with cases where the time-points of interest can be pre-computed (ie: the durations of the actions can be computed from the context at the time of application) but cannot deal with cases where the time-points must be selected from within ranges to satisfy dynamic constraints.

The figure also shows the performance of COLIN on a modified version of the Rovers Time domain in which recharging and discharging due to navigation are continuous effects and an additional continuous recharge action is available whilst the rover is navigating. We used the IPC3 problem set (the multiple rovers curve in the figure) and a second problem set where only a single rover is provided (the single rover curve). We also tested COLIN on a continuous version of the IPC3 satellite domain in which satellites have a power level that is subject to continuous change throughout the day, and operations consume power. The result is that the satellite can only perform certain tasks at certain times and sometimes must wait for sufficient power to be available. The satellite results are not presented here for space reasons.

In the Airplane Landing problem the objective is to land a collection of airplanes as close as possible to a target time, while minimising the early and late-landing penalties that decrease and increase, respectively, according to linear functions. Certain logical constraints must also be respected (the runway must be cleared between planes and landings must be scheduled before they occur). This problem uses variable length durative-actions, duration-dependent effects and requires concurrency. No other planner that we know of can solve this problem. COLIN produces the following plan to land 3 planes on 1 runway, given that it takes 40 time units to clear the runway between landings.

```
0.000: (releaseplanes)          [0.020]
0.001: (land-early kl101 main) [129.000]
0.018: (land-early kl108 main) [208.986]
0.019: (land-late kl115 main)  [288.988]
0.021: (schedule kl101 main)   [40.000]
129.002: (clear kl101 main)    [40.000]
169.003: (schedule kl108 main) [40.000]
209.005: (clear kl108 main)    [40.000]
249.006: (schedule kl115 main) [40.000]
```

It can be observed that the plan contains considerable parallel activity and, although not optimal, is of good quality. VAL [Howey and Long, 2003] reports the plan as incurring a penalty of 7270.22 while a good hand-built plan incurs 4940.3. A trivial plan, landing the planes in the order of their target landing times, costs 9450.26. The details of the domain and problem instance are available from the authors.

Our final set of experiments examine whether repeated solution of an LP, to schedule a plan at each node during search, is practical. We took the largest plan obtained in our test (142 start and end steps) and scheduled increasing length action sequences, obtaining action timestamps and updated variable bounds in the state reached at the end of the sequence. This provides a measure of the overheads incurred by using the LP during search. We also consider the case where the additional variable bounds discussed in Section 5.2 are not used. The data we obtained (not shown for lack of space) shows that the LP costs increase polynomially to a maximum of 0.05 seconds while the additional variable bounds reduce the time taken by an increasing margin (from 10% to about 20%).

7 Conclusions

Planning with mixed discrete-continuous numeric change is a challenging problem. Previous approaches have been largely based on an underlying temporal model that uses the compressing compilation of durative actions, severely compromising the range of problems that can be tackled.

We have described a new approach, built on CRIKEY3, a temporal planner that can solve problems that require concurrency. We have shown how the treatment of discrete numeric fluents in CRIKEY3 can be replaced with a more capable solution, using linear programming. This solution is sufficiently powerful to support duration-dependent effects, both in actions with context-dependent durations and in actions with bounded durations, and also continuous linear change within durative actions. We have shown how the TRPG heuristic used in CRIKEY3 can be extended to guide search in the context of duration-dependent and continuous change.

References

[Coles *et al.*, 2008] A. I. Coles, M. Fox, D. Long, and A. J. Smith. Planning with problems requiring temporal coordination. In *Proc. 23rd Nat. Conf. on AI (AAAI)*, 2008.

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

[Dierks, 2005] H. Dierks. Finding optimal plans for domains with restricted continuous effects with uppaal-cora. In *ICAPS Workshop on Verification and Validation of Model-Based Planning and Scheduling Systems*, 2005.

[Do and Kambhampati, 2001] M. Binh Do and S. Kambhampati. Sapa: a domain-independent heuristic metric temporal planner. In *Proc. European Conf. on Planning (ECP'01)*, 2001.

[Edelkamp, 2003] S. Edelkamp. Taming numbers and durations in a model-checking integrated planning system. *J. Art. Int. Research*, 20:195–238, 2003.

[Fox and Long, 2003] M. Fox and D. Long. PDDL2.1: An Extension of PDDL for Expressing Temporal Planning Domains. *J. Art. Int. Research*, 20:61–124, 2003.

[Fox and Long, 2006] M. Fox and D. Long. Modelling mixed discrete-continuous domains for planning. *J. Art. Int. Research*, 27:235–297, 2006.

[Frank and Morris, 2007] J. Frank and P.H. Morris. Bounding the resource availability of activities with linear resource impact. In *Proc. Int. Conf. on AI Planning and Scheduling, ICAPS*, 2007.

[Garrido *et al.*, 2001] A. Garrido, E. Onaindia, and F. Barber. A temporal planning system for time-optimal planning. In *Progress in AI*, volume 2258 of *LNCS*, 2001.

[Gerevini *et al.*, 2006] A. Gerevini, A. Saetti, and I. Serina. An approach to temporal planning and scheduling in domains with predictable exogenous events. *J. Art. Int. Research*, 25:187–231, 2006.

[Hoffmann, 2003] J. Hoffmann. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *J. Art. Int. Research*, 20:291–341, 2003.

[Howey and Long, 2003] R. Howey and D. Long. VAL's Progress: The Automatic Validation Tool for PDDL2.1 used in the International Planning Competition. In *ICAPS 2003 workshop: 'The Competition: Impact, Organization, Evaluation, Benchmarks'*, pages 28–37, 2003.

[Léauté and Williams, 2005] T. Léauté and B.C. Williams. Coordinating Agile Systems through the Model-based Execution of Temporal Plans. In *Proc. 20th Nat. Conf. on AI (AAAI)*, 2005.

[Li and Williams, 2008] H. Li and B.C. Williams. Generative systems for hybrid planning based on flow tubes. In *Proc. 18th Int. Conf. on Aut. Planning and Scheduling (ICAPS)*, 2008.

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

[McDermott, 2003] D. McDermott. Reasoning about Autonomous Processes in an Estimated Regression Planner. In *Proc. 13th Int. Conf. on Aut. Planning and Scheduling (ICAPS)*, 2003.

[Penberthy and Weld, 1994] S. Penberthy and D. Weld. Temporal Planning with Continuous Change. In *Proc. 12th Nat. Conf. on AI (AAAI)*, pages 1010–1015. AAAI/MIT Press, 1994.

[Shin and Davis, 2005] Ji-Ae Shin and E. Davis. Processes and Continuous Change in a SAT-based Planner. *Artificial Intelligence*, 166:194–253, 2005.

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