Planning with Linear Continuous Numeric Change

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

Reasoning with domains containing continuous numeric change remains an open challenge in planning. In this paper, we discuss how an established temporal planner, CRIKEY3, can be adapted to plan in domains containing linear continuous numeric change. In particular, we discuss how an LP can be used to scheudule action choices, and how the LPRPG heuristic (hitherto for non-temporal planning) can be adapted for use in a temporal and linear-continuous setting.

1 Introduction

In this paper we consider the problem of planning in domains with continuous linear numeric change across the execution of actions. Such change cannot always be adequately modelled under the assumption that change is instantaneous, and is a key facet of many interesting problems. For example, the battery charge in a Martian rover is constantly fluctuating, and activity needs to be planned in the context of this. Similarly, customer demand in electricity substation modelling varies throughout the day, and the discrete switching actions needed to satisfy supply constraints depend on this.

In this paper, we present an approach whereby a forward chaining temporal planner can be extended to reason with actions with continuous linear numeric change. The work we present is currently being implemented, so our focus in this paper is on how existing components can be adapted to suit our purposes. We make three contributions. First, we consider a recent heuristic for planning with numbers in non-temporal domains, and extend this to temporal domains. Second, we propose how a Linear Programme (LP) can be used to support reasoning with time and numbers during search within continuous numeric domains. Third, and finally, we discuss how the temporal-numeric heuristic we present can then be used within the continuous numeric search structure we propose, providing the search guidance necessary to underpin planning.

2 The Problem

Conventional STRIPS planning problems can be thought of as a tuple <A,I,G>, where A is the set of actions that can be applied, I is a propositional representation of the initial state, and G is a propositional representation of the goal state. Each action A, has preconditions, propositions that must be true in order for the action to be applied, and effects, facts that become true, or false, on the application of that action. The task is to find a *plan* a sequence of actions that transformes the initial state I into the goal state G.

The introduction of PDDL2.1 in 2001 (Fox & Long 2003) saw the extension of the language to deal with numeric fluents and temporal actions. PDDL2.1 is split into several *levels*. Level 1 is restricted to propositional models. Level 2 introduces numeric fluents, to capture reasoning about numeric values in the world. The use of numeric fluents is restricted to basic numeric operations. Fluents can either be assigned a specific value (either in the initial state or by an action); increased or decreased by a value; or modified according to an arithmetic expression making use of the basic operators $+, -, *$ and $\div$. The important thing to note, at level 2, is that all numeric change is instantaneous, happening at the time an action is applied: it cannot occur gradually over time. This limitation means that many important interesting features of real world problems cannot be modelled.

Level 3 of PDDL2.1 defines temporal planning problems, with the concept of *durative actions*. Durative actions are a development of the non-temporal actions of levels 1 and 2, defined to accommodate a model of time where the start and end of actions are of interest. Problems here can still be thought of as a tuple <A,I,G>, but each $a \in A$ is now defined differently. Rather than preconditions, durative actions have conditions; these can be specified to hold either at the start, the end, or across the action's execution (referred to as *invariants*). Effects are similarly modified, and can fall either at the start of the execution of an action, or upon its completion. Finally, durative actions have a duration constraint specified: a upper and lower bound on the action's duration, equal in the case of fixed-duration actions. The language to define these bounds is similar to that of numeric preconditions: each is dictated by a basic arithmetic expression.

Level 3 of PDDL2.1 also retains support for the numeric capabilities of level 2, giving rise to some interesting new challenges in planning: how do time and numbers interact? As a starting point, in level 3, numeric effects and conditions can occur wherever propositional effects and conditions occur, with numeric change being instantaneous. This is the level of PDDL2.1 that state-of-the-art planners, such as LPG (Gerevini & Serina 2002) and Sapa (Do & Kambhampati

2001) can reason with; some include support for other orthogonal features such as timed initial literals (Hoffmann & Edelkamp 2005), but the basic numeric temporal reasoning is of this form.

Level 4 of PDDL2.1 introduces a richer interaction between time and numbers: continuous numeric change. That is, rather than numeric change being discrete, values can change continuously over time, throughout the execution of an action, according to some specified numeric function. Such effects are specified through use of a differential equation, with an effect on a variable v given as $\frac{dv}{dt} = f(V) + c$, where V is the vector of state numeric variables, $f(V)$ is a mathematical function over these, and c is a constant. Solving problems with this level of expressivity remains an open challenge to the planning community.

In this paper, we present ideas to solve a subset of such problems. We restrict our attention to the case where $f(V)$ refers only to static numeric state variables; i.e. those unchanged by any action. In this case, $f(V)$ is a constant state-independent value, and hence the continuous numeric change $\frac{dv}{dt}$ is linear. Let us now take a look at an example of such an action, in a domain similar to the Generator domain (Howey & Long 2003) (the important simplification is that numeric change is linear). In this domain, a generator must run for a fixed length of time (in our case 100 seconds), and as it powers a safety critical system it is essential that there are no breaks in its running. Running the generator consumes fuel from a tank; it is possible to refill the tank whilst running the generator, but the total capacity of the fuel tank on the generator, 90 units, must not be exceeded. The action to refill the tank increases the level of fuel in the tank at a rate of 2 units per second for a duration of 10 seconds.

The action to run the generator, in PDDL2.1 level 4, is given in Figure 1. The condition on executing this action is the safety criterion: 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 PDDL, continous numeric change is specified as the rate of change of a fluent, in this case `fuel-level`, with respect to time: the #t is a special keyword denoting the time elapsed so far. Thus this effect states that the amount of fuel consumed t time units after the action has been executed is $t * 1$, or t. This represents a linear decrease in fuel level over time.

Clearly reasoning with continuous resources requires proper reasoning about numeric resources and about time. We will explore approaches to reasoning with each of these, at PDDL2.1 levels 2 and 3 respectively, before going on to present a combination of the two approaches that allows the management of continuous numeric effects where change with respect to time is linear.

3 Related Work

There is a growing body of work in the area of planning with continuous systems, also referred to as hybrid or mixed discrete-continuous systems. Within the planning community, there are several researchers who have worked in this area. Work includes the exploration of control in hybrid systems by Brian Williams and his colleagues (Léauté & Williams 2005; Li & Williams 2008). This work has focussed on model-based control of hybrid systems 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 (typically 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.

A different, although related, approach to reasoning with linear continuous dynamics is the work of Shin and Davis (Shin & Davis 2005) in TM-LPSAT, which uses a SAT model of a problem, in which linear constraints on the continuous dynamics are captured as logical variables. Once a satisfying assignment for the discrete combinatorial problem has been found, the active variables corresponding to linear constraints are harvested and the constraints assembled into a linear programme which is solved separately. If it cannot be solved, the system backtracks over the satisfying assignments. A problem with this approach is to gain adequate information from the linear programme to support useful backtracking in the SAT problem. A similar approach was explored in (Audemard *et al.* 2002), though this was not concerned with continuous change. An approach based on heuristic search is presented by McDermott in his OPTOP system (McDermott 2003). Earlier work on continuous planning was conducted by Penberthy and Weld in Zeno (Penberthy & Weld 1994), a planner that reasoned directly with the analytic forms of the differential equations describing continuous dynamics. Representation of hybrid systems in a standard planning formalism is considered in the work of Fox and Long (Fox & Long 2006), while earlier work such as (Sandewall 1994) considers the broader issues of representation of hybrid systems for inference systems.

Planning in domains with continuous processes is often coupled with management of uncertainty, since the dynamics of systems are typically hard to specify with precision. This area of research has been investigated by several researchers (Younes & Simmons 2004; Benazera *et al.* 2005).

Of course, considerable work on reasoning with hybrid systems has been carried out in other fields, including verification and real-time systems. Although much of this work is relevant, there is far too much to give a representative overview in this paper. Instead, we point the reader to the work on timed automata theory (Lynch & Vaandrager 1992) as a good starting point.

4 The LPRPG Heuristic

Since the introduction of level 2 of PDDL2.1 in 2001, the use of numeric values in planning has recieved comparatively little attention. Many planners can reason with domains including such values, but few can optimise with respect to these values, or indeed solve problems where there numeric resources are key in finding solutions to the problems. Most state-of-the-art numeric planners still use the metric relaxed planning graph (metric RPG) heuristic introduced with Metric-FF (Hoffmann 2003). The metric RPG

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

Figure 1: Generate Action from the Generator Domain

heuristic extends the 'ignore delete lists' relaxation to numbers, by ignoring 'delete' numeric effects. To achieve this, each fact layer is extended to record the lower and upper bounds on the value of each numeric variables. Initially, these correspond to the values of the variables in the state being evaluated, and as the graph is expanded, numeric effects which increase a variable are used to increase its upper bound, and effects which decrease a variable decrease its lower bound.

The metric RPG heuristic, whilst capable of providing some search guidance in numeric planning problems, does however have a number of weaknesses. One in particular is particularly detrimental to heuristic guidance: the ability to cyclically transfer numeric resources. In this case, the upper bound on a variable v can be increased by executing a pair of actions in sequence: one which increases the value of v' at the expense of v, and one which increases v at the expense of v'. As the unwanted effects are ignored, the presence of this pair of actions within the planning graph allows the upper bound on v to increase through what would be a fruitless action sequence were all the effects of the actions considered.

Recent work has addressed this issue through combining the metric RPG with a linear programme (LP), producing what is known as the LPRPG heuristic. For full details, we refer the reader to (Coles *et al.* 2008b), but we shall discuss it briefly here. The heuristic is designed for producer–consumer problems, where resource variables have a lower bound of zero, and actions increase or decrease one or more resource levels by prescribed amounts. In such a setting, an LP can be made with three classes of variables. First, those denoting the numeric variable values in the first fact layer (fixed to correspond to the state being evaluated), one for each variable v. Second, variables denoting how many times each of the n actions present in the planning graph has been applied, each denoted a_i. Third, variables denoting the values in the current fact layer, denoted v'. Constraints are used to dictate the values of this third class of variables from the first two. If $change(a_i, v)$ gives the effect of an action a_i upon a variable v (0 if no change, positive if a production action, and negative if consumption) then the value of each v' is given by:

$$v' = v + \sum_{i=[1..n]} change(a_i, v)$$

The LP can then be used to find numeric variable bounds for the current fact layer, setting the objective function to first maximise and then minimise each variable v'. As before, these then determine which actions are applicable in the next action layer. As the constraints giving the values of each v' consider both the decreasing and increasing effects, and resource variables are bounded to be non-negative, the heuristic is able to avoid cyclical resource transfer and provide better heuristic guidance in strongly numeric domains.

5 CRIKEY3

In handling continuous numeric effects it is crucial to reason properly with time. Three aspects govern the model of temporal actions according to PDDL. The first is the notion of start and end points: conditions and effects can be specified to hold and, respectively, occur at both the start and end of each action. Second, an invariant condition can be specified to hold between these, written as propositional and/or numeric 'over all' conditions. Third, and finally, a duration inequality for the action is specified: upper and lower bounds on the duration of the action, expressed as formulæ over numeric constants and state variables.

From this model of temporal actions, many state-of-the-art planners reduce the problem to non-temporal planning through 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 Temporal Graph Plan (TGP) (Smith & 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 (Coles *et al.* 2008a). Proper reasoning becomes even more important when numeric fluents may also change continuously throughout the duration of an action.

Let us return to our running example, considering the simplified generator problem. It is clear that in a solution to this 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 starts. If the `refill` action is not applied until the execution of the `generate` action is complete then the generator will run out of fuel before completing generation. It is possible to envisage this situation happening in the TGP propositional case also, where a resource is modelled propositionally as being present-or-not. For example, consider a `use_torch` action with the start effect that there is light, and an end effect to delete this proposition. Such an action models the presence of a resource, in this case light, and any action requiring this resource to be present must be executed concurrently with the resource providing action (assuming no other action adds the proposition).

In problems such as the generator problem, or the propositional torch problem, the compilation approach is not an effective means of solving the problem. In the generator case the two actions will necessarily have to be sequenced: each is instantaneous and mutex with the complied version of the other. In the torch domain, again, if the use torch action is instantaneous it is not possible for any other actions making use of the light to run in parallel with it, thus if any activity requiring light is necessary in order to reach the goal the compiled problem will be unsolveable. This clearly motivates the need for a different approach taking proper note of the temporal structure of actions: the points between the starts and ends of actions are important, and cannot be compiled away.

CRIKEY3 (Coles *et al.* 2008c) is a forward chaining temporal planner that is able to reason with problems where such coordination between actions is necessary in order to find a solution. It can handle richly temporally expressive domains specified in PDDL2.1 and is the temporal planner upon which we will build the continuous planning framework. Let us explore, then, how CRIKEY3 works. The first point to observe is that CRIKEY3 does not use the TGP action compliation; instead it uses a compilation first introduced in LPGP (Long & Fox 2003). This compilation is lossless, and instead of compiling a temporal action into a single non-temporal action, it is compiled into two separate instantaneous actions: one representing the start of the action, and another then end. To ensure a one-to-one correspondance between starts and ends of actions, additional dummy facts are used. Using this compilation the starts and ends of actions can be interleaved by the planner during search. This does make search to find a plan less efficient than if the TGP compilation is used, in cases where such a compilation is sufficient, but it remains sound and complete. If we now consider the generator domain again, it is possible to see that a solution plan does conceivably exist:

```
0:  generate_start
1:  fill_start
2:  fill_end
3:  generate_end
```

To ensure search with these start and end actions remains sound, two other considerations must be made. First, starting an action establishes invariants which must hold until the action is ended. To capture these, the invariants of any actions which have started but not yet finished are recorded in the state definition. Then, when considering which actions are logically applicable (i.e. their preconditions are satisfied), no action can be applied which violates any of these active invariants.

Second, when searching for a solution plan, we must also consider the temporal structure of the domain; i.e the duration constraints between the start and end points of actions. Temporal constraints arise from two sources: adjacent 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 $step_1..stepi$. Each of the two sources of temporal constraints are then appropriately encoded. First:

$$\epsilon \leq step_{i+1} - step_i$$

That is, each step must occur a small amount of time (ϵ) after the previous one. Separation by ϵ is necessary as if the action at $i + 1$ requires an effect of i as a precondition, the two steps cannot occur strictly concurrently: they must be nominally separated. Second, to capture action durations, for each pair of start and end actions $\langle A_{start}, A_{end} \rangle$ at steps i and j a duration constraint is added:

$$min \leq step_j - step_i \leq max$$

With this STN, it is then possible to ensure that a valid schedule exists for the action steps chosen: if the STN is inconsistent (i.e. negative cost cycles exist within the induced directed graph) the plan is invalid. Hence, by constructing an STN at each state in the search space, CRIKEY3 ensures the action choices are both logically and temporally sound.

Through using the LPGP action compilation, performing the necessary housekeeping to maintain invariants, and using an STN to check temporal feasibility, we have discussed how forward chaining planning can be used by CRIKEY3 to search for solutions to temporal planning problems. Such search requires appropriate heuristic guidance, and to this end CRIKEY3 employs a temporal relaxed planning graph (TRPG) heuristic. Unlike the planning graph heuristic used in Sapa (Do & Kambhampati 2001), the TRPG constructed in this manner is able to maintain the PDDL start–end semantics: Sapa's planning graph with compiled actions results in false dead-ends. The first fact layer in the TRPG corresponds to the state being evaluated. Following this, are timestamped action and fact layers, with action layers containing start and end actions, and fact layers their (positive) effects. To capture the temporal relationships between the starts and ends of actions, the durations are used to offset start and end points between fact layers. For a start action A_{start} in the planning graph, appearing at layer t, its end is delayed until layer $t + dur_{min}(A)$; i.e. the earliest point at which the end could be applied, given it has to follow the start. For actions that have started but not yet finished in the current state, their ends are delayed until the layer timestamped with the minimum duration of the action, minus an upper bound on the time since the start of the action. In this manner, both logical and temporal constraints are encoded within the heuristic, and as discussed in (Coles *et al.* 2008c) the search guidance is effective in domains with action coordination and deadlines.

6 A Temporal LPRPG Heuristic

In the non-temporal LPRPG heuristic, discussed in Section 4 the LP is updated and calls made to determine variable bounds at each fact layer, and then used again during solution extraction. In non-temporal planning, consulting the LP during graph construction is practical as all action durations are equal (i.e. nominal) and hence the number of layers is comparatively small. In temporal planning, however, the number of layers in the planning graph tends to be

considerably larger: action durations are not necessarily tidy multiples of each other.

To trade-off heuristic cost against heuristic quality, a temporal variant of the LPRPG heuristic can be constructed by changing the circumstances in which the LP is used. We can exploit the fact that the bounds computed by the technique used in Metric-FF are generous estimates of those computed by the LP, and delay the use of the LP until the point at which these bounds indicate the numeric goals can be satisfied. At this point, we construct an LP containing all the actions in the final action layer (each being applicable arbitrarily many times), with the additional constraint that the number of times the end of an action is applied is equal to the sum of the number of unfinished instances of that action executing in the current state and the number of times its start has been chosen in the planning graph. With this LP, we then calculate the variable bounds on v' and determine whether the numeric goals are satisfied according to these. If not, a loop is entered:

1. The LP numeric variable bounds are compared to the numeric goals to identify the variables for which insufficient consumption/production is present in the planning graph.
2. Planning graph expansion then continues until at least one new action appears in the planning graph which has the necessary effect (consumption/production) on each goal variable; at this point, the LP is updated to find new bounds, and the loop returns to 1. Alternatively, if no new actions appear before returning to 1, the state can be said to be a dead-end.

In this manner, we lose the improved information on when numeric variables are able to hold given values, as provided by the non-temporal LPRPG heuristic, but have reduced considerably the number of calls to the LP. Solution extraction proceeds as in the non-temporal case, using the LP to find action choices to satisfy numeric preconditions and goals. One of two scenarii arises when using the LP to satisfy a numeric precondition:

1. The LP indicates which actions to use, in which case, these are added to the relaxed plan as in the non-temporal case.
2. The LP is unsolvable — we have only ensured that numeric top-level goals can be satisfied before beginning solution extraction. In this case, we add the actions that would be chosen by the original metric RPG heuristic.

7 Planning with Linear Continuous Change

In this section, we shall now describe how we extend the architecture of CRIKEY3 to reason with continuous linear numeric change. We will begin, first, by setting out how plans are scheduled using an LP. This will be followed by details of how the LP can be used to give a state definition for use during search, and finally how the LPRPG heuristic can be used in conjunction with this extended state definition.

7.1 LP Action Scheduling

Of the continuous numeric change formulæ that can be specified in PDDL, the subset supported is that where the rate of change of a variable is a constant (either a domain-specified numeric constant, or a static numeric state variable). Conceptually, we augment the definition of temporal actions given in Section 5 by allowing the specification of the gradient of the continuous numeric change effected across the execution of the action. Whilst in CRIKEY3 an STN was adequate to schedule the chosen sequence of start and end actions, with the presence of continuous numeric change this is no longer adequate. As preconditions can be specified over numeric variables subject to continuous change, resolving temporal interaction cannot be separated from reasoning about numeric values. By way of example, let us return to our generator example. Intuitively, the solution is simple: run the generator for 100 seconds; within this time, refill the tank. Disregarding the interaction between time and numeric values, the fill action can come arbitrarily soon after starting the generator. Clearly, this is infeasible, as sufficient time has to have elapsed to allow the tank to be refilled without overflowing.

To allow plans for such problems to be scheduled, the plan is formulated as a linear program (LP), containing both the temporal constraints previously encoded in the STN as well as details of the action choices' numeric behaviour. The variables in the STN in CRIKEY3 represent the timestamps of each start or end action in the plan, and are restricted through sequence and duration constraints. In our LP representation, as well as having a timestamp variable for each start or end action ($step_i$), with associated temporal constraints, we have variables denoting the values of the numeric state variables at each of these points ($V_i = \{v_i \text{ for each } v\}$), and immediately following their execution (V_i'). This allows numeric preconditions and effects to be encoded within the LP. First, considering instantaneous numeric change:

- Start and end preconditions are written as constraints over V_i — they 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 $V_{i+1}..V_j$.
- Start and end instantaneous effects on a variable v are written as constraints to determine v_i' from V_i; for instance, `(increase v (+ w x))` becomes:

$$v_i' = v_i + w_i + x_i$$

What remains is to detail how numeric change occuring in the temporal gap between actions steps is encoded; i.e. the constraints dictating the relationship between V_i' and V_{i+1}. The LP is constructed by iterating through the steps of the plan, tracking the active continuous change on each variable at each step:

- Initially, for each variable v, δv_0: no continuous change is active.
- If step i is the start of an action changes v with rate m, $\delta v_{i+1} = \delta v_i + m$.
- If step i is the end of an action which changed v with rate m, $\delta v_{i+1} = \delta v_i - m$.

Variable	Constraints
$step_0$	N/A
v_0	=90
v'_0	$= v_0, \geq 0$
$step_1$	$\geq step_0 + \epsilon$
v_1	$= v'_0 - 1.(step_1 - step_0), \geq 0$
v'_1	$= v_1, \geq 0, \leq 90$
$step_2$	$= step_1 + 10$
v_2	$= v'_1 + 1.(step_2 - step_1), \geq 0, \leq 90$
v'_2	$= v_2, \geq 0$
$step_3$	$\geq step_2 + \epsilon$, and $= step_0 + 100$
v_3	$= v'_2 - 1.(step_3 - step_2), \geq 0$
v'_3	$= v_3$

Table 1: Variables and Constraints for the Linear Generator Problem

- 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 then encodes the changes occuring 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 yield one of two results: either a solution, and hence a schedule for the plan, or no solution, and thus the fact that the plan is invalid.

Returning to our example, the plan consists of four steps:

```
0:  generate_start
1:  fill_start
2:  fill_end
3:  generate_end
```

The LP representing this plan contains the variables and constraints shown in Table 1. The constraint that $v \geq 0$ is added to all relevant points during `generate` and forces `fill` to occur sufficiently early; and the constraint of $v \leq 90$ forces `fill` to occur sufficiently late. The solution assignments to $step_0$ and $step_1$ then give the timestamps of `generate` and `fill` and hence a schedule for the plan.

7.2 Extended State Definition

Discussion of the extensions to CRIKEY3 has thus far considered the scheduling of a plan containing actions with linear continuous effects. Searching for such a plan in a forward-chaining manner poses an interesting challenge: in a state reached after the execution of some actions, the values of numeric state variables depends on how much time elapses before the next action is applied. For instance, once the generator is running, there is no fixed value for the level of fuel in the tank; in effect, unlike the non-continuous case, the numeric state variables no longer hold fixed values. The significance of this becomes clear when considering the decision of which actions to apply when expanding a state. In CRIKEY3, numeric and propositional facts are known, and this information is used to determine whether actions are *logically applicable* based on state information — their preconditions are satisfied, and no invariants are violated by their effects. Whether the action is actually also *temporally applicable* is then determined by attempting to schedule the actions through the use of an STN. With the scheduling now being performed using an LP, considering both time and numbers, aspects of numeric applicability are not fully know until attempting to schedule the plan: we can safely assume propositional applicability, but the scheduler now determines *numeric–temporal applicability*.

One possible solution to this issue is to consider only propositional facts when considering which actions are applicable in a state, and when calculating its heuristic value. In this case, handling numeric values and preconditions is entirely delegated to the LP. Doing so, however, has two key weaknesses. First, many actions will be considered potentially applicable — any whose propositional preconditions are satisfied. As such, the cost of state expansion will be increased due to attempting to schedule each of these using an LP to find the subset of the reachable states which appears to be feasible. Second, any purely propositional heuristic used is consequently restricted to considering only propositional facts, effectively reducing attempts to find a successful chain of interacting numeric actions to blind search.

As a compromise, we extend the state definition used in CRIKEY3 such that rather than recording a fixed value for each numeric variable, we maintain an upper and lower bound. To determine these, we make an extension to the formulation of the LP: following a sequence of actions $1..i$, we add a further dummy action to denote 'now', with timestamp variable $step_{now}$ and numeric value variables V_{now}. As before, V_{now} is determined from V'_i according to Equation 1 and $step_{now} \geq step_i + \epsilon$. Additionally, for each action that has started but have 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. The rationale behind this is as follows: for each unfinished action A_{start}, now either represents the end of A (i.e. A_{end}), or it does not. Supposing $now = A_{end}$, then we need appropriate maximum and/or minimum duration constraints between A_{start} and now. Supposing $now \neq A_{end}$, then we know that A_{end} must come after now — at the earliest, at time $step_{now} + \epsilon$. In this case, the time elapsed between A_{start} and now cannot exceed the maximum duration of A, as necessarily the maximum duration constraint between A_{start} and A_{end} would then be broken. As in both of these cases the maximum duration constraint between A_{start} and now must be respected, it can be added to the LP.

Having extended the LP to contain the necessary dummy action to represent the state reached by the specified plan, it can now be used to find the upper and lower bounds on each numeric variable in the state by changing the objective function of the LP. Specifically, whereas previously the objective function was to minimise the timestamp of the last action, changing it to maximise and minimise each of V_{now} in turn will find the respective upper and lower bounds. For reasons of efficiency, if the value of a variable can be shown to be time-independent in the plan so far, its value can be estab-

lished as before; i.e. through applying the actions' numeric effects forwards from the initial state.

7.3 Improving LP Efficiency

In general, tightening the variable bounds within an LP reduces the time taken to find a solution. As the LP scheduler will be used once per node in the search space, any variable bounding we can make will help to reduce the overheads this incurs. One possibility is to extend the state definition further to store information for use as variable bounds when constructing the LPs to evaluate successor states. The trade-off, however, is that storing bounds requires additional memory. Hence, we shall discuss two means of improving LP bounds, one with minimal memory overheads and the other with none.

The first method we consider is to store, 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' through applying an action a, the LP scheduler is used, first with the objective function to minimise the timestamp of a. Assuming the plan can indeed be scheduled, then the value of the objective function is a lower bound on the timestamp of a in S' and in all states reachable forwards from S': there is no way in which applying further actions to S' can force this action to occur any earlier than this proven lower bound. By storing these objective function values in a state, alongside their respective planned actions, the bounds can then be used within the scheduler when evaluating successor states; i.e. the timestamp lower bounds from S can be used in the LP scheduler for S'.

The second method for adding further bounds to the LP exploits the fact that when expanding a state S to lead to S' we know the upper and lower bounds on the state variables in S. For a state S' reached by a plan of actions $1..i$, the state S corresponds to that immediately before action i. In effect, the dummy action for 'now' discussed in Section 7.2 is action i — that following the actions $1..(i-1)$ that lead to S. As the variables V_i correspond to those immediately before action i, the upper and lower bounds on the variables in S can then be applied to the respective variables in V_i. In this manner, we can add further bounds without having to expand the state definition further: we exploit the fact that we have both the previous state S and the new state S' available when evaluating S'.

7.4 Using the Temporal LPRPG Heuristic

Finally, we consider how the temporal LPRPG heuristic can be extended to provide guidance in problems with continuous numeric effects. Modifications are made in three places. First, as we no longer have a state with fixed values assigned to numeric variables, the upper and lower bounds from the state are taken to be the upper and lower bound of the first fact layer numeric variables. As such, whereas previously the values for the LP variables denoting the current state numeric variable levels were fixed, they are now free to take any value within the appropriate range. In doing so, the planning graph structure can be built as before, with no changes to the algorithm.

The second modification concerns how the linear continuous effects are encoded: change previously was instantaneous, either at the start or the end. To achieve this, we make the optimistic assumption that the continuous effects of an action are available as soon as it has started. For instance, the net effect of the `refill` action is to increase the fuel level by 20, so in the heuristic we assume this increase occurs as a start effect of the 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, its amortised effects are applied to update the upper and lower bounds in the first action layer, giving a representation of the state that one would have if the earlier start of the action had this effect.

Third, and finally, we recognise implicit conditions introduced by continuous resource consumption, and attach these to the end points of such actions. Consider the case where an action A consumes resource v across its execution, and hence requires v to be non-zero at all times. If its duration is d and the rate of consumption of v is m, then by the end of the action 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 can be added as an end precondition of A, i.e. that before A can finish, production of v must be at least what is needed by A. As a special case, for actions which have already started but not yet finished, when constructing the TRPG we know the upper bound on how much time has elapsed since the start of each of these. With $exec(A)$ denoting the maximum time since the start of A, the additional condition is then that $v \geq (d - exec(A)).m - ub(v)$; i.e. we assume as much consumption as possible falls before the start of the planning graph.

8 Conclusions

In this paper, we have discussed how a temporal planner, CRIKEY3, can be adapted to plan in problems where actions have continuous linear numeric effects. Key to the success of the approach is the use of a Linear Programme (LP) to schedule the action choices, in the context of interacting temporal and numeric constraints. The LP cannot be used to entirely decouple the continuous and non-continuous parts of planning, but minimises the changes required to the rest of the planner: storing upper- and lower-bounds on variables; and updating the heuristic with an approximation of the continuous behaviour.

In future work, our current aim is to complete the implementation of the planner and gather empirical data on the performance of the planner, along with a collection of appropriate domains.

References

Audemard, G.; Bertoli, P.; Cimatti, A.; Kornilowicz, A.; and Sebastiani, R. 2002. A SAT-based approach for solving formulas over boolean and linear mathematical propositions. In *Proceedings of the 18th International Conference on Automated Deduction*, volume 2392, 193–208. Springer-Verlag, LNAI Series.

Benazera, E.; Brafman, R.; Meuleau, N.; Mausam; and Hansen, E. A. 2005. An AO* Algorithm for Planning with

Continuous Resources. In *Workshop on Planning under Uncertainty for Autonomous Systems, associated with the International Conference on AI Planning and Scheduling (ICAPS)*.

Coles, A. I.; Fox, M.; Halsey, K.; Long, D.; and Smith, A. J. 2008a. Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence*.

Coles, A. I.; Fox, M.; Long, D.; and Smith, A. J. 2008b. A hybrid relaxed planning graph-lp heuristic for numeric planning domains. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 08)*.

Coles, A. I.; Fox, M.; Long, D.; and Smith, A. J. 2008c. Planning with problems requiring temporal coordination. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 08)*.

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

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

Fox, M., and Long, D. 2003. PDDL2.1: An Extension of PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research (JAIR)* 20:61–124.

Fox, M., and Long, D. 2006. Modelling mixed discrete-continuous domains for planning. *J. Artif. Intell. Res. (JAIR)* 27:235–297.

Gerevini, A., and Serina, I. 2002. Lpg: A planner based on local search for planning graphs with action costs. In *AIPS*, 13–22.

Hoffmann, J., and Edelkamp, S. 2005. The deterministic part of ipc-4: An overview. *Journal of Artificial Intelligence Research (JAIR)* 24:519–579.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *Journal of Artificial Intelligence Research (JAIR)* 20:291–341.

Howey, R., and Long, D. 2003. Val's progress: The automatic validation tool for pddl2.1 used in the international planning competition. In *Proceedings of the ICAPS 2003 workshop on 'The Competition: Impact, Organization, Evaluation, Benchmarks'*, 28–37.

Léauté, T., and Williams, B. 2005. Coordinating Agile Systems through the Model-based Execution of Temporal Plans. In *Proceedings of 20th National Conference on AI (AAAI)*, 114–120.

Li, H., and Williams, B. 2008. Generative systems for hybrid planning based on flow tubes. In *Proceedings of ICAPS'08*, 206–213.

Long, D., and Fox, M. 2003. Exploiting a Graphplan Framework in Temporal Planning. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS)*, 52–61.

Lynch, N., and Vaandrager, F. 1992. Forward and backward simulations for timing-based systems. In *Proceedings of Real-Time: Theory in Practice, volume 600 of Lecture Notes in Computer Science*, 397–446.

McDermott, D. 2003. Reasoning about Autonomous Processes in an Estimated Regression Planner. In *Proceedings of 13th International Conference on Automated Planning and Scheduling (ICAPS)*, 143–152. AAAI-Press.

Penberthy, S., and Weld, D. 1994. Temporal Planning with Continuous Change. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)*, 1010–1015. AAAI/MIT Press.

Sandewall, E. 1994. *Features and fluents: the representation of knowledge about dynamical sys tems, volume I.* Oxford University Press.

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

Smith, D., and Weld, D. S. 1999. Temporal Planning with Mutual Exclusion Reasoning. In *Proceedings of the 16th International Joint Conference on Artificial Intelliegence (IJCAI)*, 326–337.

Younes, H. L. S., and Simmons, R. G. 2004. Policy Generation for Continuous-Time Stochastic Domains with Concurrency. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS)*, 325–333.