ECAI 2016
G.A. Kaminka et al. (Eds.)

doi:10.3233/978-1-61499-672-9-655

Interval-Based Relaxation for General Numeric Planning

Enrico Scala and **Patrik Haslum** and **Sylvie Thiebaux** and **Miquel Ramirez**
The Australian National University and NICTA
Canberra, ACT, Australia
firstname.lastname@anu.edu.au

Abstract. We generalise the interval-based relaxation to sequential numeric planning problems with non-linear conditions and effects, and cyclic dependencies. This effectively removes all the limitations on the problem placed in previous work on numeric planning heuristics, and even allows us to extend the planning language with a wider set of mathematical functions. Heuristics obtained from the generalised relaxation are pruning-safe. We derive one such heuristic and use it to solve discrete-time control-like planning problems with autonomous processes. Few planners can solve such problems, and search with our new heuristic compares favourably with them.

1 Introduction

The ability to express quantitative information is crucial to realistically model many planning domains, in particular domains that involve interaction with physical systems. Examples include positions in time and space, quantities such as pressure, flow or volume, as well as resources.

Planning with unbounded numeric variables is significantly harder than classical planning with only finite-domain variables. Not only can the values of variables grow unboundedly, but even finite values may be reachable only asymptotically. For example, repeatedly applying the effect $x = (x+y)/2$ brings x arbitrarily close to y, whilst repeatedly applying the pair of effects $x = (x+y)/2$ and $y = x \times y$ can either diverge to $\pm\infty$ or converge to 0 or 1, depending on the starting values of x and y. In general, numeric effects can be state-dependent: The effect on x of the update $x = (x + y)/2$ depends on the current value of both x and y. Behaviours of this kind complicate reachability analysis, and therefore the construction of informative heuristics for numeric planning. Because of this, work on heuristic search for numeric planning has focused on restricted forms of numeric conditions and effects, such as linear expressions. Starting from the MetricFF planner [17], the majority of numeric planning heuristics have (implicitly or explicitly) relied on an interval relaxation, which approximates reachable values with upper and lower bounds. Only recently did Aldinger et al. [1] examine the interval-based relaxation theoretically, and show that asymptotic reachability is decidable for the full range of numeric effects expressible with arithmetic expressions (built using $+$, $-$, $\times$ and $\div$), as allowed in PDDL 2.1 [11]. However, even their analysis is restricted to problems with only a very limited form of cyclic dependencies between variables. The example above, in which the effect on both x and y depend on the value of both variables, is outside their scope.

In this paper we show that interval-based relaxed reachability analysis is feasible, and produces informative heuristic guidance, for a much wider range of numeric planning problems, involving cyclic dependencies as well as expressions using standard mathematical functions such as exponentiation, square root, and others. This is achieved by two innovations: an asymptotic relaxed reachability analysis that works also with cyclic dependencies among additive numeric effects, and a syntactic transformation of the problem which eliminates non-additive state-dependent effects, at the cost of inducing an additional relaxation.

Finally, we apply the resulting heuristic to solve time-discretised planning problems with autonomous processes, which feature complex numeric conditions and non-linear effects. Few planners have the expressive range to attempt these problems and heuristic search using our generalised interval-based relaxation heuristic is more efficient than comparable alternatives.

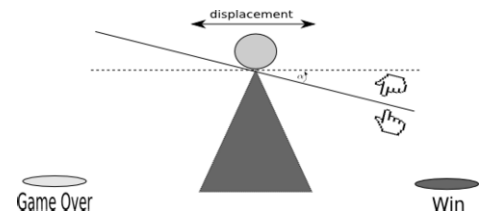

Figure 1: The ball moves along the plane, and is accelerated by tilting the plane. The goal is to make it fall into the winning hole on the right. If the ball falls off the left side, the game is lost.

2 Motivating Example

Let us consider the numeric domain represented in Figure 1: A ball, at position x on a plane, moves with velocity v, possibly falling from one of the sides. $x = 0$ at the centre, and position and velocity is positive to the right. The change in velocity is a function of the inclination α of the plane, accounting for gravity (g) and drag along the surface, proportional with a constant μ to the speed squared (like in the car domain by Bryce et al. [3]). The drag always acts in the opposite direction to the current velocity. An action models the *discretised* movement of the ball over δ_t units of time, changing variables x and v as follows:

$$x = x + v\delta_t$$
$$v = \begin{cases} v + (g\sin\alpha - \mu v^2)\delta_t & \text{when } v \geq 0 \\ v + (g\sin\alpha + \mu v^2)\delta_t & \text{when } v < 0 \end{cases}$$

The left hand side is the updated value of x and v. δ_t is a constant and determines the precision of the discretisation. The other two actions increase and decrease α by a constant amount, by tilting the plane.

The goal is to bring the ball into the hole on the right; this is achieved when x exceeds a given threshold.

This domain has both cyclic dependencies (i.e., a self-cycle in the second numeric effect) and non-linear effects (e.g., the velocity squared). Earlier work on defining heuristics for numeric planning has focused on linear effect expressions [17, 5, 13] or on *acyclic* planning tasks [1]. By removing these two assumptions, we address this class of numeric planning problems. Our key concern is the relaxation that underlies heuristics.

3 Notation and Background Material

We focus on sequential numeric planning with ground actions, corresponding to PDDL 2.1 level 2 [11] but extended with additional interpreted mathematical functions.

A state of the system is a partial assignment over propositions $\mathcal{P}$ and real-valued numeric variables $\mathcal{X}$; $\mathcal{V} = \mathcal{P} \cup \mathcal{X}$. A propositional condition is a positive literal, while a numeric condition is a tuple $\langle \xi, \trianglerighteq, 0 \rangle$ where $\trianglerighteq \in \{\geq, >, =\}$ and ξ is a numeric expression recursively defined as follows: (I) a rational constant is an expression; (II) a variable $x \in X$ is an expression; (III) if ξ' is an expression, then so are $\xi \oplus \xi'$ where $\oplus \in \{+, -, \cdot, \div\}$, ξ^n , where $n \in \mathbb{N}$, b^ξ and $\log_b(\xi)$, where $b \in \mathbb{R}$ and $b > 0$, and $\sqrt{\xi}$. Other functions (e.g., trigonometric functions, nth root, etc) can also be supported, as long as they are functions in the mathematical sense, and computable; for the interval-based relaxation, functions must also have an interval extension (cf. next section). We write $\mathrm{val}(x, s)$ and $\mathrm{val}(\xi, s)$ for the value of variable x and expression ξ in state s, respectively. The value of a variable not assigned in s is *undefined*. Undefinedness propagates through expressions, in the sense that the value of a function is undefined whenever any of its arguments is. Some functions have restricted domains, and their result is undefined when any of their arguments is outside its domain; for example, $\sqrt{x}$ is undefined whenever $x < 0$. A condition $\langle \xi, \trianglerighteq, 0 \rangle$ is unsatisfied whenever the value of ξ is undefined [11]. With slight abuse of notation, we write $x \in \xi$ to mean that variable x appears in expression ξ.

Let $\mathcal{C}$ be a set of propositional and numeric conditions. We write $s \models \mathcal{C}$ when state s satisfies all conditions in $\mathcal{C}$.

Definition 1 (Numeric Action)**.** *A numeric action a is a pair $\langle \mathrm{pre(a)}, \mathrm{eff(a)} \rangle$ where $\mathrm{pre(a)}$ is a set of propositional and numeric conditions and $\mathrm{eff(a)}$ is a set of effects. A classical effect is of the form $p = \top$ or $p = \bot$ ($p \in \mathcal{P}$). A numeric effect is $x \circ\!= \xi$, where $\circ\!= \in \{=, +=, -=\}$ and ξ is an expression over variables in $\mathcal{X}$. $\mathrm{eff(a)}$ cannot contain multiple effects on the same variable.*

We use subscripts to distinguish propositional and numeric parts (e.g., $\mathrm{eff}_{\mathrm{num}}(\mathrm{a})$ is the set of numeric effects of a). For an effect e, $\mathrm{lhs(e)}$ denotes the affected variable, $\mathrm{op}(e)$ the effect operator ($=$, $+=$ or $-=$) and $\mathrm{rhs(e)}$ denotes the right-hand side expression.

Definition 2 (Numeric Planning Problem)**.** *A numeric planning problem is a tuple $\Pi = \langle s_0, \mathcal{A}, \mathcal{G}, \mathcal{V} \rangle$ where $\mathcal{V} = \mathcal{P} \cup \mathcal{X}$ is the set of variables, s_0 is an assignment to variables in $\mathcal{V}$ that is complete for variables in $\mathcal{P}$, A is a set of numeric actions, and G is a set of propositional and numeric goal conditions.*

An effect is called *state-dependent* if the right-hand side is not a constant. Increase and decrease effects can be reformulated as assignments (e.g., $x+= \xi$ as $x = x + \xi$). However, we will see that this distinction is, surprisingly, crucial, because increase and decrease effects are *additive* operations, which assignments are, in general, not. Given an action a, we partition $\mathrm{eff}_{\mathrm{num}}(\mathrm{a})$ into sets *incr*(a), *decr*(a) and *assn*(a) of increase, decrease and assign effects, respectively. We also write *const_assn*(a) for the set of state-independent propositional and numeric assignment effects of a (e.g., $x = 5$).

Action a is applicable in state s iff $s \models \mathrm{pre(a)}$, and its execution results in state $s' = \mathrm{succ}(s, a)$ such that $\forall x \in \mathcal{V} : \mathrm{val}(x, s') =$

$$\begin{cases} rhs(e) & \text{if } \exists e \in \textit{const\_assn}(a) : \mathrm{lhs(e)} = x \\ \mathrm{val}(rhs(e), s) & \text{if } \exists e \in \textit{assn}(a) : \mathrm{lhs(e)} = x \\ \mathrm{val}(x, s) + \mathrm{val}(rhs(e), s) & \text{if } \exists e \in \textit{incr}(a) : \mathrm{lhs(e)} = x \\ \mathrm{val}(x, s) - \mathrm{val}(rhs(e), s) & \text{if } \exists e \in \textit{decr}(a) : \mathrm{lhs(e)} = x \\ \mathrm{val}(x, s) & \text{otherwise (Frame Axiom)} \end{cases}$$

Definition 3 (Plan)**.** *A plan π for $\Pi = \langle s_0, \mathcal{A}, \mathcal{G}, \mathcal{V} \rangle$ is a sequence of actions $a_0, ..., a_{n-1}$ from A such that each action in π is applicable in the state resulting from the application of its predecessors, i.e., $s_0 \models pre(a_0)$, $\mathrm{succ}(s_0, a_0) \models pre(a_1)$, etc, and $\mathrm{succ}(s_{n-1}, a_{n-1}) \models G$. A plan π is said to be optimal if, among all valid plans, it has a minimal number of actions.*

3.1 The Interval-Based Relaxation

Numeric planning is harder than propositional planning, and undecidable in the general case [15]. To obtain useful heuristics we must look for relaxations of the model that yield a computationally effective representation of the task to solve. Moreover, it is important to look for relaxations generating heuristics that are *adequate* for the task they are relaxing [17]. We consider only relaxations (and heuristics) that are *pruning-safe*. A relaxation is pruning-safe if it has no solution only when the original (non-relaxed) problem is also unsolvable; that is, it overestimates the space of reachable states.

An idea pursued in previous work in both classical and numeric planning is that of abstracting away negative effects of the actions, considering only their positive contribution towards achieving a goal or precondition. In propositional planning this amounts to ignoring the delete effects of actions. In general, it implies a possibilistic relaxed interpretation, in which variables accumulate sets of possible values monotonically [14]. Applying this principle to numeric variables, which have unbounded domains, requires a compact representation of the set of values; this is provided by the interval-based representation [1, 23]. Pioneered by Hoffman [17][1], the interval-based relaxation [1] is the underlying principle used in nearly all heuristics for numeric planning [17, 13, 5, 6]. An exception is the work of Eyerich et al. [9] on the Temporal Fast Downward system, which extends the context-enhanced additive heuristic [16] to temporal and numeric planning. However, this heuristic does not account for indirect effects: action a indirectly affects variable x when the effect e of another action b on x depends on a variable y (in $rhs(e)$) which is changed by a. Ignoring indirect effects in numeric planning makes any reachability analysis not pruning-safe (The context-enhanced additive heuristic is also not pruning-safe, but for a different reason.)

For ease of presentation, we will in the following describe only the relaxation of the numeric part of the problem. The propositional part is handled by standard delete-relaxation. The two parts interact only in the relaxed satisfaction of conjunctive conditions.

[1] Note that Hoffman does not exploit the interval representation explicitly, but uses a transformation to Linear Normal Form where variables' domains can only increase. This can be interpreted as constructing an enclosure, by lower and upper bounds, of the possible values attainable by a variable.

3.1.1 Definition of the Interval-Based Relaxation

In the interval-based relaxation (IBR), a state assigns each (defined) numeric variable to an interval of the real line, representing the set of values that the variable can possibly attain. We will refer to this as a *relaxed state*, s^+. A closed interval $x = [\underline{x}, \overline{x}]$ denotes the lower bound $\underline{x}$ and upper bound $\overline{x}$ a variable x can attain. An open interval $x = (\underline{x}, \overline{x})$ is analogous but with $\underline{x}$ and $\overline{x}$ excluded; i.e., $(\underline{x}, \overline{x}) = \{z : \underline{x} < z < \overline{x}, z \in \mathbb{R}\}$. A mixed bounded interval mixes open and closed bounds. Closed interval binary operations between two intervals x and y are defined as follows [23]:

- $x + y = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$;
- $x - y = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$;
- $x \times y = [min(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}), max(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y})]$;
- $x \div y = [min(\underline{x} \div \underline{y}, \underline{x} \div \overline{y}, \overline{x} \div \underline{y}, \overline{x} \div \overline{y}), max(\underline{x} \div \underline{y}, \underline{x} \div \overline{y}, \overline{x} \div \underline{y}, \overline{x} \div \overline{y})]$ (if $0 \notin y$ otherwise one of the bounds diverges [23]).

Binary operations between open or mixed bounded intervals follows the same rules; if an open and a closed bound contribute to the new interval bound, the result is open.

The interval extensions of other mathematical functions in the planning language (such as x^n, $\sqrt{x}$, b^x, $\log_b(x)$) can be similarly defined. The requirement is only that the result of the interval operation contains every value that could result from applying the function to any value(s) in the argument interval(s). To illustrate, we show the interval power, exponentiation and square root. Let x be an interval $[\underline{x}, \overline{x}]$, $n \in \mathbb{N}$ and $b \in \mathbb{R}$, $b > 0$. We have:

$$x^n = \begin{cases} [\underline{x}^n, \overline{x}^n], & \text{if } \underline{x} > 0 \text{ or } n \text{ is odd} \\ [\overline{x}^n, \underline{x}^n], & \text{if } \overline{x} < 0 \text{ and } n \text{ is even} \\ [0, \max(\underline{x}^n, \overline{x}^n)], & \text{if } 0 \in x \text{ and } n \text{ is even} \end{cases}$$

$$b^x = [b^{\overline{x}}, b^{\underline{x}}] \quad \text{for } 0 < b < 1$$

$$b^x = [b^{\underline{x}}, b^{\overline{x}}] \quad \text{for } b \geq 1$$

$$\sqrt{x} = \begin{cases} [\sqrt{\underline{x}}, \sqrt{\overline{x}}], & \text{if } \underline{x} \geq 0 \\ [0, \sqrt{\overline{x}}], & \text{if } \overline{x} \geq 0 \\ \text{undefined}, & \text{otherwise} \end{cases}$$

The square root of a negative value is not a real number. Hence, the interval square root is the result of restricting the argument interval to the range over which the function is defined, and undefined only if there is no such value. All three extend to open or mixed bounded intervals: for integral power, inequalities in the two first cases are non-strict; for the square root, the second case applies when $\overline{x} > 0$.

Expressions, conditions and actions in the IBR are syntactically the same as in the original PDDL problem; what changes is their interpretation. In particular, the value of an expression ξ in a relaxed state s^+, denoted $\mathrm{val}^+(\xi, s^+)$, is the interval computed using interval operations as defined above. Note that mathematical identities are not necessarily preserved by the relaxed interpretation. For example, if $\mathrm{val}^+(x, s^+) = [-2, 2]$, then $\mathrm{val}^+(x \times x, s^+) = [-4, 4]$, while $\mathrm{val}^+(x^2, s^+) = [0, 4]$. A numeric condition $\langle \xi, \trianglerighteq, 0 \rangle$ is satisfied in a relaxed state s^+ if $\mathrm{val}^+(\xi, s^+)$ is defined and there exists some value $v \in \mathrm{val}^+(\xi, s^+)$ such that $v \trianglerighteq 0$. A set of conditions (that can appear both in the goal or action preconditions) is relaxed satisfied iff each condition in the set is. We write $s^+ \models \mathcal{C}$ when s^+ satisfies $\mathcal{C}$.

An action a is applicable in s^+ if $s^+ \models \mathrm{pre(a)}$. In the IBR, action effects can only extend the set of possible values of a variable. To define the relaxed successor state, we need the convex union:

Definition 4. *The convex union between two closed intervals x, y is $x \sqcup y = [\min\{\underline{x}, \underline{y}\}, \max\{\overline{x}, \overline{y}\}]$. The extension to open or mixed bounded intervals uses open/closed bounds according to those used for x and y.*

Definition 5 (Relaxed Action Effects)**.** *Let s^+ be a relaxed state and a an action such that $s^+ \models \mathrm{pre(a)}$. The successor state $s_1^+ = \mathrm{succ}^+(s^+, a)$ is such that $\forall x \in \mathcal{X}, \mathrm{val}^+(x, s_1^+) =$*

$$\begin{cases} \mathrm{val}^+(x, s^+) \sqcup [\mathrm{rhs(e)}, \mathrm{rhs(e)}] & \text{if } x = \mathrm{lhs(e)}, e \in \textit{const\_assn}(a) \\ \mathrm{val}^+(x, s^+) \sqcup \mathrm{val}^+(rhs(e), s^+) & \text{if } x = \mathrm{lhs(e)}, e \in \textit{assn}(a) \\ \mathrm{val}^+(x, s^+) \sqcup (\mathrm{val}^+(x, s^+) + \mathrm{val}^+(\mathrm{rhs(e)}, s^+)) & \text{if } x = \mathrm{lhs(e)}, e \in \textit{incr}(a) \\ \mathrm{val}^+(x, s^+) \sqcup (\mathrm{val}^+(x, s^+) - \mathrm{val}^+(\mathrm{rhs(e)}, s^+)) & \text{if } x = \mathrm{lhs(e)}, e \in \textit{decr}(a) \\ \mathrm{val}^+(x, s_1^+) & \text{otherwise} \end{cases}$$

Applying actions' effects in the IBR can only *monotonically increase* the variables' intervals, because the convex union of two intervals contains both. Hence, this relaxation, just like the classical delete relaxation, is also monotonic with respect to condition satisfaction: what is true before an action is applied is also true after its execution. This property ensures that the relaxation over-approximates the set of reachable conditions, and therefore that it is pruning-safe.

Given a numeric planning problem Π, we denote its relaxation by Π^+. The initial state s_0^+ of Π^+ assigns to each variable $x \in \mathcal{V}$ the unit interval $[\underline{\mathrm{val}(x, s_0)}, \overline{\mathrm{val}(x, s_0)}]$, if x is defined in s_0.

3.1.2 Asymptotic Behavior of Additive Numeric Effects

In numeric planning, there may be no upper bound on the length of the action sequence needed to reach a state (or condition). Because of this, Aldinger et al. [1] considered *asymptotic reachability* in the interval-based relaxation, i.e., the variable intervals that are reachable in the limit of an infinite repetition of actions' effects. The following proposition is slightly adapted from their work. (We present it for increase effects only; the result for decrease effects is analogous.)

Proposition 1. *Let s^+ be a relaxed state, $x \in \mathcal{X}$ a numeric variable, $a \in \mathcal{A}$ an action such that $s^+ \models \mathrm{pre(a)}$, and $x \mathrel{+}= \xi$ an effect of a. Let s_{lim}^+ be the relaxed state reached in the limit by applying a an unbounded number of times in s^+. Then the upper and lower open bounds of x in s_{lim}^+ are as follows:*

- *if $\exists y' > 0 \in \mathrm{val}^+(\xi, s^+)$ then $\overline{x} = \infty$ in s_{lim}^+; and*
- *if $\exists y' < 0 \in \mathrm{val}^+(\xi, s^+)$ then $\underline{x} = -\infty$ in s_{lim}^+.*

In other words, if the right-hand side of an increase effect affecting variable x *can* attain a positive value, no matter how small, then the value of x can be made arbitrarily large by repeated application of the action. This follows directly from the monotonicity of the interval-based relaxation, since if the interval of the right-hand side of the effect, ξ, contains a positive value in s^+ then it contains that value in any successor state. The rate of increase is not constant in general, but always greater or equal to the rate in the first state.

The following notion of dependency between numeric variables was defined by Aldinger et al. We repeat it here because it is necessary for stating precisely how we generalise their work.

Definition 6 (Slightly adapted from Aldinger et al. [1])**.** *A numeric variable x_1 is directly dependent on a numeric variable x_2 in task Π if there exists an action a in $\mathcal{A}$ with a numeric effect $\langle x_1 \circ\!= \xi \rangle$ ($\circ\!= \in \{=, +=, -=\}$) such that $x_2 \in \xi$.*

A variable may depend on itself. However, formulating effects with increase and decrease operators can avoid certain cycles. For example, the assignment $x = x + 1$ causes x to self-depend, but the equivalent increase effect $x{+}{=}\ 1$ does not.

Aldinger et al. [1] proved that asymptotic reachability in the interval-based relaxation is decidable for planning problems where the dependency relation over the numeric variables has no cycle. In this paper, we *remove this restriction*. Moreover, we also show that the IBR can support an extended set of mathematical functions in the planning language, such as exponentiation and square root, which have not been supported in previous work.

We approach the derivation of a heuristic based on the interval relaxation in two parts: First, we present a procedure to determine if a condition is asymptotically reachable in the relaxed problem; only if that is the case do we then compute an estimate of the number of actions required to achieve it. The second part treated in Section 5.3.

4 The Additive Effects Transformation

Our generalisation starts from the observation that one reason why Aldinger et al. [1] required the acyclic dependency assumption is the presence of state-dependent non-additive action effects. Non-additive effects can be transformed into additive ones by simple reformulation which preserves their semantic in the real (non-relaxed) problem. An assignment $x = \xi$ can be rewritten as $x{+}{=}\ \xi - x$. The expressions $x + \xi - x$ and ξ are equivalent in real arithmetic. Formally:

Definition 7 (Additive Effects Transformation)**.** *The additive transformation of an effect* e *is*

- $\tau(e) := \mathrm{lhs(e)} \mathrel{+}= \mathrm{rhs(e)} - \mathrm{lhs(e)}$ *if* $\mathrm{op}(e)$ *is an assignment and* $\mathrm{rhs(e)}$ *is non-constant; and*
- $\tau(e) := e$ *otherwise.*

The additive effects transformation *of a numeric planning problem* Π *is obtained by applying* τ *to all effects of all actions in* Π*, and is denoted by* $\tau(\Pi)$.

Note that this tranformation does not change constant (numeric or propositional) assignments. We refer to the interval-based relaxation applied to the additive effects transformation of Π as the Additive Interval-Based Relaxation (AIBR) of Π.

Definition 8. *The additive interval-based relaxation of a numeric planning problem* Π *is the interval-based relaxation of* $\tau(\Pi)$*, and is denoted by* Π^{++}.

A natural question is what is the relation between AIBR and IBR? AIBR is an over-approximation of IBR: The space of reachable states in Π^{++} includes that of Π^{+}. This implies that heuristics based on the the AIBR are also pruning-safe. The inclusion is a simple consequence of how the interval arithmetic is defined: $x \subseteq x + y - y$ for any intervals x, y, and consequently $z \sqcup x \subseteq z \sqcup (x + y - y)$, for any z. To see how the AIBR can overestimate (asymptotic) reachability, consider a very simple planning example with two actions $a = \langle \emptyset, x = y \rangle$ and $b = \langle \emptyset, y = 1 \rangle$ and a state $s = \langle x = 0, y = 0 \rangle$. In the IBR, the (asymptotically) reachable value of x is the interval $[0, 1]$. Applying the additive effects transformation to a we get $\tau(a) = \langle \emptyset, x{+}{=}\ y - x \rangle$. From Proposition 1 it is easy to see that the asymptotically reachable value of x is now $[0, \infty)$. Applying b we have $\mathrm{val}^{+}(y, \mathrm{succ}^{+}(s^{+}, b)) = [0, 1]$, and, since intervals can only grow, $0 \in \mathrm{val}^{+}(x, \mathrm{succ}^{+}(s^{+}, \cdots))$. Hence the right-hand side of the additive effect $(y - x)$ includes 1, and applying the action repeatedly achieves arbitrary large values of x. Of course, if the task does not include any state-dependent assignment effects, the set of reachable states is exactly the same.

The transformation addresses only partially the problem, since cyclic dependencies can still occur. Crucially, however, those dependencies are only between additive effects.

5 From Numeric Actions to Supporters and the Asymptotic Relaxed Planning Graph

As observed in the previous section, we can identify the asymptotic implications of additive numeric effects on a variable by evaluating the *sign* of its right-hand side. The case of constant assignments ($x = k$ where k is a constant) is simpler as these are idempotent; repeated application of such effects does not change their initial effect.

A difficulty arises when the possible value of the right-hand side of some numeric effect depends on the state in which the effect is applied, since this state depends on the sequence of actions that were executed before it. As noted by Aldinger et al. [1], this is particularly problematic when the planning problem has cyclic dependencies. In this case there is no a-priori ordering of actions that is guaranteed to produce the complete set of reachable values. If, on the other hand, the dependency relation is acyclic, it suffices to apply the asymptotic effect of each action once, in order of the dependency relation.

5.1 Supporters Set

State-dependent numeric effects can be interpreted as a compact way of expressing conditional effects. Each of these conditional effects depends on the specific value of the right hand side of the effect, which in turn depends on the sequence of actions done before. The main idea of our asymptotic reachability analysis is to make explicit the conditions for such indirect effects to occur through the use of auxiliary actions called "supporters".

Each action a is split into a set of supporters, each modelling one possible asymptotic outcome of an effect of a. As observed in Proposition 1, an additive numeric effect can extend the interval of the affected variable to ∞ or $-\infty$. The idea of supporters is to model explictly the condition enabling these asymptotic behaviors. A supporter is similar to an action, with the difference that it can express a numeric interval effect $x = (\underline{x}, \infty)$ or $x = (-\infty, \overline{x})$. As per Definition 5, this effect updates the state to the convex union of the interval x in the current state and $(\underline{x}, \infty)$ or $(-\infty, \overline{x})$, respectively.

Definition 9 (Supporters of Action a)**.** *Each additive numeric effect* $e \in \mathrm{eff}_{\mathrm{num}}(\mathrm{a})$ *generates two supporters* e^{+} *and* e^{-}*: If* $\mathrm{op}(e)$ *is* $+{=}$*, then* e^{+} *has the precondition* $\mathrm{pre(a)} \cup \{\langle \mathrm{rhs(e)}, >, 0 \rangle\}$ *and the effect* $\mathrm{lhs(e)} = (\underline{x}, +\infty)$*; the precondition of* e^{-} *is* $\mathrm{pre(a)} \cup \{\langle \mathrm{rhs(e)}, <, 0 \rangle\}$ *and the effect* $\mathrm{lhs(e)} = (-\infty, \overline{x})$*. The supporters generated by a decrease effect are defined analogously, but with the left-hand sides of the effects swapped. A constant assignment effect* $x = k$ *generates only one supporter, with precondition* $\mathrm{pre(a)}$ *and the effect unchanged.*

Note that in actual numeric planning, assigning an interval is not possible. The supporters are used only in the AIBR to compactly represent application of an effect an arbitrary number of times.

From the set of actions $\mathcal{A}$ we generate a set of supporters Ω using this mechanism. The size of Ω is no more than $2n|\mathcal{A}|$, where n is the maximum number of effects per action. Supporters have no cyclic dependencies, according to Definition 6, because their effects are all constant assignments. The dependencies are instead captured by their

preconditions. We can therefore compute asymptotic relaxed reachability in a manner analogous to relaxed planning graph construction over Ω, as described in the next section.

5.2 Asymptotic Relaxed Planning Graph

This section describes the construction of the Asymptotic Relaxed Planning Graph (ARPG), $\mathbb{G}$, from the set of supporters. As in the classical relaxed plan graph construction, $\mathbb{G}$ is a digraph of alternating interval (corresponding to "fact") and supporter (corresponding to "action") layers. The construction of $\mathbb{G}$ starts with the unit intervals corresponding to the initial state of the planning task, and expands it with a supporter layer containing all supporters in Ω whose preconditions are relaxed satisfied, followed by a new interval layer updated with the effects of the applied supporters. The process iterates until either the goal condition is relaxed satisfied in the last interval layer, or no new supporters can be added (in which case the goal is unreacheable). Algorithm 1 formalises the process. With slight abuse of notation, we write $\mathrm{succ}^+(s^+, \mathcal{S})$ for the (relaxed) state that results from simultaneously applying all actions in supporter set $\mathcal{S}$. This is well-defined since all supporters are constant assignments and the successor state is defined by taking the convex union.

Algorithm 1: Asymptotic Relaxed Planning Graph (ARPG)

Input: Π^{++}
Output: Is $\mathcal{G}$ reachable?
$\Omega =$ supporters of $\mathcal{A}$.
$s^+ = s_0^+$.
$\mathcal{S} = \{a \in \Omega : s^+ \models \mathrm{pre(a)}\}$
while $\mathcal{S} \neq \emptyset$ *and* $s^+ \not\models \mathcal{G}$ **do**
  $s^+ = \mathrm{succ}^+(s^+, \mathcal{S})$
  $\Omega = \Omega \backslash \mathcal{S}$
  $\mathcal{S} = \{a \in \Omega : s^+ \models \mathrm{pre(a)}\}$
return $s^+ \models \mathcal{G}$

Figure 2 shows an example of the conversion from actions to supporters, and a sketch of the asymptotic reachability analysis.

Proposition 2 (Termination)**.** *The ARPG construction terminates.*

Proof Sketch: The set of supporters is finite, so, since repetitions are not allowed, the number of layers that can be built is finite. In the worst case, the construction will terminate when all applicable supporters have been tried.

Proposition 3 (Safeness)**.** *If* ARPG(Π^{++}) *returns False, then* $\mathcal{G}$ *is unreachable in* Π.

Proof Sketch: Π^{++} is a proper relaxation of Π. Assume $\mathcal{G}$ is reachable, by a plan $\pi = b_0, ..., b_m$ that takes s_0 to a state s' such that $s' \models \mathcal{G}$. The plan is executable in Π^{++} (by monotonicity) and the goal is relaxed satisifed in its final state. From the relaxed execution of π, we can extract the corresponding supporters that are used; as these are relaxed applicable, the ARPG construction returns True.

5.3 Heuristic Estimation

The ARPG construction decides only whether $\mathcal{G}$ is reachable, in the relaxed problem, by application of some, arbitrarily large, number of actions. To compute an estimate of how many actions, we can proceed to build an RPG in the standard fashion, applying (sets of)

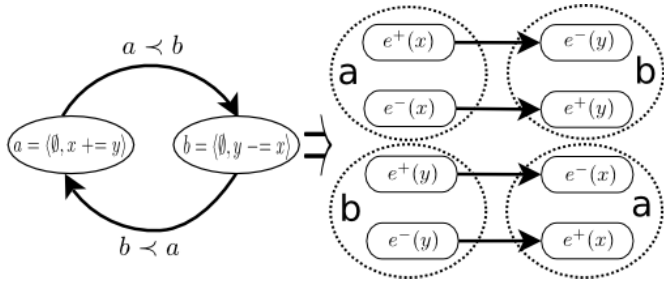

Figure 2: Left: Cyclic dependency ($\prec$) example. Action a depends on y which is modified by b; b depends on x which is modified by a. Right: Supporters generated for the two actions. Dependencies between supporters are due to the added preconditions (e.g., $pre(e^+(x)) = \{y > 0\}$). If $s_0 = \langle x = -5, y = -5 \rangle$, supporters in the first layer are $e^+(y)$ and $e^-(x)$. Asymptotically all values of x and y are reachable both in the AIBR and in the original problem. If we want to reach goal $y < -100$ in the AIBR we have to apply first $e^+(y)$ (i.e., using action b) then $e^+(x)$ (i.e., action a), and then $e^-(y)$ (i.e., action b again).

relaxed applicable actions until the goal is reached and then counting how many were needed. Algorithm 2 shows the relaxed plan computation procedure, for the sake of completeness. Because the goal has already been determined to be relaxed reachable, this process is guaranteed to terminate in a finite number of steps.

Algorithm 2 simply counts the number of actions applied before the goal is achieved. This is the heuristic estimate used in our experiments. Although it can obviously overestimate the size of the relaxed plan substantially, it is very simple to implement and still provides effective heuristic guidance, as shown in Section 7. The heuristic could be made more accurate by removing from $\mathcal{A}'$ actions that do not contribute to achieving $\mathcal{G}$, in a way that is similar to relaxed plan extraction. How to do that efficiently and in a principled way, as well as how to find an admissible estimate, is a question for future work.

Algorithm 2: Compute AIBR Estimate

Input: Π^{++}
Output: Integer
$\mathcal{A}' = \emptyset$
$s = s_0^+$
Loop
  foreach $a \in \mathcal{A}$ **do**
    if a *is relaxed applicable in* s **then**
      $s = \mathrm{succ}^+(s, a)$
      $\mathcal{A}' = \mathcal{A}' \cup \{a\}$
      if $s \models \mathcal{G}$ **then**
        return $|\mathcal{A}'|$

6 Planning with Autonomous Processes in PDDL+

Sequential numeric planning is a crucial building block to supporting more expressive planning formalisms, such as planning with autonomous processes as captured in PDDL+ [12]. In the theory of waiting, proposed by McDermott [22] for planning with autonomous processes, instantaneous actions (the agent decisions) are interleaved with waiting (environment evolution). During a waiting action, the agent observes the evolution of the world for a potentially unbounded

amount of time. During this time, the state changes according to the accumulated, *additive*, effect of active processes. The activation of processes is determined by their conditions on the evolving state.

In principle, the amount of time an agent must wait in a given situation can be arbitrarily large or small. Waiting too long it may miss a window of opportunity to take some action whose precondition is only briefly satisfied, or the dynamics of the world may change as its evolution triggers a change in the set of active processes. Fixing the minimum waiting time to a constant δt results in a time-discretised approximation of the model, and a sequential numeric planning problem. The domain presented in Section 2 is an example of such a model. As shown by Löhr et al. [20, 21], this approximation is good enough to solve a number of challenging realistic hybrid control problems. The "discretise-and-validate" approach to planning with processes, exemplified by the UPMurphi planner [8], also has planning for a time-discretised model at its core, and embeds it into an iterative scheme in which the proposed plan is checked against the continuous-time model and the discretisation refined if it is found to be invalid. Importantly, the time-discretisation of a process model typically results in a sequential planning problem with complex numeric conditions and effects. UPMurphi solves this problem by blind search; a recently developed successor system, DiNo [24], uses a heuristic based on relaxed planning graphs.

We focus here on planning for time-discretised models with instantaneous actions, processes and global constraints[2]. A global constraint is a boolean formula over propositional and numeric conditions that must be satisfied in every state along the plan trajectory, and they are equivalent to *always* constraints of PDDL. We use the global constraint mainly to model the equivalent of PDDL+ events. In many PDDL+ domains, events restrict particular plan trajectories by leading to dead-end states (e.g., blowing up the engine in the CAR domain), playing a role similar to global constraints. Whether events do in fact make the language more expressive than global constraints alone can is an open question, but they are known to be the cause of significant modeling and computational complications (e.g., infinite cascades of events [10]).

We solve the sequential numeric planning problem with a best-first search using the AIBR heuristic. Because the AIBR is monotonic, applying the effect of processes in the relaxed problem becomes optional rather than mandatory (i.e., a "may" instead of a "must" semantic). Likewise global constraints, which are always satisfied in the state being evaluated, cannot be invalidated in the relaxed problem. In spite of these approximations, the heuristic provides effective guidance, as shown in the next section.

7 Experiments

To evaluate the effectiveness of the AIBR heuristic, we perform experiments on a set of time-discretised PDDL+ domains involving autonomous processes and global constraints. We did not consider IPC numeric planning benchmarks. These have only simple (linear or constant) conditions and effects, which are already known to be handled, in most cases quite effectively, by heuristics based on the interval relaxation. Next, we present the setting and domains used, then a summary of results, an in-depth analysis for selected domains, and a comparison with two state of the art numeric planners.

[2] As observed by Fox et al. [12], durative actions can be compiled away using two instantaneous actions modeling the start and the stop of a process, which in turns captures the continuous effects of the durative action.

7.1 Setting

We implemented a numeric planner using best-first search on $f(n) = g(n) + h(n)$, where $h(n)$ is the AIBR heuristic. As in McDermott's planner [22], branching is on both (sets of) instantaneous action(s) and waiting. The waiting time is fixed to a constant δ_t. The successor state of the waiting realises the cumulative effect of all processes active at that particular moment. The planner is correct and complete for the time-discretised problem; of course, like UPMurphi [8] without the validation, the approach is incomplete for the continuous-time PDDL+ semantics (e.g., we can miss opportunities during the δ_t), and can produce invalid plans as we do not check zero-crossings within the waiting time (e.g., there could be other processes triggered, or global constraints invalidated in that interval). Since our heuristic can overestimate the actual distance to the goal, it is inadmissible and can produce suboptimal plans.

The implementation is in JAVA 1.8, and experiments were run on Ubuntu 14.04 on an Intel I5-vPro with 8 GB of memory. The planner is called Expressive Numeric Heuristic Search Planner (ENHSP), and will be publicly available.

7.2 Planning Domains

The large majority of our test domains require reasoning about autonomous non-linear processes. They include the well-known PDDL+ domains CAR [12] and GENERATOR [3], in both their linear and non-linear versions, the CONVOYS domain introduced by McDermott [22], and two new domains INTERCEPT and HVAC. To these, we add a challenging pure sequential numeric planning domain called COMPLEXPOURING.

Both CAR domain versions involve instantaneous accelerate/decelerate actions and processes updating the distance driven and speed of the car. The non-linear version features a new process, which is active when the speed v is positive and additionally accounts for the drag k, according to $\frac{dv}{dt} = -k \cdot v^2$. Like Bryce et al. [3], we test the scalability of our approach by increasing the maximum acceleration/deceleration (from 1 to 8) to enlarge the branching factor .

GENERATOR describes power generation by an engine which consumes fuel and may need (concurrent) refilling from various fuel tanks. The linear [12] and non-linear [3] versions both have turn-on/turn-off generator and start/stop refuelling actions but different refuelling dynamics. In the non-linear case, the rate of increase of the fuel level f caused by refuelling is given by $\frac{df}{dt} = \alpha.t^2$ where α is a constant and t the refuelling time. Problem instances differ by increasing the number of tanks needed to achieve the goal of generating power for a given duration. Dead-ends appear whenever the planner delays refuelling for too long or when refueling is impossible due to the lack of space in the generator's tank. As in previous work [3], the largest instance has 8 tanks, all of which are needed.

CONVOYS features a set of convoys that must reach specific locations starting from their initial positions. The time needed by convoy c to move between two positions a and b on the map depends on the speed v_c of the convoy and on the inverse of the square of the traffic $T_{a,b}$ in that specific road segment. There is a process per convoy and road segment, which updates the distance travelled according to $\frac{dD_{c,a,b}}{dt} = -\frac{v_c}{T^2_{a,b}}$ as long as the segment end point is not reached. Instances in this domain involves up to 8 locations and 4 convoys.

INTERCEPT involves a vampire v and a bird b in the two-dimensional space $\mathbb{Q}^2$. Two processes describe their movement, given as a function of the speed in the two dimensions. The task is to launch the vampire (initially stationary) at the right time and with

the right speed for it to intercept the bird. This happens when the Euclidean distance between the two is lower than a given radius r, which is encoded using the constraint $(x_v - x_b)^2 + (y_v - y_b)^2 \leq r^2$. Instances scale according to the initial distance between the vampire and the bird, and by the radius defining the goal condition.

HVAC is an abstraction of a ventilation, air-conditioning and cooling (HVAC) control problem [19][3]. For each zone l in a building, two HVAC control parameters, the supply air flow rate a_l^{SA} and supply air temperature T_l^{sa}, must be adjusted from time to time so as to ensure that the zone temperature T_l meets given constraints. These constraints reflect the schedule of activities occuring at the zone. In our abstraction, there are instantaneous actions to increase/decrease T_l^{SA} and a_l^{SA} by a given amount. A single process updates the actual time, and a process per zone computes the rate of temperature change in this zone as $\frac{dT_l}{dt} = \beta.a_l^{SA}.(T_l^{SA} - T_l)$, where β is a constant. Time in this representation is implicitly captured by the progress of the plan, in contrast to the MIP formulation presented in [19], where each variable is explicitly located on the timeline. Moreover, since nonlinear optimisation solvers struggle with this problem, the MIP formulation linearises the bilinear terms in the above equation. Instances in this domain are generated by increasing the number of scheduled activities from 1 to 16.

COMPLEXPOURING is the problem of filling buckets using water tanks arranged in a complex directed network. Each tank and bucket has a given capacity and initial volume of water. This is a purely sequential problem which has no processes but just an action that enables the transfer of water from one container to a connected one. The transfer is modelled by Toriccelli's law which states that the volume V of water left in a tank with an initial volume U of water after t seconds with an open tap is $V = (-kt - \sqrt{U})$, with $t \in (0, \frac{\sqrt{U}}{k})$, where k is a constant that depends on the cross-sectional area of the tank, the size of the tap, and gravity [18]. In our formulation this law is discretised to model the change happening after 1 second. In order to achieve the goal, we may have to transfer water from different tanks into intermediary ones, and so on. In this domain, we study the complexity arising from two different sources, leading to two different instance sets. In the first set, tanks arranged in a flat structure all feed into a single bucket and we vary the number of tanks from 2 to 10. In the second set, we vary the structure of the network of tanks, which includes from 3 to 10 tanks and 1 or 2 buckets to fill.

7.3 Summary of Results

Table 1 reports number of instances (I), coverage (C), run time (T), plan length (PL), and number of expanded nodes (Exp) for all the domains described in the previous section. The planner was able to solve all the instances provided.

Domain	I	C	T	PL	Exp
CAR	8	8	0.1	17.2	30.6
CAR (NL)	8	8	0.3	20.8	353.9
GENERATOR	8	8	3.7	1005.5	1006.5
GENERATOR (NL)	8	8	4.9	1005.5	1006.5
CONVOYS	6	6	25.1	23	935
INTERCEPT	10	10	1.5	114.4	2750.9
HVAC	16	16	16.5	110.6	9235.4
COMPLEXPOURING	17	17	4.9	14.1	1693

Table 1: Overall picture of the experimental results. PDDL+ domains ran with $\delta_t = 1$ (apart from INTERCEPT, which requires $\delta_t = 0.1$ to make the instances solvable). Time-out is set to 1800 seconds. Plan Length includes both actions and waiting actions.

[3] We only consider the control part of the problem and ignore external influences such as the temperature of adjacent rooms.

	With Heuristic Guidance				Blind Search		
	Exp	Eval	PL	T	Exp	PL	T
1	34	68	20	0.10	359209	20	96
2	73	148	20	0.14	NA	NA	NA
3	388	948	21	0.34	NA	NA	NA
4	448	1002	21	0.36	NA	NA	NA
5	472	1158	21	0.39	NA	NA	NA
6	472	1158	21	0.39	NA	NA	NA
7	472	1158	21	0.41	NA	NA	NA
8	472	1158	21	0.41	NA	NA	NA

(a) Non-Linear CAR

	With Heuristic Guidance				Blind Search		
	Exp	Eval	PL	T	Exp	PL	T
1	1003	1005	1002	2.70	NA	NA	NA
2	1004	1009	1003	3.05	NA	NA	NA
3	1005	1014	1004	3.77	NA	NA	NA
4	1006	1020	1005	4.60	NA	NA	NA
5	1007	1027	1006	5.01	NA	NA	NA
6	1008	1035	1007	5.93	NA	NA	NA
7	1009	1044	1008	6.86	NA	NA	NA
8	1010	1054	1009	7.65	NA	NA	NA

(b) Non-Linear GENERATOR

	With Heuristic Guidance				Blind Search		
	Exp	Eval	PL	T	Exp	PL	T
1	8	7	7	0.02	7	7	0.06
2	12	22	6	0.05	64	6	0.15
3	386	688	13	0.54	NA	NA	NA
4	1389	1651	15	1.85	NA	NA	NA
5	16	30	8	0.05	269	8	0.27
6	7	12	6	0.05	66	6	0.17
7	26	72	11	0.86	88661	11	30.2
8	204	742	14	1.03	NA	NA	NA
9	309	1131	22	1.01	NA	NA	NA
10	50	144	11	0.19	88563	11	19.7
11	429	1606	16	1.15	NA	NA	NA
12	408	1889	19	1.50	NA	NA	NA
13	1163	6097	21	3.25	NA	NA	NA
14	18356	109777	24	51.79	NA	NA	NA
15	5933	33500	22	19.38	NA	NA	NA
16	35	171	14	0.82	NA	NA	NA
17	50	363	10	1.22	NA	NA	NA

(c) COMPLEXPOURING

Table 2: Detailed results

The low average number of expanded nodes in the domain is indicative of heuristic effectiveness; the next section provides more details. Many domains have non-linear dependencies among numeric variables, and some require (implicit) temporal reasoning (e.g., GENERATOR, INTERCEPT). The heuristic is more inaccurate in these domains, especially HVAC which has many global constraints. Since AIBR is monotone, it does not capture negative effects on them.

7.4 In-Depth Analysis

Next, we present a more in-depth analysis of heuristic guidance for a subset of domains. AIBR informativness and impact on plan length is evaluated comparing it with blind search ($f(n) = g(n)$, i.e, uniform-cost search). We used uniform-cost search rather than uninformed depth-first search since the latter only solved a few instances of one domain (with plans 10–100 times longer). Since, the search space is usually infinite, depth-first search often fails to terminate also for solvable instances.

Car domain. Table 2a shows the number of expanded (Exp) and evaluated nodes (Eval), plan length (PL) and run-time (T) for the non-linear instances. The increasing spectrum of possible accelerations in instances 1 to 8 only marginally affects ENHSP's runtime. Blind search finds minimal-length plans, but is not able to scale up beyond instance 1. The plans produced using the heuristic, whilst suboptimal, are still of a good quality.

Non-linear Generator. Table 2b shows data for each instance. Here the number of necessary plan steps depends on the time the generator has to run, and on the number of tanks that must be used. With a

shortest plan length of about a 1000 actions and an average branching factor of 3 actions, even the smallest instance has a huge search space. Despite this, the AIBR heuristic solves all instances, whilst blind search solves none. Surprisingly, the heuristic overestimation favours states where fewer actions are possible (a tank can be used just once in this domain). This causes planner to favour refuelling over waiting, thus avoiding the generator running out of fuel.

Complex Pouring. Table 2c reports the data for each instance. Instances 1-9 belong to the set of instances with a flat network structure (see Subsection 7.2) The largest instance (instance 9) has a larger volume of liquid to be poured so as to increase the length of plans. Instances 10-16 involving a more complex network of tanks. The depth of the network affects the length of plans. Blind search was able to produce plans within the timeout but only for a subset of these instances. This is because, except for the instances requiring more than 11 actions, the state space is small enough to be blindly explored. Interestingly, none of the plans produced with the heuristics is longer than the plan produced by blind search. It turns out that, despite its inadmissibility, our heuristic leads to optimal plans in this domain.

ENHSP outperforms blind search on the other domains. Blind search only solved instances from the linear version of CAR (expanding on the average 387605 nodes), and 3 instances of CONVOYS (average expanded nodes: 10621).

7.5 Comparative Analysis

We compared ENHSP with dReal [3] and UPMurphi [8], both of which follow the planning via model-checking paradigm. Neither of them is strictly designed for sequential numeric planning, but they were the closest comparable planners that we were aware of and that were available at the time of writing. Another planner with similar capabilities was proposed by Bajada et al. [2], and two more systems for planning with non-linear processes have only recently become available [4, 24]. Examining their comparative performance, and that of ENHSP on newly proposed benchmarks [24], is a question for future work.

Table 3 reports the planners' runtimes on the CAR and GENERATOR domains. The UPMurphi runtimes were obtained by running the latest version of the planner[4] in its standard configuration with a discretisation step of 1 sec, on the same PDDL+ formalisation the we use for ENHSP. The dReal runtimes are taken from Bryce et al.'s paper [3]. This is because dReal does not support PDDL with processes natively; a model-checking formulation of each instance has to be written manually, which includes a mode for each possible combination of processes (note that there could be an exponential number of modes to consider). We strived to make our PDDL+ formulations as close as possible to those in the dReal distribution[5].

dReal uses some heuristic guidance, but the heuristic is intended to estimate the number of jumps between different modes, and ignores the numeric part of the problem. UPMurphi automatically translates PDDL+ to the model-checking formulation, but unlike dReal, it uses blind search and is not guided by any heuristic. Table 3 shows that the AIBR heuristic provides better search guidance than the mechanisms employed by these two planners.

However, these results should only be considered as indicative, rather than as a fair comparison. There are several differences between the three planners which must be taken into account. For instance, dReal is an approximate planner where preconditions and goals are satisfied up to an approximation error. ENHSP and UPMurphi may also produce unsound plans, but for a different reason, due to discretisation of time and lack of zero-crossing check. To remedy this would require introducing a plan validation step, and subsequent refinement of the discretisation [8]. dReal only finds plan with a bounded number of steps, specified in advance, so in general must be run several times, with increasing step bounds. The CPU times in Table 3 are for a single run only, in which the planner is given the optimal number of steps for each problem. (CPU times for the whole process, on some instances, can be found in [4].) In contrast, ENHSP does not require any a priori bound.

Instance	1	2	3	4	5	6	7	8
CAR								
ENHSP	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
dReal	1.1	1.2	1.2	1.2	1.2	1.3	1.3	1.2
UPMur.	0.3	0.6	1.0	1.5	1.8	2.1	2.3	2.6
NLCAR								
ENHSP	0.1	0.1	0.3	0.4	0.4	0.4	0.4	0.4
dReal	16.7	16.7	16.3	16.8	16.6	16.8	17.4	16.6
UPMur.	4.9	42.7	84.3	113	139	171	187	204
GENERATOR								
ENHSP	1.9	2.6	2.7	3.1	4.0	4.4	5.1	5.6
dReal	3.1	15.6	134.7	1699	TO	TO	TO	TO
UPMur.	113.4	MO	MO	MO	MO	MO	MO	MO
NLGENERATOR								
ENHSP	2.7	3.1	3.8	4.6	5.0	5.9	6.9	7.7
dReal	12.8	71.6	1696	TO	TO	TO	TO	TO
UPMur.	128	MO	MO	MO	MO	MO	MO	MO

Table 3: Comparison of the run times (in sec.) of ENHSP, dReal and UPMurphi. Each instance can end with a solved situation, timeout (TO), or out of memory (MO).

8 Conclusions

The interval-based relaxation is the natural extension of the principle of monotonic (delete-free) relaxation to numeric planning [17], and the basis of most numeric planning heuristics [13, 6, 7, 5, 24]. However, these apply only to restricted problems, for example having linear conditions and effects. Up to now, it has not been clear if (asymptotic) reachability in the relaxation is even decidable for unrestricted problems, which may have cyclic dependencies [1]. We have shown that relaxed reachability is computable, through a novel asymptotic relaxed planning graph formulation, for problems with additive action effects, and moreover that non-additive state-dependent effects can be removed by a syntactic problem transformation, at the price of a further relaxation. The resulting additive interval-based relaxation (AIBR) and heuristics derived from it are pruning-safe.

Sequential numeric planning problems with complex non-linear effects arise naturally as time-discretisations of planning in the presence of autonomous continuous-time processes [8, 20], and efficient and general numeric planners are a key building block for supporting more expressive forms of planning. Evaluation of a planner using a heuristic obtained from the AIBR on various non-linear sequential planning problems and time-discretised PDDL+ problems with global constraints showed it to be competitive with some planners designed for such problems [3, 8].

In addition to extending the comparison to other recent planners, our future work is aimed at understanding how to make the heuristic more accurate and/or better integrated with the search engine used, for example through the extraction of helpful actions [17].

Acknowledgements This work is supported by ARC project DP140104219, "Robust AI Planning for Hybrid Systems". NICTA is funded by the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

[4] Available at `http://github.com/gdellapenna/UPMurphi`
[5] `http://github.com/danbryce/dreal`

REFERENCES

[1] Johannes Aldinger, Robert Mattmüller, and Moritz Göbelbecker, 'Complexity of interval relaxed numeric planning', in *Proc. of KI 2015: Advances in Artificial Intelligence*, pp. 19–31, (2015).

[2] Josef Bajada, Maria Fox, and Derek Long, 'Temporal planning with semantic attachment of non-linear monotonic continuous behaviours', in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pp. 1523–1529, (2015).

[3] Daniel Bryce, Sicun Gao, David J. Musliner, and Robert P. Goldman, 'Smt-based nonlinear PDDL+ planning', in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pp. 3247–3253, (2015).

[4] Michael Cashmore, Maria Fox, Derek Long, and Daniele Magazzeni, 'A compilation of the full PDDL+ language into SMT', in *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, (2016).

[5] Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long, 'Colin: Planning with continuous linear numeric change', *Journal of Artificial Intelligence Research (JAIR)*, **44**, 1–96, (2012).

[6] Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long, 'A hybrid LP-RPG heuristic for modelling numeric resource flows in planning', *Journal of Artifificial Intelligence Research (JAIR)*, **46**, 343–412, (2013).

[7] Amanda Jane Coles, Andrew I. Coles, Maria Fox, and Derek Long, 'Forward-chaining partial-order planning', in *Proc. of International Conference on Automated Planning and Scheduling (ICAPS-10)*, (2010).

[8] 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, Thessaloniki, Greece, September 19-23, 2009*, (2009).

[9] Patrick Eyerich, Robert Mattmüller, and Gabriele Röger, 'Using the context-enhanced additive heuristic for temporal and numeric planning', in *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*, (2009).

[10] Maria Fox, Richard Howey, and Derek Long, 'Validating plans in the context of processes and exogenous events', in *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pp. 1151–1156, (2005).

[11] 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).

[12] Maria Fox and Derek Long, 'Modelling mixed discrete-continuous domains for planning', *J. Artif. Intell. Res. (JAIR)*, **27**, 235–297, (2006).

[13] Alfonso Gerevini, Ivan Saetti, and Alessandro Serina, 'An approach to efficient planning with numerical fluents and multi-criteria plan quality', *Artificial Intelligence*, **172**(8-9), 899–944, (2008).

[14] Peter Gregory, Derek Long, Maria Fox, and J. Christopher Beck, 'Planning modulo theories: Extending the planning paradigm', in *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, (2012).

[15] Malte Helmert, 'Decidability and undecidability results for planning with numerical state variables', in *Proc,. of International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*, pp. 44–53, (2002).

[16] Malte Helmert and Hector Geffner, 'Unifying the causal graph and additive heuristics', in *Proc. of the 18th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 140–147, (2008).

[17] Jörg Hoffmann, 'The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables', *Journal of Artificial Intelligence Research (JAIR)*, **20**, 291–341, (2003).

[18] Richard Howey and Derek Long, 'VALs progress: The automatic validation tool for PDDL2.1 used in the international planning competition'.

[19] BoonPing Lim, Menkes van den Briel, Sylvie Thiébaux, Scott Backhaus, and Russell Bent, 'Hvac-aware occupancy scheduling', in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pp. 679–686, (2015).

[20] Johannes Löhr, Patrick Eyerich, Thomas Keller, and Bernhard Nebel, 'A planning based framework for controlling hybrid systems', in *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS)*, (2012).

[21] Johannes Löhr, Patrick Eyerich, Stefan Winkler, and Bernhard Nebel, 'Domain predictive control under uncertain numerical state information', in *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS)*, (2013).

[22] Drew V. McDermott, 'Reasoning about autonomous processes in an estimated-regression planner', in *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003), June 9-13, 2003, Trento, Italy*, pp. 143–152, (2003).

[23] Ramon E. Moore, R. Baker Kearfott, and Michael J. Cloud, *Introduction to Interval Analysis*, SIAM, 2009.

[24] Wiktor Piotrowski, Maria Fox, Derek Long, Daniele Magazzeni, and Fabio Mercorio, 'Heuristic planning for PDDL+ domains', in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI), New York (NY), USA, July 9-15*, (2016). Shorter version in Proc. AAAI-16 Workshop on Planning for Hybrid Systems.