Interval Based Relaxation Heuristics for Numeric Planning with Action Costs

Johannes Aldinger and Bernhard Nebel

Department of Computer Science, University of Freiburg, Freiburg, Germany
{aldinger,nebel}@informatik.uni-freiburg.de

Abstract. Many real-world problems can be expressed in terms of states and actions that modify the world to reach a certain goal. Such problems can be solved by automated planning. Numeric planning supports numeric quantities such as resources or physical properties in addition to the propositional variables from classical planning. We approach numeric planning with heuristic search and introduce adaptations of the relaxation heuristics $h_{\max}$, h_{add} and h_{FF} to interval based relaxation frameworks. In contrast to previous approaches, the heuristics presented in this paper are not limited to fragments of numeric planning with instantaneous actions (such as linear or acyclic numeric planning tasks) and support action costs.

1 Introduction

Whereas domain-independent planning has proven successful, numeric quantities such as physical properties (e.g. velocity) and resources (e.g. fuel level) cannot be modeled in classical planning. As many real world problems feature numeric quantities, we aim to advance domain-independent planning by including numeric quantities, leading to *numeric planning*.

The performance of applying informed search algorithms such as hill-climbing or best-first search to planning depends on the quality of the underlying heuristic. One challenge is to design good heuristic estimators for numeric planning. We are interested in adapting the forward chaining *delete relaxation* heuristics $h_{\max}$, h_{add} and h_{FF} to numeric planning. The *delete relaxation* from classical planning ignores negative interactions between actions by neglecting delete effects: effects that falsify propositional variables. As such the set of achieved propositions grows monotonically, and heuristics can be computed in polynomial time. For numeric planning, intervals offer a way to compactly represent an over-approximation of arbitrarily many values that can be achieved by a variable.

Another challenge is that numeric actions are non-idempotent operations: applying the same numeric effect to a state more than once can yield a new distinct successor every time. This makes heuristics based on a basic *interval relaxation* only polynomial in the length of a shortest relaxed plan [13]. Recently, Aldinger et al. [2] proposed a more sophisticated interval-based relaxation framework for

numeric planning that captures arbitrarily many repetitions of applying a numeric action in one step. The plan existence problem in this *repetition relaxation* is polynomial in the input for tasks with acyclic dependencies.

Previous work on numeric relaxation heuristics is either restricted to a fragment of numeric planning, e.g. Metric FF [13] is restricted to linear tasks whereas MIPS [9], Colin [8] and ENHSP [18] only deal with uniform action costs. The $h_{\max}$ and h_{add} variants of Scala et al. [17] correspond roughly to the *planning graph* approach with a *repetition relaxation* in this paper. We are interested in adaptations which offer heuristic guidance for *all* numeric planning tasks with instantaneous actions including actions with non-linear effects and non-uniform action cost. Notably, we are also interested in computing a h_{FF}-like heuristic, i.e., a heuristic basing its estimate on the extraction of valid relaxed plans.

In this paper, we explore the design space of numeric relaxation heuristics with regard to two relaxation methods (*interval* or *repetition* relaxation), two methods of aggregating heuristic costs (*max* and *sum*), and two search techniques for relaxed reachability (a *planning graph method* and *priority queues*). We identify tractable combinations and derive heuristics which also take action costs into account. We propose a new method to handle tasks with cyclic dependencies in the numeric effects which differs from Scala et al. [18]. Finally we present a generalization to the marking method of relevant operators used by h_{FF}, which explicates target values in the intervals to extract relaxed plans.

2 Interval Relaxation

A delete relaxation is a simplification of a planning instance where facts which are achieved once remain achieved. Thus, the set of achieved values grows monotonically. In relaxed classical planning, actions are idempotent and therefore, this growth is bounded by the number of actions in the planning task. In numeric planning, actions are non-idempotent operations and the number of values a variable can attain is unbounded even by executing a single action repeatedly. Aiming towards a tractable relaxation for numeric planning, intervals are an obvious choice to represent the achieved values of a variable, as they offer a compact representation. Furthermore, the number of action applications can be restricted as well. We discuss two relaxation frameworks that approach this challenge differently.

The depth of a relaxed planning graph is restricted to the length of a shortest relaxed plan, which allows us to compute an *interval relaxed* planning graph very much like in classical planning. However, desired heuristic properties (such as admissibility for $h_{\max}$) can not be guaranteed in this framework. The *repetition relaxation* uses a semi-symbolic representation of intervals to simulate the behavior of arbitrary many action repetitions at once. This makes relaxed actions idempotent, and the plan existence problem can be decided in polynomial time. In this section, we give a condensed overview of these relaxation frameworks, borrowing notation from Aldinger et al. [2].

Interval arithmetic [19] uses an upper and a lower bound to enclose the actual value of a number. Intervals $[\underline{x}, \overline{x}]$ contain all rational numbers from $\underline{x}$ to $\overline{x}$. We refer to the lower bound of an interval x by $\underline{x}$ and to the upper bound by $\overline{x}$. Intervals can be *closed* or *open* and we denote closed interval bounds by brackets $[\underline{\cdot}, \overline{\cdot}]$ and open interval bounds by parentheses $(\underline{\cdot}, \overline{\cdot})$.

A numeric planning task $\Pi = \langle \mathcal{V}_P, \mathcal{V}_N, \mathcal{A}, \mathcal{I}, \mathcal{G}, \gamma \rangle$ is a 6-tuple, where $\mathcal{V}_P$ is a set of propositional variables with domain $\{true, false\}$, $\mathcal{V}_N$ is a set of numeric variables with domain $\mathbb{Q}^\infty := \mathbb{Q} \cup \{-\infty, \infty\}$, $\mathcal{A}$ is a set of actions, $\mathcal{I}$ the initial *state*, $\mathcal{G}$ a goal *condition* and $\gamma : \mathcal{A} \to \mathbb{Q}^+$ is a function assigning a strictly positive rational cost to each action. A *state* is a (full) mapping from variables $\mathcal{V} := \mathcal{V}_P \cup \mathcal{V}_N$ to values from their respective domain. A *numeric expression* $(\xi_1 \circ \xi_2)$ is an arithmetic expression with operators $\circ \in \{+, -, \times, \div\}$ and expressions ξ_1 and ξ_2 recursively defined over variables $\mathcal{V}_N$ and constants from $\mathbb{Q}$. A *numeric constraint* $(\xi \trianglerighteq 0)$ compares numeric expressions ξ to 0 with $\trianglerighteq \in \{\geq, >, =\}$ and a (goal or action) *condition* is a conjunction of propositions and *numeric constraints*. *Numeric effects* are assignments $(v \circ\!= \xi)$ where v is a variable from $\mathcal{V}_N$, $\circ\!= \in \{:=, +=, -=, \times=, \div=\}$ and ξ is a *numeric expression*. Actions in $\mathcal{A}$ have the form $\langle \text{pre}, \text{eff} \rangle$ and consist of a *condition* pre and a set of *effects* eff containing at most one truth assignment for each propositional variable and at most one *numeric effect* for each numeric variable.

The semantic of a numeric planning task is straightforward: conditions are satisfied in a state if all propositions evaluate to *true* and all numeric constraints are satisfied, where numeric expressions are evaluated recursively. The evaluation of expression ξ in state s is denoted by $s(\xi)$. Actions are *applicable* in a state, iff the precondition is satisfied and none of its effects causes a division by zero. The successor state s' obtained by applying an action in state s is s, expect for variables that appear in an *effect*. Propositional variables are assigned the new truth value and the values $s'(v_n)$ of each numeric variable v_n are evaluations of the expression on the right hand side of the effect in s applied to $s(v_n)$ according to the assignment operator. A *plan* π is a sequence of consecutively applicable actions that leads from $\mathcal{I}$ to a state satisfying $\mathcal{G}$.

We consider two relaxation frameworks: the *interval relaxation* and the *repetition relaxation*. Syntactically, they differ only slightly from the unrelaxed task. The domains of numeric variables $\mathcal{V}_N$ are now intervals. The interpretation of constants and the values of variables in the initial state are degenerate (one element) intervals. The semantics of both relaxations are based on interval arithmetic. Numeric expressions evaluate to intervals, and numeric constraints are satisfied, if numbers exist within these intervals that satisfy the constraint. Propositional effects that would set a variable to *false* are ignored.

Numeric effects ensure monotonicity by use of the convex union $r = x \sqcup y$ where $\underline{r} = \min(\underline{x}, \underline{y})$ and $\overline{r} = \max(\overline{x}, \overline{y})$. The successor state s' obtained by applying an action in s is again s, except for variables v that appear on the left hand side of numeric effects. These variables are mapped to $s'(v) = s(v) \sqcup \text{eval}(v)$, where $\text{eval}(v)$ is a relaxation dependent evaluation of the numeric effect. For the *interval relaxation*, $\text{eval}(v)$ is the evaluation of the effect right-hand side, using

interval arithmetic on the intervals from s. For the *repetition relaxation* $\text{eval}(v)$ is the result of simulating arbitrary many repetitions of the effect in isolation.

The idea behind the *repetition relaxation* is that relaxed actions become idempotent if arbitrary many repetitions are handled at once. The evaluation of arbitrary many action applications does not have to be computed with an actual simulation. More efficiently, it is sufficient to consider the *behavior* of numeric effects. If an *additive* effect ($+=$ and $-=$) extends an interval bound of a variable in a state once, it can extend that bound to any value by applying the action multiple times. The repetition result of an additive effect only depends on whether the evaluated effect interval contains negative or positive numbers, but it does not depend on the amount. For multiplicative effects, the assignment can contract or expand depending on whether the effects intervals contains numbers with absolute value greater or less than 1 and switch signs if it contains negative elements. This observation allows us to decompose the evaluation of numeric effects into *behavior classes* $\mathcal{B} = \{(-\infty, -1), \{-1\}, (-1, 0), \{0\}, (0, 1), \{1\}, (1, \infty)\}$ and then unite these partial effects to obtain the interpretation of $\text{eval}(v)$ in the *repetition relaxation*. Detailed semantics are found in the original paper [2]. We remark that the decomposition into *behavior classes* bases the evaluation $\text{eval}(v)$ on the values of $s(v)$ before the application. As such, $s'(v)$ can hit new behavior classes. As the number of behavior classes is restricted, the repetition relaxed actions are "pseudo-idempotent": they can change up to three times (v has a different behavior if it is negative, zero or positive). Another source of non-idempotence comes from the interaction between actions which becomes especially problematic if these dependencies are cyclic. As such, the authors of the *repetition relaxation* deem it only feasible for tasks with acyclic dependencies.

2.1 Cyclic Dependencies

The interval which is reached by applying a numeric effect $v_n \circ= \xi$ *depends* on the values of all variables in ξ. This dependency relation induces a *dependency graph*. If the *dependency graph* is acyclic, sequences of actions are pseudo-idempotent as the values of the variables stabilize in topological order.

Cyclic dependencies can make sequences of actions non-idempotent. It is an open research question, whether the interval that is reached after repeatedly applying a sequence of actions causing a cycle can be determined in polynomial time. In order to enforce a topology nevertheless, we can break cycles by introducing auxiliary variables. The check for cycles can be done in polynomial time by algorithms checking for connected components in the dependency graph. In the heuristic, we include special cycle breaker actions which can reinsert the values of the auxiliary variables to the higher level original variable in a controlled manner. The implementation of these cycle breaker actions opens design space. Tractable heuristics have to bound the number of reinsertions.

The most coarse cycle breaker action sets changing variables to $(-\infty, \infty)$: an interval which can not be extended thus ensuring idempotence. A little more accurate is to only set an interval bound to infinity if the interval changed into the respective direction. This relates very much to the *additive effects transformation*

[18], which compiles assignments $x := \xi$ into increase effects $x \mathrel{+}= \xi - x$. Both approaches relax cyclic effects even further by assuming that a shifted bound can be extended arbitrary often by the same margin. An advantage of our method is the restriction of cycle handling to the cycle breaker actions, which does not affect the preconditions of all other actions.

3 Numeric Planning Graph Heuristics

We want to solve a *delete relaxed* simplification of a planning problem in order to guide search in the original one. For classical planning, the relaxed plan existence problem is easy: starting from the initial state, we can iteratively apply all applicable actions to the relaxed state in parallel. The procedure terminates when a fix-point is reached. The *relaxed parallel planning graph* [14] is a graph representation of this planning procedure. It consists of alternating state layers of reachable propositions and action layers of actions which are applicable in that propositional state. The forward propagation heuristics h_{add} [6, 5] and its admissible counterpart $h_{\max}$ [4] estimate the cost $\gamma(v_p)$ to achieve the propositions v_p for each achieving action a by $\gamma(v_p) := \min_a(\gamma(v_p), \gamma(a) + \gamma(\text{pre}(a)))$. Propositions that hold in the initial state are initialized to cost $\gamma(v_p) = 0$ and to $\gamma(v_p) = \infty$ otherwise. The action precondition cost $\gamma(\text{pre}(a))$ is an estimate of the cost of a set of propositions. The heuristics differ in how the cost of this set is estimated. For $h_{\max}$, the most expensive proposition cost $\gamma(\text{pre}(a)) := \max_{v_p \in \text{pre}(a)} \gamma(v_p)$ is used, while h_{add} uses the *sum* of all preconditions $\gamma(\text{pre}(a)) := \sum_{v_p \in \text{pre}(a)} \gamma(v_p)$ instead. The h_{FF} heuristic [14] improves on h_{add} by *marking* actions that are required to compute the h_{add} estimate regressively, and as such it computes a relaxed plan, using its cost as h_{FF} estimate.

3.1 Heuristic Estimators for Numeric Planning

We discuss tractable extensions of the heuristics $h_{\max}$, h_{add} and h_{FF} in the two interval based relaxation frameworks introduced in the previous section: the *interval relaxation* and the *repetition relaxation*. In a purely propositional setting, facts can be seen as variable-value pairs, where the value of propositional variables are subsets of $\{\mathit{true}, \mathit{false}\}$. Similarly, a "fact" for numeric planning is a variable-value pair where the values of numeric variables are intervals. In contrast to classical planning, there are infinitely many such variable-value pairs. Tractable heuristics have to restrict the number of considered numeric facts.

Numeric facts occur as *implicit preconditions* of the actions of the planning task. A numeric effect $v \circ\!= \xi$ can reach a certain interval $s'(v)$ by certain combinations of the intervals $s(v)$ and $s(\xi)$ before the application of the action. In general, there are infinitely many possible combinations of $s(v)$ and $s(\xi)$ to reach $s'(v)$ and thus, the pairs $\langle v, s(v)\rangle$ and all pairs of variables $\langle v_e, s(v_e)\rangle$ where v_e appears in the expression ξ are implicit preconditions. Similarly, numeric constraints $\xi \trianglerighteq 0$ can be satisfied by infinitely many target values $q_e \in s(\xi)$

satisfying $q_e \trianglerighteq 0$. These target values q_e can be reached by combinations of values of the variables in ξ, making them implicit preconditions, too.

A numeric relaxation heuristic has to ensure that first, values in the precondition enable the required values in the effect and second, that the number of considered numeric facts is bounded. We discuss a *planning graph* based and a *priority queue* based approach to restrict this number, both of which are motivated by the method of computing the cost formulas in classical planning.

The first approach is based on the parallel *planning graph* representation for relaxed planning. The considered facts are restricted to one variable-value pair for each variable in the state layers of the planning graph. For interval based planning, several parallel actions can alter the same variable, in which case the convex union of the individual results is used. The cost of a variable-value pair is again the cost of the cheapest achiever, and the cost of a set of such numeric facts is the sum or the maximum of each variable-value pair. Implicit preconditions from constraints and effects are included in the "fact set cost".

A different approach to restrict the considered numeric facts is to use a generalized Dijkstra algorithm for estimating the fact or fact set costs. Facts are processed according to a *priority queue* storing the cost to achieve them. As other actions can alter a variable while a new value is still in the priority queue, the considered numeric facts are convex unions of the effects values reachable at enqueue time and the variable's value at dequeue time.

Both approaches, the *planning graph* and the *priority queue* based approach restrict the number of considered variable-value pairs. Note that other approaches are conceivable, e.g. Scala et al. [17] discriminate facts by condition type. We will now discuss combinations of both approaches with *interval* or *repetition* relaxation which guarantee polynomial time bounds on the heuristic computation.

3.2 Heuristics Based on Planning Graphs

In the relaxed *planning graph*, the length of a shortest relaxed plan restricts the maximal number of layers required until the goal formula is satisfied for the first time. Therefore, heuristic cost estimations are polynomial in the output size $|\mathcal{A}| \times |\mathcal{V}| \times |\pi^\star|$ where $\pi^\star$ is a shortest (but not necessarily cheapest) plan. Variations of this approach are often used for numeric planning [9, 7, 13, 18].

A weakness of this approach is that, for the *interval relaxation*, $h_{\max}$ does not compute admissible estimates in tasks with action costs, and the cost estimates for h_{add} can be higher than the estimates that would be computed by the formulas for classical planning. The reason is that a fact can be achieved at a better cost in a deeper layer in the planning graph. This is a problem for numeric planning, because no a priori bound can be given on how deep the better value could be found. Opposed to relaxed classical planning, where heuristic computation terminates when a fix-point is reached and no new facts are added or reached at a cheaper cost from one layer to the next, *interval relaxed* "facts" will usually not reach a fix-point. Instead, graph generation terminates as soon as the goal formula is satisfied for the first time. Admissibility of $h_{\max}$ can be

enforced by setting the cost of all actions to the cheapest cost among action costs applicable in the current layer, which kind of obviates the use of action costs[1].

A combination of the *repetition relaxation* with the *planning graph* based variable-value pair selection strategy is not as promising as other combinations. The repetition relaxation is coarser than the interval relaxation as it aggregates arbitrary many repetitions. The planning graph approach is less accurate than the priority queue based approach when several actions of different cost alter the same variable: the result is the convex union of all individual results but the cost is the cost of the best achiever. The approach combines the downsides of the components without really making up for that.

3.3 Heuristics Based on Priority Queues

Priority queue based heuristics in the *interval relaxation* framework lack tractability as unboundedly many cheap actions can be processed before relevant ones.

The number of variable-value pairs which are inserted into the *priority queue* has to be bounded. Unfortunately, even *repetition relaxed* actions, or sequences thereof can be non-idempotent. Effects in the *repetition relaxation* are applied on the intervals fixed to the preceding state. Interactions of an action with itself are not considered when computing the behavior of the action. Similarly, a sequence of actions can be non-idempotent, even if every single action is idempotent. There are three sources of non-idempotence: variables hitting new behavior classes, interacting variables and, finally, cyclic dependencies between variables.

First, applying a numeric effect can cause a variable to hit a new behavior class. This type of non-idempotence does not impair tractability, as the behavior of a variable can change at most three times ($v > 0$, $v = 0$, $v < 0$).

Second, the result of a numeric effect depends on the variables in the assigned expression. As such, actions are reenqueued whenever an implicit precondition achieves a new value. For many planning tasks, the dependency relation between variables is acyclic. While the plan existence problem in the *repetition relaxation* is polynomial for acyclic tasks [2] this does not restrict the number of queue insertions polynomially. The problem occurs if cheap actions depend on many other more expensive actions which reside in topologically higher layers of the dependency graph. Theorem 1 of the Addendum to this paper [3] shows an example requiring exponentially many enqueue operations. The heuristic becomes tractable if actions which have an *implicit precondition* on a variable are only enqueued after all topologically higher variables have been processed. However, this means that variables in the lower layers have to wait for variables in a higher layer regardless of their cost. As the values achieved by the topologically higher variables might not be required, admissibility of $h_{\max}$ is impaired with this approach. For h_{add}, overestimating the heuristic value is acceptable, allowing us to block enqueuing of topologically lower variables until topologically higher variables are processed. A workaround for $h_{\max}$ is to compute $h_{\max}$ in two phases.

[1] Scala et al. [17] run into a similar problem using "asynchronous subgoaling" and have to set the cost of hard conditions to 0 to ensure admissibility of $h_{\max}$.

In the first phase, maximally reachable intervals for all variables are determined, and tractability is ensured by delaying topologically lower variables. Then, in a second phase, the maximally reachable values from the first phase are used for the assign effects. Topological dependencies do not have to be respected with maximally reachable intervals, as now action sequences are idempotent.

The third source of idempotence are cycles in the *dependency graph*, which can be broken by introducing auxiliary variables as presented in Section 2.1.

Bounding non-idempotence ensures that *priority queue* based algorithms can compute *repetition relaxed* estimates for $h_{\max}$ and h_{add} in polynomial time.

3.4 h_{FF}-Based Heuristics

The h_{FF} heuristic computes a *relaxed plan*, and uses the cost of this plan as heuristic estimate. As in classical planning, this plan is computed *regressively* by greedily marking required facts and actions based on the h_{add} estimates. The marking procedure captures beneficial interactions such as an action enabling precondition facts of several others. Numeric planning offers even more room for improving h_{FF} over h_{add} by not having to fully enable *implicit preconditions*, as numeric facts do not necessarily have to enable the whole reachable interval. A major contribution of this paper is a generalization of the marking procedure to numeric planning, which selects explicit *target values* in the precondition fact intervals in order to determine the necessary part of numeric preconditions.

Example 1. Let $s_0(v_1) = [0, 0]$ and $s_0(v_2) = [1, 1]$ and actions $a_1 = \langle P, v_1 \mathrel{+}= v_2 \rangle$ and $a_2 = \langle \emptyset, v_2 := 5 \rangle$ with cost $\gamma(a_1) = 1$ and $\gamma(a_2) = 10$ having to satisfy a condition $\mathcal{C} : v_1 - 1 \geq 0$. Applying a_1 is sufficient to satisfy $\mathcal{C}$. However, as a_1 has a precondition P, a_2 could be applicable before a_1 making $s(v_2) = [1, 5]$ with $\gamma = 10$. Explicating target values allows us to set $q_2 = 1$ for v_2, making h_{FF} chose $s_0(v_2) = [1, 1]$ with $\gamma = 0$ instead, thus marking a_1 but not a_2.

For the *repetition relaxed* approach, explicating target values also allows to determine the actual number of repetitions required for each action, and thus, the *repetition relaxed* h_{FF} heuristic can compute an *interval relaxed* plan which is more accurate than accounting for relaxed actions only once.

The progression step generates a sequence of relaxed states with the property that the intervals for each variable are monotonically increasing and the last state satisfies the goal condition. Starting from the goal conditions, we explicate target values in these intervals regressively, while marking actions enabling them. The h_{FF} estimate is then the cost of marked actions.

3.5 Explication of Target Values

Given a sequence of relaxed states determined by a progressive reachability analysis and the actions achieving the respective reachable intervals, we want to explicate target values in these intervals so that the resulting plan has minimal cost. The explication procedure has to respect *local target value constraints*

for each action, which ensure that all implicit and explicit preconditions of the action are satisfied and that the action achieves the desired values. These *local target value constraints* generate a set of feasible sub-intervals, where each choice of an explicit target value can lead to some relaxed plan.

The *global target value optimization problem* is then an optimization problem that selects target values in the feasible sub-intervals which minimize the cost of the resulting relaxed plan. Each target value choice influences the *local target value constraints* of all preceding states. The global target value optimization problem is NP-complete (see Theorem 2 of the Addendum [3]) and we will therefore propose approximate target value selection strategies for the local constraints and drop the requirement that the extracted plan has to be optimal.

The sequence of relaxed states starts in a state s_0 consisting of degenerate point intervals, the state for which the h_{FF} estimate has to be computed. The generated relaxed state sequence depends on the progression method: with the planning graph approach, relaxed states are the "fact layers" of the planning graph, whereas with a priority queue based approach the states are given by all intervals reachable with cost equal to the priority at enqueue time. The last state of the reachability sequence satisfies the goal condition.

The *local target value constraints* ensure that for each numeric effect of a given action a, all implicit and explicit preconditions are satisfied in the relaxed state before the application of the action, and that the execution of its numeric effects enables the target values desired from the global optimization component.

Three basic target value conditions ensure satisfaction of a *local target value constraint* of an action: Its explicit preconditions have to be satisfied (1). This requires numeric expressions evaluate to a desired target value (2). Finally, the numeric effects have to reach the desired target value (3). These basic target value conditions then restrict the intervals of the preceding state by sub-intervals containing feasible selection choices for the global target value optimization.

The first basic target value condition has to ensure that a constraint $\xi \trianglerighteq 0$ is satisfied in the state $s(\xi)$ preceding the action. We know that the action is applicable in the progression $s \vDash s(\xi) \trianglerighteq 0$. Therefore, the feasible sub-intervals $s(\xi) \cap [0, \infty)$ for "$\geq$", $s(\xi) \cap (0, \infty)$ for "$>$" or $s(\xi) \cap [0, 0]$ for "$=$" are non-empty.

Example 2. Let ξ evaluate to $s(\xi) = [-2, 3]$. The constraint $\xi \geq 0$ can be satisfied by any value in the feasible sub-interval $[0, 3]$, the result of $[-2, 3] \cap [0, \infty)$.

The second basic target value condition has to ensure that a *numeric expression* $\xi_1 \circ \xi_2$ evaluates to the target value q. Feasible sub-intervals can be obtained by first determining potential partners which make the expression evaluate to q by solving the equations $I_1 \circ s(\xi_2) = s(q)$ and $s(\xi_1) \circ I_2 = s(q)$ for I_1 or I_2 respectively, where I_1 and I_2 are intervals containing all numbers that have a partner in $s(\xi_2)$ or $s(\xi_1)$ respectively so that the expression evaluates to q. The feasible sub-intervals for the optimization component are then $s(\xi_1) \cap I_1$ and $s(\xi_2) \cap I_2$. Special care has to been taken if the inversely reachable partner "intervals" come from a division by an interval containing 0, an interval opera-

tion which can cause gaps in the resulting interval. We cannot relax the partner property, and therefore, the respective partner interval I_1 or I_2 is split.

Example 3. Let $s(\xi_1) = [-1, \frac{1}{2}]$ and $s(\xi_2) = [-3, 2]$ for $\xi_1 \times \xi_2$ enabling $q = 2$. The partner interval $I_1 = [2, 2] \div [-3, 2]$ is split into $I_1^1 = [2, 2] \div [-3, 0) = (-\infty, -\frac{2}{3}]$ and $I_1^2 = [2, 2] \div (0, 2] = [1, \infty)$. The partner interval $I_2 = [2, 2] \div [-1, \frac{1}{2}]$ is split into $I_2^1 = [2, 2] \div [-1, 0) = (-\infty, -2]$ and $I_2^2 = [2, 2] \div (0, \frac{1}{2}] = [4, \infty)$. The feasible sub-intervals are then $s(\xi_1) \cap I_1^1 = [-1, -\frac{2}{3}]$ and $s(\xi_2) \cap I_2^1 = [-3, -2]$ as the disjunctions $s(\xi_1) \cap I_1^2$ and $s(\xi_2) \cap I_2^2$ are empty.

Finally, the third basic target value condition ensures that a *numeric effect* $v \circ\!= \xi$ reaches the desired target value q, where reaching a target value has to be considered in a relaxed sense respecting the underlying relaxation semantics (repetition or interval relaxation). For the *interval relaxation*, this third basic type can be reduced to the second basic target value condition by interpreting the assignment $v \circ\!= \xi$ as an assignment of the expression $v := v \circ \xi$ with the appropriate assignment operator. This expression $v \circ \xi$ (or only ξ in the case of $:=$) has then to reach the required target value q. Care has to be taken here if the target value q was only reached because of the convex union of the relaxation semantics. When searching for partner intervals I_1 and I_2 we can use intervals $(-\infty, q]$ or $[q, \infty)$ instead of the degenerate interval $[q, q]$, where the diverging bound depends on whether the addition of q extends the upper or the lower bound of the value of v in the previous state $q > \overline{s(v)}$ or $q < \underline{s(v)}$. These values are larger (smaller) than needed and contain q because of the convex union.

Example 4. Let $s(v) = [0, 0]$ be assigned an expression $s(\xi) = [1, 2]$ by a numeric effect $v \mathrel{+}= \xi$ and let the required target value $q = \frac{1}{2}$. The corresponding expression $[0, 0] + [1, 2]$ evaluates to $[1, 2]$ which does not contain $q = \frac{1}{2}$. The partner intervals I_1 and I_2 use $[\frac{1}{2}, \infty)$ for q and we obtain $I_1 = [\frac{1}{2}, \infty) - [1, 2] = [-\frac{3}{2}, \infty)$ and $I_2 = [\frac{1}{2}, \infty) - [0, 0] = [\frac{1}{2}, \infty)$ So the target values can be chosen in $[0, 0] \cap [-\frac{3}{2}, \infty) = [0, 0]$ for $s(v)$ and $[1, 2] \cap [\frac{1}{2}, \infty) = [1, 2]$ for $s(\xi)$.

For the *repetition relaxation*, the feasible sub-intervals are intersections of the behavior classes involved to establish the desired target value with the value in the preceding state. The number of required repetitions is determined analogously to the repetitions from Theorem 6 of Aldinger et al. [2].

With these three basic target value conditions we can ensure that the propagation of values satisfying the *local target value constraints* leads to the desired target value. The explication process can propagate a list of feasible sub-intervals for the implicit and explicit preconditions to the previous layer.

The *global target value optimization problem* is now the problem to find a cost minimal set of target values for each feasible sub-interval of the *local target value constraints*. The cost of a target value is the cost of the achieving action (multiplied by the number of required repetitions for the repetition relaxation) plus the constraint cost of all of its (implicit and explicit) preconditions.

Determining target values in the feasible sub-intervals optimally is intractable. Therefore, our explication process selects locally promising target values. The

values of all variables have to reach the point intervals from s_0 at the end of the regression procedure, making proximity to the starting values $s_0(v)$ an indicator for good target values. An exception to this rule are open intervals in the *repetition relaxation*. As open intervals are only generated by contracting effects, values close to open interval bounds can only be reached by applying the contraction repeatedly and it is advisable to keep a safety margin to open interval bounds.

4 Implementation and Experiments

The Fast Downward planning system [12] is a modular planning system that is widely used in classical planning. We extend Fast Downward to support numeric planning from PDDL 2.1, layer 2 [10] as well as selected features from PDDL 3 such as global constraints. The original Fast Downward does not support floating point numbers and thus, major modifications had to be performed.

While classical planning tasks are restricted to actions with integer valued action costs, numeric planning tasks come with more sophisticated metric expressions instrumenting over several variables. Numeric Fast Downward (NFD) supports linear state-independent instrumentation effects[7], which are evaluated in the initial state and compiled into a rational valued action cost. Instrumentation variables are detected automatically and stored separately which allows search algorithms to prune states that only differ in these variables.

We implemented the heuristics $h_{\max}$, h_{add} and h_{FF} in the two most promising combinations of relaxation and "fact" selection scheme identified in the previous section: the *planning graph* approach in an *interval relaxation* (identified by the superscript h^{gi}) and the *priority queue* based approach in the *repetition relaxation* (identified by the superscript h^{qr}).

4.1 Experiments

We performed experiments on various numeric domains [16, 11, 1] comparing NFD to Metric FF [13] and two configurations of ENHSP: subgoaling with redundant constraints $\hat{h}^{radd}_{hbd+}$ [17] and h_{AIBR} [18]. We used greedy best first search for all NFD heuristics. Experiments were run on a cluster with a timeout of 30 minutes for each instance. Table 1 shows the number of solved instances on various domains. A star indicates errors in ENHSPs preprocessing component.

Our heuristics perform slightly worse than Metric FF for the planning domains that both planners can solve. In principle, the heuristic estimates from h^{gi}_{FF} should resemble Metric FF most. Differences can probably be attributed to different search search algorihms. Metric FF is restricted to linear tasks and cannot find solutions for nonlinear problems such as geo-rovers or jumpbot. ENHSP excels at block-grouping and performs roughly as good as most NFD configurations on the domains both planners can solve. ENHSP ignores action costs and uses unit cost actions instead. There was no domain in the benchmarks which exploits this weakness. The *priority queue* based *repetition relaxed* h^{qr}_{FF} heuristic

Table 1. Solved instances on various numeric domains

Domain	Metric FF	ENHSP		NFD			
		$\hat{h}^{radd}_{hbd+}$	h_{AIBR}	h^{gi}_{add}	h^{qr}_{add}	h^{gi}_{FF}	h^{qr}_{FF}
block-grouping (192)	22	**122**	62	15	18	14	28
counters (78)	**36**	14	34	21	11	21	31
depots (22)	**20**	⋆	⋆	7	10	10	11
driverlog (40)	**34**	⋆	⋆	29	31	30	30
farmland (50)	9	0	**50**	17	10	19	23
geo-rovers (21)	0	⋆	⋆	1	0	1	**2**
jumpbot (20)	0	3	15	**17**	0	15	13
plant-watering (51)	**22**	12	**22**	0	15	0	15
rovers (20)	**12**	⋆	⋆	0	3	1	4
satellite (40)	26	21	22	12	23	19	**29**
settlers (20)	**9**	⋆	⋆	0	0	0	2
sokoban (325)	0	37	0	**70**	**70**	69	**70**
zenotravel (20)	**20**	⋆	⋆	7	7	8	9
Sum (899)	210	209	205	196	198	207	**267**

does not only solve most instances, but it also solves several instances in each of the domains, indicating that our heuristics offer guidance for all domains.

The jumpbot domain [1] is particularly interesting as it models physical properties in a dynamic world. It features cyclic, non-linear effects for turning, accelerating or deccelerating the robot, as well as classical preconditions. Therefore, it can neither be solved by control engineering [15] nor by planners requiring linear tasks such as Metric FF.

5 Conclusion

We discussed different approaches to tractable heuristics for interval relaxed numeric planning and considered different relaxation frameworks: the *interval relaxation* and the *repetition relaxation* with different restriction schemes for the variable-value pairs considered during heuristic exploration: one motivated from the *planning graph*, another from a *priority queue*. We highlighted critical combinations that impair tractability of the heuristic or restrict algorithms to a subset of numeric planning tasks. Furthermore, we generalized the marking procedure of h_{FF}.

We implemented the well known planning graph heuristics $h_{\max}$, h_{add} and h_{FF} from classical planning in these frameworks and established heuristics which are suitable for all numeric planning tasks expressible in PDDL 2.1, layer 2. We showed experimentally that the general heuristics can find plans even for planning tasks with cycles, non-linear effects and action costs, providing a baseline for future approaches.

Acknowledgments This work was supported by the DFG through grants EXC1086 (BrainLinks-BrainTools) and NE 623/13-2 (HYBRIS-2).

References

[1] Aldinger, J.: The Jumpbot Domain for Numeric Planning. Tech. Rep. 279, University of Freiburg (2016)

[2] Aldinger, J., Mattmüller, R., Göbelbecker, M.: Complexity of Interval Relaxed Numeric Planning. In: Proceedings of the 38th German Conference on Artificial Intelligence (KI 2015) (2015)

[3] Aldinger, J., Nebel, B.: Addentum to 'Interval Based Relaxation Heuristics for Numeric Planning with Action Costs'. Tech. Rep. 280, University of Freiburg (2017)

[4] Bonet, B., Geffner, H.: Planning as Heuristic Search: New Results. In: Proceedings of the 5th European Conference on Planning (ECP 1999). pp. 360–372 (1999)

[5] Bonet, B., Geffner, H.: Planning as Heuristic Search. Artificial Intelligence 129(1–2), 5–33 (2001)

[6] Bonet, B., Loerincs, G., Geffner, H.: A Robust and Fast Action Selection Mechanism for Planning. In: Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI 1997/ IAAI 1997). pp. 714–719 (Jul 27–31 1997)

[7] Coles, A., Coles, A., Fox, M., Long, D.: A Hybrid LP-RPG Heuristic for Modelling Numeric Ressource Flows in Planning. Journal of Artificial Intelligence Research 46 (JAIR 2013) pp. 343–412 (2013)

[8] Coles, A., Fox, M., Long, D., Smith, A.: A Hybrid Relaxed Planning Graph-LP Heuristic for Numeric Planning Domains. In: Proceedings of the 20th International Conference on Automated Planning and Search (ICAPS 2008) (2008)

[9] Edelkamp, S.: Generalizing the Relaxed Planning Heuristic to Non-Linear Tasks. In: Proceedings of the 27th German Conference on Artificial Intelligence (KI 2004) (2004)

[10] Fox, M., Long, D.: `PDDL2.1` : An Extension to `PDDL` for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research 20 (JAIR 2003) pp. 61–124 (2003)

[11] Francès, G., Geffner, H.: Modeling and Computation in Planning: Better Heuristics from More Expressive Languages. In: Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS 2015) (2015)

[12] Helmert, M.: The Fast Downward Planning System. Journal of Artificial Intelligence Research 26 (JAIR 2006) pp. 191–246 (2006)

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

[14] Hoffmann, J., Nebel, B.: The FF Planning System: Fast Plan Generation Through Heuristic Search. Journal of Artificial Intelligence Research 14 (JAIR 2001) pp. 253–302 (2001)

[15] Löhr, J., Eyerich, P., Keller, T., Nebel, B.: A Planning Based Framework for Controlling Hybrid Systems. In: Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012) (2012)

[16] Long, D., Fox, M.: An Overview and Analysis of the Results of the 3rd International Planning Competition. Journal of Artificial Intelligence Research 20 (JAIR 2003) pp. 1–59 (2003)

[17] Scala, E., Haslum, P., Thiébaux, S.: Heuristics for Numeric Planning via Subgoaling. In: Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016). pp. 655–663 (2016)

[18] Scala, E., Haslum, P., Thiébaux, S., Ramírez, M.: Interval-Based Relaxation for General Numeric Planning. In: Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016). pp. 655–663 (2016)
[19] Young, R.C.: The Algebra of Many-valued Quantities. Mathematische Annalen 104 pp. 260–290 (1931)