Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

Temporal Planning with Semantic Attachment of Non-Linear Monotonic Continuous Behaviours

Josef Bajada and **Maria Fox** and **Derek Long**
Department of Informatics, King's College London
London WC2R 2LS, United Kingdom
{josef.bajada, maria.fox, derek.long}@kcl.ac.uk

Abstract

Non-linear continuous change is common in real-world problems, especially those that model physical systems. We present an algorithm which builds upon existent temporal planning techniques based on linear programming to approximate non-linear continuous monotonic functions. These are integrated through a semantic attachment mechanism, allowing external libraries or functions that are difficult to model in native PDDL to be evaluated during the planning process. A new planning system implementing this algorithm was developed and evaluated. Results show that the addition of this algorithm to the planning process can enable it to solve a broader set of planning problems.

1 Introduction

Real-world temporal planning problems often involve non-linear continuous change to numeric variables. This is especially the case for scenarios that model the physical world. Examples of such instances include problems that involve fluid dynamics or systems dealing with power management. It is also difficult to model these problems in conventional PDDL since the constructs provided by the language to express continuous change [Fox and Long, 2003] are limited to arithmetic operators.

On the other hand, current state-of-the-art temporal planners are becoming more effective at solving planning problems that involve linear continuous change. One of the reasons for this is their ability to leverage the efficiency and flexibility of linear programming to model the temporal and numeric aspects of the problem. Examples of such planners include TM-LPSAT [Shin and Davis, 2005] and COLIN [Coles *et al.*, 2009; 2012], together with its successors POPF [Coles *et al.*, 2010] and OPTIC [Benton *et al.*, 2012].

In this work we present an approach that extends these techniques to solve problems that also involve non-linear continuous change to numeric variables. In order to allow more complex expressions to be incorporated in continuous effects, while retaining the same structure of PDDL, we propose to integrate such continuous functions in planning domains using *semantic attachment* [Dornhege *et al.*, 2009].

We present an algorithm that approximates non-linear change by converging iteratively to a solution. Each semantically attached function can have a dedicated margin of error, within which each non-linear continuous effect is required to be estimated. The system also enables the user to optionally specify this as a parameter within the problem definition.

A PDDL 2.1 temporal planner, uNICOrn, was developed in order to evaluate this algorithm, and we show that it can handle durative actions of a variable duration (with duration inequalities) that have linear and non-linear continuous effects. Concurrent durative actions affecting the same numeric fluent are also supported. This planner was evaluated using two domains with non-linear continuous effects. Results show the effectiveness of this algorithm in solving this class of problems and the impact of decreasing the error tolerance on its overall performance.

2 Background

Continuous change has an important role when modelling real-world problems. This was recognised in PDDL 2.1 [Fox and Long, 2003], where durative actions and continuous effects were introduced to enable the capability to model temporal planning problems [Cushing *et al.*, 2007] and time-dependent numeric updates. Continuous effects are defined in terms of a rate of change of a numeric variable with respect to time. Listing 1 shows an example of a durative action modelling an airplane flight. The `fuel-level` of the `airplane` decreases at its `fuel-consumption-rate` with respect to time, denoted by the special operand `#t`.

```
(:durative-action fly
  :parameters (?p - airplane ?a ?b - airport)
  :duration (= ?duration (flight-time ?a ?b))
  :condition (and
    (at start (at ?p ?a))
    (over all (inflight ?p))
    (over all (>= (fuel-level ?p) 0)))
  :effect (and
    (at start (not (at ?p ?a)))
    (at start (inflight ?p))
    (at end (not (inflight ?p)))
    (at end (at ?p ?b))
    (decrease (fuel-level ?p)
        (* #t (fuel-consumption-rate ?p)))))
```

Listing 1: A durative action in PDDL [Fox and Long, 2003].

However, real-world problems commonly also involve non-linear continuous behaviour. This is especially the case when the problem being modelled involves physical processes. One such example, used as a benchmark in this work, is Torricelli's Law. This states that the velocity, v, with which liquid drains from a tank in relation to the height, h, of the surface of the liquid in the tank is $v = \sqrt{2gh}$ (where g is gravity). Consequently, v will decrease as h decreases.

While current temporal planners are quite effective in dealing with linear continuous change, most of them struggle when non-linear processes are introduced. While it is in fact possible to model some non-linear functions in PDDL, through a continuous effect on the rate of change used in another continuous effect, most planners are still not capable of handling such kind of models.

One of the few planners that does handle non-linear change is UPMurphi [Penna *et al.*, 2009]. It uses an approach which involves time discretisation. A plan is found when the required conditions are satisfied for all discrete time points and the plan is then validated using a separate plan validator, VAL [Howey *et al.*, 2004]. The main disadvantage with this approach is that it does not scale up very well, with long plans necessitating an increase in the discretisation value to obtain a solution within a reasonable time. Another more recent approach [Bryce *et al.*, 2015] involves combining DPLL-style SAT solving together with a differential equation solver, capable of integrating continuous effects formulated as Ordinary Differential Equations (ODE), and an Interval Constraint Propagation (ICP) solver. In this case, the user specifies the error precision within which the non-linear effects can be evaluated. Even in this case, the main issue is scalability as the problem size increases.

In this work we try to address the issue of scalability by handling a subset of non-linear continuous effects. More specifically, we consider those that manifest a monotonic increase or decrease between two discrete time-points in the plan. Our approach is based on establishing a linear approximation of the function between the specific time-points. It does not require any time discretisation, but makes use of an iterative method to converge to the required error tolerance. Furthermore, it is not restricted to ODEs, but virtually any continuous monotonic function can be attached through a semantic attachment mechanism.

3 Non-linear Temporal Planning Tasks

The language used by uNICOrn is PDDL 2.1 [Fox and Long, 2003], augmented with a set M of semantically attached functions provided by external modules. We restrict attention to continuous monotonic functions. These kinds of functions are very common when modelling physical systems and were thus the primary focus of this work.

Let such a planning task with semantic attachments be defined as $\Pi = \langle \rho, \vartheta, M, O_{inst}, O_{dur}, s_0, G \rangle$, where:

- ρ is the set of atomic propositional facts.
- ϑ is the set of numeric fluents.
- M is the set of continuous monotonic functions available through semantically attached modules, taking the form $m : S \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$, where S is the set of possible states. $m(s,t)$ maps the duration, t, spent in a state $s \in S$ to the value of the function at that time.
- O_{inst} is the set of instantaneous actions.
- O_{dur} is the set of durative actions.
- s_0 is the initial state of the problem, consisting of a subset of ρ and a mapping of ϑ to numeric values.
- G is a set of goal conditions that must be satisfied.

An instantaneous action, $a_{inst} \in O_{inst}$ is an action that can occur at a specific time point in a plan. It can be applied to a state $s \in S$ if its precondition expression, $pre(a_{inst})$, is satisfied by that state. Preconditions can be propositional facts of the form $f \in \rho$, or numeric conditions of the form $\langle exp, \otimes, c \rangle$, where exp is an arithmetic expression whose variables correspond to numeric fluents in ϑ, the operator $\otimes \in \{=, <, \leq\}$, and c is a numeric constant. An action's precondition expression, $pre(a_{inst})$, combines a set of propositional and numeric conditions into a logical expression using conjunction, disjunction and negation. An action, a_{inst}, is applicable to a state, s, if such precondition expression is satisfied by s, formally $s \models pre(a_{inst})$.

Each instantaneous action also has a set of instant effects, $\mathit{eff}(a_{inst})$. When an action is applied to a state s, a new state, s', is obtained from applying its effects; $s' = \mathit{eff}(a_{inst})(s)$ [Hoffmann, 2003]. These effects could add or remove propositional facts, and also change the value of a numeric variable $v \in \vartheta$ based on the evaluation of an arithmetic expression. Numeric effects can either *increase* or *decrease* the value of v, or *assign* it to a completely new value.

A durative action, $a_{dur} \in O_{dur}$, follows similar rules, but spans over a period of time in a plan. Conditions and effects can occur at the start, at the end, or during the execution of a durative action. Each durative action $a_{dur} \in O_{dur}$ has a start condition expression, $startCond(a_{dur})$, that has to be satisfied by the state preceding the start of the action, an end condition expression, $endCond(a_{dur})$ that needs to be satisfied by the state preceding the end of the action, an invariant condition expression, $inv(a_{dur})$, that must hold throughout its execution, and a set of temporal constraints, $durCond(a_{dur})$, on its duration. A durative action also has start effects, $\mathit{startEff}(a_{dur})$, that are applied when the action commences, end effects, $\mathit{endEff}(a_{dur})$, applied when the action stops executing, and continuous effects $\mathit{contEff}(a_{dur})$, representing continuous change to a numeric fluent occurring throughout the execution of the action. These continuous effects can be linear changes of the form $v_{s'} = v_s + \delta v \cdot t$, where v_s and $v_{s'}$ correspond to the value of variable v in states s and s' respectively, δv is the rate of change of v, and t corresponds to the duration spent in state s. Continuous effects can also be non-linear functions encapsulated through semantic attachment, of the form $v_{s'} = m(s,t)$, where $m \in M$.

Similar to the approach used by other temporal planners, such as LPGP [Long and Fox, 2003], LPG [Gerevini *et al.*, 2003], and COLIN [Coles *et al.*, 2012], each durative action a_{dur} is split into two *snap actions*: $a_{\vdash}$, representing the conditions and effects when the durative action starts, and $a_{\dashv}$, representing the conditions and effects when the durative ac-

tion ends. Thus, *snap actions* have the same structure of instantaneous actions, where $pre(a_\vdash) = startCond(a_{dur})$, $\mathit{eff}(a_\vdash) = \mathit{startEff}(a_{dur})$, $pre(a_\dashv) = endCond(a_{dur})$, and $\mathit{eff}(a_\dashv) = \mathit{endEff}(a_{dur})$. The set of operators of Π can thus be defined as $O = O_{inst} \cup \{a_\vdash, a_\dashv \mid a_{dur} \in O_{dur}\}$.

A temporal state is defined as $s = \langle F, V, P, Q, C\rangle$, where:

- F is the set of atomic propositions holding in the state, with $F \subseteq \rho$.
- V maps the numeric variables to their respective value, with $V : \vartheta \to \mathbb{R}$
- P is the plan, a sequence of actions $(a_0, a_1, ..., a_n)$ where $a_i \in O$, to reach the state s from s_0.
- Q is a list of executing durative actions that have commenced, but not yet terminated, together with the step index at which they were started in the plan P. Each element in Q takes the form $\langle a_\vdash, i\rangle$, where $i \in [0..n]$.
- C is a set of temporal constraints of the form $lb \leq (t(j) - t(i)) \leq ub$ for step indices $\{i, j\} \subseteq \{0, ..., n\}$ in plan P, where lb is the lower-bound and ub is the upper-bound of the temporal interval between steps i and j, and $t : \mathbb{N} \to \mathbb{R}_{\geq 0}$ determines the time at which a step is to be executed.

Whenever an action $a \in O$ is applied and appended to the plan P to obtain a new state s', the propositions F and numeric fluents V are updated according to the effects, $\mathit{eff}(a)$, as explained above. Furthermore, if a is a start or end *snap action* of a durative action a_{dur}, Q and C need to be updated accordingly. If $a = a_\vdash$, Q is updated with an entry containing the *snap action* and the index of the step, $Q' = Q \cup \{\langle a_\vdash, |P|\rangle\}$. A constraint is also added to C to enforce that $\varepsilon \leq t(|P|) - t(|P| - 1)$, where ε is a very small non-zero amount enforcing a total ordering between actions [Coles *et al.*, 2012]. If $a = a_\dashv$, new constraints of the form $lb \leq (t(|P|) - t(i)) \leq ub$ are added to C, given that the corresponding entry for the start *snap action*, $\langle a_\vdash, i\rangle$, is in Q. These constraints reflect the duration condition, $durCond(a_{dur})$, of the durative action. The entry $\langle a_\vdash, i\rangle$ is then removed from Q of the new state s'.

The invariant conditions that must hold throughout the duration of a state s consist of the invariant conditions of all the actions for which an entry exists in Q, as defined in Equation 1, where $Dom(Q) = \{a_\vdash | \langle a_\vdash, i\rangle \in Q\}$ and a_{dur} is the durative action commenced by $a_\vdash$.

$$stateInv(s) = \bigcup_{a_\vdash \in Dom(Q)} inv(a_{dur}) \tag{1}$$

According to PDDL 2.1 semantics, continuous effects take place while in a state, as time progresses from one step in the plan to the next. Variables affected by continuous change are time-dependent, since their value does not depend solely on which actions are chosen, but also on their duration. In our planning framework, continuous effects on variables can either *increase* or *decrease* the value of a numeric fluent by a constant rate with respect to a state's duration. It can thus be formally defined as a tuple $\langle v, \delta v\rangle$, where $v \in \vartheta$ and δv is the rate of linear change. In the case of semantically attached non-linear effects, δv will be determined through an iterative method that converges to a value within the required error bound. The list of continuous effects taking place between state s_i and s_{i+1}, denoted $\mathit{stateContEff}(s_i)$, consists of all the continuous effects of durative actions executing during that interval. This corresponds to those durative actions whose start *snap action* is in $Dom(Q)$ for state s_i.

4 Planning and Scheduling

In order to find a valid temporal plan for a problem with continuous effects the computation of the values of time-dependent numeric fluents must be performed in conjunction with the scheduling process. The approach we use is similar to the one used in COLIN [Coles *et al.*, 2012], where a Linear Program (LP) is used to find a feasible assignment to the numeric variables and the timestamps of each *happening* [Fox and Long, 2003], such that all the action preconditions and temporal constraints are satisfied. However, there are some key differences in the way numeric fluents are represented in the LP and effects are propagated throughout the plan.

A plan is found by performing a forward search from the initial state and applying actions in O that have their preconditions satisfied, obtaining new states from the effects of each of the applied actions. Just as in COLIN, at every new state being evaluated, an LP is constructed to determine whether the plan to reach that state is actually possible given the temporal and numeric constraints. A variable is used for the time of each *happening*, with the objective function set to minimise the value of the variable corresponding to the time of step n. The temporal constraints in C are translated into the corresponding linear constraints of the LP.

Definition 1. A numeric fluent evaluated at step k of a plan is *time-dependent* at k if either:

i) it is modified by a continuous effect from a durative action, a_{dur}, where $a_\vdash$ takes place at step $h < k$ and $a_\dashv$ takes place at step $i > h$, or

ii) at step $i \leq k$ the value of the numeric fluent is computed from an expression involving the non-constant duration of a durative action, or

iii) it is updated by a *time-dependent* effect expression from an action at step $i < k$,

and there is no step j at which the fluent is assigned to a non-time-dependent value, where $i < j < k$.

Definition 2. An expression evaluated at step k of a plan is *time-dependent* at k if one of its terms is a *time-dependent* numeric fluent at step k.

Let $\tilde{\vartheta} \subseteq \vartheta$ be the set of fluents for which there exists at least one step in the plan at which they are *time-dependent*. Expressions that do not depend on fluents in $\tilde{\vartheta}$ are evaluated prior to the scheduling stage and excluded from the LP. Apart from potentially reducing the number of variables, this approach also helps to prune actions deemed inapplicable due to non-time-dependent numeric preconditions, thus eliminating the need to compute their LP. This approach also allows more complex non-time-dependent numeric expressions to be incorporated at the endpoints of each action, which otherwise would not be supported by the LP. One example of this would

be the case where the discrete numeric effect at the start or end of a durative action is the result of a multiplication of two or more numeric fluents. Furthermore, for each step k, conditions and effects that involve fluents in $\tilde{\vartheta}$, but that are not *time-dependent* at k, are also excluded from the LP and are instead computed by the planner.

For each $v \in \tilde{\vartheta}$, two variables v_i and v'_i are added to the LP at each step i, corresponding to the value of v before and after applying the action at step i respectively. The value of v_0 corresponds to that of the initial state, and constraints are added to enforce that $v_0 = s_0.V(v)$. Values computed by the planner for steps in which v is non-time-dependent are also enforced in the LP in the same way.

The *time-dependent* preconditions of each action in the plan at each step i are encoded as constraints over $\tilde{\vartheta}_i$. The *time-dependent* invariants for s_i, $tdInv(s_i) \subseteq stateInv(s_i)$, are also encoded in a similar fashion over $\tilde{\vartheta}'_i$ and $\tilde{\vartheta}_{i+1}$. The *time-dependent* instantaneous effects of each action in the plan at each step i on a variable $v \in \tilde{\vartheta}$ are encoded as constraints over the relationship between v_i and v'_i.

Continuous effects on a variable $v \in \tilde{\vartheta}$ at step i are represented with constraints on the relationship between v'_i and v_{i+1}. COLIN is capable of combining concurrent continuous effects on the same variable v by sequentially accumulating the increase or decrease of its rate of change with each start or end action that has a continuous effect on v [Coles *et al.*, 2012]. However, this approach does not support cases where an action modifies the rate of change of a continuous effect of another action that has already started. Figure 1 illustrates one such example, where action a performs a linear increase on variable v at rate y, and the start *snap action* $b_\vdash$ updates y, affecting the rate of change of the rest of the continuous effect of a.

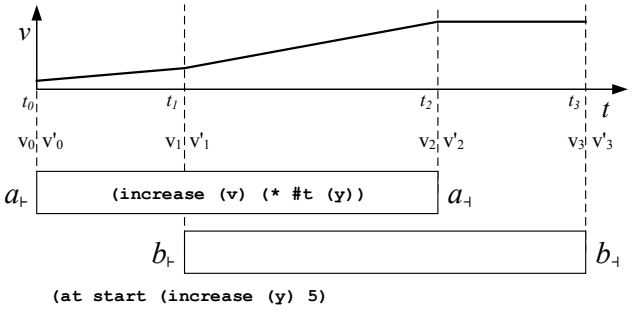

Figure 1: Updating the rate of change of a continuous effect.

Rather than accumulating the rate of change sequentially, we compute a fresh value for the cumulative rate of change, Δv_i, for each step i, as defined in Equation 2, where $\Xi_i^v = \{\delta v | \langle v, \delta v \rangle \in stateContEff(s_i)\}$ and $stateContEff(s_i)$ is the list of active continuous effects taking place throughout the duration of the state s_i. The value for the subsequent step is then computed using Δv_i, as shown in Equation 3.

$$\Delta v_i = \sum_{\delta v_i \in \Xi_i^v} \delta v_i \tag{2}$$

$$v_{i+1} = v'_i + \Delta v_i (t(i_{i+1}) - t(i)) \tag{3}$$

This approach allows for discrete updates to the rate of change of a linear continuous effect to take place, effectively supporting piecewise linear continuous effects, with a different value for δv in each segment.

5 Non-Linear Iterative Convergence

With the planning framework for linear continuous effects defined, we now propose a mechanism with which non-linear monotonic continuous effects, semantically attached to the planning task, can be included. Each function, $m(s,t)$, available through a semantically attached module, provides the value of that function, at a state s, for time $t \geq 0$. This is essentially the interface between the temporal planner and the external module. The fact that the functions are restricted to be monotonic makes it easier to verify any invariant conditions that need to hold throughout the continuous effect (although this is limited to cases where the sum of the effects on a variable v within a step i, Δv_i, is also monotonic, and the invariant condition is constant throughout its duration).

Theorem 1. For a continuous function $m : S \times \mathbb{R}_{\geq 0} \to \mathbb{R}$, conditioned by a state $s \in S$, there exists a linear function $\tilde{m} : \mathbb{R}_{\geq 0} \to \mathbb{R}$ for each $t \geq 0$, where $m(s,0) = \tilde{m}(0)$ and $m(s,t) = \tilde{m}(t)$.

Proof. Let $v_0 = m(s,0)$ and $v_t = m(s,t)$. There exists a linear function of the form $\tilde{m}(t) = \delta v \cdot t + v_0$ where $\delta v = (v_t - v_0)/t$, in which case $\tilde{m}(0) = v_0$ and $\tilde{m}(t) = v_t$. □

As shown in Theorem 1, a non-linear continuous function can be intercepted by a linear one at the vertical axis (where $t = 0$) and at any other given time-point. If the desired duration of a continuous numeric effect is known beforehand, the linear function $\tilde{m}$ can effectively replace the non-linear one m and still provide the correct updated value for a continuously changing variable at time t. This reduction makes it computable through the LP-based planning framework. However, the main challenge is how to determine the duration, and thus select the right linear approximation.

We propose an iterative method, that starts from computing δv_i at one of the known bounds of the duration of the state, determined from its constraints C, and then iteratively improves its value until the error between the approximated change on v and the real one is less than or equal to a predefined error value for that function, e_m, as described in Algorithm 1. $\delta\tilde{V}$ maps the step index i, numeric fluent v, and continuous effect m to the approximated rate of change for that variable contributed from effect m.

Γ_s refers the list of non-linear effects applied to the steps of plan P of state s, with each element consisting of $\langle i, v, m \rangle$, where i is to the index of the *happening* after which the non-linear continuous effect takes place (up to step $i+1$), $v \in \tilde{\vartheta}$ corresponds to the *time-dependent* numeric fluent updated by the effect, and m corresponds to an instance of a continuous effect involving a semantically attached function.

The function `scheduleWithLP`$(s, \delta\tilde{V})$ sums up the rates of change on each variable $v \in \tilde{V}$ at each step, as defined

Algorithm 1: Non-linear Iterative Convergence

Function `NonlinSchedule(`s, $\delta\tilde{V}$`)`

Data: A temporal state s and a set of initial approximations $\delta\tilde{V}$ of the rates of change for each non-linear effect m, on each variable v, at each step, i.

Result: A schedule for the plan to reach s, or Failure

converged $\leftarrow true$

schedule $\leftarrow$ `scheduleWithLP(`$s, \delta\tilde{V}$`)`

if schedule $= \emptyset$ **then**

  return Failure

for $\langle i, v, m\rangle \in \Gamma_s$ **do**

  $d \leftarrow$ schedule$(i+1) -$ schedule(i)

  $\delta v \leftarrow m(s_i, d)$

  if $|(\delta v \cdot d) - (\delta\tilde{V}(i,v,m) \cdot d)| > e_m$ **then**

    converged $\leftarrow false$

    if *(($\delta v < 0$) and ($\delta\tilde{V}(i,v,m) < \delta v$)) or (($\delta v > 0$) and ($\delta\tilde{V}(i,v,m) > \delta v$))* **then**

      $\delta\tilde{v} \leftarrow (\delta\tilde{V}(i,v,m) + \delta v)/2$

      $s.C \leftarrow s.C \cup \{(d \leq t(i+1) - t(i))\}$

    else

      $\delta\tilde{v} \leftarrow \delta v - (\delta\tilde{V}(i,v,m) - \delta v)/2$

      $s.C \leftarrow s.C \cup \{(t(i+1) - t(i) \leq d)\}$

  $\delta\tilde{V}(i,v,m) \leftarrow \delta\tilde{v}$

if converged **then**

  return schedule

else

  return `NonlinSchedule(`$s, \delta\tilde{V}$`)`

in Equation 2. It combines these cumulative rates of change with the *time-dependent* preconditions of the actions in P and the temporal constraints C to build a linear program that computes a `schedule`. This consists of a list of start times for each step in the plan.

From this `schedule`, the actual durations of each state can be extracted, and the deviation of each approximated non-linear effect from its real value can be quantified. If this exceeds the maximum error defined for that effect, e_m, the flag `converged` is set to *false*, indicating that further iterations are needed to converge to the acceptable error threshold. A new gradient approximation for that continuous effect is then computed, and new temporal constraints on the duration of step i are added to $s.C$. These depend on whether the gradient is negative or positive, and whether the approximation over-estimated or under-estimated the actual value. If all non-linear effects were estimated within the acceptable error threshold then the `schedule` is returned. Otherwise, the same procedure is called again recursively, with s carrying the new set of temporal constraints C and $\delta\tilde{V}$ updated with new gradient approximations.

Figure 2 illustrates the intuition behind the non-linear iterative convergence algorithm. Starting from the value v_{ub} at one of the known bounds of the state's duration, ub, the algorithm interpolates a linear approximation intercepting the non-linear function at $t = 0$ and $t = ub$. A linear program is used to determine the duration for the state considering the rest of the constraints, establishing it to be $d0$. At this point the error e_{d0} is computed from the value of the real function, and it is determined that it exceeds the maximum allowed error, e_m, for that function. A new gradient approximation that halves the error at $d0$ is then used and the LP is computed once again, obtaining $d1$. The procedure is repeated until the error is small enough to accept the value.

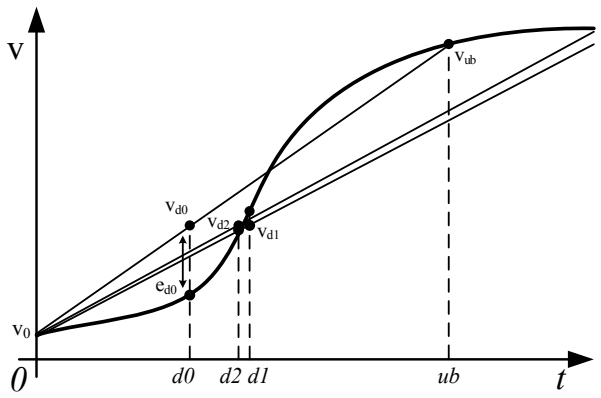

Figure 2: Non-linear iterative convergence on a monotonic continuous function.

6 Empirical Evaluation

The non-linear convergence algorithm was integrated with the planning framework described in Sections 3 and 4. The implemented planner, uNICOrn, performs a breadth-first search (BFS) starting at the initial state and expanding each temporal state, s, according to the set of applicable actions. An action, $a \in O$, is deemed applicable if the propositional and non-time-dependent preconditions in $pre(a)$ are satisfied by the temporal state, and if a `schedule` for the plan of the new state, $s' = \mathit{eff}(a)(s)$, is found by the LP.

The node expansion process of the BFS was slightly modified to suit better the temporal characteristics of the planning problem. If the temporal state being expanded has no executing actions ($Q = \emptyset$), a dummy goal action a_g, where $pre(a_g) = G$ and $\mathit{eff}(a_g) = \emptyset$, is checked for applicability and if it is, its resultant temporal state is put at the head of the BFS open list. This handles cases where the plan of the current temporal state can actually achieve the goal conditions if subjected to the right schedule. On the other hand, if the temporal state has executing actions ($|Q| > 0$) and the goal conditions have been satisfied, the corresponding applicable end *snap actions* are added to the BFS open list before adding the rest of the applicable actions. This gives a preference to evaluate *snap actions* that end executing durative actions before starting new ones.

The system was evaluated using two domains, the *Tanks* domain (described below), and the *Car* domain [Fox, 2006].

In both cases, their non-linear characteristics were encapsulated in semantically attached functions, and both were modelled using PDDL 2.1. Tests were performed using an Intel® Core™ i7-3770 CPU @ 3.40GHz.

The *Tanks* domain consists of *tanks* and *buckets*, where each tank has a spigot at the bottom, through which the liquid inside the tank is allowed to flow out by gravity. The velocity with which the liquid flows out is not constant but follows Torricelli's law of fluid dynamics. This velocity decreases over time, in relation to the height of the liquid in the tank, until it reaches zero. The height itself also decreases non-linearly depending on the outflow velocity.

These two non-linear functions were semantically attached to the planning domain through `Torr.drain-rate` and `Torr.height-change`, where `Torr` is the alias of the module providing these external functions. The module also introduces a special type `Tank`, and numeric fluents for the `height`, `surface-area` and `hole-area`, representing the initial height of the fluid in a tank, the surface area of the fluid in a tank, and the area of the hole in the spigot, respectively. These are used by the module to calculate the drain rate and change in height according to Torricelli's law. A `Bucket` has a maximum `capacity` and `volume` of liquid it contains.

```
(:durative-action fill
 :parameters (?t - Torr.Tank
              ?b - Bucket)
 :duration (and (>= ?duration 0))
 :condition (and
   (over all (>= (Torr.height ?t) 0))
   (over all
       (<= (volume ?b) (capacity ?b)))
   (at start (not(filling ?b)))
   (at start (not(filled-from ?t))))
 :effect (and (at start (filling ?b))
    (at start (filled-from ?t))
    (increase (volume ?b)
      (* #t (Torr.drain-rate ?t)))
    (decrease (Torr.height ?t)
      (* #t (Torr.height-change ?t)))
    (at end (not(filling ?b)))))
```

Listing 2: The `fill` durative action.

Listing 2 shows the PDDL for the `fill` durative action, which updates the `volume` of the `Bucket ?b` together with the `height` of `Tank ?t`, in relation to the duration of the action. Throughout the action the `height` of the tank must remain non-negative, and the `volume` of liquid contained in the bucket must be less than or equal to its `capacity`. The goal of the problem is to fill the bucket up to within one hundred units of its capacity, and make use of all the tanks available. Listing 3 shows the goal condition with three tanks.

```
(:goal (and
  (> (volume b1) (- (capacity b1) 100))
  (<= (volume b1) (capacity b1))
  (filled-from tank1)
  (filled-from tank2)
  (filled-from tank3)))
```

Listing 3: Goal condition for 3-tank problem.

Listing 4 shows the output plan for the Tanks problem with 3 tanks, with error tolerance $e_m = 0.0001$ and $\varepsilon = 0.001$. Changing the error tolerance value will have an impact on the durations of the actions in the plan since they have non-linear continuous effects.

```
0.0000: (fill tank1 bucket1) [11176.1667]
11176.1677: (fill tank2 bucket1) [11176.1666]
22352.3353: (fill tank3 bucket1) [8423.8748]
```

Listing 4: 3-tank plan with an Error Tolerance of 0.0001

Figure 3 shows the time taken to produce a plan for this domain, for up to ten tanks. In each case the capacity of the bucket was set such that all tanks need to be used to achieve the goal volume. Each problem was also executed with a range of error tolerance values to analyse the impact of the increase in the number of iterations needed to converge.

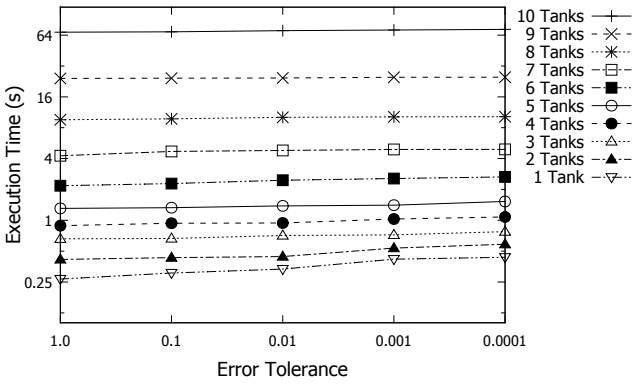

Figure 3: Performance of uNICOrn on the problem instances for the *Tanks* domain with decreasing error tolerance.

While the time needed to solve the problem increases exponentially with the number of tanks, partly due to a higher branching factor that needs to be explored by the breadth-first search, decreasing the error tolerance required has a low impact on the over all planning process. Table 1 compares the performance of uNICOrn (with 0.1 error tolerance) with that of UPMurphi (with 1.0 time discretisation) using a maximum of 3GB memory on the same setup. Missing timings indicate that the planner ran out of memory.

Tanks	1	2	3	4	5	6	7
uNICOrn	0.31s	0.43s	0.67s	0.94s	1.33s	2.28s	4.70s
UPMurphi	0.16s	0.64s	31.44s	•	•	•	•

Table 1: Performance of uNICOrn and UPMurphi on problem instances of the *Tanks* domain.

The *Car* domain [Fox, 2006] includes a durative action, `drive`, which increases the distance travelled with respect to the vehicle's velocity, and two instantaneous actions, `accelerate` and `decelerate`, which increase and decrease the acceleration of the vehicle by one unit respectively. With a non-zero acceleration, the velocity increases linearly over time, making the distance travelled with respect to time non-linear. The `drive` action can only end if the car is stationary.

The goal is to travel a distance of thirty units. Two instances of the domain were tested, one without any time bounds, and one with an upper bound of fifty on the duration of the `drive` action, in order to increase the constraints on the plan. The results of these tests are shown in Table 2.

	uNICOrn					UPMurphi	
	1	0.1	0.01	0.001	0.0001	1.0	0.1
Unbounded	0.76s	0.76s	0.83s	0.88s	0.92s	4.72s	•
Bounded	0.97s	1.09s	1.13s	1.16s	1.17s	4.34s	•

Table 2: Performance of uNICOrn and UPMurphi on two problem instances of the *Car* domain, with decreasing error tolerance / time discretisation.

7 Conclusion

Non-linear change is present in many real-world contexts, and the ability to support such functions within a planning system makes it more useful in applications that stand to benefit from such technology. We have presented an approach that builds on existent LP-based techniques, that are already quite effective at handling linear change, and enhanced them with the capabilities to handle piecewise linear continuous effects and also approximate monotonic non-linear change through an iterative improvement method. The planning framework is designed to support the inclusion of virtually any non-linear monotonic continuous function in a planning domain, through a semantic attachment mechanism.

The results from evaluating the new planning system on domains with non-linear continuous change show that the impact of this algorithm on the overall performance is low. Decreasing the error tolerance did not correspond to any significant performance degradation. These findings open up new possibilities on how non-linear continuous behaviour can be managed within a planning system through the use of linear interpolation and approximations.

8 Acknowledgements

This research was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) as part of the project entitled *The Autonomic Power System* (Grant Ref: EP/I031650/1). The authors would also like to thank the anonymous reviewers for their valuable feedback.

References

[Benton *et al.*, 2012] J. Benton, Amanda Coles, and Andrew Coles. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, pages 2–10, 2012.

[Bryce *et al.*, 2015] Daniel Bryce, David Musliner, and Robert Goldman. SMT-Based Nonlinear PDDL+ Planning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI-15)*, 2015.

[Coles *et al.*, 2009] Amanda Coles, Andrew Coles, Maria Fox, and Derek Long. Temporal Planning in Domains with Linear Processes. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 2009.

[Coles *et al.*, 2010] Amanda Coles, Andrew Coles, Maria Fox, and Derek Long. Forward-Chaining Partial-Order Planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS 2010)*, pages 42–49, 2010.

[Coles *et al.*, 2012] Amanda J. Coles, Andrew I. Coles, Maria Fox, and Derek Long. COLIN: Planning with Continuous Linear Numeric Change. *Journal of Artificial Intelligence Research*, 44:1–96, January 2012.

[Cushing *et al.*, 2007] William Cushing, Subbarao Kambhampati, Mausam, and Daniel S. Weld. When is Temporal Planning Really Temporal? In *20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2007.

[Dornhege *et al.*, 2009] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. Semantic Attachments for Domain-Independent Planning Systems. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pages 114–121, 2009.

[Fox and Long, 2003] Maria Fox and Derek Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[Fox, 2006] Maria Fox. Modelling Mixed Discrete-Continuous Domains for Planning. *Journal of Artificial Intelligence Research*, 27:235–297, 2006.

[Gerevini *et al.*, 2003] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. Planning Through Stochastic Local Search and Temporal Action Graphs in LPG. *Journal of Artificial Intelligence Research*, 20:239–290, 2003.

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

[Howey *et al.*, 2004] Richard Howey, Derek Long, and Maria Fox. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning using PDDL. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 294–301. IEEE Comput. Soc, 2004.

[Long and Fox, 2003] Derek Long and Maria Fox. Exploiting a Graphplan Framework in Temporal Planning. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS 2003)*, volume 1, pages 52–61, 2003.

[Penna *et al.*, 2009] Giuseppe Della Penna, Daniele Magazzeni, Fabio Mercorio, and Benedetto Intrigila. UPMurphi: A Tool for Universal Planning on PDDL+ Problems. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pages 106–113. AAAI, 2009.

[Shin and Davis, 2005] Ji-Ae Shin and Ernest Davis. Processes and Continuous Change in a SAT-Based Planner. *Artificial Intelligence*, 166(1-2):194–253, 2005.