Dissertation zur Erlangung des Doktorgrades der Technischen
Fakultät der Albert-Ludwigs-Universität Freiburg im Breisgau

Relaxation Heuristics for Numeric Planning

vorgelegt von
Johannes Aldinger, M.Sc.

am 31. Juli 2018

Betreut von
Prof. Dr. Bernhard Nebel

Tag der Disputation:
30. November 2018

Dekan:
Prof. Dr. Oliver Paul

Prüfungskommission:
Prof. Dr. Bernd Becker (Vorsitz)
Prof. Dr. Andreas Podelski (Beisitz)
Prof. Dr. Bernhard Nebel (Betreuer)
Prof. Dr. Jörg Hoffmann (Prüfer)

Zusammenfassung

Ein wichtiges Merkmal von intelligenten Agenten ist, das sie *denken* bevor sie *handeln*. *Handlungsplanung* ist das Teilgebiet der *Künstlichen Intelligenz*, welches sich mit dieser Art von *Denken* beschäftigt. Das Ergebnis eines solchen Denkprozesses ist eine Handlungsanweisung für die Schritte des Agenten. Gegeben ein durch Zustände und Aktionen beschriebenes *Modell* der Welt, ist ein *Plan* eine Sequenz von Aktionen, welche einen initialen Weltzustand in einen Zustand überführt, in welchem eine gewünschte Zielbedingung erfüllt ist.

Im klassischen Planen werden die Zustände der Welt durch Variablen mit beschränktem Wertebereich beschrieben. Allerdings ist dieser Formalismus für viele reale Anwendungsbereiche nicht ausdrucksstark genug. Um etwa Ressourcen (z.B. die verbleibende Menge Benzin im Tank) oder physikalsiche Größen (z.B. die aktuelle Geschwindigkeit eines Autos) zu modellieren, werden numerische Variablen benötigt. In dieser Arbeit beschäftigen wir uns mit *numerischer* Handlungsplanung bei welcher Variablen auch kontinuierliche Werte zugewiesen werden können.

Im Allgemeinen ist die Existenz einer Lösung eines numerischen Planungsproblems *unentscheidbar*. Daher untersuchen wir Näherungslösungen für numerische Planungsprobleme, welche die Kosten von bestimmten numerischen Fakten abschätzen, insbesondere solche, die zum erreichen eines Ziels benötigt werden. Im Fokus dieser Arbeit stehen hierbei Intervall-basierte Relaxiserungsheuristiken, da diese auf viele numerische Probleme angewendet werden können, insbesondere auch auf Probleme mit nicht-linearen Änderungen. Bei einigen Problemen bieten die vorgestellten Relaxierungsheuristiken eine gute Orientierungshilfe, aber selbst in Fällen bei denen die Informationsqualität schwach ist, ermöglichen Relaxierungsheuristiken eine grundlegende Orientierung für Probleme in welchen spezialisierte Lösungen nicht verfügbar sind.

Abstract

An important feature of intelligent agents is that they *think* before they *act*. Automated planning is the area in Artificial Intelligence that is concerned with this art of thinking. The result of such a reasoning process is an instruction for the steps of the agent. Given a world modeled by states and actions, a *plan* is a sequence of actions which transforms an initial world situation into a situation that satisfies a goal condition, thus solving the planning problem.

Classical planning models the world state with finite-domain variables. However, this formalism is not expressive enough for many real-world applications. Numeric variables are required to model resources (e.g. the fuel level in a tank) or physical quantities (e.g. the current velocity of a car). In this dissertation we elaborate on *numeric* planning where the variables can take continuous values.

In general, the existence of a solution to numeric planning problems is *undecidable*. Therefore, we are interested in heuristics for numeric planning that estimate the cost of reaching certain numeric facts, particularly ones necessary to reach a goal. The focus of this work are interval-based relaxation heuristics, as they are applicable to a wide range of numeric planning problems including problems with non-linear change. The proposed relaxation heuristics offer good guidance on some problems, but even in cases where the information quality is poor, a basic guidance is valuable for problems where no specialized solutions are available.

Acknowledgments

I have received support and encouragement from a great number of individuals. My deepest gratitude is to my advisor, Prof. Bernhard Nebel, who always managed to keep the balance between giving me the freedom to explore on my own and the guidance necessary to complete this thesis.

Further gratitude goes to the colleagues from the Foundations of Artificial Intelligence groups in Freiburg and Basel, foremost to Robert Mattmüller, but also to Thomas Keller, Patrick Eyerich, Malte Helmert, Gabi Röger, Mara Göbelbecker, Christian Dornhege, Andreas Hertle, Roswitha Hilden, Petra Geiger, Uli Jakob, Stefan Wölfl, Tim Schulte, Florian Geißer, Johannes Löhr, Daniel Kuhner, Felix Burget, Benedikt Wright, David Speck and Jendrik Seipp.

Last but not least, I thank my wife Meike and my children Clara and Noah for their support. Thank you!

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- oder Beratungsdiensten (Promotionsberaterinnen oder Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.

Freiburg, 13. Januar 2019. Johannes Aldinger

Contents

Chapter 1

Introduction

A defining characteristic of intelligence is the ability of *thinking* before *acting*. *Automated planning* is the area of artificial intelligence which is concerned with this art and practice. Given a world description, a set of actions that modify the world and a goal formulation, *planning* seeks to find a sequence of actions that transforms the current world situation into one that satisfies the goal. The difficulty of a planning task depends on the assumptions made on the properties of this world model. Actions can alter the world with deterministic or with non-deterministic effects. The effects of an action can either be applied instantaneously or after a duration, and if actions are durative, the change can either happen at discrete moments or continuously. Actions can be applicable concurrently or only sequentially. The world can be discrete or continuous. State variables can be fully accessible or observed only partially. The world can be static with the only change happening by actions of the acting agent or exogenous influence can alter the world dynamically. There can be several agents. The objective can either be to reach a designated goal state or to maximize a reward function.

The easiest configuration is known as *classical planning* where the unique and fully observable initial world state is described by discrete finite-domain variables. A single agent can sequentially apply actions with deterministic and instantaneous effects. Depending on the objective of a classical planning problem we distinguish *optimal* planning where we are only interested in plans which are optimal regarding a certain cost function and *satisficing* planning, where a good plan quality is preferable but not necessarily required. Classical planning has been extensively studied in the recent years and many planning systems have emerged, e.g., the FF Planning System [HN01], Fast Downward [Hel06] and many others. However, classical planning is not expressive enough for many practically relevant planning problems. Modeling of physical properties (e.g. velocity) or resources (e.g. fuel level) requires real-valued variables. Many interesting problems require reasoning about continuous quantities and therefore, they can not be expressed appropriately in classical planning. While there exist planning problems which rely on an expressivity that goes beyond numeric variables, e.g., concurrent temporal actions, or the modeling of exogenous effects of a dynamic environment, there are many interesting problems that can be solved by *numeric planning* but not by *classical planning*.

We define *numeric planning* as single agent planning in a world with a unique

and fully observable initial world state which is described by variables whose domain are continuous numbers. Instantaneous actions with deterministic effects are applied sequentially in order to achieve a state satisfying a goal formula.

A predominant approach to solve classical planning problems is heuristic search. In this thesis we study approaches to obtain heuristic guidance by solving a simplified planning problem that can be solved efficiently and use the solution of the simplified problem to guide search in the original problem. We investigate *relaxation heuristics* for numeric planning, a simplification where negative interactions are removed from the planning problem. Such *relaxations* are well known as they are one of the first successful approaches for classical planning [BG99].

1.1 Outline

This thesis is structured as follows: Chapter 2 introduces the formal definition of the *numeric planning tasks* that we are interested in this dissertation. We will briefly discuss the complexity of numeric planning and the representation of numbers, before we give an overview over related work.

Chapter 3 introduces the *delete relaxation* for classical planning and elaborates on different ideas to extend the *relaxation* concept to numeric planning. Interval-based relaxations turn out to be suited very well and we advance into the *interval relaxation* in more depth. Afterwards we discuss a more sophisticated interval based relaxation, the *repetition relaxation*, where arbitrary many repetitions of the same numeric effect are aggregated to a single step.

Based on these relaxations for numeric planning, we discuss relaxation heuristics Chapter 4, functions that are computed on these simplified problems and than can be used to guide search in the original numeric planning problem.

In Chapter 5, we introduce the Numeric Fast Downward (NFD) planning system, a planning system suitable for numeric planning problems that is equipped with the heuristics presented in Chapter 4.

In Chapter 6 we present the Jumpbot domain as well as other benchmark problems for numeric planning. Afterwards, we show the experimental results from comparing different configurations of our NFD planner in terms of plan cost, coverage and algorithmic quality. We conclude our experiments with a comparison to other numeric planners. Finally, we examine an *agile Earth observation* scenario that is motivated by a real world application.

Finally, Chapter 7 concludes the thesis.

1.2 Contribution

During the doctoral process, the author has contributed to research in several areas of automated planning.

Interval based relaxations and their *complexity* were analyzed at the ICAPS-2015 Workshop on Heuristics and Search for Domain-independent Planning (HSDIP 2015) [AMG15a] as well as a paper published at the 38th German Conference on Artificial Intelligence [AMG15b]. The results build the foundations of Chapter 3.

Chapter 4 is based on our work on *relaxation heuristics for numeric planning*, which is published at the 40th German Conference on Artificial Intelligence (KI 2017) [AN17c] with an extended abstract presented at the 10th International Symposium on Combinatorial Search (SoCS 2017) [AN17b] and Technical Report 280 containing some proofs [AN17a].

In Chapter 6, we briefly introduce the JUMPBOT benchmark domain which was first described in Technical Report 279 [AL16]. At the end of the chapter we also discuss an application scenario, where we plan slew maneuvers of an *agile Earth observation satellite* that has to find a trajectory to scan as many observation sites on the Earth surface as possible. The corresponding paper was presented at the ICAPS-2013 Workshop on Planning in Continuous Domains (PCD 2013) [AL13]. Additional contributions in the area of *Earth observation planning* have been published in the Jahrbuch der Deutschen Gesellschaft für Luft- und Raumfahrt (DGLR2013) [LAWW13].

Outside the scope of this thesis are publications on a *brain-controlled robotic assistant for users with limited communication skills* that appear at the 2017 European Conference on Robotics (ECMR 2017) [BFK+17] which is based on a robot control framework which is presented at the 2018 International Conference on Intelligent Robots and Systems (IROS 2018) [KAB+18]. An article in the Journal of Robots and Autonomous Systems (RAS) (2018) [KFB+18] is still under review.

References to Author's Contributions

[AL13] Johannes Aldinger and Johannes Löhr. "Planning for Agile Earth Observation Satellites." In: *Proceedings of the ICAPS-2013 Workshop on Planning in Continuous Domains (PCD 2013)*. 2013, pp. 9–17.

[AL16] Johannes Aldinger and Johannes Löhr. *The Jumpbot Domain for Numeric Planning*. Tech. rep. 279. University of Freiburg, Apr. 2016.

[AMG15a] Johannes Aldinger, Robert Mattmüller, and Moritz Göbelbecker. "Complexity Issues of Interval Relaxed Numeric Planning". In: *Proceedings of the ICAPS-2015 Workshop on Heuristics and Search for Domain-independent Planning (HSDIP 2015)*. 2015, pp. 4–12.

[AMG15b] Johannes Aldinger, Robert Mattmüller, and Moritz Göbelbecker. "Complexity of Interval Relaxed Numeric Planning". In: *Proceedings of the 38th German Conference on Artificial Intelligence (KI 2015)*. Ed. by Steffen Hölldobler, Markus Krötzsch, Rafael Peñaloza, and Sebastian Rudolph. Vol. 9324. LNAI. Springer-Verlag, 2015, pp. 19–31.

[AN17a] Johannes Aldinger and Bernhard Nebel. *Addendum to 'Interval Based Relaxation Heuristics for Numeric Planning with Action Costs'*. Tech. rep. 280. University of Freiburg, Oct. 2017.

[AN17b] Johannes Aldinger and Bernhard Nebel. "Extended Abstract: Interval Based Relaxation Heuristics for Numeric Planning with Action Costs". In: *Proceedings of the 10th International Symposium on Combinatorial Search (SoCS 2017)*. 2017, pp. 155–156.

[AN17c] Johannes Aldinger and Bernhard Nebel. "Interval Based Relaxation Heuristics for Numeric Planning with Action Costs". In: *Proceedings of the 40th German Conference on Artificial Intelligence (KI 2017)*. Ed. by Gabriele Kern-Isberner, Johannes Fürnkranz, and Matthias Thimm. Vol. 10505. LNAI. Springer-Verlag, 2017, pp. 15–28.

[BFK+17] Felix Burget, Lukas D.J. Fiederer, Daniel Kuhner, Martin Völker, Johannes Aldinger, Robin T. Schirrmeister, Chau Do, Joschka Boedecker, Bernhard Nebel, Tonio Ball, and Wolfram Burgard. "Acting Thoughts: Towards a Mobile Robotic Service Assistant for Users with Limited Communication Skills". In: *Proceedings of the European Conference on Robotics (ECMR 2017)*. 2017, pp. 385–390.

[KAB+18] Daniel Kuhner, Johannes Aldinger, Felix Burget, Mara Göbelbecker, Wolfram Burgard, and Bernhard Nebel. "Closed-Loop Robot Task Planning Based on Referring Expressions". In: *Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2018)*. 2018.

[KFB+18] Daniel Kuhner, Lukas Fiederer, Felix Burget, Martin Völker, Johannes Aldinger, Robin Schirrmeister, Chau Do, Wolfram Burgard, and Bernhard Nebel. "A Deep Learning Framework for BCI Control of a Robotic Service Assistant using Intelligent Goal Formulation". In: *Journal of Robots and Autonomous Systems (RAS)* (2018), to appear.

[LAWW13] Johannes Löhr, Johannes Aldinger, Stefan Winkler, and Georg Willich. "Automated Planning for Earth Observation Spacecraft under Attitude Dynamical Constraints". In: *Jahrbuch der Deutschen Gesellschaft für Luft- und Raumfahrt (DGLR2013)*. 2013.

Chapter 2

Numeric Planning Tasks

In this chapter we define the structure of problems that we want to solve in this thesis. We begin with a formal definition of *numeric planning tasks*. Afterwards, we discuss the computational complexity of numeric planning and the representation of real numbers. We conclude the chapter with a summary of related work.

2.1 Numeric Planning Task Definition

A *numeric planning task* is a formalization of the problem of reaching a goal state in a world modeled by *continuous* variables. The static and fully observable world can be altered sequentially by actions with instantaneous effects performed by a single agent. Figure 2.1 depicts a small example plan in a world described with only two state variables v_0 and v_1. The sequence of actions $\pi = \langle a_0, a_1, a_2 \rangle$ transforms the initial state s_0 to s_1 then s_2 and finally s_3, a state that satisfies a goal condition (e.g. $-v_1 - 1 > 0$).

Formally, a numeric planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G}, \kappa \rangle$ is a 5-tuple where $\mathcal{V}$ is a set of numeric variables v with domain $\mathbb{Q}_\perp^\infty := \mathbb{Q} \cup \{-\infty, \infty, \perp\}$: the rational numbers with positive and negative infinity and an "undefined" value $\perp$. $\mathcal{A}$ is a set of actions, $\mathcal{I}$ the initial *state*, $\mathcal{G}$ a goal *condition* and $\kappa : \mathcal{A} \to \mathbb{Q}^+$ is a function assigning a strictly positive rational cost to each action. A *state* is a (full) mapping from variables from $\mathcal{V}$ to values from $\mathbb{Q}_\perp^\infty$. The value of variable v in state s is denoted by $s(v)$. A *numeric expression* is either a variable v from $\mathcal{V}$, a constants c from $\mathbb{Q}$, or of the form $\xi_1 \circ \xi_2$ where $\circ \in \{+, -, \times, \div\}$

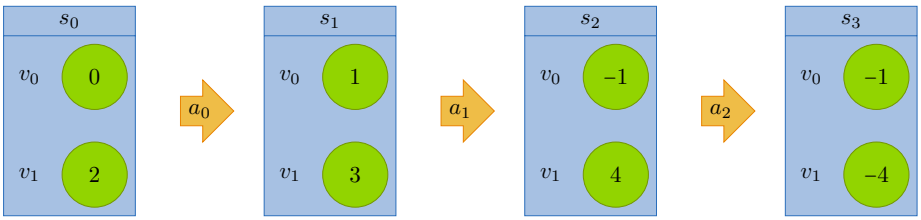

Figure 2.1: A numeric plan with instantaneous actions modifying two state variables v_0 and v_1 with a sequence of actions $\langle a_0, a_1, a_2 \rangle$.

is an arithmetic operator and expressions ξ_1 and ξ_2 are recursively defined. A *numeric constraint* $\xi \trianglerighteq 0$ compares expressions ξ to 0 with $\trianglerighteq \in \{\geq, >, =\}$ and a (goal or action) *condition* is a conjunction of *numeric constraints*. *Effects* are assignments $v \circ\!= \xi$ where v is a variable from $\mathcal{V}$, $\circ\!= \in \{+=, -=, \times=, \div=, :=\}$ 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 *effect* for each variable.

The semantics of a numeric planning task is straightforward. Numeric expressions $\xi_1 \circ \xi_2$ for $\circ \in \{+, -, \times, \div\}$ are recursively evaluated in state s: $s(\xi_1 \circ \xi_2) = s(\xi_1) \circ s(\xi_2)$. Constants $c \in \mathbb{Q}$ are mapped to themselves $s(c) = c$ and variables $v \in \mathcal{V}$ to the respective value in their domain. Division by zero results in the undefined value $\perp$, i.e. if $\circ = \div$ and $s(\xi_2) = 0$, $s(\xi_1 \div 0) = \perp$. If either $\xi_1 \in \{-\infty, \infty, \perp\}$ or $\xi_2 \in \{-\infty, \infty, \perp\}$, $s(\xi_1 \circ \xi_2)$ evaluates to either $-\infty$, ∞ or $\perp$ as intuitively expected, i.e. adding a finite number to an infinite value remains infinity, the difference of two infinite values is undefined and undefined values are propagated. The semantic adheres to the IEEE 754 floating point guidelines where "$\perp$" corresponds to the NaN value.

For conditions, satisfaction in state s is defined in the obvious way: numeric constraints $\xi \trianglerighteq 0$ are satisfied in s if the arithmetic comparison of the evaluated expression is, i.e. $s \vDash (\xi \trianglerighteq 0)$ iff $s(\xi) \trianglerighteq 0$. The conjunction of conditions c_i and c_j is satisfied, if both conditions are, i.e. $s \vDash c_i \wedge c_j$ iff $s \vDash c_i$ and $s \vDash c_j$.

Actions $a = \langle \text{pre}, \text{eff} \rangle$ are applicable in s iff $s \vDash \text{pre}$. The successor state $\text{app}(a, s) = s'$ resulting from an application of a in s is defined as follows: for each numeric effect $\text{eff}_i \in \text{eff} = \{\text{eff}_1, \ldots, \text{eff}_n\}$ of the form $v \circ\!= \xi$ with $\circ\!= \in \{+=, -=, \times=, \div=\}$, $s'(v) = s(v) \circ s(\xi)$ where $\circ \in \{+, -, \times, \div\}$ is the arithmetic operator corresponding to the respective assignment operator. If eff_i is a numeric assignment $v := \xi$, then $s'(v) = s(\xi)$. Finally, if a variable v does not occur in the left-hand side of any effect, then $s'(v) = s(v)$.

A plan π is a (possibly empty) sequence of actions $\langle a_1, \ldots, a_n \rangle$ that leads from $\mathcal{I}$ to a state satisfying $\mathcal{G}$ such that each action is applicable in the state that follows by executing the plan up to that action, i.e. $s_0 = \mathcal{I}$, $s_i = \text{app}(a_{i-1}, s_{i-1})$ where $s_{i-1} \vDash \text{pre}(a_i)$ for $1 \leq i \leq n$ and $s_n \vDash \mathcal{G}$.

2.2 Complexity of Numeric Planning

Automated planning is computationally hard even in its most basic form as STRIPS [FN71] planning, modeling propositional variables in a closed world, where action preconditions are positive literals and actions are not allowed to have conditional effects. Even this restricted classical planning formalism is PSPACE-complete [Byl94]. Numeric planning is *undecidable* [Hel02]. Even though completeness of numeric planning algorithms can therefore not be achieved in general, numeric planners can find plans or an assurance that the problem is unsolvable for many practical tasks.

2.2.1 Representation of Numbers

Real numbers are generally not representable in a computer and have to be approximated. Asymptotic complexity is usually measured with respect to the size of the representation of the input. Analyzing the complexity of numeric

planning is therefore only sensible if the representation is bounded. Moreover, the time complexity depends on the definition of a time step. Two different cost models are typically used to define such a step [CLRS09]: the *uniform cost model* assumes that basic operations have constant cost independent of the representation size of the involved numbers. The *logarithmic cost model* assigns a cost proportional to the number of bits required to store involved numbers. For rational numbers, the result of addition and subtraction are numbers with a similar representation requirement. However, for multiplication or division the representation size of the result requires an order of twice the representation size of its factors. In the following chapters we base our analysis on a *uniform cost model*. Therefore, polynomial *input* complexity results shown in this thesis would translate to polynomial *output* complexity results if we would base our step size on a *logarithmic cost model* instead, as the representation size can grow exponentially in the input for a polynomial number of steps in a *logarithmic cost model*.

In our implementation we use 64-bit floating point numbers. For a fixed representation size, the complexity of a step is obviously constant for all basic operations. However, not all numbers are representable by floating point numbers. While we are aware of error propagation issues that come from rounding of values to the closest representable number [Gol91], such representation problems did not become apparent in our practical applications and are beyond the scope of this thesis.

2.3 Related Work

Numeric planning received attention of the automated planning community when planning paradigms with increased expressiveness were advocated for in the International Planning Competition (IPC) 2002 [LF03]. Among the participating planners MIPS [EH01] and LPG [GS02], the Metric-FF planning system [Hof03] is most relevant to our work, as it also follows the approach to extending the concept of a delete relaxation to numeric planning. However, the numeric benchmark domains of the IPC 2002 were rather simple extensions of the classical domains that also model numeric resources. The numeric effects in these benchmarks are mostly restricted to linear tasks that assign constant expressions (e.g. actions that *decrease* the energy of a rover by 3). This restricted expressivity can be exploited. The Metric-FF planning system [Hof03] tries to convert the planning task into a *linear numeric* task, which ensures that variables can "grow" in only one direction. When high values of a variable are beneficial to fulfill the preconditions, *decrease* effects are considered harmful and a generalization of the *delete relaxation* concept translates into ignoring *decrease* effects. Edelkamp [Ede04] proposes a generalization of this approach to non-linear tasks which then roughly corresponds to the interval based relaxation that will be proposed in the upcoming sections. Still, we were the first to properly formalize and analyze numeric extensions to the relaxation idea.

On the application side, several planners support numeric quantities, albeit usually only restricted to a subset of numeric planning tasks, most commonly to linear planning problems. Additionally, most planners ignore action costs, a strategy that usually increases coverage at the expense of plan quality. In the following, we will enlarge upon other relaxation based approaches for numeric

planning and briefly discuss other alternatives.

The Colin Planning System [CFLS08; CCFL13] combines a planning graph based *relaxation* approach with a solver for linear programs (LP). The *relaxation heuristic* is built by a *planning graph* based progression similar to the planning graph based approach discussed in Section 4.3.2. While intervals are used to represent the values of variables at each layer of the planning graph, these intervals are not determined by an *interval relaxation* (cf. Section 3.3) in the way we propose it in this thesis. Instead, Coles, Fox, Long, and Smith identify variables that are used to model *resources*, and they identify actions that *produce* or *consume* such resources. Based on this change, they create a *mixed integer linear program* (MILP), where the variables of the linear program model either planning variables or actions copied for each fact layer. The actions variables indicate whether an action has been applied or not in the respective layer of the graph. The constraints of the MILP ensure that resources can only be *consumed* if they have been *produced* before. Solving the LP relaxed program, where action variables are allowed to attain values between 1 and 0, results in tighter bounds as compared to the *interval relaxation* approach at the cost of having to solve a linear program for each layer of the planning graph. More severely, the approach requires variables to model *resources* and change on such resources has to be *linear*. However, if these conditions are met, the LP-approach has a huge structural advantage over *relaxation* based approaches, which suffer from the *cyclic resource transfer* problem which is common in many logistics settings. When a *resource* can be transferred from one location to another, the corresponding actions appear to *consume* the *resource* at the first location, and *produce* it at the second, when in fact the available amount of the *resource* does not change at all. While *relaxations* relax this property and erroneously assume that an arbitrary amount of the resource could be *produced* by such actions, LP-based approaches handle such situations very well. As such, the hybrid LP-RPG approach of Coles, Fox, Long, and Smith excels at resource heavy domains such as SETTLERS, but it is not as well suited for domains that model physical properties such as JUMPBOT.

The ENHSP planning system [SHTR16; SHT16] is a recent planner for numeric planning problems that is most closely related to our work. There are several configurations that draw on different techniques to approach numeric planning. Most closely related to our work is an *interval-based* relaxation approach by Scala, Haslum, Thiébaux, and Ramírez [SHTR16]. The *additive interval based relaxation* heuristic h_{AIBR} pursues a similar idea to a *repetition relaxation* (cf. Section 3.4) with a *planning graph* based approach to generate relaxed state progression sequences (cf. Section 4.3.2). One major difference is the handling of *cyclic effects* (cf. Section 4.2) $v := \xi$ where a variable v is assigned an expression ξ that depends on variable v. The *additive effect transformation* transforms such assignment effects $v := \xi$ to additive effects $v \mathrel{+}= \xi - v$. If a variable extends its upper bound by assigning expression ξ to it, in the relaxation, this transformation translates to the assumption that an arbitrary upper bound can be reached by applying the action cycle repeatedly, and analogously for the lower bound. After this transformation, the task does not contain effects that are critical (cf. Section 3.4.2) and as such heuristics can be computed in polynomial time. Now the *additive interval based relaxation heuristic* h_{AIBR} is computed by identifying *supporters*: conditions which support that an additive effect extends the upper bound (or the lower bound respectively) of the affected

variable. If these *support* conditions are met, the intervals upper (lower) bound is set to infinity in the next layer of the planning graph. This relates very much to the *repetition relaxation* (cf. Section 3.4) where the *repetition successor* of an additive effect sets respective bounds to infinity as well. The heuristic estimate of the *additive interval based relaxation* heuristic h_{AIBR} is then the sum of all actions that appear in the planning graph. Opposed to the extensions of the FF heuristic that we propose in Section 4.4, actions that do not contribute towards reaching the goal are not eliminated from this estimate. Additionally the approach assumes unit action costs.

A different approach used in ENHSP is based on generalizations of the forward propagation heuristics $h_{\max}$ and h_{add} [BLG97; BG99] in a way differing from our approach in Section 4.3. In classical planning, the cost of a proposition can be estimated recursively by the cost of the best achieving action. The cost of achieving an action is estimated by the action cost plus the most expensive ($h_{\max}$) or the sum (h_{add}) of all costs of achieving all propositions occurring in the precondition in isolation. Scala, Haslum, and Thiébaux [SHT16] extend this approach by estimates for the cost of *numeric conditions* that are further differentiated into *simple* and *hard* conditions. *Simple* conditions compare *linear expressions* to zero where all variables in the expression are only affected by simple increase or decrease effects, and can be computed more accurately than *hard* conditions, whose estimate is based on an *interval based relaxation* resembling *additive interval based relaxation heuristic* h_{AIBR}. Additionally, redundant constraints can be added to the planning problem, which help to avoid severe underestimation of the heuristic estimate. We compare against this *hybrid subgoaling heuristics* $\hat{h}^{radd}_{hbd+}$ with redundant constraints in our experiments (cf. Chapter 6).

Apart from generalizations of the delete relaxation idea, there are approaches that are based on planning as satisfiability [KS92]. *Satisfiability modulo theories* (SMT) allows sat-solvers to outsource *theory*-solvers which deal with the numeric aspects of a SAT problem. Examples of numeric planning systems that pursue such an approach include the Springroll planning system Scala, Ramírez, Haslum, and Thiébaux [SRHT16] and the RANTANPLAN planner [BAV15] which combines an SMT approach with linear programs as well.

Chapter 3

Relaxations

A *relaxation* is an approximation of a difficult problem by a related problem that has fewer constraints which makes it easier to solve.

For planning, we can ignore negative interactions between the actions, thus making the problem monotone. In *classical* STRIPS [FN71] planning, the planning task can be transformed into a positive normal form where all preconditions require propositional variables to be *true*. Negative interactions are relaxed by ensuring that all propositions which are achieved once cannot be invalidated during the planning process. This *delete relaxation* ignores effects that *delete* propositions from the world model, i.e. a *delete relaxation* ignores effects that would set propositional variables to *false*.

We are interested in relaxation approaches for numeric planning that can be computed in polynomial time. *Interval* based relaxations are particularly interesting as they allow us to represent arbitrarily many values of a variable compactly.

This chapter is based on our paper presented at the 38th German AI Conference [AMG15b] and the corresponding workshop paper presented at the ICAPS 2015 HSDIP workshop [AMG15a]. This chapter is structured as follows: we start with providing the background of relaxations for classical planning in Section 3.1 before we discuss different extensions of the relaxation concept to numeric planning in Section 3.2. In Section 3.3 we delve deeper into interval based relaxations. Finally, we present the *repetition relaxation*, in Section 3.4: an interval based relaxation that deals with the problem that numeric actions are non-idempotent operations by applying actions arbitrarily often in one step.

3.1 Delete Relaxation for Classical Planning

The idea of a *delete relaxation* is that facts that are achieved once can not be deleted in future steps. The propositional STRIPS encoding of classical planning operates under a closed world assumption and only lists the true propositions in a state. In STRIPS, planning tasks are in a positive normal form where all (action and goal) conditions require propositions to be true (a restriction that can be ensured by introducing inverted propositions to the planning task) so that it is always beneficial for propositions to be true in order to make actions applicable or reach the goal. The effects of an action can then be separated into *add effects*

which add propositional facts (set the value of the corresponding variable to *true*) and *delete effects* which "delete" propositional facts (set the value of the corresponding variable to *false*). A relaxation which ignores the delete effects of all actions introduces a simplified monotonous planning problem where the application of each action can only add new propositions to the current state.

More recent planning systems are usually not restricted to propositional state variables. Instead they use the SAS^+ formalism [BN93], which allows for (finite-domain) multi-valued variables. Here, the counterpart of a "delete relaxation" is a relaxation which replaces the domains of variables in the relaxation by a *set* of reachable values. E.g. in a LOGISTICS domain with a variable modeling the position of a truck, the truck could be located in several cities at the same time in the relaxation. Instead of setting a variable to a new value, relaxed actions add the new value to the set of reachable values.

3.1.1 Complexity of Relaxed Classical Planning

In *classical planning*, the search space is finite and we will show that the relaxed plan existence problem can be decided in polynomial time, even for *classical* SAS^+ planning with conditional effects. As the search space is bounded in relaxed classical planning, and as the delete relaxation is monotone, each fact has to be achieved at most once. Similarly, applying a relaxed action could set some variables to new values. However, if this same action would be applied repeatedly, these values would not change again (at least for STRIPS-planning). This property is known as *idempotence*.

Definition 1 (Idempotence). An effect $e = \mathrm{eff}(v)$ on a variable $v \in \mathcal{V}$ is *idempotent* if $\mathrm{app}_e(s) = \mathrm{app}_e(\mathrm{app}_e(s))$ for all states s, where $\mathrm{app}_e(s) = \mathrm{app}(e, s)$.

An effect is *idempotent* if applying it to a state results in the same state as applying it to a state and then applying the effect to the resulting state again. Idempotence is a beneficial property that gives an a priori bound on the number of effects that have to be applied. Yet, idempotence is no necessary requirement for the polynomiality of the plan existence problem. The important aspect is the existence of an a priori bound on the number of effects that have to be applied. In classical planning formalisms that allow for conditional effects, applying an action can achieve a new value which also enables a conditional effect, so the number of required action applications is bounded by the number of conditional effects. We denote this property as *pseudo-idempotence*.

Definition 2 (Pseudo-Idempotence). An effect $e = \mathrm{eff}(v)$ on a variable $v \in \mathcal{V}$ is *pseudo-idempotent* if there exists a $k \in \mathbb{N}$ so that $\mathrm{app}_e^{k+1}(s) = \mathrm{app}_e^k(s)$ for all states s, where recursively $\mathrm{app}_e^n(s) = \mathrm{app}_e(\mathrm{app}_e^{n-1}(s))$ with base case $\mathrm{app}_e^0(s) = s$.

After k applications of the same effect, we reach a fix-point. We also say that e is *k-pseudo-idempotent* where the special case $k = 1$ is known as *idempotence* and $k = 0$ is known as *identity*.

Actions in classical planning with conditional effects are *k-pseudo-idempotent* for $k = |\mathrm{cond\_eff}(a_{\max})|$, where $a_{\max}$ is the action with most conditional effects in the planning task. We can now show polynomiality of the relaxed plan existence problem with the help of a *fix-point algorithm for relaxed parallel planning*.

Definition 3 (Parallel Fix-Point Algorithm). A *parallel fix-point algorithm* for *relaxed classical planning* is a planning algorithm that starts with the initial state and iteratively applies all applicable actions to the current relaxed state in parallel.

The algorithm terminates either if a fix point is reached of if the current relaxed state satisfies the goal condition.

As propositions are only added but never deleted in *relaxed planning*, the states accumulate more and more propositions in each step, or more and more values of a variable for a SAS^+ encoding.

Theorem 1 (Polynomiality of Relaxed Plan Existence Problem). *The relaxed plan existence problem is polynomial in* $|\mathcal{A}| \times \mathrm{cond}_{\max} \times |\mathcal{V}| \times \mathrm{dom}_{\max}$ *where* $\mathrm{cond}_{\max}$ *is the maximal number of conditional effects among all actions in the planning task plus one for unconditional effects, and* $\mathrm{dom}_{\max}$ *is the maximal number of values in the domain among all variables in the* SAS^+ *encoding of the relaxed classical planning task.*

Proof. The polynomial complexity of the relaxed plan existence problem is shown by applying the *parallel fix-point algorithm* (cf. Definition 3) for relaxed planning. The number of values that can be achieved is bounded by $|\mathcal{V}| \times \mathrm{dom}_{\max}$. In each iteration that does not reach a fix point, at least one new value has to be achieved for one of the variables. This bounds the number of parallel steps by the number actions in the task $|\mathcal{A}|$ times the number of conditional effects of the action with most conditional effects plus one $\mathrm{cond}_{\max}$. As no action can invalidate the preconditions of other actions, a serialized plan can be obtained by ordering parallel actions arbitrarily. □

The state space is infinite in *numeric planning* and we can use the concept of *pseudo-idempotence* to derive a relaxation framework where the plan existence problem can be decided in polynomial time as well.

3.2 Relaxations for Numeric Planning

In this section, we discuss extensions of delete relaxation to numeric planning. The idea behind delete relaxation is that values of a variable that are achieved once remain achieved. We discuss several ways to extend this concept to numeric planning.

3.2.1 Accumulation Semantics

In the *accumulation semantics*, instead of *changing* their values, variables *accumulate* all values achieved so far. The number of accumulated values after k parallel steps is finite, but generally exponential in k. Therefore, it quickly becomes infeasible to maintain the set of possible values, as can be seen in the following example.

Example 1. Let Π be a planning task with initial state $\mathcal{I}(v) = 0$ and actions $a_1 = \langle \emptyset, \{v \mathrel{+}= 1\} \rangle$ and $a_2 = \langle \emptyset, \{v \mathrel{\div}= 2\} \rangle$. Denoting by $r_k(v)$, $k = 0, \ldots, 3$, the reachable values of v after k parallel steps, we get $r_0(v) = \{0\}$, $r_1(v) = \{0, 1\}$, $r_2(v) = \{0, \frac{1}{2}, 1, 2\}$ and $r_3(v) = \{0, \frac{1}{4}, \frac{1}{2}, 1, \frac{3}{2}, 2, 3\}$, an exponentially growing set of values.

Besides this observation for *bounded* plan existence, one can also show that *unbounded* plan existence with respect to the accumulation semantics is still undecidable. To see this, we can adapt the undecidability proof for numeric planning by Helmert [Hel02]. A reduction of the search for solutions to Diophantine equations to numeric planning with respect to the accumulation semantics shows that the latter problem is undecidable, since solutions to Diophantine equations have to be integers and the delete relaxation does not relax this property.

3.2.2 Accumulation Semantics for Positive Tasks

One possible approach to dealing with the exploding number of accumulated values is the restriction to tasks where higher values are always better. Then, instead of storing *all values* a variable has attained so far, it is sufficient to store (an upper bound on) the *highest value*. A sufficient criterion for this is that all preconditions and goals have the form $v - c \trianglerighteq 0$, where v is a numeric variable and c a constant, and that all numeric effects only add or subtract a positive constant to or from a variable. The Metric-FF planner uses this type of relaxation, and Hoffmann [Hof03] shows that a large class of problems can be compiled into the required linear normal form.

3.2.3 Interval Semantics

One can handle a larger class of tasks with higher precision by only making the assumption that one of the *extreme* values, the highest or lowest, is best. This necessitates keeping track of two values for each variable v, a lower bound $\underline{v}$ and an upper bound $\overline{v}$. As long as preconditions and goals are comparisons of variables to constants and effects only add or subtract constants, it is insubstantial whether one considers the values between $\underline{v}$ and $\overline{v}$ reached or not. By considering the entire interval reached in a relaxed sense, one can handle even more expressive tasks. In particular, when allowing divisions in effects, besides "higher values are better" and "lower values are better", one also has the objective "values closer to c are better" for constants c. Then, if c is in the interval between $\underline{v}$ and $\overline{v}$, one may assume that values arbitrarily close to c can be reached, whereas otherwise, one can assess the proximity to c achieved so far. The Planning Domain Definition Language PDDL [MGH+98] that is used to model most planning problems allows for algebraic base operations which are also supported by interval arithmetic [You31]. As such, we consider interval relaxation a viable approach and focus on it in the following section.

3.3 Interval Relaxation

In this section we investigate the *interval relaxation* for numeric planning. We start by discussing the underlying interval arithmetic before we formally define interval relaxed planning tasks. We then discuss the complexity of the plan existence problem in this relaxation.

3.3.1 Interval Arithmetic

Interval arithmetic uses an upper and a lower bound to enclose the actual value of a variable. Closed intervals $[\underline{x}, \overline{x}] = \{q \in \mathbb{Q} \mid \underline{x} \leq q \leq \overline{x}\}$ contain all rational

numbers from $\underline{x} \in \mathbb{Q}$ to $\overline{x} \in \mathbb{Q}$. Throughout this thesis we refer to the lower bound of an interval x by $\underline{x}$ and to the upper bound by $\overline{x}$. The set $\mathbb{I}_c = \{[\underline{x}, \overline{x}] \mid \underline{x} \leq \overline{x}\}$ contains all closed intervals. Open intervals $(\underline{x}, \overline{x}) = \{q \in \mathbb{Q}^\infty \mid \underline{x} < q < \overline{x}\}$ and the respective set of open intervals $\mathbb{I}_o = \{(\underline{x}, \overline{x}) \mid \underline{x} < \overline{x}\}$, as well as half open intervals $[\underline{x}, \overline{x}) = \{q \in \mathbb{Q}^\infty \mid \underline{x} \leq q < \overline{x}\}$ and $(\underline{x}, \overline{x}] = \{q \in \mathbb{Q}^\infty \mid \underline{x} < q \leq \overline{x}\}$ and the respective sets $\mathbb{I}_{co} = \{[\underline{x}, \overline{x}) \mid \underline{x} < \overline{x}\}$ and $\mathbb{I}_{oc} = \{(\underline{x}, \overline{x}] \mid \underline{x} < \overline{x}\}$ are defined analogously. Note that open intervals are also allowed to contain positive or negative infinity, which is not allowed for closed interval bounds. The set of mixed-bounded intervals is $\mathbb{I}_m = \mathbb{I}_c \cup \mathbb{I}_o \cup \mathbb{I}_{oc} \cup \mathbb{I}_{co}$. Finally, $\emptyset$ is the empty interval and $\mathbb{I} = \mathbb{I}_m \cup \emptyset$ the set of all intervals.

Sometimes we want to make general statements on intervals where the openness is not relevant for the statement. In such situations we denote intervals with *indeterminate parentheses* $(\!|\underline{x}, \overline{x}|\!)$. If we want to refer to values $q \in x$ we use an *indeterminate comparison operator* $\lhd$ where $\underline{x} \lhd q$ is interpreted as $\underline{x} < q$ if the lower bound of x is open and $\underline{x} \leq q$ if the lower bound is closed. Analogously $q \lhd \overline{x}$ is interpreted as $q < \overline{x}$ if the upper bound is open and as $q \leq \overline{x}$ otherwise. The symmetric comparator "$\rhd$" is defined analogously.

Numbers q can be identified with the degenerate interval $[q, q]$. The interval resulting from basic arithmetic operations $x \circ y$ where $x, y \in \mathbb{I}$ is defined as $\{(q_x \circ q_y) | q_x \in x \text{ and } q_y \in y\}$. If both intervals x and y are non-empty, the basic operations can be computed as:

- addition: $(\!|\underline{x}, \overline{x}|\!) + (\!|\underline{y}, \overline{y}|\!) = (\!|\underline{x} + \underline{y}, \overline{x} + \overline{y}|\!)$,
- subtraction: $(\!|\underline{x}, \overline{x}|\!) - (\!|\underline{y}, \overline{y}|\!) = (\!|\underline{x} - \overline{y}, \overline{x} - \underline{y}|\!)$,
- multiplication: $(\!|\underline{x}, \overline{x}|\!) \times (\!|\underline{y}, \overline{y}|\!) = (\!|\min(\underline{xy}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{xy}), \max(\underline{xy}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{xy})|\!)$,
- division: the result of a division depends on whether or not y contains zero. $(\!|\underline{x}, \overline{x}|\!) \div (\!|\underline{y}, \overline{y}|\!) = (\!|\min\left(\underline{x}/\underline{y}, \underline{x}/\overline{y}, \overline{x}/\underline{y}, \overline{x}/\overline{y}\right), \max\left(\underline{x}/\underline{y}, \underline{x}/\overline{y}, \overline{x}/\underline{y}, \overline{x}/\overline{y}\right)|\!)$ if $0 \notin (\!|\underline{y}, \overline{y}|\!)$. Otherwise, at least one of the bounds diverges to $\pm\infty$, and one typically treats division $(\!|\underline{x}, \overline{x}|\!) \div (\!|\underline{y}, \overline{y}|\!)$ as multiplication $(\!|\underline{x}, \overline{x}|\!) \times \frac{1}{(\!|\underline{y}, \overline{y}|\!)}$ and splits $\frac{1}{(\!|\underline{y}, \overline{y}|\!)}$ into $\frac{1}{(\!|\underline{y}, 0)} = (-\infty, \frac{1}{\underline{y}}]$ and $\frac{1}{(0, \overline{y}|\!)} = (\!|\frac{1}{\overline{y}}, \infty)$. In case 0 is on the bound of the interval, only one of the split intervals contributes to the result. Finally, $\frac{1}{[0,0]} = \emptyset$, and the division evaluates to the empty interval $\emptyset$.

For more detailed information on interval arithmetic we refer to the literature [MKC09].

The openness of interval bounds is determined as follows: For *arithmetic operations* between interval bounds, whenever one of them contributes to the new interval bound, the bound is open, and only if both contributing bounds are closed the new bound is closed. For *minimum or maximum*, if there is a tie in the minimal or the maximal value, the bound is closed if at least one contributing bound is closed as well, and it is open otherwise.

Example 2. The product $(-2, 4] \times (-6, 3]$ is $(-24, 12]$. The lower bound is the result of 4×-6 and the left bound is open because one of the factors (namely -6) has an open bound. The new upper bounds is computed by the maximum of -2×-6 and 4×3 both yielding a value of 12. Because the contributing pair 4×3 consists of two closed bound factors, one of the values contributing to the maximum is closed and thus, the upper bound is closed.

The computation instructions for the basic operations $x \circ y$ can also be used for intervals with infinite bounds. The arithmetic operations yield the intuitively expected results that adhere to the IEEE 754 floating point guidelines as in the definition of a numeric planning task (cf. Section 2.1). The undefined value $-\infty + \infty = \bot$ cannot arise from interval addition or subtraction $x \pm y$, as $-\infty$ is always the lower bound and ∞ always the upper bound of an interval (e.g., $(-\infty, -\infty) \notin \mathbb{I}$). For multiplication, the critical values are multiplications of $\pm\infty$ and 0. We obtain the desired behavior $\{(q_x \times q_y) | q_x \in x \text{ and } q_y \in y\}$ by setting such values to 0 instead of $\bot$ with the rationale that an interval with an infinite bound also contains values $q < \infty$ (or $q > -\infty$) for which the result would be 0. The bound of this special zero is the same as the bound of the factor zero.

Example 3. The interval multiplication of $[3, \infty)$ and $(-5, 0]$ results in

$$[3, \infty) \times (-5, 0] = (\!|\min(-15, 0, -\infty, \mathbf{0}), \max(-15, 0, -\infty, \mathbf{0})|\!) = (-\infty, 0].$$

The special case $(-\infty, \infty) \times [0, 0]$ is covered by our definition as well:

$$(-\infty, \infty) \times [0, 0] = (\!|\min(\mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}), \max(\mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0})|\!) = [0, 0]$$

This leaves us with divisions by the point interval zero $[0, 0]$ that result in the empty interval $\emptyset$. As there exist no $q \in \emptyset$, there also exist no pair $q_x \circ q$ or respectively $q \circ q_x$ and the result of such arithmetic operations results in the empty interval $\emptyset \circ x = x \circ \emptyset = \emptyset$.

3.3.2 Interval Relaxed Numeric Planning Tasks

We will now formally define *interval relaxed* numeric planning tasks. In order to capture the relaxation idea "whatever we achieved once remains achieved" in numeric planning, intervals are expanded monotonously. As such, the result of a relaxed numeric action has to include the original values. The result of a relaxed numeric effect is the *convex union* of the old value and the new value.

Definition 4 (Convex Union). Let $x, y \in \mathbb{I}$ be intervals. The *convex union* $u = x \sqcup y$ is the convex hull of x and y:

$$x \sqcup u := \text{conv}(x, y) = \{\lambda q_x + (1 - \lambda) q_y | 0 \leq \lambda \leq 1, q_x \in x, q_y \in y\}.$$

For $x, y \in \mathbb{I}_m$ the *convex union* is the interval u with $\underline{u} = \min(\underline{x}, \underline{y})$ and $\overline{u} = \max(\overline{x}, \overline{y})$. Whether the bounds of u are open or closed depends on whether those of x and y are open or closed. If at least one interval of the *convex union* is empty, $x \sqcup \emptyset = \emptyset \sqcup x = x$ where $x \in \mathbb{I}$.

If there is a gap between the intervals x and y, i.e. $\overline{x} < \underline{y}$ or $\overline{y} < \underline{x}$, u also includes intermediate values not present in either x or y (namely all values from $\overline{x}$ to $\underline{y}$ or from $\overline{y}$ to $\underline{x}$ respectively).

We can now define *interval relaxed planning tasks*. The interval relaxation of a numeric planning task differs only marginally from the original task description on a syntactic level. The only difference are the domains of numeric variables which are now mapped to intervals. We allow mixed-bounded intervals in relaxed planning tasks even though open bounds can only be generated by limit value considerations which will become relevant for the repetition relaxation in the upcoming Section 3.4.

Definition 5 (Interval Relaxed Planning Task). Let Π be a numeric planning task (cf. Section 2.1). The corresponding *interval relaxed* planning task $\Pi^+ = \langle \mathcal{V}^+, \mathcal{A}^+, \mathcal{I}^+, \mathcal{G}^+, \kappa^+ \rangle$ of Π is a 5-tuple where $\mathcal{V}^+$ are the variables from Π with the domains replaced by mixed-bounded intervals $\mathrm{dom}(v) = \mathbb{I}_m$ for all $v \in \mathcal{V}^+$. The initial state $\mathcal{I}^+$ is derived from $\mathcal{I}$ by replacing numbers $\mathcal{I}(v)$ with degenerate intervals $\mathcal{I}^+(v) = [\mathcal{I}(v), \mathcal{I}(v)]$. Actions $\mathcal{A}^+ = \mathcal{A}$, the goal condition $\mathcal{G}^+ = \mathcal{G}$ and the cost function $\kappa^+ = \kappa$ remain unchanged.

The semantics of Π^+ draws on interval arithmetic. *Numeric expressions* are defined recursively. Constant expressions are interpreted as $s^+(c) = [c, c]$. Numeric expressions ξ_1 and ξ_2 are recursively evaluated $s^+(\xi_1 \circ \xi_2) = s^+(\xi_1) \circ s^+(\xi_2)$ for $\circ \in \{+, -, \times, \div\}$ where "$\circ$" now operates on intervals. For (goal and action) conditions, the relaxed semantics of numeric constraints is defined as follows: let ξ be a numeric expression and $\trianglerighteq \in \{\geq, >, =\}$ a comparison operator. Then $s^+ \vDash (\xi \trianglerighteq 0)$ iff $\exists q \in s^+(\xi)$ with $q \trianglerighteq 0$. This implies that intervals can be "greater" and "less" than zero at the same time. Note that for expressions that evaluate to $\bot$, $s^+ \nvDash (\bot \trianglerighteq 0)$ as $s^+(\bot) = \emptyset$.

For numeric actions, the relaxation idea that "values remain achieved" requires the intervals of variables to be monotonically expanding. This translates to assigning the convex union of the old and the new value to the variable v altered by a numeric effect $v \circ\!= \xi$ where $\circ\!= \in \{+=, -=, \times=, \div=, :=\}$. The state $\mathrm{app}^+(a, s^+) = s^{+\prime}$ resulting from an application of a is then

$$s^{+\prime}(v) = \begin{cases} s^+(v) \sqcup (s^+(v) + s^+(\xi)) & \text{for } += \\ s^+(v) \sqcup (s^+(v) - s^+(\xi)) & \text{for } -= \\ s^+(v) \sqcup (s^+(v) \times s^+(\xi)) & \text{for } \times= \\ s^+(v) \sqcup (s^+(v) \div s^+(\xi)) & \text{for } \div= \\ s^+(v) \sqcup s^+(\xi) & \text{for } := . \end{cases}$$

As we use the convex union from Definition 4, $s^{+\prime}(v)$ contains all values between the old value $s^+(v)$ of v and the evaluated expression $s^+(v) \circ s^+(\xi)$. Again, $s^{+\prime}(v) = s^+(v)$ if v occurs in no effect. Note that unlike expressions, variables never evaluate to the undefined value $\bot$: each variable is assigned a defined value in the initial state, and the intervals are monotonously expanded.

A relaxed plan π^+ is again a sequence of actions $\langle a_1^+, \ldots, a_n^+ \rangle$ that leads from $\mathcal{I}^+$ to a state satisfying $\mathcal{G}^+$ such that each action is applicable in the state that follows by executing the plan up to that action.

Example 4. Applying $a = \langle \emptyset, x \times\!= \xi \rangle$ in a state with $s^+(x) = [8, 10]$ and $s^+(\xi) = [-\frac{1}{2}, \frac{1}{2}]$ leads to $s^{+\prime}(x) = [8, 10] \sqcup ([8, 10] \times [-\frac{1}{2}, \frac{1}{2}]) = [8, 10] \sqcup [-5, 5] = [-5, 10]$.

The intervals $s^+(v)$ of a relaxed planning task are generated by continuously expanding the bounds of a degenerate interval $\mathcal{I}^+(v) = [\mathcal{I}(v), \mathcal{I}(v)]$. The *initial value* of this point interval is relevant in different contexts. In order to highlight that this initial value $\mathcal{I}(v)$ is a number, opposed to the interval $\mathcal{I}^+(v)$, we introduce a new symbol i_v for it:

Definition 6 (Initial Value). Let v be a variable that occurs in an *interval relaxed planning task*. We refer to the point $i_v \in \mathbb{Q}$ in the unrelaxed initial state $i_v = \mathcal{I}(v) = \underline{\mathcal{I}^+(v)} = \overline{\mathcal{I}^+(v)}$ as *initial value* of v.

We sometimes interpret the notion of an *initial value* more broadly and also apply it to expressions ξ with the same meaning, namely the point $i_\xi = \mathcal{I}(\xi)$ which the expression evaluates to in the initial state of the unrelaxed planning task. As all operations are monotone, $i_v \in s^+(v)$ in all reachable states s^+.

3.3.3 Interval Relaxation Complexity

The search space is infinite in *numeric planning*. As such, an a-priori bound on the required number of effects that have to be applied is not known. Nevertheless, we can use the parallel approach from classical planning (cf. Definition 3) for interval relaxed numeric planning as well. The monotonicity requirement ensures that interval relaxed actions can not invalidate the preconditions of other actions and can therefore be applied in parallel. While the sequential application of a set of actions does not necessarily yield the same intervals as the parallel application of this set, the sequentially reachable intervals are super-intervals of the intervals reachable from parallel execution. The complexity of the plan existence problem is therefore polynomial in $\mathcal{O}(\text{len}(\pi^+))$, where $\text{len}(\pi^+)$ is the length of a shortest plan.

If relaxed *numeric* planning operators were idempotent operations (cf. Definition 1), polynomiality in the *input* of the relaxed plan existence problem could be achieved even though the search space is unbounded. However, the number of values that can be reached by a numeric planning operator is unbounded, and applying the same effect repeatedly to a variable is possible and often necessary to achieve a goal.

Example 5. Let Π^+ be an interval relaxed numeric planning task with $\mathcal{V}^+ = \{v\}$, $\mathcal{A}^+ = \{a\}$ with $a = \langle \emptyset, v \mathrel{+}= 1 \rangle$, $\mathcal{I}^+ = \{v \mapsto [0, 0]\}$ and $\mathcal{G}^+ = \{v \geq 10\,000\}$.

The shortest plan is to apply action a 10 000 times. Then, $v \mapsto [0, 10\,000]$ which satisfies $\mathcal{G}^+$.

To overcome this problem, we will introduce a relaxation that simplifies the repeated application of actions by considering the values that are reachable by repeatedly applying such effects in one step (cf. Chapter 3.4) which makes effects *pseudo-idempotent* (cf. Definition 2), at least for *acyclic* numeric planning tasks (cf. Definition 12). We transform the planning task into a semi-symbolic representation that captures repeated application of numeric actions. We define *repetition relaxed* planning tasks, where we simulate the behavior of applying numeric effects arbitrarily often *independently*. As we will see later, the independence assumption is not justified for numeric effects $v \mathrel{\circ}= \xi$ where the assigned expression ξ contains (or indirectly depends on) the affected variable v. However, we succeed in showing that the plan existence problem is polynomial for tasks with *acyclic dependencies* as interval relaxed plans can be constructed from *repetition relaxed* plans in polynomial time.

3.4 Repetition Relaxation

In order to show polynomiality of the *interval-based* relaxation for numeric planning, we have to solve the problem that numeric actions are non-idempotent operations. The *repetition relaxation* is an interval-based relaxation which relaxes the *interval relaxation* of a numeric planning task even further. Instead

of applying a numeric effect only once, repetition relaxed actions are made pseudo-idempotent by yielding the interval that is reachable by applying the action arbitrarily often. While we use the *repetition relaxation* as a tool to show polynomiality for the *interval relaxation* at first, the idempotent actions in the repetition relaxed semantics are also beneficial to compute relaxation heuristics for numeric planning (cf. Chapter 4).

3.4.1 Repetition Relaxed Numeric Planning Tasks

Repetition relaxed planning tasks use mixed-bounded intervals to capture the attainable values of a numeric variable. We are interested in the behavior of numeric effects in the limit. This limit can be computed compactly by analyzing the *behavior* of the effect's arithmetic operator.

Example 6. Let a_1 be an action with an additive effect $v_1 \pm= c$ for $c \in \mathbb{Q}$ and $\pm= \in \{+=, -=\}$. If operator and constant c share the algebraic sign, applying a in a state s extends the upper bound of $s(v)$ and can extend that bound to any value by applying a multiple times. The same is true for the lower bound if the algebraic sign of operator and c are different. The result of applying an additive effect $v \pm= \xi$ arbitrarily often in s only depends on whether the evaluation of the assigned expression ξ contains negative values, zero or positive values.

The behavior of multiplicative effects $v \divideontimes= \xi$ where $\divideontimes= \in \{\times=, \div=\}$ is slightly more complex. Multiplicative effects can *contract* or *expand* depending on whether ξ contains elements with absolute value greater one and switch signs if ξ contains negative elements, resulting in up to seven different behaviors of ξ.

In order to define the semantics of *repetition relaxed* planning tasks, we first define the semantics of applying a numeric effect k times to an interval with the limit of applying it arbitrarily often to an interval.

Definition 7 (k-Repetition Successor). Let $x, e \in \mathbb{I}_m$ be mixed bounded intervals and $x \circ= e$ with $\circ= \in \{+=, -=, \times=, \div=, :=\}$ an assignment operating on intervals. Let $x_0 = x$ and

$$x_{i+1} = \begin{cases} x_i \sqcup (x_i + e) & \text{for } += \\ x_i \sqcup (x_i - e) & \text{for } -= \\ x_i \sqcup (x_i \times e) & \text{for } \times= \\ x_i \sqcup (x_i \div e) & \text{for } \div= \\ x_i \sqcup e & \text{for } := . \end{cases}$$

The *k-repetition successor* by $\mathrm{succ}^k_{\circ=}(x, e) = \bigcup_{i=0}^{k} x_i$ is the result of assigning e k times to x.

The limit of the *k-repetition successor* is then the result of applying the numeric effect arbitrarily often to an interval.

Definition 8 (Repetition Relaxed Interval Successor). The *repetition relaxed interval successor* $\mathrm{succ}_{\circ=}(x, e) = \mathrm{succ}^{\infty}_{\circ=}(x, e) = \bigcup_{i=0}^{\infty} x_i$ is the result of assigning e arbitrarily often to x.

As $x_{i+1} \supseteq x_i$ by definition of the convex union (cf. Definition 4) and because all x_i are convex, the resulting set $\text{succ}_{\circ=}(x, e)$ is an interval. However, open intervals can be generated in the limit even if x and e are closed.

For convenience we allow x or e to be empty with the consequential intervals $\text{succ}_{\circ=}(x, \emptyset) = x$, $\text{succ}_{\circ=}(\emptyset, e) = \emptyset$ for $\circ= \in \{+=, -=, \times=, \div=\}$ and $\text{succ}_{:=}(\emptyset, e) = e$.

As the repetition relaxed interval successor $\text{succ}_{\circ=}(x, e)$ is the limit of applying e arbitrarily often to x, $\text{succ}_{\circ=}(x, e)$ is an idempotent operation.

We can now define *repetition relaxed* numeric planning tasks.

Definition 9 (Repetition Relaxed Planning Task). Let Π be a planning task. The *repetition relaxation* of Π is a 5-tuple $\Pi^\# = \langle \mathcal{V}^\#, \mathcal{A}^\#, \mathcal{I}^\#, \mathcal{G}^\#, \kappa^\# \rangle$. The domains of numeric variables $\text{dom}(v) = \mathbb{I}_m$ for $v \in \mathcal{V}^\#$ are mixed-bounded intervals. The initial state $\mathcal{I}^\#$ maps variables again to the degenerate interval $\mathcal{I}^\#(v) = [\mathcal{I}(v), \mathcal{I}(v)]$. Actions $\mathcal{A}^\# = \mathcal{A}$, the goal condition $\mathcal{G}^\# = \mathcal{G}$ and the cost function $\kappa^\# = \kappa$ remain again unchanged.

As with the pure *interval relaxation*, the repetition relaxation differs mainly in the semantics of numeric effects. The semantics of *numeric expressions* operates on mixed-bounded intervals which are also supported by interval arithmetic. The interpretation of a numeric expression is given as $s^\#(\xi_1 \circ \xi_2) = s^\#(\xi_1) \circ s^\#(\xi_2)$ for expressions ξ_1 and ξ_2 and $\circ \in \{+, -, \times, \div\}$. Again, numeric constraints $\xi \trianglerighteq 0$, where ξ is a numeric expression and $\trianglerighteq \in \{\geq, >, =\}$ is a comparison operator, are satisfied in state $s^\# \vDash (\xi \trianglerighteq 0)$ iff $\exists q \in s^\#(\xi)$ with $q \trianglerighteq 0$, and $s^\# \nvDash (\bot \trianglerighteq 0)$.

The semantics of *numeric effects* captures the repeated application of actions. We use the *repetition relaxed interval successor* $\text{succ}_{\circ=}(x, e)$ of Definition 8 which is the interval obtained from applying a numeric assignment $x \circ= e$ arbitrarily often to an interval x. Numeric effects $v \circ= \xi$ are defined by fixing the variable $x = s^\#(v)$ and the expression $s^\#(\xi)$ to the interval they evaluate to before the application of the effect. The state $\text{app}^\#(a, s^\#) = s^{\#\prime}$ resulting from an application of a with effect $\text{eff} = \{\text{eff}_1, \ldots, \text{eff}_n\}$ is then $s^{\#\prime}(v) = \text{succ}_{\circ=}(s^\#(v), s^\#(\xi))$ for numeric effects $\text{eff}_i = (v \circ= \xi)$, and $s^{\#\prime}(v) = s^\#(v)$ if v occurs in no effect.

Repetition relaxed plans $\pi^\#$ are again sequences of actions $\langle a_1^\#, \ldots, a_n^\# \rangle$ that lead from $\mathcal{I}^\#$ to a state satisfying $\mathcal{G}^\#$ such that each action is applicable in the state that follows by executing the plan up to that action. The repetition relaxation $\Pi^\#$ of a planning task relaxes Π^+ further and plans for Π^+ are still plans for $\Pi^\#$. The reason is that each repeated action application can only extend the interval of affected numeric variables more.

3.4.2 Efficient Computation of the Repetition Relaxed Interval Successor

Fixing expressions ξ of numeric effects $v \circ= \xi$ to the interval $e = s(\xi)$ they evaluate to in the current state is beneficial for the computation of the successor. Changes in the assignment (which can be an arbitrary arithmetic expression) do not have to be considered immediately. This allows us to compute the repetition relaxed interval successor in a computationally efficient closed form.

$:=$		$\tilde{e}$
		$(-\infty,\infty)$
$\tilde{x}$	$(-\infty,\infty)$	$(\!\lvert\min(\underline{x},\underline{\mathbf{e}}),\max(\overline{x},\overline{\mathbf{e}})\rvert\!)$

$+=$		$\tilde{e}$		
		$(-\infty,0)$	$\{0\}$	$(0,\infty)$
$\tilde{x}$	$(-\infty,\infty)$	$(-\infty,\overline{x})$	$(\!\lvert\underline{x},\overline{x}\rvert\!)$	$(\!\lvert\underline{x},\infty)$

$-=$		$\tilde{e}$		
		$(-\infty,0)$	$\{0\}$	$(0,\infty)$
$\tilde{x}$	$(-\infty,\infty)$	$(\!\lvert\underline{x},\infty)$	$(\!\lvert\underline{x},\overline{x}\rvert\!)$	$(-\infty,\overline{x})$

$\times=$		$\tilde{e}$						
		$(-\infty,-1)$	$\{-1\}$	$(-1,0)$	$\{0\}$	$(0,1)$	$\{1\}$	$(1,\infty)$
$\tilde{x}$	$(-\infty,0)$	$(-\infty,\infty)$	$(\!\lvert\underline{x},-\underline{x}\rvert\!)$	$(\!\lvert\underline{x},\underline{x}\times\underline{\mathbf{e}}\rvert\!)$	$(\!\lvert\underline{x},0]$	$(\!\lvert\underline{x},0)$	$(\!\lvert\underline{x},\overline{x}\rvert\!)$	$(-\infty,\overline{x})$
	$\{0\}$				$[0,0]$			
	$(0,\infty)$	$(-\infty,\infty)$	$(\!\lvert-\overline{x},\overline{x}\rvert\!)$	$(\!\lvert\overline{x}\times\underline{\mathbf{e}},\overline{x}\rvert\!)$	$[0,\overline{x}\rvert\!)$	$(0,\overline{x}\rvert\!)$	$(\!\lvert\underline{x},\overline{x}\rvert\!)$	$(\!\lvert\underline{x},\infty)$

$\div=$		$\tilde{e}$						
		$(-\infty,-1)$	$\{-1\}$	$(-1,0)$	$\{0\}$	$(0,1)$	$\{1\}$	$(1,\infty)$
$\tilde{x}$	$(-\infty,0)$	$(\!\lvert\underline{x},\underline{x}\div\underline{\mathbf{e}}\rvert\!)$	$(\!\lvert\underline{x},-\underline{x}\rvert\!)$	$(-\infty,\infty)$	$\emptyset$	$(-\infty,\overline{x}\rvert\!)$	$(\!\lvert\underline{x},\overline{x}\rvert\!)$	$(\!\lvert\underline{x},0)$
	$\{0\}$		$[0,0]$		$\emptyset$		$[0,0]$	
	$(0,\infty)$	$(\!\lvert\overline{x}\div\underline{\mathbf{e}},\overline{x}\rvert\!)$	$(\!\lvert-\overline{x},\overline{x}\rvert\!)$	$(-\infty,\infty)$	$\emptyset$	$(\!\lvert\underline{x},\infty)$	$(\!\lvert\underline{x},\overline{x}\rvert\!)$	$(0,\overline{x}\rvert\!)$

Table 3.1: Partial behaviors for numeric effects

The successors $\mathrm{succ}_{\circ=}(x,e)$ of numeric effects $x \circ= e$ are defined by the limit $\bigcup_{i=0}^{\infty} x_i$ and we are interested in determining the result of such an effect as efficiently as possible. The idea is to consider the *behavior* of the change that the assignment e causes on x. Possible assignment behaviors are extension, contraction, flip and constancy. Which of these behaviors occur only depends on which of up to 21 symbolic *behavior classes* are covered by x and e. The behavior classes for x are $\mathbb{X} = \{(-\infty,0),\{0\},(0,\infty)\}$, and for e they are $\mathbb{E} = \{(-\infty,-1),\{-1\},(-1,0),\{0\},(0,1),\{1\},(1,\infty)\}$. We decompose e and x into the behavior classes they hit, i.e, where $e \cap \tilde{e} \neq \emptyset$ for a behavior class $\tilde{e} \in \mathbb{E}$ and $x \cap \tilde{x} \neq \emptyset$ for a behavior class $\tilde{x} \in \mathbb{X}$, respectively. Table 3.1 contains partial behaviors $\mathcal{T}_{\circ=}(x,e)$ for $\circ= \in \{:=,+=,-=,\times=,\div=\}$ where $\mathcal{T}_{\circ=}(x,e)$ is only defined if $x \subseteq \tilde{x} \in \mathbb{X}$ and $e \subseteq \tilde{e} \in \mathbb{E}$. $\mathcal{T}_{\circ=}(x,e)$ is the table entry with column $\tilde{x}$ and row $\tilde{e}$ in the table with the corresponding $\circ=$ operator. Divisions by the degenerate zero interval $[0,0]$ result in the empty interval $\emptyset$.

Before we will show that the partial behaviors $\mathcal{T}_{\circ=}(x,e)$ correspond to the *repetition relaxed interval successor* $\mathrm{succ}_{\circ=}(x,e)$ if x and e hit the respective behavior classes, we show the monotonicity of $\mathrm{succ}_{\circ=}(x,e)$ in the following Lemma.

Lemma 2 (Monotonicity of k-Repetition Successor). *The intervals x_i in the k-*repetition successor *(cf. Definition 7) are monotonically expanding, i.e. $x_i \subseteq x_{i+1}$ for $0 \leq i \leq k$.*

Proof. Monotonicity follows inductively from the definition of the k-repetition successor. Independently of the assignment operator, the interval x_{i+1} is defined as convex union of the interval x_i with a possibly empty or undefined interval. Whereas the monotonicity is therefore not necessarily strict, the convex union (cf. Definition 4) ensures that the k-repetition successor is monotone. □

Theorem 3 (Partial Behavior Table Entries are Correct). *The partial behaviors $\mathcal{T}_{\circ=}(x,e)$ equal $\mathrm{succ}_{\circ=}(x,e)$ for $x \subseteq \tilde{x} \in \mathbb{X}$ and $e \subseteq \tilde{e} \in \mathbb{E}$.*

We prove Theorem 3 exemplarily for two of the less obvious entries in Table 3.1. The proofs for the remaining cases can be done similarly and are therefore omitted. Our proceeding is to show $\mathrm{succ}_{\circ=}(x,e) = \mathcal{T}_{\circ=}(x,e)$ for a given table entry by showing $\mathrm{succ}_{\circ=}(x,e) \subseteq \mathcal{T}_{\circ=}(x,e)$ and $\mathrm{succ}_{\circ=}(x,e) \supseteq \mathcal{T}_{\circ=}(x,e)$. The repetition relaxed interval successor $\mathrm{succ}_{\circ=}(x,e)$ is defined as limit for $k \to \infty$ of $\mathrm{succ}^k_{\circ=}(x,e)$ (cf. Definition 8) and is monotone (cf. Lemma 2). Therefore, we usually determine a parameter k so that $q \in x_i$ for $i \geq k$ and $q \in \mathcal{T}_{\circ=}(x,e)$.

Proof for multiplication, $x \subseteq \tilde{x} = (0,\infty)$ and $e \subseteq \tilde{e} = (0,1)$: We have to show that $\mathrm{succ}_{\times=}(x,e) = (0,\overline{x}\rangle\!\rangle$. We take advantage of the contracting behavior of e. In the progression defining $\mathrm{succ}_{\times=}(x,e)$, starting from $x_0 = x$ we get

$$
\begin{aligned}
& x_{i+1} \\
&= x_i \sqcup (x_i \times e) \\
&= (\!|\underline{x_i} \sqcup (x_i \times e), \overline{x_i \sqcup (x_i \times e)}|\!) \\
&= (\!|\min(\underline{x_i}, \underline{x_i} \times \underline{e}, \underline{x_i} \times \overline{e}, \overline{x_i} \times \underline{e}, \overline{x_i} \times \overline{e}), \max(\overline{x_i}, \underline{x_i} \times \underline{e}, \underline{x_i} \times \overline{e}, \overline{x_i} \times \underline{e}, \overline{x_i} \times \overline{e})|\!) \\
&= (\!|\underline{x_i} \times \underline{e}, \overline{x_i}|\!)
\end{aligned}
$$

The last step can is shown inductively: as all numbers in $x_0 = x \subseteq (0,\infty)$ and $e \subseteq (0,1)$ are positive, so are numbers in the product $x \times e$, and therefore, inductively all numbers in x_i. The minimum is obtained for $\underline{x_i} \times \underline{e}$ and the maximum for $\overline{x_i}$ because e is a contraction ($0 < \underline{e} \leq \overline{e} < 1$). This even allows us to write x_i in a closed form namely $x_i = (\!|\underline{x} \times \underline{e}^i, \overline{x}|\!)$.

"$\subseteq$": In order to prove $\mathrm{succ}_{\times=}(x,e) \subseteq (0,\overline{x}\rangle\!\rangle$, we show that there exists an index $k \in \mathbb{N}$ with $q \in x_k$ and $x_k \subseteq (0,\overline{x}\rangle\!\rangle$. With our preliminary consideration we can reformulate x_k to $(\!|\underline{x} \times \underline{e}^k, \overline{x}|\!)$. The subset relation follows directly: the *lower bound* x_k is the product of $k+1$ numbers all of which are greater zero and as such, $0 < \underline{x_k} \lhd q$ ("$\lhd$" is interpreted as "$<$" or as "$\leq$" depending whether the lower bound is open or closed, cf. Section 3.3.1). For the *upper bound* we have $q \lhd \overline{x} \leq \overline{x}$ and therefore, we can conclude that $q \in (0,\overline{x}\rangle\!\rangle$.

"$\supseteq$": Now we have to show the converse direction $\mathrm{succ}_{\times=}(x,e) \supseteq (0,\overline{x}\rangle\!\rangle$. Let $q \in (0,\overline{x}\rangle\!\rangle$. We have to show that $q \in \mathrm{succ}_{\times=}(x,e)$ respectively that there exists a $k \in \mathbb{N}$ with $q \in x_k$. We can determine such a k by using the closed form again. For the upper bound we get $q \lhd \overline{x} \leq \overline{x}$ and for the lower bound we have to solve the inequality

$$
\begin{aligned}
& \underline{x} \times \underline{e}^k \lhd q \\
\Leftarrow \quad & \underline{e}^k \lhd q \div \underline{x} \\
\Leftarrow \quad & \log_{\underline{e}}(\underline{e}^k) \rhd \log_{\underline{e}}(q \div \underline{x}) \qquad (\underline{e} < 1 \to \log \text{ flips the inequality sign}) \\
\Leftarrow \quad & k \rhd \log_{\underline{e}}(q \div \underline{x})
\end{aligned}
$$

A suitable $k \in \mathbb{N}$ is $k = \lfloor \log_{\underline{e}}(q \div \underline{x}) \rfloor + 1$. □

Proof for division, $x \subseteq \tilde{x} = (-\infty,0)$ and $e \subseteq \tilde{e} = (-\infty,-1)$: We have to show that $\mathrm{succ}_{\div=}(x,e) = (\!|\underline{x}, \underline{x} \div \overline{e}|\!)$.

"$\subseteq$": We prove $\mathrm{succ}_{\div=}(x,e) \subseteq (\!|\underline{x}, \underline{x} \div \overline{e}|\!)$ by showing that every from $q \in \mathrm{succ}_{\div=}(x,e)$ there exists a $k \in \mathbb{N}$ with $q \in x_k = (\!|\underline{x_k}, \overline{x_k}|\!)$ so that $x_k \subseteq (\!|\underline{x}, \underline{x} \div \overline{e}|\!)$. We show the subset relation inductively for all $k \in \mathbb{N}$.
Base case: For $q \in x_0 = x$, it is easy to show that also $q \in (\!|\underline{x}, \underline{x} \div \overline{e}|\!)$. For the

lower bound $q \geq \underline{x}$ follows trivially from $\underline{x_0} = \underline{x}$. For the upper bound we know from $x \subseteq (-\infty, 0)$ and $e \subseteq (-\infty, -1)$ that $\underline{x}$, $\overline{x}$, $\underline{e}$ and $\overline{e}$ are all negative and therefore $\underline{x} \div \overline{e} > 0 > \overline{x}$ so the upper bound on the right hand side is always greater than the upper bound of $\overline{x_0}$ and we have $\underline{x} = \underline{x_0} \leq q \leq \overline{x_0} < \underline{x} \div \overline{e}$.
Inductive step, lower bound: We show that the lower bound $\underline{x_{i+1}} = \underline{x}$, with the induction hypothesis that $\underline{x_i} = \underline{x}$ holds. The new bound $\underline{x_{i+1}} = \underline{x_i \sqcup (x_i \div e)} = \min(\underline{x_i}, \underline{x_i} \div \underline{e}, \underline{x_i} \div \overline{e}, \overline{x_i} \div \underline{e}, \overline{x_i} \div \overline{e})$. We will now show that $\min(\underline{x_i}, \underline{x_i} \div \underline{e}, \underline{x_i} \div \overline{e}, \overline{x_i} \div \underline{e}, \overline{x_i} \div \overline{e}) = \underline{x_i}$. Because $-\infty < \underline{e} < \overline{e} < 0$ and $\underline{x_i}$ are negative, we can immediately rule out $\underline{x_i} \div \underline{e}$ and $\underline{x_i} \div \overline{e}$ from attaining the minimum. If $\overline{x_i}$ is also negative, $\overline{x_i} \div \underline{e}$ and $\overline{x_i} \div \overline{e}$ are positive as well and we are done. So we can assume that $\overline{x_i} \geq 0$. But then we have the following chain of inequalities

$$\begin{aligned} \underline{x_i} &= \underline{x} && (3.1)\\ &= (\underline{x} \div \overline{e}) \div \overline{e} && (3.2)\\ &\leq \overline{x_i} \div \overline{e} && (3.3)\\ &\leq \overline{x_i} \div \underline{e} && (3.4) \end{aligned}$$

where 3.1 because of induction hypothesis, 3.2 by division by the neutral element $1 = \overline{e} \div \overline{e}$ and associativity, 3.3 by plugging in the induction hypothesis $\overline{x_i} \leq (\underline{x} \div \overline{e})$ where the inequality sign is flipped as $\overline{e} < 0$ and finally, 3.4 because $\underline{e} \leq \overline{e} < 0$ are both negative and $|\underline{e}| \geq |\overline{e}|$. Whether $\overline{x_i}$ is positive or negative, the minimum is attained at $\underline{x_i}$ and therefore $\underline{x_{i+1}} = \underline{x_i} \geq \underline{x}$
Inductive step, upper bound: We now have to show that $\overline{x_{i+1}} \leq \underline{x} \div \overline{e}$. The new bound $\overline{x_{i+1}}$ is computed as $= \max(\overline{x_i}, \underline{x_i} \div \underline{e}, \underline{x_i} \div \overline{e}, \overline{x_i} \div \underline{e}, \overline{x_i} \div \overline{e})$. We will show now that the maximum is either attained at $\underline{x_i} \div \overline{e}$ or at $\overline{x_i}$. The fraction of two negative numbers attains its maximum when the dividend has its highest absolute value and the divisor has its lowest absolute value. As $\underline{e} < \overline{e} < 0$, the dividend has to be negative, and among the two candidates $\underline{x_i}$ and $\overline{x_i}$, $\underline{x_i}$ maximizes the fraction as it is either the only negative candidate or it has the higher absolute value. As $\underline{e} < \overline{e} < -1$ the lowest absolute value of the divisor is attained at $\overline{e}$. Thus, $\max(\underline{x_i} \div \underline{e}, \underline{x_i} \div \overline{e}, \overline{x_i} \div \underline{e}, \overline{x_i} \div \overline{e}) = \underline{x_i} \div \overline{e}$. With the induction hypothesis $\underline{x_i} \geq \underline{x}$ we also have $\underline{x_i} \div \overline{e} \leq \underline{x} \div \overline{e}$ as $\overline{e}$ is negative and flips the inequality sign. If the maximum is attained at $\overline{x_i}$ instead, the inductive step follows directly from the hypothesis $\overline{x_i} \leq \underline{x} \div \overline{e}$.
"$\supseteq$": We have to show that $\mathrm{succ}_{\div=}(x, e) \supseteq (\!|\underline{x}, \underline{x} \div \overline{e}|\!)$. Let $q \in (\!|\underline{x}, \underline{x} \div \overline{e}|\!)$. We have to show that it then also follows that $q \in \mathrm{succ}_{\div=}(x, e)$. As $\underline{x_k} = \underline{x}$ for all $k \in \mathbb{N}$ we only have to show that there exists a $k \in \mathbb{N}$ with $q \in x_k = (\!|\underline{x_k}, \overline{x_k}|\!)$ because $x_{i+1} \subseteq x_i$ and therefore $q \in \mathrm{succ}_{\div=}(x, e) = \bigcup_{i=0}^{\infty} x_i$. The maximum is obtained after $k = 1$ steps because the maximum to compute $\overline{x_{i+1}} = \underline{x_i} \div \overline{e}$ only depends on the lower bound $\underline{x_i}$ which equals $\underline{x}$ for all $k \geq 0$. □

This decomposition allows us to compute partial behaviors in constant time. Partitioning x and e into behavior classes and building the union of all partial behaviors results in an interval.

Lemma 4. *The union of the decomposition of the intervals x and e into behavior classes $\bigcup_{\tilde{x} \in \mathbb{X}, \tilde{e} \in \mathbb{E}} \mathcal{T}_{\circ=}(x \cap \tilde{x}, e \cap \tilde{e})$ is an interval.*

Proof. Let r be the union $\bigcup_{\tilde{x} \in \mathbb{X}, \tilde{e} \in \mathbb{E}} \mathcal{T}_{\circ=}(x \cap \tilde{x}, e \cap \tilde{e})$ of all partial behaviors. Each table entry $\mathcal{T}_{\circ=}(x_p, e_p)$ for the partitions $x_p = x \cap \tilde{x}$ and $e_p = e \cap \tilde{e}$ is an interval. We prove by contradiction that r has to be an interval. Assume that r is not an

interval. As r consists of intervals this means that r has gaps. Each table entry $\mathcal{T}_{\circ=}(x_p, e_p)$ equates to the repetition relaxed interval successor $\mathrm{succ}_{\circ=}(x_p, e_p)$ for the respective partial intervals x_p and e_p (cf. Theorem 3). Repetition relaxed interval successors are defined with the convex union (cf. Definition 8), which ensures that each interval $\mathcal{T}_{\circ=}(x_p, e_p) = \mathrm{succ}_{\circ=}(x_p, e_p) \supseteq x_p$ is a super-interval of the partition x_p. Therefore, if r had a gap, the split of x into x_p would not be a partitioning but a split into intervals introducing a gap, which contradicts the partitioning of x_p. □

With such a decomposition, repetition relaxed numeric effects can now be computed using table look-ups. Unfortunately, the union of the partial behaviors of an effect does not equal the succeeding interval according to the semantics.

False Hypothesis 5. *The repetition relaxed interval successor* $\mathrm{succ}_{\circ=}(x, e)$ *of an effect* $x \circ= e$ *equals the union of the successors obtained by decomposition of the effect into behavior classes, i.e.* $\bigcup_{\tilde{x}\in\mathbb{X},\tilde{e}\in\mathbb{E}} \mathrm{succ}_{\circ=}(x \cap \tilde{x}, e \cap \tilde{e}) = \mathrm{succ}_{\circ=}(x, e)$.

Hypothesis 5 does not hold in general, as the following example illustrates. The successor can grow into behavior classes which were not covered initially:

Example 7. Let $a = \langle \emptyset, \{v \times= \xi\}\rangle$ have an effect on v in a state with $s^\#(v) = [1, 4]$ and $s^\#(\xi) = [-\frac{1}{2}, 2]$. The partial behaviors are:

$$\begin{aligned}
\mathrm{succ}_{\times=}([1,4],[-\tfrac{1}{2},0)) &= [-2,4]\\
\mathrm{succ}_{\times=}([1,4],[0,0]) &= [0,4]\\
\mathrm{succ}_{\times=}([1,4],(0,1)) &= (0,4]\\
\mathrm{succ}_{\times=}([1,4],[1,1]) &= [1,4]\\
\mathrm{succ}_{\times=}([1,4],(1,2]) &= [1,\infty).
\end{aligned}$$

However, the union $\bigcup_{\tilde{x}\in\mathbb{X},\tilde{e}\in\mathbb{E}} \mathrm{succ}_{\times=}(s^\#(v) \cap \tilde{x}, s^\#(\xi) \cap \tilde{e}) = [-2, \infty)$, differs from $\mathrm{succ}_{\times=}(s^\#(v), s^\#(\xi)) = (-\infty, \infty)$.

Yet, this counterexample does not invalidate our approach. The number of behavior classes is restricted, and therefore, new classes can only be hit a restricted number of times. We correct the hypothesis by including the partial behaviors $\mathcal{T}_{\circ=}(x, e)$ of the classes hit by x in a nested fix-point iteration:

Definition 10 (Decomposition Successor). Let $x, e \in \mathbb{I}_m$ be mixed-bounded intervals and the sequence $\langle x_j\rangle$ for a parameter $j \in \mathbb{N}_0$ be recursively defined as

$$\begin{aligned}
x_0 &= x\\
x_{j+1} &= \bigcup_{\tilde{x}\in\mathbb{X},\tilde{e}\in\mathbb{E}} \mathrm{succ}_{\circ=}(x_j \cap \tilde{x}, e \cap \tilde{e}).
\end{aligned}$$

The *decomposition successor* $\widetilde{\mathrm{succ}}_{\circ=}(x, e) = \bigcup_{j=0}^{\infty} x_j$ is the fix-point of this sequence.

Now, newly attained behavior classes become part of the decomposition in the next iteration, and we will later show that $\mathrm{succ}_{\circ=}(x, e) = \widetilde{\mathrm{succ}}_{\circ=}(x, e)$ (cf. Theorem 7).

Example 8. Recall Example 7 starting with $x_0 = s^\#(v) = [1, 4]$ and $e = s^\#(\xi) = [-\frac{1}{2}, 2]$. The successor $x_1 = \bigcup_{\tilde{x}\in\mathbb{X},\tilde{e}\in\mathbb{E}} \text{succ}_{\times=}(x_0 \cap \tilde{x}, e \cap \tilde{e}) = [-2, \infty)$ hits new behavior classes for x. In the next iteration, the combination of the newly achieved behavior class $\tilde{x} \cap x_1 = (-\infty, 0) \cap [-2, \infty) = [-2, 0)$ and $e = [-\frac{1}{2}, 2]$ contains among others the successor $\text{succ}_{\times=}([-2, 0), (1, 2]) = (-\infty, 0)$. The union still contains partial behaviors that set the upper bound to ∞, so $\text{succ}_{\times=}(x_1 \cap \tilde{x}, e \cap \tilde{e}) = (-\infty, \infty)$. Now, a fix-point is reached and $\widetilde{\text{succ}}_{\circ=}(x, e) = \text{succ}_{\circ=}(x, e)$.

The fix-point is reached after at most two steps, and computing the *decomposition successor* by partitioning the effect into behavior classes is therefore 2-pseudo-idempotent.

Lemma 6 (Fix Point of Decomposition Successor). *The* decomposition successor *sequence $\langle x_j \rangle$ (c.f. Definition 10) converges after at most two steps.*

The *decomposition successor* sequence $\langle x_j \rangle$ consists of intervals (cf. Lemma 4) x_j which are monotonously expanding, because each partial effect is based on the convex union (cf. Definition 8). The sequence converges as soon as $x_j = x_{j-1}$. A bound of x_{j-1} can only be extended if some behavior class $\tilde{x} \in \mathbb{X}, \tilde{e} \in \mathbb{E}$ yields an interval $\mathcal{T}_{\circ=}(x_{j-1} \cap \tilde{x}, e \cap \tilde{e})$ that extends x_{j-1}. The evaluated expression $s^\#(\xi) = e$ is fixed for interval successors, and only changes on x_{j-1} can cause x_j to change again.

We will now investigate all possible progressions of the sequences $\langle x_j \rangle$ based on the assignment operator $\circ= \in \{:=, +=, -=, \times=, \div=\}$ to show Lemma 6.

Proof for $x := e$: The interval $x_0 = x$ is assigned e and

$$\begin{aligned} x_1 &= (\!|\min(\underline{x_0}, \underline{e}), \max(\overline{x}_0, \overline{e})|\!) \\ x_2 &= (\!|\min(\underline{x_1}, \underline{e}), \max(\overline{x_1}, \overline{e})|\!) \\ &= (\!|\min(\min(\underline{x_0}, \underline{e}), \underline{e}), \max(\max(\overline{x_0}, \overline{e}), \overline{e})|\!) \\ &= (\!|\min(\underline{x_0}, \underline{e}), \max(\overline{x}_0, \overline{e})|\!) \\ &= x_1 \end{aligned}$$

In the case of an assignment $x := e$ the fix-point is already reached after one step. □

Proof for $x \mathrel{+}= e$: Let again $x_0 = x$. For the interval x_1, depending on whether $e \cap (-\infty, 0)$ is empty or not, the lower bound $\underline{x_1}$ will be either $\underline{x_0}$ or $-\infty$. Similarly, the upper bound $\overline{x_1}$ will be either $\overline{x_0}$ or ∞ depending on whether the behavior class $e \cap (0, \infty)$ is hit. The exact same reasoning holds for x_2 and because e is fixed, $x_2 = x_1$ and again, after one step a fix-point is reached. □

Proof for $x \mathrel{-}= e$: The proof is analogous to the case for $x \mathrel{+}= e$ with the behavior classes of e exchanged. Again, a fix-point is reached after one step. □

Proof for $x \mathrel{\times}= e$: Let again $x_0 = x$. The behaviors for multiplication depend on the combinations of behavior classes $\tilde{x}$ and $\tilde{e}$ that are hit. The value of x_1 is determined by all partial behaviors which are hit by x_0 and by e. Table 3.2 is a copy of Table 3.1 for multiplication and division where all entries that are relevant for this proof are either marked in black, green or red. The interval bounds that are marked in black are already contained in x_0 and thus, the

$\times=$		$\tilde{e}$						
		$(-\infty,-1)$	$\{-1\}$	$(-1,0)$	$\{0\}$	$(0,1)$	$\{1\}$	$(1,\infty)$
$\tilde{x}$	$(-\infty,0)$	$(-\infty,\infty)$	$(\!\lfloor \underline{x}, -\underline{x} \rfloor\!)$	$(\!\lfloor \underline{x}, \underline{x} \times \underline{e} \rfloor\!)$	$(\!\lfloor \underline{x}, 0 \rfloor\!]$	$(\!\lfloor \underline{x}, 0)$	$(\!\lfloor \underline{x}, \overline{x} \rfloor\!)$	$(-\infty, \overline{x}]\!)$
	$\{0\}$	$[0,0]$						
	$(0,\infty)$	$(-\infty,\infty)$	$(\!\lfloor -\overline{x}, \overline{x} \rfloor\!)$	$(\!\lfloor \overline{x} \times \underline{e}, \overline{x} \rfloor\!)$	$[0, \overline{x}]\!)$	$(0, \overline{x}]\!)$	$(\!\lfloor \underline{x}, \overline{x} \rfloor\!)$	$(\!\lfloor \underline{x}, \infty)$

$\div=$		$\tilde{e}$						
		$(-\infty,-1)$	$\{-1\}$	$(-1,0)$	$\{0\}$	$(0,1)$	$\{1\}$	$(1,\infty)$
$\tilde{x}$	$(-\infty,0)$	$(\!\lfloor \underline{x}, \underline{x} \div \underline{e} \rfloor\!)$	$(\!\lfloor \underline{x}, -\underline{x} \rfloor\!)$	$(-\infty,\infty)$	$\emptyset$	$(-\infty, \overline{x}]\!)$	$(\!\lfloor \underline{x}, \overline{x} \rfloor\!)$	$(\!\lfloor \underline{x}, 0)$
	$\{0\}$	$[0,0]$			$\emptyset$	$[0,0]$		
	$(0,\infty)$	$(\!\lfloor \overline{x} \div \overline{e}, \overline{x} \rfloor\!)$	$(\!\lfloor -\overline{x}, \overline{x} \rfloor\!)$	$(-\infty,\infty)$	$\emptyset$	$(\!\lfloor \underline{x}, \infty)$	$(\!\lfloor \underline{x}, \overline{x} \rfloor\!)$	$(0, \overline{x}]\!)$

Table 3.2: Partial behaviors for multiplication and division. Unchanged bounds are marked in black, idempotent bounds in green and non-idempotent bounds in red.

respective partial behaviors can not extend a bound of x_1. Interval bounds marked in green can cause a change but the respective partial behaviors are idempotent in the sense that the sequence $\langle x_j \rangle$ would converge after 1 step if only these behaviors are hit. E.g. for $\tilde{e} = (-\infty, -1)$ and $\tilde{x} = (-\infty, 0)$ the interval $x_1 = (-\infty, \infty)$ is a fix-point and cannot be extended any further. For $\tilde{e} = \{0\}$, and $\tilde{x} = (0, \infty)$ the new bound $[0, \overline{x}]\!)$ does not only extend the lower bound but it even hits a new behavior class. However, the only behavior class hit is $\tilde{x} = \{0\}$ which does not contain expanding successors independently of e. All other partial behaviors marked in green expand the respective bound in an idempotent way as well.

This restricts critical combinations to assignments e that hit $\tilde{e} = \{-1\}$ and $\tilde{e} = (-1, 0)$. The critical bound extensions are marked in red in Table 3.2. Here, the new value of the lower bound depends on the value of the upper bound and vice versa. As such, $\langle x_j \rangle$ could be a sequence that alternately extends the lower and the upper bound without ever reaching a fix-point. However, we will show now that such a sequence cannot be constructed and a fix-point is already reached after two steps. We will now systematically analyze combinations of $\underline{e}$ and $\overline{e}$ to show that the *decomposition successor* is 2-*pseudo-idempotent*. In the following case analysis, we first differentiate the lower bound of the effect $\underline{e}$, and then the upper bound $\overline{e}$ in a nested differentiation. We combine adjacent behaviors if appropriate to simplify our proof.

Case 1 ($\underline{e} < -1$). If e hits the behavior class $\tilde{e} = (-\infty, -1)$, $x_1 = (-\infty, \infty)$ unless $x_0 = [0, 0]$ and in both cases a fix-point is already reached after one step.

Case 2 ($\underline{e} = -1$). If we hit the critical behavior class $\tilde{e} = \{-1\}$ with $\underline{e}$ we have to differentiate the behavior class hit by the upper bound $\overline{e}$ as well.

Case 2.1 ($\overline{e} = -1$). If also $\overline{e} = -1$, $x_1 = x_2 = (\!\lfloor \min\{\underline{x}, -\overline{x}\}, \max\{\overline{x}, -\underline{x}\} \rfloor\!)$.

Case 2.2 ($\overline{e} \in (-1, 0)$). Both behavior classes of $\mathbb{E}$ reach the same values $\underline{x} \times \underline{e} = \underline{x} \times -1 = -\underline{x}$ and $\overline{x} \times \underline{e} = \overline{x} \times -1 = -\overline{x}$. Therefore we reach the same values as in Case 2.1: $x_1 = x_2 = (\!\lfloor \min\{\underline{x}, -\overline{x}\}, \max\{\overline{x}, -\underline{x}\} \rfloor\!)$.

Case 2.3 ($\overline{e} \in [0, 1]$). For $\tilde{e} = \{0\}$, $\tilde{e} = (0, 1)$ and $\tilde{e} = \{1\}$, all new behavior intervals are $(\!\lfloor \min\{\underline{x}, -\overline{x}\}, \max\{\overline{x}, -\underline{x}\} \rfloor\!)$ as the only extended (green) bounds add values up to zero.

Case 2.4 ($\overline{e} \in (1, \infty)$). If both $\underline{x}$ and $\overline{x}$ are negative, we get $x_1 = (-\infty, -\underline{x}]\!)$ from $x_1 = (\!\lfloor \min\{\underline{x}, -\infty\}, \max\{-\underline{x}, \underline{x} \times \underline{e}, 0, \overline{x}\} \rfloor\!)$ and we reach a fix-point after

the upper bound (which is now positive) gets expanded by $\tilde{e} = (1, \infty)$ yielding $x_2 = (-\infty, \infty)$. For $\underline{x} < 0$ and $\overline{x} > 0$, the expanding behavior $\tilde{e} = (1, \infty)$ already yields $x_1 = (-\infty, \infty)$ after one step, but if both $\underline{x}$ and $\overline{x}$ are positive, we get a progression analogous to the negative case and $x_1 = (\!|-\overline{x}, \infty)$ and $x_2 = (-\infty, \infty)$ again. The behaviors from $\tilde{x} = \{0\}$ do not add new values.

Case 3 ($\underline{e} \in (-1, 0)$). We will now consider cases where the lower bound is in the second critical behavior class.

Case 3.1 ($\overline{e} \in (-1, 0)$). Again, we reach a fix-point after one step: the interval $x_1 = (\!|\min\{\underline{x}, \overline{x} \times \underline{e}\}, \max\{\overline{x}, \underline{x} \times \underline{e}\}|\!)$ can not be extended because e is contracting for all $q \in e$. If $\tilde{x} = (-\infty, 0)$ is hit, the upper bound is extended to $\overline{x_1} = \underline{x} \times \underline{e}$ but then for x_2 the newly hit behavior class neither extends the lower bound ($\overline{x_1} \times \underline{e} > \underline{x}_1$ nor the upper bound ($\overline{x_1} = \overline{x_1}$). And analogously for $\tilde{x} = (0, \infty)$.

Case 3.2 ($\overline{e} \in [0, \infty)$). The reasoning for the other behavior classes that could be hit by $\overline{e}$ is analogous as for the Cases 2.3–2.4 where $\underline{e} = -1$.

Case 4 ($\underline{e} \in [0, \infty)$). No critical behavior class is hit and a fix-point is reached after one step.

Concluding, effects intervals where $\underline{e} \in [-1, 0)$ and $\overline{e} \in (1, \infty)$ require two steps until they converge to a fix point and all other intervals reach the fix-point of the *decomposition successor* after only one step. □

Proof for $x \div= e$: Let again $x_0 = x$. The proof for division is somewhat similar to the case of multiplication. However, the number of critical combinations is much lower because a division by a number close to zero causes at least one bound diverge to infinity, often reaching the fix-point interval $(-\infty, \infty)$ immediately. As for multiplication, we marked the relevant interval bounds in Table 3.2. Interval bounds are depicted in black if the bound is not extended at all. If the bound is extended in an idempotent way, we mark the bound green. Finally, potentially critical bound extensions are marked in red. Again, we systematically analyze all possible combinations of $\underline{e}$ and nest the differentiation of the upper bound $\overline{e}$.

Case 1 ($\underline{e} \in (-\infty, -1]$). We start with cases where the lower bound hits one of the potentially critical behavior classes.

Case 1.1 ($\overline{e} \in (-\infty, -1]$). We reach a fix-point $(\!|\min\{\underline{x}, \overline{x} \div \overline{e}\}, \max\{\overline{x}, \underline{x} \div \overline{e}\}|\!)$ after one step as in the proof for multiplication with $\overline{e} = -1$ and $\overline{e} \in (-1, 0)$ (cf. Case 2.1–2.2 and Case 3.1). The argumentation is the same as for multiplication because we perform the reciprocal operation.

Case 1.2 ($\overline{e} \in (-1, \infty)$). We immediately reach the fix-point $(-\infty, \infty)$ as the behavior class $\tilde{e} = (-1, 0)$ is hit.

Case 2 ($\underline{e} \in (-1, \infty)$). No critical behavior classes are hit and we converge after one step.

Concluding, all repetition relaxed interval divisions reach a fix-point after only one step. □

We know from Lemma 6 that the sequence defining the *decomposition successor* $\widetilde{\text{succ}}_{\circ=}(x, e)$ reaches a fix-point after at most two steps. As such, it is feasible to compute the *repetition relaxed interval successor* with the help of reformulating it to the *decomposition successor*, which is shown by the following Theorem:

Theorem 7. *Let $x, e \in \mathbb{I}_m$ be intervals. The repetition relaxed interval successor* $\text{succ}_{\circ=}(x, e)$ *of an interval assignment $x \circ= e$ equates to the fix-point of the* decomposition successor*, i.e.* $\widetilde{\text{succ}}_{\circ=}(x, e) = \text{succ}_{\circ=}(x, e)$.

Proof. We show equality by proving that $\widetilde{\text{succ}}_{\circ=}(x, e) \subseteq \text{succ}_{\circ=}(x, e)$ as well as $\widetilde{\text{succ}}_{\circ=}(x, e) \supseteq \text{succ}_{\circ=}(x, e)$.

To see that $\widetilde{\text{succ}}_{\circ=}(x, e) \subseteq \text{succ}_{\circ=}(x, e)$ consider the first decomposition of the sequence $\langle x_j \rangle$ defining $\widetilde{\text{succ}}_{\circ=}(x, e)$: all partial effects $\text{succ}_{\circ=}(x \cap \tilde{x}, e \cap \tilde{e})$ are operations on subsets of x and e. The *repetition relaxed interval successor* $\text{succ}_{\circ=}(x, e)$ is monotone in both arguments i.e. $\breve{x} \subseteq x \wedge \breve{e} \subseteq e \Rightarrow \text{succ}_{\circ=}(\breve{x}, \breve{e}) \subseteq \text{succ}_{\circ=}(x, e)$. Therefore, x_1 is the union of sub-intervals which are all part of $\text{succ}_{\circ=}(x, e)$. Iteratively for x_2 we know that $x_1 \subseteq \text{succ}_{\circ=}(x, e)$. Again $\text{succ}_{\circ=}(x, e)$ is monotone in both arguments and all partial effects if x_2 are subsets of $\text{succ}_{\circ=}(x_1, e)$. As x_2 is already the fix-point of $\widetilde{\text{succ}}_{\circ=}(x, e)$ (cf. Lemma 6), we have $\widetilde{\text{succ}}_{\circ=}(x, e) \subseteq \text{succ}_{\circ=}(x, e)$.

The converse direction $\widetilde{\text{succ}}_{\circ=}(x, e) \supseteq \text{succ}_{\circ=}(x, e)$ is shown by contradiction. Let $q \in \text{succ}_{\circ=}(x, e)$ but not in $\widetilde{\text{succ}}_{\circ=}(x, e)$. Both successor functions are defined recursively starting with $x_0 = x$. Therefore, $q \notin x_0$. Let $\text{succ}^k_{\circ=}(x, e)$ be the sequence of intervals defining $\text{succ}_{\circ=}(x, e)$. There has to be a $k > 0$ with $x_{k+1} = x_k \sqcup (x_k \circ e)$ so that $x_k \subset \widetilde{\text{succ}}_{\circ=}(x, e)$ but $x_{k+1} \not\subset \widetilde{\text{succ}}_{\circ=}(x, e)$. After k steps, the bound of the successor is extended beyond the decomposition $\widetilde{\text{succ}}_{\circ=}(x, e)$ for the first time. As $x_k \subset \widetilde{\text{succ}}_{\circ=}(x, e)$, $q \notin x_k$ but the value has to be in the second part of the interval x_{k+1}: $q \in (x_k \circ e)$. The resulting interval depends on $\underline{x_k}$, $\overline{x_k}$, $\underline{e}$, $\overline{e}$ and in case of division also on whether $0 \in e$. Each combination of these extreme bounds is contained in one partial behavior $\mathcal{T}_{\circ=}(x_k, e)$, which also contains all values entailed by the convex union up to this new value. If $(x_k \circ e)$ hits a new behavior class or extends the bounds within a behavior class, this is a contradiction to $\widetilde{\text{succ}}_{\circ=}(x, e)$ being a fix-point. If $(x_k \circ e)$ stays within a behavior, this is a contradiction to $\mathcal{T}_{\circ=}(x_k, e)$ being correct (cf. Theorem 3). Thus, such a k cannot be found, and therefore, it is impossible for $q \in \text{succ}_{\circ=}(x, e)$ but not $q \in \widetilde{\text{succ}}_{\circ=}(x, e)$. □

The decomposed successor $\widetilde{\text{succ}}_{\circ=}(x, e)$ allows as to compute the result of applying an action a in state $s^\#$, $\text{app}^\#(a, s^\#)$, under the repetition relaxed semantics in constant time. This allows us to use the *parallel fix-point algorithm* (cf. Definition 3) from the classical case analogously: apply all applicable actions in parallel until a fix-point is reached, where the value of a variable is the convex union of all values that can be achieved by applying an action to the current relaxed state.

Theorem 8. *The parallel fix-point algorithm for repetition relaxed planning is sound, i.e. if the algorithm outputs an alleged plan, it is indeed a plan for $\Pi^\#$.*

Proof. Actions are only applied if the precondition is fulfilled and only outputs an alleged plan if the goal condition is satisfied in the last relaxed state. □

k	$s_k^\#(v)$	$s_k^\#(\xi)$
0	$[-1,-1]$	$[0,0]$
1	$[-1,0]$	$[-0.5,0]$
2	$[-1,0.5]$	$[-0.75,0]$
3	$[-1,0.75]$	$[-0.875,0]$
4	$[-1,0.875]$	$[-0.9375,0]$
5	$[-1,0.9375]$	$[-0.96875,0]$
6	$[-1,0.96875]$	$[-0.984375,0]$
7	$[-1,0.984375]$	$[-0.9921875,0]$
	$\vdots$	$\vdots$

Table 3.3: Progression of the state $s_k^\#(v)$ for applying an action k times whose effect multiplies v by $\xi = -\frac{v+1}{2}$.

We are interested in sequential and not in parallel plans the following theorem shows that sequential plans can be obtained from parallel plans easily:

Theorem 9. *Ordering parallel actions arbitrarily in a parallel repetition relaxed numeric plan yields a valid sequential plan.*

Proof. Applying an action can only extend the intervals of affected variables (i.e. variables which appear as effects of the action). Therefore, if a numeric constraint is satisfied in a state s, it is also satisfied in the successor state s' after applying an arbitrary relaxed action, i.e. $s \vDash \xi \trianglerighteq 0 \Rightarrow s' \vDash \xi \trianglerighteq 0$. As the preconditions of all parallel actions remain achieved, the order in which parallel actions are applied can not invalidate any action that was applicable in parallel.

For the effects, we again have monotonicity in both arguments of $\mathrm{succ}_{\circ=}(x,e)$ i.e. $\breve{x} \supseteq x \wedge \breve{e} \supseteq e \Rightarrow \mathrm{succ}_{\circ=}(\breve{x},\breve{e}) \supseteq \mathrm{succ}_{\circ=}(x,e)$. While the resulting intervals may not be the same as for parallel application, the intervals reached by sequentially applying actions are super-intervals of the intervals reachable by applying actions in parallel. □

Unfortunately, the parallel algorithm does not necessarily terminate. In the definition of the semantics of a repetition relaxed planning task, we fix the effect $s^\#(\xi) = e$ and the value of the variable $s^\#(v) = x$ even if ξ depends on v. However, this implicit independence assumption is not justified. Critical entries are marked in **bold red** in Table 3.1. They occur for multiplicative effects that contract x and flip the arithmetic sign at the same time, as well as for assignment effects $\mathcal{T}_{:=}(x,e) = (\!|\min(\underline{x},\underline{\mathbf{e}}), \max(\overline{x},\overline{\mathbf{e}})|\!)$. In such cases, v does not only depend on ξ, but ξ also depends on v and the evaluated intervals $s^\#(\xi) = e$ and $s^\#(v) = x$ are cyclically dependent.

Example 9. Let $a = \langle \emptyset \to \{v \times\!= \xi\} \rangle$ for $\xi = -1 \times ((v+1) \div 2)$ and let $\mathcal{G} = \{v - 1 \geq 0\}$ be the goal. Let furthermore $s^\#(v) = s_0^\#(v) = [-1,-1]$. Applying action a arbitrarily often according to the repetition semantics yields the progression $s_k^\#$ for k action applications as depicted in Table 3.3.

The interval $s^\#(v)$ does *not* only change a restricted number of times, so the fix-point algorithm for interval relaxed numeric planning will not terminate.

If we succeeded in directly computing the fix-point to which the intervals converge with a symbolic interval we could continue the fix-point algorithm from here. In Example 9 we would set $s^{\#'}(v) = [-1, 1)$ resulting in $s^{\#'}(\xi) = (-1, 0]$.

Unfortunately, to the best of our knowledge, a general and yet efficient (i.e. polynomial time computable) approach to determine this fix-point is not known. Neither are we aware of a proof that such a general approach does not exist. The problem in Example 9 is that ξ depends on v. We conjecture that we could substitute variables that depend on v by a function that depends on v, at least if there is only one cyclic dependency. If the arithmetic expressions do not contain division, resolving these functions results in higher order polynomials that depend on v. A necessary conditions for equations $v = p(v)$ where $p(v)$ is a higher order polynomial, is that the derivation $p'(v) = 0$. So candidate fix-points coincide with polynomial roots of $p'(v)$. However, besides that there are no closed form solutions for polynomials with a degree of 5 or more, the values of v are intervals and it is not apparent which bound is relevant at which step. This is already under the simplifying assumptions that there is only one cycle and arithmetic expressions do not contain division, and a general computation instruction of the fix-point could become even more involved with several cycles and divisions.

Thus, we restrict planning tasks to contain only effects where the assigned expression is independent of the affected variable. For our heuristics we discuss methods that handle such cyclic dependencies in Section 4.2. For now, we restrict ourselves to planning tasks where the aforementioned problem does not occur. We show that such planning tasks are solvable in polynomial time.

Definition 11 (Dependency of Variables). A variable v_1 is *directly dependent* on a variable v_2 in task Π if there exists an $a \in \mathcal{A}$ with a numeric effect $v_1 \circ= \xi$ so that v_2 occurs in ξ.

Definition 12 (Acyclic Dependency Task). A planning task with acyclic *direct dependency* relation is an *acyclic dependency* task.

Note that the definition of *direct dependence* is purely syntactic. A task is considered to be *cyclic* even if a semantically identical task is *acyclic*. Examples include inapplicable actions with a dependent effect, or semantically equivalent representations of effects (e.g. in the effect $v := v + 5$, v is *dependent* on itself while it is not for $v \mathrel{+}= 5$).

Example 10. Consider a planning task with the four actions depicted on the left in Figure 3.1. The *dependency graph* of this task, depicted on the right, is *acyclic* and as such a *topology*. While v_1 occurs in an effect of action a_1, the assignment $\xi = 2$ of the division only consists of constants and therefore, v_1 does not *depend* from other variables. *Dependency* can be induced by assignments of a single variable, e.g., v_3 depends on v_1 because of the first effect of a_2, as well as by more sophisticated expressions, e.g., v_4 depending on v_1 and v_2 because of the second effect of a_2. A variable can occur in the assigned expression multiple times (v_1 in a_3's first effect on v_5) and the same dependency relation can be induced by several effects, e.g., v_8 depends on v_4 because of the second effect of a_1 and also because of the second effect of a_4.

The *dependency graph* of this planning task is *acyclic* and variables of the task can be partitioned into *dependency layers* $V_0^{\#} = \{v_1, v_2\}$, $V_1^{\#} = \{v_3, v_4, v_5\}$,

$$a_1 : \left\langle \varnothing, \left\{ \begin{array}{l} v_1 \div\!= 2 \\ v_8 \mathrel{+}= v_2 \times (v_4 \div 3) \end{array} \right\} \right\rangle,$$

$$a_2 : \left\langle \varnothing, \left\{ \begin{array}{l} v_3 \mathrel{-}= v_1 \\ v_4 := (3 + v_1) \div v_2 \end{array} \right\} \right\rangle,$$

$$a_3 : \left\langle \varnothing, \left\{ \begin{array}{l} v_5 := (v_1 - v_2) \div (3 \times v_1) \\ v_6 \div\!= v_3 \div v_4 \end{array} \right\} \right\rangle,$$

$$a_4 : \left\langle \varnothing, \left\{ \begin{array}{l} v_7 \times\!= (v_5 - 3) \\ v_8 \mathrel{+}= v_4 - v_7 \end{array} \right\} \right\rangle.$$

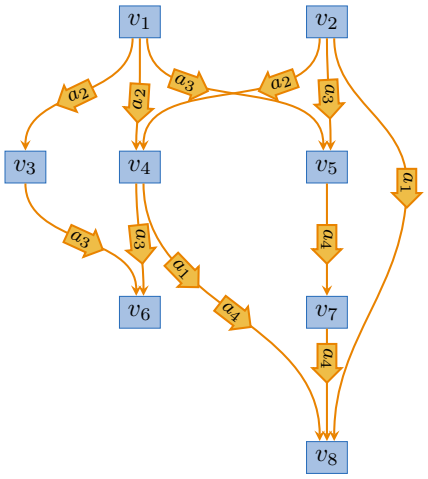

Figure 3.1: Left: a planning task with 4 actions, right: the corresponding *dependency graph* which is *acyclic*.

$V_2^\# = \{v_6, v_7\}$ and $V_3^\# = \{v_8\}$. A possible topological ordering is $v_2 \prec v_1 \prec v_4 \prec v_3 \prec v_5 \prec v_6 \prec v_7 \prec v_8$.

Theorem 10. *The parallel fix-point algorithm for repetition relaxed planning terminates for acyclic dependency tasks.*

Proof. As the planning task has acyclic dependencies, the direct dependency relation induces a topology. Let a *phase* of the algorithm be a sequence of parallel action applications, where no new action becomes applicable. During each phase, we consider numeric effects in topological order concerning the dependency graph. Let $V_l^\# \subseteq V^\#$ be the variables in dependency layer l. We iterate over the layers $k \geq 0$ of the topology assuming that a fix-point is reached for all variables $V_k^\#$ starting with the variables $V_0^\#$ that only depend on constants. Variables $V_{k+1}^\#$ only depend on variables $V_l^\#$ with $0 \leq l \leq k$ or on constants. A fix-point is reached for all those variables by induction hypothesis. Inductively, we can assume that the expressions of numeric effects that alter the variables of layer $V_{k+1}^\#$ are fixed. Therefore, the successor $\mathrm{succ}_{\circ=}(x, e)$ of an effect $v \circ\!= \xi$ with $x = s^\#(v)$ and $e = s^\#(\xi)$ does not change the variable more than twice, because for fixed intervals, the successor converges after two steps (cf. Lemma 6). The number of steps required for one phase is k where k is the number of topology layers which is bounded by the number of variables $|\mathcal{V}^\#|$.

The number of phases is restricted, with the same argument as for the fix-point algorithm in the classical case (cf. Theorem 1). Preconditions cannot be invalidated and during each phase at least one previously inapplicable action must become applicable. The number of phases is therefore restricted to the number of actions in the planning task.

Finally, actions are 2-pseudo-idempotent (cf Lemma 6) and can be evaluated in two steps.

In conclusion, repetition relaxed plans have at most $2 \cdot |\mathcal{V}^\#| \cdot |\mathcal{A}^\#|$ steps. □

Theorem 11. *The parallel fix-point algorithm for repetition relaxed planning is complete for acyclic dependency tasks.*

Proof. We prove completeness by contradiction and show that it is impossible that the algorithm terminates and reports "unsolvable" although a plan exists. Now assume there is a plan, but the algorithm terminates and reports "unsolvable". All actions are applied as soon as they are applicable, so a satisfiable condition must have been unsatisfied. Therefore, a satisfiable numeric constraint was not achieved by the algorithm. This implies that an effect $v \circ\!= \xi$ would have been able to assign a value to a variable that was not reached by our algorithm. Therefore, the successor defined by the semantics $\mathrm{succ}_{\circ=}(s^{\#}(v_n), s^{\#}(\xi))$ has to be different from the successor computed by the algorithm $\widetilde{\mathrm{succ}}_{\circ=}(s^{\#}(v_n), s^{\#}(\xi))$, which contradicts (cf. Theorem 7). □

Until now we have an algorithm which can compute parallel plans for repetition relaxed planning tasks in polynomial time for acyclic dependency tasks. With Theorem 9 we can also straighten parallel plans into repetition relaxed sequential plans. Yet, we are interested in plans for the interval relaxation without the symbolic description of numeric variables. We will show that we can derive interval relaxed plans π^{+} from repetition relaxed plans $\pi^{\#}$ in Theorem 15. The idea of our proof is to regressively explicate the number of action applications from a plan obtained by the fix-point algorithm for repetition relaxed planning, where a regressive step is the application of one action. In order to determine the number of repetitions we also explicate *target values* in the intervals of each variable which are sufficient to satisfy all numeric constraints of subsequent actions or the goal. The number of action repetitions k that is required to reach the chosen target values is obtained from the behavior class enabling the target value and coincides with the repetitions from the proof of Theorem 3. Before we prove constructively that interval relaxed plans can be derived from repetition relaxed ones in Theorem 15, we provide some preliminaries.

In order to determine target values for each variable, we have to decompose numeric expressions so that target values in the decomposition achieve the target value in the composed expression.

Definition 13 (Target Value Decomposition). Let $x, y \in \mathbb{I}_m$ be mixed bounded intervals, $\circ \in \{+, -, \times, \div\}$ and $q \in x \circ y$. We refer to this number q as *target value*. A *target value decomposition* for $x, y, \circ$ and q is a pair $\langle q_x, q_y \rangle$ where $q_x \in x$, $q_y \in y$ and $q_x \circ q_y = q$.

Lemma 12 (Existence of Target Value Decomposition). *Let $x, y \in \mathbb{I}_m$ be mixed bounded intervals, $\circ \in \{+, -, \times, \div\}$ and $q \in x \circ y$. Then, there exist a target value decomposition for $x, y, \circ$ and q.*

Proof. By definition, $x \circ y = \{(q_x \circ q_y) | q_x \in x \text{ and } q_y \in y\}$. Let $q \in x \circ y$ then there exists a pair $\langle q_x, q_y \rangle$ so that $q = q_x \circ q_y$. □

We can compute a target value decomposition by restricting the feasible target value choices of q_x to values which have a partner $q_y \in y$ so that $q_x \circ q_y = q$.

For $q \in x \circ y$ where $q \in \mathbb{Q}$ and $x, y \in \mathbb{I}_m$, we are looking for target values $q_x \circ q_y = q$. Feasible choices for q_x are not only restricted to the interval x, but they also need a *partner* $q_y \in y$. Depending on $\circ \in \{+, -, \times, \div\}$ this results in

the following values for q_x:

$$\begin{aligned} q_x + q_y = q &\rightarrow q_x = q - q_y && \text{for addition} \\ q_x - q_y = q &\rightarrow q_x = q + q_y && \text{for subtraction} \\ q_x \times q_y = q &\rightarrow q_x = q \div q_y && \text{for multiplication} \\ q_x \div q_y = q &\rightarrow q_x = q \times q_y && \text{for division.} \end{aligned}$$

The set x_c of all potential candidates q_x which have a partner in y can then be determined in the following way, using the reciprocal operation:

$$x_c = \begin{cases} [q,q] - y & \text{for addition} \\ [q,q] + y & \text{for subtraction} \\ [q,q] \div y & \text{for multiplication} \\ [q,q] \times y & \text{for division.} \end{cases}$$

The feasible choices for q_x are then all values in $x_c \cap x$. This intersection is not necessarily an interval, as x_c is computed by means of a division by an interval possibly containing zero.

Example 11. Let $x = [2,4]$ and $y = [3,7]$, and $q = 10$ in the interval $x + y = [2,4] + [3,7] = [5,11]$. All candidates x_c that have a partner in y so that $q_x + q_y = 10$ are $[10,10] - [3,7] = [3,7]$. Feasible choices for q_x are all target values in the intersection $[2,4] \cap [3,7] = [3,4]$. We can chose an arbitrary $q_x \in [3,4]$ and its corresponding partner to obtain a *target value decomposition* e.g. $\langle 3,7\rangle$, $\langle 4,6\rangle$ or $\langle 3.217, 6.783\rangle$.

In order to decompose target values in a state sequence that is generated by a relaxation, intervals enclose values that cannot be reached in reality. As such, target values q could be in the unreachable gap of divisions by an interval containing zero. While q is not *reachable* in this case, it is *relaxed reachable*. If we are to decompose q into target values for x and y this means that we are looking for target values so that relaxed reachability of q is implied by them. If q lies inside a gap, we can reach a value on both sides of the gap if we include an additional target value for y. Therefore, a relaxed *target value decomposition* is a triple that includes an auxiliary target value.

Definition 14 (Target Value Decomposition (Relaxed Semantics)). Let $x, y \in \mathbb{I}_m$ be mixed bounded intervals, $\circ \in \{+,-,\times,\div\}$ and $q \in \mathbb{Q}$ be a number in the result $q \in \mathrm{conv}(x \circ y)$. A *relaxed target value decomposition* for $x, y, \circ$ and q is a triple $\langle q_x, q_y^{\vdash}, q_y^{\dashv}\rangle$ so that $q_x \in x$, $q_y^{\vdash}, q_y^{\dashv} \in y$ and $q \in \mathrm{conv}([q_x, q_x] \circ [q_y^{\vdash}, q_y^{\dashv}])$.

Lemma 13 (Existence of Target Value Decomposition (Relaxed Semantics)). *Let $x, y \in \mathbb{I}_m$ be mixed bounded intervals, $\circ \in \{+,-,\times,\div\}$ and $q \in \mathrm{conv}(x \circ y)$. Then, there exists a relaxed target value decomposition $\langle q_x, q_y^{\vdash}, q_y^{\dashv}\rangle$ for $x, y, \circ$ and q.*

Proof. If $q \in x \circ y$ in the unrelaxed interval arithmetic (cf. Section 3.3.1), there exists a target value decomposition with Lemma 12 and we can set $q_y^{\vdash} = q_y^{\dashv} = q_y$. Otherwise, we can assume that $x \circ y$ is a division splitting $x \div y$ where $0 \in y$ so that the result is split into two intervals and that q lies in the gap between these intervals. We will show now that every triple $\langle q_x, q_y^{\vdash}, q_y^{\dashv}\rangle$ where $q_x \in x$,

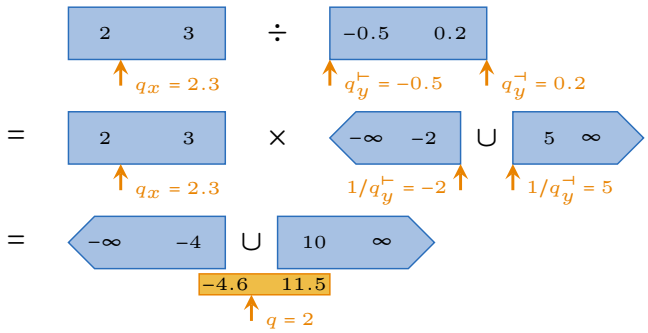

Figure 3.2: The target value $q = 2$ lies in the gap of $[2, 3] \div [-0.5, 0.2]$

$q_y^\vdash, q_y^\dashv \in y$ and $q_y^\vdash < 0$ and $q_y^\dashv > 0$ is a target value decomposition of q. Such target values exist, as divisions by y only introduce a gap, if it contains values less than zero as well as values greater than zero.

Let $e_1, e_2 \in \mathbb{I}_m$ be the two intervals resulting from the division $x \div y$, where $e_1 = x \times (-\infty, \frac{1}{\underline{y}}\rparen$ and $e_2 = x \times \lparen\frac{1}{\overline{y}}, \infty)$ (cf. Section 3.3.1). As $q_y^\vdash$ and $q_y^\dashv$ differ in their algebraic sign, so do $q_x \div q_y^\vdash$ and $q_x \div q_y^\dashv$. By the definition of interval division and depending on whether q_x is positive or negative, one of these results is in e_1 and the other in e_2. In both cases $q \in \mathrm{conv}([q_x, q_x] \circ [q_y^\vdash, q_y^\dashv])$, as q is in the gap between e_1 and e_2. □

Example 12. Let $x = [2, 3]$ and $y = [-0.5, 0.2]$ be intervals and $q = 2 \in \mathrm{conv}([2, 3] \div [-0.5, 0.2]) = (-\infty, \infty)$ be a target value in the relaxed result as illustrated in Figure 3.2. This target value lies in the gap resulting from a division by an interval containing zero and is therefore only relaxed reachable. Nevertheless we can find a relaxed target value decomposition, in our example $\langle 2.3, -0.5, 0.2\rangle$ where $q_x = 2.3 \in x = [2, 3]$, $q_y^\vdash = -0.5 \in y = [-0.5, 0.2]$ and $q_y^\vdash < 0$ as well as $q_y^\dashv = 0.2 \in y = [-0.5, 0.2]$ and $q_y^\dashv > 0$ resulting in $q_x \div q_y^\vdash = -4.6$ and $q_x \div q_y^\dashv = 11.5$ so that $q \in [-4.6, 11.5]$.

Instead of a single target value q_y, relaxed target value decomposition provides a pair of target values $q_y^\vdash$ and $q_y^\dashv$ (cf. Lemma 13). If we recursively decompose an expression to determine target values for all variables occurring in the expression, using two target values for each sub-expression would cause an exponential blow-up in the number of target values to determine for the expression. However, we can usually save one of the target values if we have access to the *initial value* (cf. Definition 6) i_y of y. Depending on whether i_y is positive or negative, we can continue the decomposition with either $q_y^\vdash$ or with $q_y^\dashv$. The exception to this rule are the rare occurrences where the initial value is inaccessible, most notably if y originates from the result of a numeric expression that is undefined in the initial state (e.g. $\xi = v_4 \div 0$).

Next, we determine a target value regression for numeric effects based on the *repetition relaxed interval successor* $\mathrm{succ}_{\circ=}(x, e)$ (cf. Definition 8), i.e. target values in the intervals $q_x \in x$ and $q_e \in e$ contributing to the interval successor, so that a desired target value q is reached.

As the computation of the repetition relaxed interval successor $\mathrm{succ}_{\circ=}(x, e)$ is based on the 2-pseudo-idempotent (cf. Lemma 6) decomposition successor,

the *target value regression* of a numeric effect is defined with an auxiliary target value again, this time for the effect interval e.

Definition 15 (Target Value Regression). Let $x, e \in \mathbb{I}_m$ be mixed bounded intervals and $x \circ\!= e$ be an effect where $\circ\!= \in \{+=, -=, \times=, \div=\}$ and $q \in \mathrm{succ}_{\circ=}(x, e)$.

Then, a *target value regression* $\langle q_x, q_e^{\vdash}, q_e^{\dashv}, k\rangle$ is a quadruple where $q_x \in x$, $q_e^{\vdash}, q_e^{\dashv} \in e$ and $k \in \mathbb{N}$ so that $q \in \mathrm{succ}^k_{\circ=}([q_x, q_x], [q_e^{\vdash}, q_e^{\dashv}])$.

Lemma 14 (Existence of Target Value Regression). *Let $x, e \in \mathbb{I}_m$ be mixed bounded intervals and $q \in \mathrm{succ}_{\circ=}(x, e)$. There exists a* target value regression $\langle q_x, q_e^{\vdash}, q_e^{\dashv}, k\rangle$ *for q.*

Proof. We first distinguish whether q is already in the interval x before the application of the effect or not.

Case 1 ($q \in x$). If q is already in x, all quadruples $\langle q, q_e^{\vdash}, q_e^{\dashv}, 0\rangle$ with $q_e^{\vdash}, q_e^{\dashv} \in e$ are target value regressions, and $q \in \mathrm{succ}^0_{\circ=}([q, q], [q_e^{\vdash}, q_e^{\dashv}]) = [q, q]$ follows directly from the definition.

Case 2 ($q \notin x$). As $q \in \mathrm{succ}_{\circ=}(x, e)$ but not in x, the target value q has to be achieved by the repeated application of the effect $x \circ\!= e$. From Theorem 7 we can compute the successor interval $\mathrm{succ}_{\circ=}(x, e)$ by $\widetilde{\mathrm{succ}}_{\circ=}(x, e)$. We will now look at the sequence $\langle x_j\rangle$ defining $\widetilde{\mathrm{succ}}_{\circ=}(x, e)$.

The sequence $\langle x_j\rangle$ is defined as union (cf. Lemma 6) of partial behaviors $\mathcal{T}_{\circ=}(x_{j-1} \cap \tilde{x}, e \cap \tilde{e})$ (cf. Table 3.1) for behavior classes $\tilde{x} \in \mathbb{X}$ and $\tilde{e} \in \mathbb{E}$. This sequence reaches a fix-point after one step for all assignment operators $\circ \in \{+, -, \times, \div\}$ except for multiplications that are both *expanding* and *flipping*. Thus, most of the time, there exists at least one combination of behavior classes $\tilde{x}$ and $\tilde{e}$ so that $q \in x_1 = \mathcal{T}_{\circ=}(x_0 \cap \tilde{x}, e \cap \tilde{e})$. This partial behavior extends the bounds of $x_0 = x$ to an interval containing the desired target value as $q \notin x$ but $q \in \mathcal{T}_{\circ=}(x_0 \cap \tilde{x}, e \cap \tilde{e})$. The sole exception for expanding and flipping multiplications is then treated in the respective sub-case.

Case 2.1 ($x := e$). For assignment effects, $\langle x_j\rangle$ reaches a fix-point after one step where $x_1 = x \sqcup e$. If $q \in e$, the quadruple $\langle q_x, q, q, 1\rangle$ is a target value regression for all $q_x \in x$ as $q \in \mathrm{succ}^1_{:=}([q_x, q_x], [q, q]) = [q_x, q_x] \sqcup [q, q] = [\min(q, q_x), \max(q, q_x)]$. Otherwise, there has to be a gap between x and e and q is in this gap. Thanks to the convex union, any pair of target values q_x and q_e will enclose the gap and therefore also q in which case $\langle q_x, q_e, q_e, 1\rangle$ is a *target value regression* for all $q_x \in x$ and all $q_e \in e$.

Case 2.2 ($x \pm\!= e$). For additive effects where $q \notin x$, we can reach q from any pair of target values $q_x \in x \cap \tilde{x}$ and $q_e \in e \cap \tilde{e}$ from the behavior class that achieves q. If $q > \overline{x}$ it is the behavior class that extends the upper bound of x and vice versa. A repetition count $r \in \mathbb{Q}^+$ corresponding to such q_x and q_e is obtained by the solution of the equation $q_x \pm r \cdot q_e = q$ and $k \in \mathbb{N}$ is then obtained by rounding r up to the next integer $k = \lceil r\rceil$. A target value regression is then the quadruple $\langle q_x, q_e, q_e, k\rangle$.

The target value $q \in \mathrm{succ}^{\lceil r\rceil}_{\pm=}([q_x, q_x], [q_e, q_e])$ as the arithmetic operations are continuous and the interval bounds q_x and $q_x \pm \lceil r\rceil \cdot q_e$ enclose q.

Case 2.3 ($x \divideontimes\!= e$, $q \in x_1$). Even for multiplicative effects, the sequence $\langle x_j\rangle$ converges after one step most of the time, so that $q \in x_1$. As $q \notin x = x_0$,

there exists again one combination of behavior classes $\tilde{x} \in \mathbb{X}$ and $\tilde{e} \in Be$ that achieves q. Any pair of target values $q_x \in x \cap \tilde{x}$ and $q_e \in e \cap \tilde{e}$ is suitable to reach q.

Similarly to the additive effects (cf. Case 2.2), a repetition constant $r \in \mathbb{Q}^+$ is obtained by solving equations $q_x \circledast q_e^r = q$ and again, $k = \lceil r \rceil$ yields an integer $k \in \mathbb{N}$. Note that k is determined analogously to the k from the proof of Theorem 3. A target value regression is then the quadruple $\langle q_x, q_e, q_e, \lceil r \rceil \rangle$ where $q \in \text{succ}_{\circledast=}^{\lceil r \rceil}([q_x, q_x], [q_e, q_e])$ with the same reasoning that multiplication and division are continuous operations (at least within the behavior class which exclude the singularity of division).

Case 2.4 ($x \circledast= e$, $q \notin x_1$). In case the decomposition successor requires two steps to reach a fix-point, we are in the scenario where we first have to flip the algebraic sign of q_x by multiplying with a negative target value $q_e^\vdash < 0$, and then, we have to expand the resulting interval by repetitively multiplying it with $q_e^\dashv > 1$.

Quadruples $\langle q_x, q_e^\vdash, q_e^\dashv, k+1 \rangle$ are then target value regressions if $q_x \in x$, $q_e^\vdash \in e \cap (-\infty, 0)$ and $q_e^\dashv \in e \cap (1, \infty)$. Such target values exist, as $q \notin x_1$ and there are no other combinations that require two steps (cf. Lemma 6).

The corresponding k is determined by solving the equation $q_x \times q_e^\vdash \times q_e^{\dashv r} = q$ for r. Again $k = \lceil r \rceil$ is obtained by rounding r up to the next integer. As we first have to flip the algebraic sign by multiplying q_x with $q_e^\vdash$, we need an additional repetition. This way, $q \in \text{succ}_{\circledast=}^{\lceil r \rceil + 1}([q_x, q_x], [q_e^\vdash, q_e^\dashv])$, which can basically be traced back to Case 2.3 which becomes obvious if we think of $q'_x = q_x \times q_e^\vdash$ as intermediate regression target value for x after one step. □

This allows us to formulate the following plan equivalence theorem:

Theorem 15. *Let $\pi^\# = \langle a_1, a_2, \ldots, a_n \rangle$ be a sequential repetition relaxed plan. There exist $k_1, \ldots, k_n \in \mathbb{N}^+$ such that $\pi^+ = \langle a_1^{k_1}, a_2^{k_2}, \ldots, a_n^{k_n} \rangle$ is an interval relaxed plan, where $a_i^{k_i}$ denotes a sequence of k_i repetitions of action a_i.*

Proof. In a first step we identify the intervals $s_i^\#$ reachable after the i-th repetition relaxed step. As $\pi^\#$ is a valid plan, the intermediate states $s_i^\#$ satisfy the preconditions of the succeeding action $s_i^\# \vDash \text{pre}(a_{i+1})$ or the goal $s_n^\# \vDash \mathcal{G}$. Generally, these intervals can be more extended than they need to be in order to satisfy the numeric constraints of action a_{i+1} or the goal $\mathcal{G}$. This is particularly relevant for open interval bounds where the sequence defining the repetition relaxed interval success converges to 0 or diverges to $\pm\infty$. We will regressively determine explicit target values in the intervals of $s_i^\#$ which are sufficient to satisfy the conditions of a_{i+1} or of $\mathcal{G}$. A regressive step consists of three tasks: first, a *target value determination task* where we determine up to two target values for each variable in $s_{i+1}^\#$, so that reaching both target values for each variable is sufficient to satisfy all constraints of a_{i+1} or of $\mathcal{G}$. Second, a *target value regression task* where we ensure that each target value is either reachable in $s_i^\#$ or by an effect of the action a_i. Third, a *repetition determination task*, where the number of repetitions of action a_i is determined as maximum repetition count from the *target value regressions* (cf. Definition 15) identified in the second task.

Target value determination task: For each variable in state $s_i^\#$ we have to determine up to two target values so that all conditions of a_i (or the goal condition $\mathcal{G}$ for $s_n^\#$) are satisfied. Two target values are sufficient, even if a variable v appears in the expressions of more than two numeric constraints. The state sequence $\langle s_i^\# \rangle$ is monotonously expanding for each variable v, starting with *initial values* (cf. Definition 6) i_v. If there are several occurrences of target values q_v for the same variable v in the constraints, using the highest target value q_v^+ and the lowest target value q_v^- among them is sufficient. If $q_v^- \in s_i^\#(v)$ and $q_v^+ \in s_i^\#(v)$, the interval $s_i^\#(v)$ includes all other target values as well, because the convex union is used in the relaxation. In fact, if all target values on one variable are greater (less) than the initial value i_v, only the highest (lowest) is sufficient to satisfy all constraints.

The values that are required to satisfy a constraint of the action a_i (or of the goal if $i = n$) are obtained by first choosing an arbitrary target value $q_\xi \trianglerighteq 0$ with $q_\xi \in s_i^\#(\xi)$ from each constraint $\xi \trianglerighteq 0$. Recursive target value decomposition of the intervals $s_{i+1}^\#(\xi_1)$ and $s_{i+1}^\#(\xi_2)$ from the expressions $\xi = \xi_1 \circ \xi_2$ are then decomposed into the corresponding sub-expressions until the sub-expressions are variables or constants. Existence of a target values decomposition is ensured by Lemma 13. Additional target values are propagated upwards from the target value regression task. The sketched procedure identifies up to two target values for each variable which ensure that all constraints for actions (or the goal) of the following layer are satisfied.

Target value regression task: Given an action a_i and up to two target values q_v^+ and q_v^- for each variable as well as the intervals $s_i^\#(v)$ that each variable evaluates to before the application of the action, we have to determine a target value regression for each target value. As actions have at most one effect on each variable, we perform a target value regression according to Definition 15 for all target values and the corresponding effects. If the target value is already contained in the interval of the preceding state $s_i^\#(v)$, we do not have to apply the action at all, the repetition count is $k_q = 0$ and we can propagate the target value regressively upwards to the preceding layer. As the target value determination task determines up to two target values for each variable, we have to perform up to two target value regression tasks for each variable, all of them include a repetition constant k_q.

Repetition determination task: The action a_i has to be repeated k times, where $k = \max(k_q)$ among all k_q identified in the target value regression task. □

Example 13 (Target Value Determination Task). Let $s(v_x) = [0, 1)$, $s(v_y) = [0, 1)$ and $s(v_z) = (1.7, 3]$ and let $a = \{(v_x + v_y) - v_z > 0, v_a \mathrel{+}= 1\}$ be an action with conditions on all 3 variables. The expression $\xi = (v_x + v_y) - v_z$ is decomposed into the sub-expression $\xi_{xy} = v_x + v_y \mapsto [0, 2)$ and $\xi = \xi_{xy} - v_z \mapsto [-3, .3)$. From $s(\xi) > [0, 0]$ we can choose an arbitrary target value in the interval $(0, .3)$ and choose an arbitrary $q = 0.1$ satisfying $q > 0$. Now, we have to recursively find appropriate target values q_x, q_y and q_z in the sub-expressions.

First, we perform a target value decomposition of $q = 0.1$ for intervals $a = s(\xi_{xy}) = [0, 2)$, $b = s(v_z) = (1.7, 3]$ with operator "$-$". From the set of candidates $a_c = [0.1, 0.1] + (1.7, 3] = (1.8, 3.1]$ we can chose an arbitrary $q_a \in (1.8, 3.1] \cap [0, 2) = (1.8, 2)$ e.g. $q_a = 1.9$ with partner $q_b = q_a - q = 1.9 - 0.1 = 1.8 \in (1.7, 3]$. As $b = s(v_z)$ this already yields the first target value

$q_z = q_b = 1.8$. We now continue by a target value decomposition of $q = q_a = 1.9$ for intervals $c = s(v_x) = [0, 1)$ and $d = s(v_y) = [0, 1)$ with operator "+". The set of candidates $c_c = [1.9, 1.9] - [0, 1) = (0.9, 1.9]$ intersected with $c = [0, 1)$ yields $(0.9, 1.9] \cap [0, 1) = (0.9, 1)$ and we chose $q_c = 0.95 \in (0.9, 1)$ arbitrarily. With the corresponding $q_d = q - q_c = 1.9 - 0.95 = 0.95 \in [0, 1)$ we have now successfully determined target values for all variables, namely $q_x = q_c = 0.95$ for v_x, $q_y = q_d = 0.95$ for v_y and $q_z = q_b = 1.8$ for v_z.

Example 14 (Target Value Regression Task). Consider again the scenario of Example 13. In this example we demonstrate the target value regression for the target value $q_x = 0.95$.

Assume that $q_x = 0.95$ has to be reached in $s_2(v_x) = [0, 1)$ as the result of applying a repetition relaxed action $a_1 = \langle \emptyset, \{v_x := \xi\} \rangle$ with $\xi = (v_a + 1)$ in state s_1 with $s_1(v_a) = [-1, 0)$, $s_1(v_x) = [0, 0]$ and therefore $s_1(\xi) = [0, 1)$.

A *target value regression* (cf. Definition 15) of intervals $x = s_1(v_x) = [0, 0]$, $y = s_1(\xi) = [0, 1)$ and target value $q = 0.95$ is $\langle 0, 0.95, 0.95, 1 \rangle$. Then, a target value decomposition of $q_e = 0.95$ for the expression $\xi = (v_a + 1)$ is $\langle -0.05, 1 \rangle$ yielding for $q_a = -0.05$ for v_a and $q_c = 1$ for the constant 1. The action a_0 has to be applied $k = 1$ times.

Let now $a_0 = \langle \emptyset \to \{v_a \div= 2\} \rangle$ be the action that manipulated v_a from $s_0(v_a) = [-1, -1]$ to $s_1(v_a) = [-1, 0)$ in the repetition relaxed plan. A target value regression of $x = s_0(v_a) = [-1, -1]$ and $y = s_0(2) = [2, 2]$ with $q = -0.05$ is $\langle -1, 2, 2, 5 \rangle$. While the target values -1 and 2 are obvious for point intervals, the number of repetitions k is determined by solving $-0.05 = -1 \div 2^k$ so $2^k = 20$ and $k = \log_2(20) \approx 4.3$. As such a_0 has to be applied 5 times.

Theorem 16 (Polynomiality of Plan Equivalence). *The repetition explication procedure that derives an interval relaxed plan π^+ from a repetition relaxed plan $\pi^\#$ described in the proof of Theorem 15 is in P.*

Proof. During each step in the plan there are at most 2 target values for each variable. The total number of target value regressions that has to be found in one step is therefore restricted to $2 \cdot |\mathcal{V}|$.

For each target value, there is at least one action enabling the respective target value. We have to recursively determine a target value decomposition for each precondition of the enabling action. The target value regression demands for two more expressions to reach a target value namely $q_e^\vdash$ and $q_e^\dashv$ for the expression ξ assigned to variable v. Recursively determining a *target value decomposition* (cf. Lemma 13) is linear in the size of the expression, unless it is a division by an interval containing zero, where all target values $q_y^\vdash$ and $q_y^\dashv$ are a target value decomposition for $q_y^\vdash < 0$ and $q_y^\dashv$. As we have access to the *initial values* of all variables, we can evaluate the expression generating y and replace either $q_y^\vdash$ or $q_y^\dashv$ by the initial value i_y. The only exception to this rule is if the expression ξ_y evaluates to the undefined value $\perp$ in the initial state. While we have to find a *decomposition* for both $q_y^\vdash$ and $q_y^\dashv$, this blow-up can not happen for all recursive expressions as divisions by intervals containing zero evaluate to $(-\infty, \infty)$ with a singularity at 0 which is not hit by recursive target values $q^\vdash < 0$ and $q^\dashv > 0$.

Summarizing, the number of target values that have to be determined is restricted to the size of all numeric expressions occurring in the preconditions

of actions that have to be applied plus twice the size of the expression in the effect. □

Theorem 17. *The problem to generate an interval relaxed numeric plan is in* P *for tasks with* acyclic dependencies *(cf. Definition 12).*

Proof. The fix-point algorithm for repetition relaxed planning tasks is sound (Theorem 8), complete (Theorem 11) and terminates in polynomial time (Theorem 10). Thus, generating a repetition relaxed plan $\pi^{\#}$ is in P. An interval relaxed plan π^{+} can be constructed from $\pi^{\#}$ (Theorem 16) in polynomial time. □

In order to asses the quality of a heuristic, Hoffmann [Hof03] define basic criteria that relaxations should meet.

Definition 16 (Adequate Relaxation). A relaxation is *adequate* if it is *admissible*, offers *basic informedness*, and if the plan existence problem for the relaxation is in P.

A relaxation is *admissible* if any plan π for the original task Π is also a relaxed plan for Π^{+}.

A heuristic offers *basic informedness* if the empty plan is a plan for Π iff it is a plan for Π^{+}.

Theorem 18. *The interval relaxation is* adequate *for acyclic dependency tasks.*

Proof. Admissibility: After each step of the original plan π, the value of numeric variables of the original task is contained in the interval of the respective variable. Admissibility follows from the semantics of comparison constraints that hold if they do for any pair of elements from the two intervals.

Basic informedness: No (goal or action) conditions are dropped from the task. Relaxed numeric variables are mapped to degenerate intervals that only contain one element. Therefore, conditions in the original task $\xi \trianglerighteq 0$ correspond to interval constraints $[s(\xi), s(\xi)] \trianglerighteq [0, 0]$, a constraint that is satisfied iff it is satisfied in the relaxed task.

Polynomiality: As a corollary to Theorem 17, we can also conclude that interval relaxed numeric plan existence is in P for tasks with *acyclic dependencies.* □

In this chapter we presented interval algebra as a means to carry the concept of a delete relaxation from classical to numeric planning. We introduced the interval relaxation which uses intervals to compactly over-approximate the set of reachable values. As interval based operations are non-idempotent operations in numeric planning, we introduced the repetition relaxation that handles arbitrary many repetitions of an action in a single step. We then showed that interval relaxed plans can be explicated from repetition relaxed plans in polynomial time. As such, we could prove that the original interval relaxation is adequate for acyclic dependency tasks, i.e. tasks where the expressions of numeric effects do not depend on the affected variable. The proposed relaxation advanced the state of the art even though adequacy of interval relaxation was only shown for the restricted set of acyclic dependency tasks. However, the requirement of acyclic dependency for numeric expressions is a proper generalization of expressions ξ being required to be constant, a requirement for other

state-of-the-art approaches, e.g. [Hof03], which is met in many practically relevant problems. The complexity of the approach for arbitrary interval relaxed planning problems remains an open research issue, though. In the upcoming section we will present heuristics based on the interval-based relaxations introduced in this chapter. For the heuristics, we also discuss practical solutions of cyclic dependency tasks, where the planning problem is relaxed even more if a cyclic dependency occurs.

Chapter 4

Numeric Relaxation Heuristics

Informed search algorithms such as hill-climbing or best-first search are an effective approach to solve planning problems. The performance of these informed search algorithm depends on the quality of the underlying heuristic. In order to use informed search algorithms for *numeric* planning problems, we require good heuristic estimators which are suitable for numeric planning. In this chapter, we discuss adaptations of the forward chaining *delete relaxation* heuristics $h_{\max}$, h_{add} and h_{FF} to numeric planning. The chapter is based upon work published at the 40th German Conference on Artificial Intelligence [AN17c] supplemented by an extended abstract presented at the 10th International Symposium on Combinatorial Search [AN17b]. Furthermore, several proofs were first presented in a technical report [AN17a].

In this chapter, we explore the design space of numeric relaxation heuristics with regard to two relaxation methods (*interval* or *repetition* relaxation) introduced in Chapter 3, two methods of aggregating heuristic costs (*maximum* 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. 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.

4.1 Interval Based Relaxations

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. Intervals offer a compact representation of an over-approximation the achievable values. However, using only an interval representation is not sufficient to derive tractable relaxation heuristics for numeric planning. In relaxed classical STRIPS planning, actions are idempotent and therefore, the monotonic growth is bounded by the number of actions as well as the number of facts in the planning task. If we allow for conditional effects in classical planning, actions become *pseudo-idempotent* (cf. Definition 2), but the growth is still

bounded by the number of actions and the number of the conditional effects, as each condition that is achieved remains achieved. By contrast, 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, we have to restrict the number of action applications within the heuristic. We consider two relaxation frameworks which approach this challenge differently: the *interval relaxation* that is described in more depth in Chapter 3.3 and the *repetition relaxation* which is described in Chapter 3.4.

The depth of a relaxed planning graph is restricted to the length of a shortest parallel relaxed plan (cf. Section 3.3.3), which allows us to compute an interval relaxed planning graph very much like in classical planning. However, desired heuristic properties (such as admissibility for $h_{\max}$ for non-unit cost tasks) can not be guaranteed in this framework. The repetition relaxation uses a semi-symbolic representation of intervals to simulate the behavior of arbitrarily many action repetitions at once. This relaxation makes relaxed actions pseudo-idempotent, i.e. actions can change the value of a variable only a restricted number of times and thus, the repetition relaxed plan existence problem can be decided in polynomial time.

4.2 Cyclic Dependencies

The interval which is reached by applying an effect $v \circ\!= \xi$ *depends* on the values of all variables in ξ (cf. Definition 12) in interval based relaxations. This dependency relation induces a *dependency graph*. If the dependency graph is acyclic, and if actions are *pseudo-idempotent* (cf. Definition 2), sequences of actions are pseudo-idempotent as well, as the values of the variables only depend on other variables which stabilize in topological order.

Cyclic dependencies can make sequences of actions non-idempotent. To the best of our knowledge, we are not aware 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 linear time by algorithms checking for strongly connected components (e.g., Tarjan's algorithm [Tar72]) in the dependency graph. If v is a variable that occurs in a cycle, we can break the cycle by introducing an auxiliary variable v' and replace v in the effect of the action that leads to the cycle by v'. Now in order to guarantee completeness of the reachability, we have to reinsert the value of v' to v if the values of these variables differ. However, we can do so in a controlled manner by including special cycle breaker actions. The implementation of these cycle breaker actions opens design space. Tractable heuristics have to bound the number of re-insertions.

The coarsest cycle breaker action sets changing variables to $v := (-\infty, \infty)$: an interval which can not be extended, thus ensuring idempotence of the cycle when $v' = (-\infty, \infty) = v$. A more accurate estimate is obtained by only setting the interval bound of v to positive or negative infinity if the interval of v' changed into the respective direction. The cost of traversing the cycle can be used as cost for the cycle breaker action. This relates very much to the *additive effects transformation* [SHTR16], which compiles assignments $x := \xi$ into increase

effects $x \mathrel{+}= \xi - x$. Both approaches relax cyclic effects even further by assuming that if a bound can be extended once, it can be extended arbitrarily often by the same margin at the same cost. 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 and thus avoids an unnecessary blow-up. Then again, in the presence of several cycles on the same variable, handling all cycles by the same action is a coarser simplification than the separate handling done by Scala, Haslum, Thiébaux, and Ramírez [SHTR16].

4.3 The Maximum and the Additive Heuristic

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 forward propagation heuristics h_{add} [BLG97; BG01] and its admissible counterpart $h_{\max}$ [BG99] estimate the cost $\kappa(p)$ to achieve the propositional fact p by a fix-point equation that can be computed with the recurrence relation

$$\kappa_{t+1}(p) := \min\left(\kappa_t(p), \min_{a \in \text{ach}(a)} (\kappa(a) + \kappa_t(\text{pre}(a)))\right) \tag{4.1}$$

where $\text{ach}(a)$ is the set of all actions $a \in \mathcal{A}$ that *achieve* p at time step t.

Propositions that hold in the state for which the heuristic is computed are initialized to cost $\kappa_0(p) = 0$, and to $\kappa_0(p) = \infty$, otherwise. Whereas the action cost $\kappa(a)$ is given by the planning task, the action precondition cost $\kappa(\text{pre}(a))$ 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

$$\kappa_t(\text{pre}(a)) := \max_{p \in \text{pre}(a)} \kappa_t(p) \tag{4.2}$$

is used, while h_{add} uses the *sum* of all preconditions

$$\kappa_t(\text{pre}(a)) := \sum_{p \in \text{pre}(a)} \kappa_t(p) \tag{4.3}$$

instead.

The recurrence relation defines a sequence. In classical planning, the fix-point of this sequence exists and is reached after a finite number of steps (cf. Section 3.1.1).

The h_{FF} heuristic [HN01] 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.

4.3.1 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 chapter: the *interval relaxation* (Section 3.3) and the *repetition relaxation* (Section 3.4).

Applying an analogous recurrence in numeric planning demands for numeric interpretations of *facts*, *achievers* and *preconditions* in Formula 4.1, Formula 4.2 and Formula 4.3. Propositional *facts* p or $\neg p$ can be seen as variable-value pairs $v_p \mapsto \mathit{true}$ or $v_p \mapsto \mathit{false}$ of a propositional variable v_p with domain $\{\mathit{true}, \mathit{false}\}$. The interpretation of *facts* as variable-value pairs generalizes naturally to multi-valued variables with a finite domain as well. Following this approach, a *numeric fact* is a variable-value pair $v \mapsto q$ where $v \in \mathcal{V}$ and $q \in \mathbb{Q}$. If we view numeric actions as grounded actions with conditional effects, we can define a *grounded interpretation* of *achievers* and *preconditions* in the following way. An *achiever* of such a numeric fact is a ground action that has an effect assigning q to v and the *precondition* of this action are variable-value pairs for all variables occurring in the action's precondition so that all numeric constraints are satisfied, as well as variable-value pairs for all variables occurring in the effect so that the desired numeric fact $v \mapsto q$ is achieved.

Example 15. Let $a = \langle \{v_1 - v_2 > 0\}, \{v_3 \mathrel{+}= v_4 \times v_1\} \rangle$ be an action and $v_3 \mapsto 5$ be a fact. One possible grounded interpretations of a that achieves $v_3 \mapsto 5$ is $\langle \{v_1 \mapsto 7, v_2 \mapsto 5, v_3 \mapsto 1, v_4 \mapsto 4, v_1 \mapsto 1\}, \{v_3 \mapsto 5\} \rangle$ where $v_1 \mapsto 7$ and $v_2 \mapsto 5$ satisfy the action precondition $7 - 5 > 0$, and $v_3 \mapsto 1, v_4 \mapsto 4$ and $v_1 \mapsto 1$ achieve the desired effect as the new value for v_3 is $1 + (4 \times 1) = 5$. Note that the grounded interpretation may include several different *precondition facts* on the same variable (in our case $v_1 \mapsto 7$ and $v_1 \mapsto 1$) which makes the action only applicable in the relaxation.

This grounded interpretation is not practical: numeric constraints $\xi \trianglerighteq 0$ in the action precondition can be satisfied by potentially infinitely many numeric facts. Similarly for the effect, infinitely many combinations of facts could achieve a certain fact of the altered variable. Explicitly listing all achievers and all corresponding preconditions would result in an infinite set of achieving actions.

As we are interested in tractable relaxation heuristics, we cannot pursue this approach, and, opposed to classical planning with a finite domain representation, it is not possible to compute a separate cost estimate for each fact. In interval based relaxations, we use a different interpretation of *facts*, *achievers* and *preconditions* that is computationally tractable. As we cannot consider all variable-value pairs, we have to aggregate several variable-value pairs to one fact. As such, a *relaxed numeric fact* is a variable-interval pair. Now, in order to define achievers of such facts in a tractable and yet meaningful way, we can investigate the structure that is produced by the recurrence relation (cf. Formula 4.1): cost values $\kappa_{t+1}(p)$ are assigned a value depending on the cost estimates $\kappa_t(p)$ *before*. For numeric relaxations we pursue a similar approach, and make this recursive structure explicit. A *relaxed state progression sequence* is a recursive graph structure consisting of *states*, *facts* and *action edges* that are constrained by the underlying relaxation. In this structure, states contain a fact for each variable of the underlying relaxed planning task and *action edges* $\langle s_i, a, s_j \rangle$ connect state s_i to state s_j with action a.

Definition 17 (Relaxed State Progression Sequence). *Relaxed state progression sequences* are recursively defined as follows: let $s_0 = \mathcal{I}$ be the initial state of the underlying relaxation. Then, $\langle s_0 \rangle$ is a *relaxed state progression sequence*.

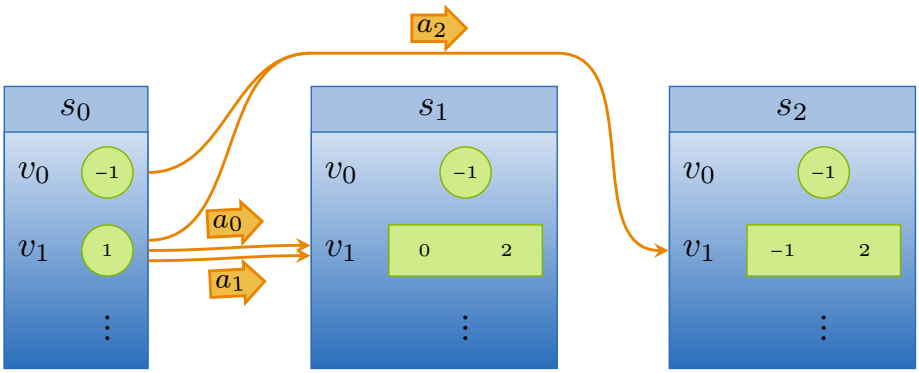

Figure 4.1: An exemplary relaxed state progression sequence.

Now let $\langle s_0, \ldots, s_{n-1} \rangle$ be a relaxed state progression sequence. Then, recursively, $\langle s_0, \ldots, s_{n-1}, \mathcal{A}_n, s_n \rangle$ where $\mathcal{A}_n$ is a set of action edges $\langle s_i, a, s_j \rangle$ is a *relaxed state progression sequence* iff

- $s_i \vDash \text{pre}(a)$ for all $\langle s_i, a, s_j \rangle \in \mathcal{A}_n$, i.e. the "starting" state of each action edge satisfies the action's precondition.
- $s_j = s_n$ for all $\langle s_i, a, s_j \rangle \in \mathcal{A}_n$, i.e. all action edges "end" in the new state s_n
- $s_n = s_{n-1} \sqcup \bigsqcup_{\langle s_i, a, s_j \rangle \in \mathcal{A}_n} \text{app}(a, s_i)$ where $\text{app}(a, s_i)$ is the state obtained by applying action a to state s_i according to the underlying relaxation, and the convex union of two states is the convex union of the intervals of corresponding variables. I.e. the intervals assigned to variables v in state s_n are the convex union of all effects on v from actions "ending" in s_n.

The initial state consists of a set of *relaxed numeric facts* where all intervals are degenerate point intervals. In order to make use of relaxed state progression sequences, we are particularly interested in sequences that reach a goal, i.e. sequences where the last state satisfies the goal condition of the underlying planning task.

For now, we assume that such a sequence is given and look at ways to generate such a sequence thereafter. Instead of computing cost estimates for *all* numeric facts, we are only interested in estimates for the facts that appear in the relaxed state progression sequence. Achievers and preconditions of such relaxed numeric facts are defined as follows. An action edge $\langle s_i, a, s_j \rangle$ *achieves* a fact $v \mapsto x$ in state s_j if a has an effect on v. By definition of the sequence, the effect of a is then included in the interval $s_j(v)$. *Preconditions* of an achiever $\langle s_i, a, s_j \rangle$ are all facts in s_i whose corresponding variable appear in the action's precondition or effect.

Example 16. Consider the state sequence depicted in Figure 4.1 where $s_0(v_1) = [1, 1]$ and let $a_0 = \langle \emptyset, v_1 \mathrel{-}= 1 \rangle$ and $a_1 = \langle \emptyset, v_1 \mathrel{+}= 1 \rangle$. The effects $v_1' \mapsto [0, 1]$ from a_0 and $v_1' \mapsto [1, 2]$ from a_1 are aggregated to the fact $v_1 \mapsto [0, 2]$ in s_1. Both action edges $\langle s_0, a_0, s_1 \rangle$ and $\langle s_0, a_1, s_1 \rangle$ *achieve* $v_1 \mapsto [0, 2]$ in s_1 in the given relaxed state progression sequence.

As relaxed numeric facts aggregate many values, there are *implicit preconditions* on the variables occurring in the achieving action and the values that lead to the desired effect are not explicitly present as in the grounded interpretation. The facts that are used to compute the precondition of an achiever $\kappa_t(\text{pre}(\langle s_t, a, s_j\rangle))$ are then all facts $v \mapsto s_t(v)$ for variables v that appear in the effect of a.

Example 17. Consider Example 16 from Figure 4.1 again. The precondition that is considered by Formula 4.2 or Formula 4.3 is $v_1 \mapsto [1,1]$ in s_0 for a_0 as well as for a_1. Moreover, let $a_2 = \langle\{v_2 \geq 0\}, \{v_1 := v_0\}\rangle$, and $s_0(v_2) = 5$. The precondition are then the facts $v_0 \mapsto [-1,-1], v_1 \mapsto [1,1]$ and $v_2 \mapsto [5,5]$ in s_0.

With this definition of *facts*, *achievers* and *preconditions*, we can apply Formula 4.1, Formula 4.2 and Formula 4.3 to the facts in the relaxed state progression sequence. Note that facts appear in the context of a state. In Formula 4.1, the cost $\kappa_t(p)$ of a fact p with the form $v \mapsto x$ is the cost of p in s_t. Usually, the fact cost that is estimated for $\kappa_{t+1}(p')$ is the cost of a different fact $p' \neq p$, namely p' of the form $v \mapsto x'$. As such, $\kappa_t(p) = \infty$, unless $x = x'$ and therefore $p = p'$.

Example 18. Consider Figure 4.1 again. The cost $\kappa_0(v_1 \mapsto [0,2]) = \infty$ and therefore, $\kappa_1(v_1 \mapsto [0,2])$ is achieved by action a_0 and a_1 but not by the "idle arc". On the other hand, $\kappa_1(v_0 \mapsto [-1,-1]) = 0$ as the same fact $v_0 \mapsto [-1,-1]$ exists in s_0 as well.

We discuss a *planning graph* based and a *priority queue* based approach to generate such relaxed state progression sequences, 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. In Formula 4.1, this corresponds to generating new states from the recurrence by evaluating all new costs in parallel. As such, $j = i + 1$ for all action edges $\langle s_i, a, s_j\rangle$. The "state layers" of the planning graph are the relaxed states in the state progression sequence. Facts aggregate the intervals of all actions that have an effect on the corresponding variable.

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. In Formula 4.1, this corresponds to generating a new state for the fact with the least cost. This way, the states in the relaxed state progression sequence aggregate facts with equal cost.

Both approaches, the *planning graph* and the *priority queue* based approach restrict the number of considered facts. Note that other approaches are conceivable, e.g. Scala, Haslum, and Thiébaux [SHT16] view propositional facts as conditions and generalize the formulas to a recurrence based on numeric constraints. We will now discuss the planning graph and the priority queue based approach to generate a relaxed state progression sequences in more detail and analyze combinations of both fact aggregation approaches with interval or repetition relaxation which guarantee polynomial time bounds on the heuristic computation.

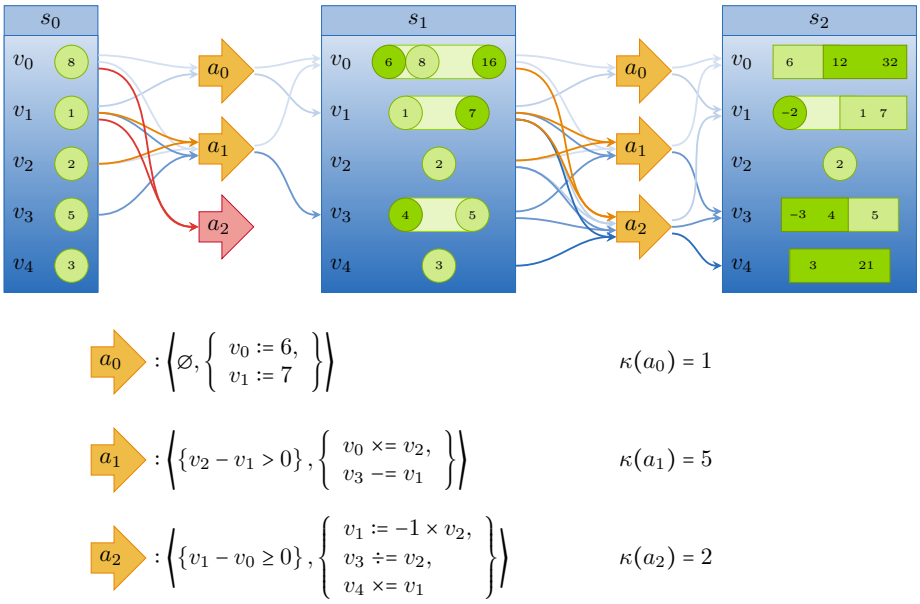

Figure 4.2: A relaxed numeric planning graph with five variables v_0 to v_4 and the progression of applying the three actions a_0 to a_2 in parallel.

4.3.2 Heuristics Based on Planning Graphs

The *relaxed parallel planning graph* [HN01] is a graph representation of the *parallel fix-point algorithm* from Definition 3. It consists of alternating state layers of reachable propositions and action layers of actions which are applicable in that propositional state.

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 cost minimal regarding κ) plan. Variations of this approach are often used for numeric planning [Ede04; CCFL13; Hof03; SHTR16].

Just as in classical planning, a planning graph consists of alternating *fact* and *action layers*. A *fact layer* is a relaxed state that consists of a variable-interval pair for each variable $v \in \mathcal{V}$ of the planning task. Starting from $s_0 = \mathcal{I}$ we iteratively apply all applicable actions in parallel until we reach a relaxed state that satisfies the goal condition. We will now illustrate the process using the following example.

Example 19. Figure 4.2 depicts a planning graph with blue relaxed states s_0, s_1 and s_2. We depict the values of all variables occurring in the planning task in green and use circles for degenerate point intervals. We use a dark green to indicate the sub-intervals that are achieved by applying the action of an action edge in the corresponding starting state. The interval that was reachable in the previous state is indicated in medium green. The values that are not directly reached but that are enclosed by the convex union are depicted in light green. In the *action layer*, we depict applicable actions in orange and inapplicable

actions in red. Arrows drawn from *relaxed states* to actions indicate the implicit preconditions of the action. Arrows from variables occurring in satisfiable action preconditions are drawn in orange and the unsatisfiable precondition of a_2 in the first action layer is drawn in red. Additionally, blue arrows indicate the implicit preconditions for each of the actions' effects. The blue gradient in the background of the state boxes indicates the color of the effect: implicit preconditions for the effects on variable v_0 are light blue, gradually getting darker and effects on v_4 are indicated in dark blue. This scheme is also used for the colors of arrows from the actions to the altered variables in the subsequent relaxed states.

Starting with the fact layer s_0, we apply all actions in parallel. The preconditions of a_0 and a_1 are satisfied in $\mathcal{I} = s_0$: $s_0 \vDash \emptyset$ and $s_0 \vDash v_2 - v_1 > 0$ as $s_0(v_2 - v_1) = s_0(v_2) - s_0(v_1) = 2 - 1 = 1 > 0$, whereas the precondition of a_2, $v_1 - v_0 \geq 0$, is not satisfied in s_0 as $s_0(v_1 - v_0) = [1,1] - [8,8] = [-7,-7] \not\geq 0$. Action a_0 has two effects: $v_0 := 6$ and $v_1 := 7$, which are applied in the subsequent fact layer s_1. For v_1, the newly achieved value 7 results in the fact $v_1 \mapsto [1,7]$ because of the convex union with the old value $s_0(v_1) = [1,1]$. For v_0, the fact considered in s_1 is an aggregate of $[8,8]$, the old value in s_0, the effect interval $[6,6]$ of a_0 as well as the effect $v_0 \times\!= v_2$ from action a_1 which sets v_0 to $s_0(v_0) \times s_0(v_2) = [8,8] \times [2,2] = [16,16]$. The convex union of all three is $[8,8] \sqcup [6,6] \sqcup [16,16] = [6,16]$, and therefore $v_0 \mapsto [6,16]$ in s_1. Finally, a_1 also has an effect on v_3 where $s_0(v_3) - s_0(v_1) = [5,5] - [1,1] = [4,4]$ yields the interval $[4,5]$. In the following action layer, action edges $\langle s_1, a, s_2 \rangle$ are considered. Action a_0, while applicable, does not achieve any value beyond s_1. Applying the effects of action a_1 yields $s_1(v_0) \times s_1(v_2) = [6,16] \times [2,2] = [12,32]$ which results in the fact $v_0 \mapsto [6,32]$ for v_0 and applying the effect $s_1(v_3) - s_1(v_1) = [4,5] - [1,7] = [-3,4]$ for v_3 resulting in the fact $v_3 \mapsto [-3,5]$ in s_2. Action a_2 becomes applicable in s_1 as $s_1(v_1 - v_0) = [1,7] - [6,16] = [-15,1] \geq [0,0]$. The first effect of a_2 sets v_1 to $s_1(-1 \times v_2) = [-1,-1] \times [2,2] = [-2,-2]$ and again, values between -2 and 1 are considered reachable for fact $v_1 \mapsto [-2,7]$ because of the convex union. The second effect which would achieve $[4,5] \div [2,2] = [2,2.5]$ for v_3 is subsumed by the effect of a_1. Finally, $s_2(v_4) = s_1(v_4) \sqcup (s_1(v_4) \times s_1(v_1)) = [3,3] \sqcup ([3,3] \times [1,7]) = [3,21]$.

Given a planning graph based relaxed state progression sequence, we can determine cost estimates. For planning with action costs, one source of heuristic imprecision comes from the aggregation of different achieved values to one fact.

Example 20. Consider the fact $v_0 \mapsto [6,16]$ in s_1 in Figure 4.2. Actions a_0 and a_1 both achieve new values for v_0. However, they differ in the action cost: $\kappa(a_0) = 1$ whereas $\kappa(a_1) = 5$ resulting in a fact cost $\kappa_1(v_0 \mapsto [6,16]) = \min(\kappa(a_0) + \kappa_0(\mathrm{pre}(a_0)), \kappa(a_1) + (\kappa_0(\mathrm{pre}(a_1)))) = \min(1+0, 5+0) = 1$. As such, achieving values in the sub-interval $(8,16]$ is assumed to cost 1 in the heuristic whereas it would cost 5 in an actual task.

In spite of this underestimation of the true cost, one weakness of the *planning graph* approach is that $h_{\max}$ does not compute admissible estimates in tasks with action costs when paired with an *interval relaxation*. The reason is that a fact can be achieved at a better cost in a deeper layer in the planning graph. Likewise, the cost estimates for h_{add} can be higher than the estimates that would be computed by the formulas for classical planning. Opposed to classical

planning, the facts change from one relaxed state to the next, and therefore, we cannot update the cost estimate any more.

Example 21. Consider the planning task from Figure 4.2 and let $\mathcal{G} : 3 - v_3 \geq 0$ be the goal. It could be achieved by the action sequence $\langle a_0, a_2 \rangle$ with cost $\kappa(a_0) + \kappa(a_2) = 1 + 2 = 3$. However, a_2 has implicit preconditions on v_0 and v_1 to satisfy the numeric constraint $v_1 - v_0 \geq 0$ and on v_3 and v_2 for the relevant numeric effect. The cost estimates in s_1 are $\kappa_1(v_0 \mapsto [6, 16]) = 1)$, $\kappa_1(v_1 \mapsto [1, 7]) = 1)$, $\kappa_1(v_2 \mapsto [2, 2]) = 0)$ and $\kappa_1(v_3 \mapsto [4, 5]) = 5$. This yields $\max(1, 1, 0, 5) = 5$ for the precondition and with $\kappa(a_2) = 2$ we get $\kappa_2(v_3 \mapsto [-3, 5]) = 7$.

The absence of an a priori bound for the depth in which a fact could be achievable with better cost is a problem for numeric planning which also appears in the context of termination. 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, unboundedly many new *interval relaxed* facts will have to be considered. To avoid an unbounded blow-up, we terminate graph generation as soon as the goal formula is satisfied for the first time, even though this strategy impairs *admissibilty* of the heuristic, i.e. it is possible that the heuristic overestimates the actual cost and thus, algorithms like $A^\star$ [Nil80] are not guaranteed to find optimal solutions. 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 counteracts the purpose of using action costs in the first place. Scala, Haslum, and Thiébaux [SHT16] run into a similar problem using "asynchronous subgoaling" and have to set the cost of hard conditions to 0 to ensure admissibility of $h_{\max}$.

A combination of the *repetition relaxation* with the *planning graph* based approach is not as promising as other combinations. The repetition relaxation is coarser than the interval relaxation as it aggregates arbitrarily many repetitions. The planning graph approach is less accurate than the priority queue based approach that we will introduce in the upcoming section. 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. Splitting up the result into sub-intervals with different cost estimates makes the computation of heuristics intractable and goes into a similar direction of a relaxation of numeric planning with an *accumulation semantics* (cf. Section 3.2.1). As such, a *planning graph based* approach with the *repetition relaxation* combines the downsides of the components without really making up for that.

4.3.3 Heuristics Based on Priority Queues

In classical planning, the cost estimates Formula 4.1 and Formula 4.2 or Formula 4.3 are usually computed with a generalized Dijkstra algorithm [Dij59]. Estimating the fact costs this way is more efficient than building a planning graph. This method translates into an approach that builds a different relaxed state progression sequence, namely one where the states are ordered by increasing cost, where the cost of a state is the cost of the most expensive fact within the state. Facts are processed according to a *priority queue* storing the cost to achieve them. Whenever a variable attains a new value, all actions that have an implicit precondition on this variable are triggered. All variables that appear

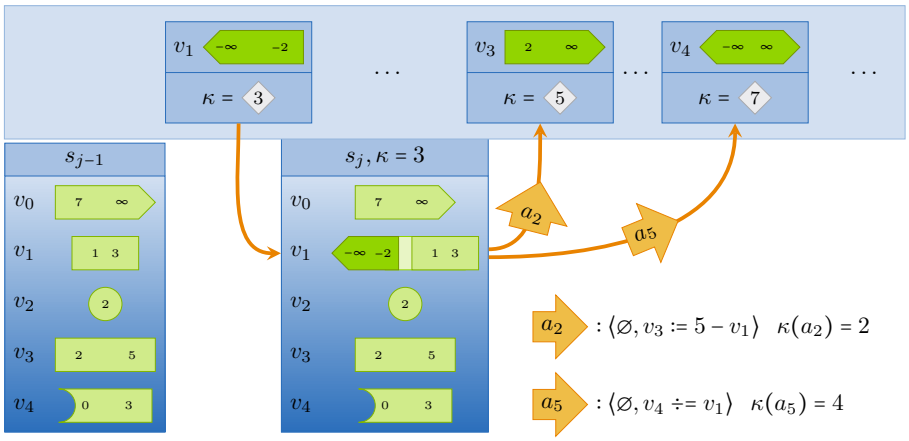

Figure 4.3: *Priority queue based* state sequence. The fact $v_1 \mapsto (-\infty, -2]$ is dequeued from the priority queue and triggers actions a_2 and a_5.

in the triggered actions effect can then enqueue new variable-interval pairs into the priority queue if an interval extends a bound. The priority of these new facts is the current priority increased by the cost $\kappa(a)$ of the action a.

In order to generate a *relaxed state progression sequence*, action edges start in the state in which the action becomes applicable and is enqueued. The enqueued fact is the interval $\text{app}(s_i, v)$, i.e. the interval obtained by applying the effect to the current state s_i. The index j of the ending state s_j depends on the number of states generated with a priority between i and j. In general and opposed to relaxed state progression sequences generated by a planning graph based approach, action edges span several nodes. Again, several intervals are aggregated for a relaxed state in order to ensure monotonicity. For the *priority queue based* approach, the facts in a relaxed state aggregate the newly reached interval form the priority queue with the interval of that variable in the previous state s_{j-1}. This is necessary as actions can achieve values for a variable which has unprocessed values achieved by other actions on the priority queue. Again, the process is best illustrated on the basis of an example, and we explain the priority queue approach in the following example.

Example 22. Figure 4.3 depicts a *relaxed state progression sequence* generated by a priority queue based approach. The priority queue is depicted in the top and contains three facts: $v_1 \mapsto (-\infty, -2]$ reachable at cost $\kappa = 3$, $v_3 \mapsto [2, \infty)$ at cost $\kappa = 5$ and $v_4 \mapsto (-\infty, \infty)$ at cost $\kappa = 7$. The state s_j reachable with priority $\kappa = 3$ is obtained by dequeueing $v_1 \mapsto (-\infty, -2]$. It extends the bound of v_1 from $s_{j-1}(v_1) = [1, 3]$ to $s_j(v_1) = (-\infty, 3]$. The intervals reachable in the previous state are marked in light green, and the new value for v_1 in dark green. Then, actions a_2 and a_5 which have implicit preconditions on v_1 are triggered. Action a_2 has an effect on v_3 an can reach $s_j(v_3) \sqcup (s_j(5) - s_j(v_1)) = [2, 5] \sqcup ([5, 5] - (-\infty, 3]) = [2, 5] \sqcup [2, \infty) = [2, \infty)$, and enqueues $v_3 \mapsto [2, \infty)$ with a cost of $\kappa = 3 + \kappa(a_2) = 5$ as v_3 reaches a new value. Similarly, a_5 is triggered an enqueues $s_j(v_4) \sqcup (s_j(v_4) \div s_j(v_1)) = (0, 3] \sqcup ((0, 3] \div (-\infty, 2]) = (0, 3] \sqcup (-\infty, \infty) = (-\infty, \infty)$ at cost $\kappa = 3 + \kappa(v_4) = 7$.

The number of facts which are inserted into the priority queue has to be

bounded in order to derive tractable heuristics. We will show that combining a priority queue based approach with an interval relaxation framework is infeasible as unboundedly many cheap actions can be processed before relevant ones in the following example.

Example 23. Let Π be a planning task where $\mathcal{I} = \{v_1 \mapsto [0,0], v_2 \mapsto [0,0]\}$ with two actions $a_1 = \langle \emptyset, v_1 \mathrel{+}= 1 \rangle$ and $a_2 = \langle \emptyset, v_2 \mathrel{+}= 5 \rangle$ with $\mathcal{G} = \{v_2 > 0\}$ and cost $\kappa(a_1) = 1$ and $\kappa(a_2) = 10\,000$. As a_1 is orders of magnitude cheaper, a_1 will enqueue $10\,000$ facts for v_1 before considering a_2 for the first time. It is also easy to extend the example by several variables and actions so that a combinatorial explosion worsens the already prohibitive result.

Therefore, we concentrate on pairing the *priority queue based* approach to generate a relaxed state progression sequence with the *repetition relaxation* whose effects are *2-pseudo-idempotent* (cf. Lemma 6) operations.

Unfortunately, sequences of repetition relaxed actions can be non-idempotent even though actions are 2-pseudo-idempotent. Effects in the repetition relaxation are applied on the intervals fixed to the preceding state and applying these numeric effects is 2-pseudo-idempotent. However, interactions of effects on variables that appear in the assigned numeric expression are not considered when computing the result. Similarly, a sequence of actions can be non-idempotent, even if every single action is idempotent. For the computational complexity it does not matter so much whether such interactions come from several effects of the same action or from interacting effects of several actions. More important is whether the planning task is acyclic or not (cf. Definition 12).

While the plan existence problem in the repetition relaxation is polynomial for acyclic tasks (cf. Theorem 16) this is not sufficient to 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. A change in the value of a variable v triggers all actions that have an effect *depending* on v, that is v appears in the assigned expression of one of the actions effects. If the dependency graph is acyclic, the repetition relaxation can be computed in polynomial time by evaluating variables according to the topology of that graph. This way, the intervals of topologically higher variables have already converged and changes in topologically lower layers can not influence the values of variables in topologically higher layers. Unfortunately, for planning tasks with action costs, priority queue based approaches alter variables in an order of the cost to achieve new values, and this order does not necessarily respect the topology. As such, the number of required enqueue operations can become exponential in the number of variables.

Theorem 19. *The number of insertions of numeric facts into the priority queue can become super-polynomial even if the dependency graph is acyclic.*

Proof. Let $\mathcal{V}_N = \{v_0, \ldots, v_n\}$ be a set of variables and $\mathcal{A} = \{a_n\} \cup \{a_{ij} | 0 \leq i < j \leq n\}$ be a set of $\frac{n^2+n}{2} + 1$ actions with costs $\kappa(a_{ij}) = 2^i$, and $\kappa(a_n) = 2^n$. So there are n actions with first index $i = 0$ all costing $\kappa = 2^0 = 1$, $n-1$ actions with first index $i = 1$ and cost $\kappa = 2$ and so forth.

Let each action $a_{ij} = \langle \mathrm{pre}_{ij}, \mathrm{eff}_{ij} \rangle$ have an empty precondition $\mathrm{pre}_{ij} = \emptyset$ and a single assign effect $\mathrm{eff}_{ij} = (v_i := v_j + 2^i)$ on variable v_i. The additional

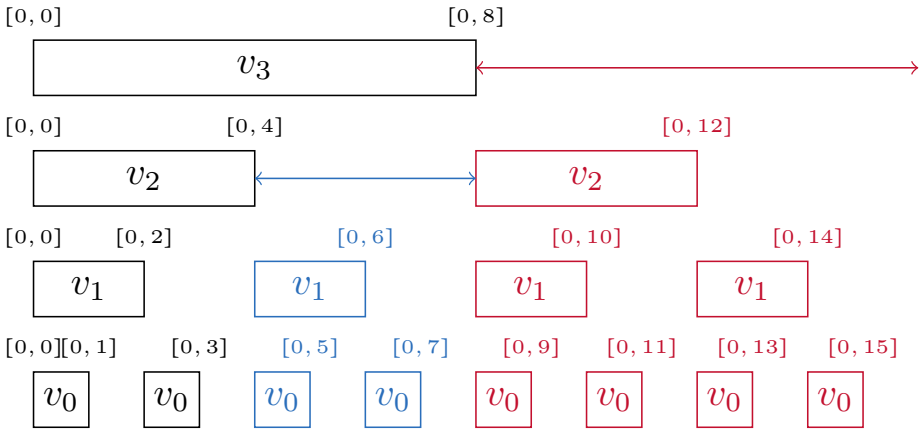

Figure 4.4: Enqueue and dequeue times of variables for $n = 3$.

action a_n has an effect $\text{eff}_n = (v_n := 2^n)$. This way, each variable v_i has a *dependency* on all variables v_j with higher index: $v_i \prec v_j$ iff $i < j$. All variables are initialized to $v_i \mapsto [0, 0]$. The idea of this construction is to have exactly one fact be dequeued at each time step (cost) from 1 to $2^{n+1} - 1$. The lower bound of all intervals remains at 0 for all variables, and the upper bound of the altered variable is extended to a value coinciding to the current cost. Initially, all actions are applicable, and as such all of them enqueue their effect facts with their respective costs. Note that all actions a_{ij} with the same first index i have the same effect $v_i \mapsto [0, 2^i]$ as $v_j \mapsto [0, 0]$ for all j. Therefore, among the initial enqueues, only the first fact to be dequeued for each variable will extend the respective interval.

We determine the number of required enqueue operations enq inductively: action a_n has no dependencies and its effect $(v_n := 2^n)$ enqueues the fact $v_n \mapsto [0, 2^n]$ with cost $\kappa = 2^n$ once: $\text{enq}(v_n) = 1$. We will now show inductively that $\text{enq}(v_{i-1}) = 2 \cdot \text{enq}(v_i)$. Whenever a fact referring to a variable v_i is enqueued, so is a fact referring to v_{i-1} by construction. Whenever an action achieves a new value for higher-layer variable k with $i < k$, it will trigger action a_{ik} for v_i and $a_{(i-1)k}$ for v_{i-1}. E.g. consider the example depicted in Figure 4.4 for $n = 3$. Action a_3 sets v_3 to $[0, 8]$ and triggers a_{23} on v_2, a_{13} on v_1 and a_{03} on v_0 enabling the respective intervals to reach 12, 10 or 9 respectively. The second enqueue operation on v_{i-1} is triggered by v_i changing itself. Because of action $a_{(i-1)i}$, v_{i-1} is not only enqueued at the same time as v_i, but it is enqueued again whenever v_i is dequeued. E.g. consider the triggers of a_{23} and a_{13} by setting v_3 to $[0, 8]$ from Figure 4.4 again: v_1 is not only triggered at the same time as v_2, but again, when v_2 reaches its new value $[0, 12]$.

The initial situation is slightly different in that there are n actions enqueued for each variable n instead of one. Save for the overhead of $\frac{n^2-n}{2}$ enqueue operations in the initially state, this already sets an upper bound of $\text{enq}(v_{i-1}) = 2 \cdot \text{enq}(v_i)$ enqueue operations for each variable. It remains to be shown that this upper bound is actually reached. Independent of k, a_{ik} costs twice as much as $a_{(i-1)k}$, and therefore, the effect triggered by v_k on v_{i-1} will be dequeued before the effect triggered on v_i. Therefore, all changes triggered at *enqueue*

time are processed before v_i is dequeued. We still have to ensure that the facts triggered from v_i at *dequeue* time are processed before v_i is enqueued again. E.g. in Figure 4.4, for v_2 we have to ensure that the changes marked in blue fit into the space marked by the blue double-arrow or that the actions triggered by v_3 marked in red fit into the region marked by the red double-arrow. Changes in v_{i-1} trigger changes in v_{i-2} and so on. The execution of the sequence of actions $a_{(i-1)(i)}, \ldots, a_{01}$ costs $\sum_{k=0}^{i-1} 2^k = 2^i - 1$ which is cheaper than 2^i. Therefore, also the changes triggered at dequeue time are processed before the variable is re-enqueued, resulting in twice as many enqueue operations for v_{i-1} as compared to v_i.

The total number of enqueue operations is then $\sum_{i=0}^{n} \mathrm{enq}(v_i) = \sum_{i=0}^{n} 2^i = 2^{n+1} - 1$ plus the overhead of $\frac{n^2-n}{2}$ enqueue operations in the initial state which is exponential in the input size. □

The heuristic becomes tractable if actions which have an *implicit precondition* on a variable v 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. 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.

If the *dependency graph* contains cycles, we can introducing auxiliary variables as presented in Section 4.2.

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

4.4 FF Plan Extraction 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.

Independently of the approach to generate it, a *relaxed state progression sequence* (cf. Definition 17) is 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. We will now investigate a generalization of a *marking* method for numeric planning.

4.4.1 Generalized Marking Procedure

In relaxed numeric planning, the progression step of the heuristic creates a relaxed state progression sequence. Given such a sequence, we can derive a sequential relaxed plan with the help of a *bucket plan* as follows.

Definition 18 (Bucket Plan). A *bucket plan* is a possibly empty sequence $\langle \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$ where *action buckets* $\mathcal{B}_i$ are sets of actions a for $1 \leq i \leq n$.

Bucket plans are useful if we have precedence constraints on some actions, but other actions can be ordered arbitrarily. E.g., in a relaxed planning graph, the buckets $\mathcal{B}_i$ correspond to the action layers of the relaxed planning graph.

There are two natural ways to derive a bucket plan from a given relaxed state progression sequence: action edges $\langle s_i, a, s_j \rangle$ are already sorted by ending state s_j in the action edge sets $\mathcal{A}_j$. Therefore, $\langle \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$ for buckets $\mathcal{B}_j = \{a | \langle s_i, a, s_j \rangle \in \mathcal{A}_j\}$ is a corresponding bucket plan. Alternatively, action edges can be sorted by starting state, and $a \in \mathcal{B}_{i+1}$ if there exist an action edge $\langle s_i, a, s_j \rangle$ starting in s_i in the relaxed state sequence.

A *sequential relaxed plan* can be derived from a *bucket plan* by ordering actions from each bucket arbitrarily.

However, the bucket plan or the corresponding sequential plans contain many actions that are not *relevant* for achieving the goal. While the cost of a relaxed plan can be used as a heuristic estimate, the cost of all actions in a relaxed state progression sequence is not very informative. In classical planning, a *marking* of actions in relaxed state progression is a function that assigns a label *marked* or *not marked* to each action. A labeling procedure can then determine a subset of relevant actions and only mark those. The sum of the costs of marked actions is then a much better heuristic estimate and the h_{FF} heuristic is based on such a marking approach.

For numeric planning, we want to restrict the actions in the plan to relevant actions as well. Whereas marking action edges like in classical planning is sufficient for the *interval relaxation* (cf. Section 3.3), actions can not be applied arbitrarily often in an actual plan and we are also interested in interval relaxed plans for the *repetition relaxation* (cf. Section 3.4). Relevance of a repetition relaxed action therefore also includes the number of repetitions for which an action has to be executed in order to achieve a desired effect. As such, we generalize the concept of a *marking* from a function that assigns *marked* or *not marked* to each actions to a function $\mu : \bigcup_{j=1}^{n} \mathcal{A}_j \to \mathbb{N}^{\geq 0}$ assigning a repetition counter to each action edge. Here, $\mu = 0$ corresponds to *not marked* and numbers greater or equal than one $\mu \geq 1$ correspond to the label *marked*.

A *marking pursuing state sequence* is a structure that allows us to verify whether a restriction to marked action edges is sufficient to satisfy the goal condition as well: we compute a new state sequence with the same structure but different values of the variables, where we only apply marked actions and verify that the last state of this sequence satisfies the goal condition. The intervals in the marking pursuing state sequence are more restrained compared to the original state sequence.

Definition 19 (Marking Pursuing State Sequence). Let $\langle s_0, \ldots, \mathcal{A}_n, s_n \rangle$ be a *relaxed state progression sequence* and $\mu : \bigcup_{j=1}^{n} \mathcal{A}_j \to \mathbb{N}^{\geq 0}$ be a marking.

The *marking pursuing state sequence* $\langle s'_0, \ldots, \mathcal{A}'_n, s'_n \rangle$ is a new state sequence where $s'_0 = s_0$ as in the *original state sequence*.

The set of action edges $\mathcal{A}'_j$ contains an edge $\langle s'_i, a, s'_j \rangle$ iff

- there exists an edge with corresponding state indices $\langle s_i, a, s_j \rangle \in \mathcal{A}_j$,
- $\mu(\langle s_i, a, s_j \rangle) \geq 1$, i.e. the corresponding edge in $\mathcal{A}_j$ is *marked*,

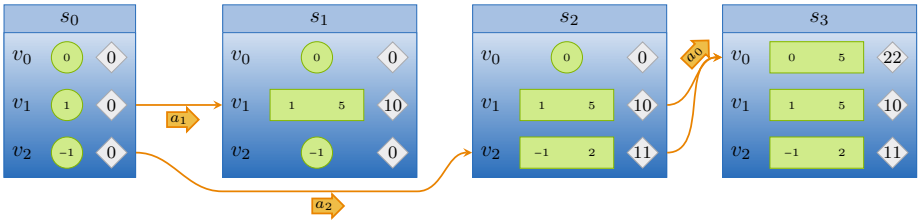

Figure 4.5: State sequence where a generalized marking is beneficial.

- $s'_i \vDash pre(a)$, i.e. the action is applicable in its starting state

.

The state s'_j is inductively given by the convex union

$$s'_j = s'_{j-1} \sqcup \bigsqcup_{\langle s'_i, a, s'_j \rangle \in \mathcal{A}'_j} \text{app}^{\mu(\langle s_i, a, s_j \rangle)}(a, s'_i)$$

where $\text{app}^{\mu(\langle s_i, a, s_j \rangle)}$ is the repeated application of the effect in the *interval relaxation semantics* according to the repetition counter of the marking.

The marking is *valid* if the last state of the marking pursuing state sequence satisfies the goal condition. The *cost* of a marking is the sum of the costs of all marked actions multiplied with their respective repetition count. A marking is *optimal* if it is *valid* and the cost of the marking is minimal among all valid markings.

Our marking procedure generalizes the marking method from classical planning in several ways. In classical planning actions are idempotent and therefore, it is sufficient to mark actions only once. This makes the exact time point at which an action is marked insubstantial. In contrast, for numeric planning it might be necessary to execute actions at different time steps and thus, the labeling procedure has to mark action edges that contain a starting state and an ending state in addition to the action itself. As the number of facts in classical planning is bounded, the heuristics do not have to generate a relaxed state progression sequence, and an optimal marking is one that marks the actions of an *optimal relaxed plan*. In numeric planning, the actions of an *optimal* relaxed plan might not be present in the state progression sequence, and a *marking* is *optimal* only regarding this structure.

Like in classical planning, 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 facts in a given state.

Example 24. Let $\mathcal{I} = \{v_0 \mapsto [0,0], v_1 \mapsto [1,1], v_2 \mapsto [-1,-1]\}$ as depicted in Figure 4.5 and actions $a_0 = \langle v_2 > 0, v_0 \mathrel{+}= v_1 \rangle$, $a_1 = \langle \emptyset, v_1 := 5 \rangle$ and $a_2 = \langle \emptyset, v_2 \mathrel{+}= 3 \rangle$ with costs $\kappa(a_0) = 1$, $\kappa(a_1) = 10$ and $\kappa(a_2) = 11$. Let furthermore $\mathcal{G} = \{v_0 > 0\}$.

Whereas marking actions a_2 and a_0 yields a valid plan, $\langle a_2, a_0 \rangle$ with cost 12, a_1 is applicable before a_0, the fact $v_2 \mapsto [1,5]$ in state s_2 has a cost $\kappa_2 = 10$ in the given relaxed state progression sequence, which leads to an estimated cost

of $\kappa_3(v_0 \mapsto [0,5]) = 22$. Yet, a *marking* procedure that ignores the irrelevant action a_1 allows us to find the better plan.

In order to determine a marking of a relaxed state progression sequence, we will illustrate a marking procedure based on the regressive explication of target values using *target value decomposition* (cf. Definition 14) and *regression* (cf. Definition 15) from Theorem 15. But first, we will show that an *optimal marking* can not be computed efficiently. We will show that the cost bounded marking problem is NP-complete as in classical planning. Analogously to classical planning, the costs annotated to each fact by the progression search only matter to approximate a "good" valid marking, but they are insubstantial for the complexity proof where a non-deterministic oracle *guesses* a marking which is then verified by a polynomial checking procedure.

We show NP-completeness of an optimal marking by considering the corresponding decision problem where the cost of the marking is bounded β.

Theorem 20. *Marking actions in an relaxed state progression sequence so that the cost of the corresponding interval relaxed plan is bounded by β is weakly NP-complete.*

Proof. NP-hardness of the marking problem for numeric planning is evident as it is a generalization of the h_{FF} marking problem for classical planning. Reducing minimum set cover to h^+: the cost of an optimal relaxed plan, can therefore also be used to show NP-hardness of the generalized numeric marking problem.

Membership in NP is shown by guessing a marking of action edges. For the interval relaxation, actions are marked with $\mu = 1$, however, for the repetition relaxation we also have to guess a number of repetitions for each action. The plan cost bound β bounds the number of repetitions that can be chosen for each action a, namely $\lfloor \frac{\beta}{\kappa(a)} \rfloor$. For very small action costs $\kappa(a)$ this number can become large which is the reason for pseudo-polynomial membership and *weak* NP-completeness.

The maximal number of repetitions $\lfloor \frac{\beta}{\min_{a \in \mathcal{A}} \kappa(a)} \rfloor$ also ensures that the rational numbers on the interval bounds can be represented in polynomial space.

The checking procedure that verifies that a given marking is indeed a relaxed plan, the *marking pursuing state sequence* (cf. Definition 19), is a simple progression scan that is restricted to the marked actions. In each relaxed state of this sequence, all marked actions are executed, unless they are not applicable in which case the marking is either suboptimal or not a valid relaxed plan. If the last state of the verification procedure satisfies the goal condition, the marking is valid. The procedure runs in time polynomial in $\lfloor \frac{\beta}{\min_{a \in \mathcal{A}} \kappa(a)} \rfloor \times l$: the number of marked actions and the length of the sequence l, thus demonstrating membership in NP. □

4.4.2 Target Value Explication

As we have shown in the previous section, an *optimal marking* can not be found in polynomial time unless P = NP. We propose a marking method for numeric planning that approximates an optimal marking based on the h_{add} estimates as it is done in the h_{FF} heuristic from classical planning.

Given a relaxed state progression sequence, we have to mark action edges, so that the marking pursuing state sequence satisfies the goal $\mathcal{G}$. Opposed to

the repetition relaxation, we are not allowed to apply actions arbitrarily often in this plan. Instead, action edges are marked with a repetition count that allows the action to reach certain target values: numbers which are sufficient to satisfy the constraints in the remaining state sequence. As these target values are not known upfront we explicate target values and the repetition counts in a *regressive* procedure based on *target value decomposition* (cf. Definition 14) and *target value regression* (cf. Definition 15) from Theorem 15.

The sequence of relaxed states starts in a state $\mathcal{I} = s_0$ 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.

We start by explicating target values which satisfy the goal constraints $\xi \trianglerighteq 0$ by choosing an arbitrary target value $q_\xi \in s_n(\xi)$ with $q_\xi \trianglerighteq 0$ for each constraint. This choice is *arbitrary*, as the regressive target value explication procedure has to be tractable, and as we only have one cost estimate for all reachable values, we cannot test several target values efficiently. Nevertheless, we will propose heuristic rules to estimate target values that usually lead to a good marking. Now given this target value q_ξ, we determine target values for all variables occurring in the expression ξ with the help of target value decomposition, For each variable, we select an achieving action for the target values assigned to the corresponding variable. Target value regression generates new target values in the "starting"-state of the action edge, so that satisfying the implicit preconditions in the preceding state is sufficient to reach the desired target value in the "ending" state. Additional target values are generated in the "starting"-state in order to satisfy the precondition of the action. Again, we chose arbitrary target values q_ξ for the preconditions $\xi \trianglerighteq 0$ of the action, where $q_\xi \in s_i(\xi)$ in the "starting"-state s_i and generate target values for the variables occurring in the expression by target value decomposition. Then, the regression procedure continues with the relaxed state s_{j-1} preceding the current state s_j in the state sequence. Note that in general, this state differs from the starting state s_i of the action edge for which the new target values are generated.

Example 25. Recall the state sequence depicted in Figure 4.5, and assume that we have to achieve a target value for variable v_2 in state s_2. The target value is achieved by applying a_2 from s_0 and therefore, the new target values will be generated for s_0. However, the regression continues with the search for achievers of the target values in the preceding relaxed state s_1.

The number of target values that has to be achieved for a given variable in a given state is restricted to two (cf. Theorem 15), as each variable starts with the *initial value* (cf. Definition 6) and expands monotonously. As such, keeping the lowest target value q^- less than the initial value and the greatest target value q^+ greater the initial value is sufficient as monotonicity automatically achieves all target values between q^- and q^+ as well.

The regression terminate as soon as all target values reach the initial values of the corresponding variables.

Given a relaxed state progression sequence as well as a set of target values that have to be reached in a certain relaxed state s_j, we can use target value

regression to determine target values in the preceding states s_i as well as a repetition count that is sufficient to enable the desired target values. However, in general, the same interval can be *achieved* by several actions in the same step. Moreover, not all actions that achieve *some* value for an interval in the current step also achieve the desired target value. As in classical planning, we estimate the cost of achieving a target value with the help of the h_{add} fact cost estimates, and select the most promising action according to the cost of the respective action evaluated by the sum of precondition fact estimates in the respective starting state s_i. With a concrete target value at hand we can identify *valid* achievers and select the one with the best h_{add} estimate. We can even stick to Formula 4.1 more closely again by allowing for the "idle arc" fact in s_{j-1} again, if the target value is already reachable in the previous state, and the "idle action" is a valid achiever even though the interval changed from s_{j-1} to s_j.

We have to select *arbitrary* target values satisfying numeric constraints of action preconditions as well as side constraints in the target value decomposition and target value regression. 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 initial values i_v an indicator for good target values. An exception to this rule are open intervals in the repetition relaxation. Open intervals are only generated as the result of a contracting effect that approaches zero, which can then be moved by other effects to arbitrary positions. Therefore, it is advisable to keep a safety margin to open interval bounds.

Example 26. Figure 4.6 depicts a relaxed state progression sequence. We start with the goal $\mathcal{G} = \{v_2 - 15 > 0\}$ and chose an arbitrary target value $q_{\mathcal{G}} = 1$ that satisfies the goal condition. The target value decomposition of the expression $\xi = v_2 - 15$ yields the pair $\langle 16, 15\rangle$ to achieve $q_{\mathcal{G}} = 1$ and we can continue with the target value $q_{v_2} = 16$ for v_2 and $q_{15} = 15$ for the constant 15.

There are two achievers for the fact $v_2 \mapsto (-\infty, \infty)$ in s_3: a_1 and a_2. The relevant effect of a_1 is $v_2 := v_1 \div 2$. However, while assigning $s_2(v_1 \div 2) = [-1, 10] \div [2, 2] = [-\frac{1}{2}, 5]$ to v_2 extends v_2 as compared to s_2 (values from 2 to 5 become reachable), the effect does not reach the target value $q = 16$ and thus, it is not a valid achiever. On the other hand, a_2 has the effect $v_2 := v_0 \times v_1$ which assigns $s_2(v_0 \times v_1) = [0, \infty) \times [-1, 10] = (-\infty, \infty)$ containing $q = 16$.

The target value regression $\langle 2, 16, 16, 1\rangle$ of the effect $v_2 := v_0 \times v_1$ assigns the initial value $q_{v_2} = 2$ to v_2 and $q_\xi = 16$ to the expression $\xi = v_0 \times v_1$. A possible target value decomposition $\langle 8, 2\rangle$ results in the target values $q_{v_1} = 8$ for and $q_{v_0} = 2$ in s_2.

The target value $q_{v_0} = 2$ in s_2 is achievable by action a_0 for a cost of $\kappa_1(v_0 \mapsto [0, 2]) + \kappa_1(v_1 \mapsto [-1, 0]) + \kappa(a_0) = 1 + 1 + 1 = 3$. However, $q_{v_0} = 2$ is reachable in the preceding state s_1 and as such, the "idle arc" $v_0 \mapsto [0, 2]$ is a valid achiever as well. And as the "idle arc" has a better h_{add} estimate, we propagate the target value $q_{v_0} = 2$ to the preceding relaxed state s_1.

The target value $q_{v_1} = 8$ is achieved by action a_1 with effect $v_1 := 5 \times v_0$ and the *target value regression* $\langle -1, 8, 8, 1\rangle$ and *decomposition* $\langle 5, 1.6\rangle$ of $\xi = v_0 \times v_1$ generates target values $q_{v_0} = 1.6$ and $q_{v_2} = -1$ in s_1. Now we have two target values $q_{v_0} = 1.6$ and $q_{v_0} = 2$ for v_0 which are both greater than the initial value 0 and thus, we only have to achieve the second one $q_{v_0} = 2$.

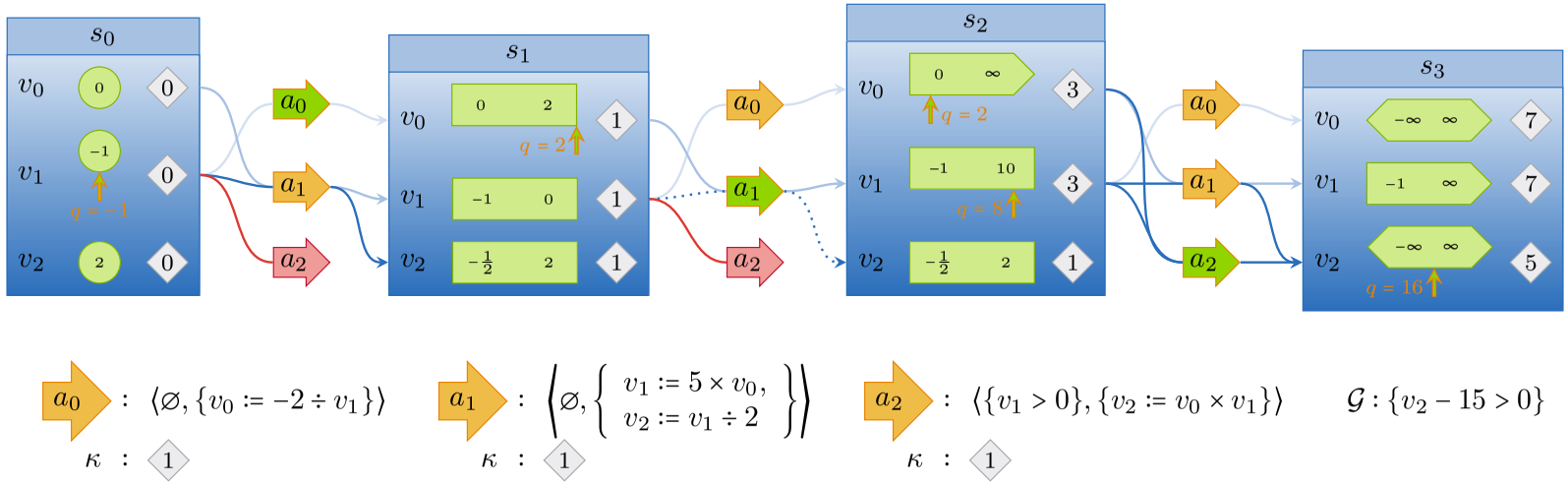

Figure 4.6: Relaxed state progression sequence where some actions are marked in green and the target value explication used in h_{FF}.

Finally, a_0 with the effect $v_0 := -2 \div v_1$ can achieve $q_{v_0} = 2$ with the *target value regression* $\langle 0, 2, 2, 1\rangle$ and the decomposition for $q_\xi = 2$ of $\xi = -2 \div v_1$ is $\langle -2, -1\rangle$ with $q_{v_1} = -1 = i_{v_1}$ so that all target values are regressed to the respective initial values.

As all actions are *marked* with a repetition count of 1, the resulting relaxed plan is then $\langle a_0^1, a_1^1, a_2^1\rangle$.

Target values are of great practical help for a regressive marking procedure which has to select numeric facts that do not have to be achieved completely. Additionally, target values allow us to select a valid achiever for the relevant value in a fact that is achieved by several actions in the relaxed state progression sequence. This opens the question whether we can derive a marking from target values and vice versa so that every plan that corresponds to the one corresponds to the other as well so that we could use "a marking" and "a set of target values" interchangeably.

Given a marking of actions in a relaxed state progression sequence, a set of target values that yield the same fact progression as the one obtained from applying all marked actions can be derived with the help of the *marking pursuing state sequence* (cf. Definition 19) as follows: For each marked action, we look at the facts in the effect and for each variable that extends the assigned interval, we select the new bounding element as target value. In a marking pursuing state sequence, effects are applied with a restricted number of repetitions (the marking) and therefore, all bounds are closed. The sole exception to this are divisions by an interval containing zero in which case bounds can diverge to infinity. In this case a sufficiently large / small number is used as target value. Such a number can easily be determined by executing the plan corresponding to the marking once, and by monitoring the constraints that have to be achieved. Whereas a single action can therefore assign two target values to a fact at most, each fact can have several achievers. However, it is sufficient to only store the greatest and the lowest value among the target values from all achievers.

Now for the converse direction we want to generate a marking given a set of target values in the relaxed state sequence efficiently. If we go through the state sequence starting from the initial state s_0, we can copy the intervals from state s_i to state s_{i+1} until we reach a target value q_v for a variable v where $q_v \notin s_{i+1}$ that has to be achieved in s_{i+1}. We have to *mark* a valid achiever for this target value and extend the interval $s_{i+1}(v)$ to the target value q_v. While this procedure allows us to find *some* marking for a given set of target values, we cannot find the *best achiever* efficiently, if there are several achievers for the same fact. The reason is that actions can have positive side effect on other target values that have to be achieved in the same or a following step. To see that finding the best achiever cannot be done efficiently unless $\mathsf{P} = \mathsf{NP}$, we can reduce minimum set cover to this problem: the subsets are given by unit cost actions whose effects achieves exactly one target value for the next relaxed state.

Finding a best achiever would be important if we would have access to an *optimal* set of target values from which we could then derive an *optimal* marking. In the next section we will discuss *optimality* of a marking with respect to the optimality of a corresponding plan.

4.4.3 Marked Action Scheduling

One major advantage of the estimate of the h_{FF} heuristic over the h_{add} estimate is that it captures beneficial side-effects of an action that achieves several relevant facts at once. In the numeric setting, actions are marked in respect to action edges. If we want to exploit beneficial interactions to the same degree as in classical planning, we have to aggregate repetitions from marking the same action edge several times, but we would also like to aggregate repetitions of the same action from several action edges that can be executed in parallel. Given a marking of a *relaxed state progression sequence* (cf. Definition 17) we want to derive an *optimal* relaxed plan that *corresponds* to the marking.

Before we define what we mean by *plan correspondence*, we have to generalize *bucket plans* (cf. Definition 18) to allow for repetitions of the same action. The *action buckets* $\mathcal{B}_i$ of a bucket plan $\langle \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$ contain actions a with a repetition count r.

Definition 20 (Plan Correspondence). Let $\langle s_0, \ldots, \mathcal{A}_n, s_n \rangle$ be a *relaxed state progression sequence* with $n+1$ states, $\mu : \bigcup_{j=1}^{n} \mathcal{A}_j \to \mathbb{N}^{\geq 0}$ be a marking and $\langle \mathcal{B}_1, \ldots, \mathcal{B}_m \rangle$ be a *bucket plan* with m buckets that contain actions with a repetition cardinality.

The bucket plan *corresponds* to marking of the relaxed state progression sequence if the number of states exceeds the number of buckets by one, and if for each action edge $\langle s_i, a, s_j \rangle$, $\sum_{a^r \in B_k} r \geq \mu(\langle s_i, a, s_j \rangle)$ for $i < k \leq j$. I.e. the sum of repetitions r of action a in buckets with an index between i and j is greater or equal to the number of repetitions that the action edge is marked with.

Figuratively speaking, a bucket plan corresponds to the marking of a relaxed state progression sequence, if we can put a bucket between all states of the sequence and the action of an action edge is contained in the bucket between its starting state and its ending state at least as often as it is marked.

Whereas bucket plans can again be obtained in two very natural ways by the bucket at the start or by the bucket at the end of an action edge, where actions of an action edge $\langle s_i, a, s_j \rangle$ that are marked r times are inserted into Bucket $\mathcal{B}_{i+1}$ or respectively in bucket $\mathcal{B}_j$ with a cardinality of r, we can usually distribute the actions among the buckets in a way that beneficial interactions are exploited more effectively. Encouragingly, an optimal corresponding plan, i.e. a plan whose cost is minimal among all plans that correspond to a given marking, can be scheduled efficiently.

The relaxed states s_i and s_j of an action edge can be seen as time bounds which determine the earliest and the latest execution time of an action. The *deploy time* i of an action is given by the index of the starting state and the *deadline* j is the index of the ending state. If several actions edges of the same action are *marked* and the execution time of these action edges overlaps, we can save executions by applying the action at a moment where the action enables several target values at once, i.e. by putting it into a bucket that is feasible for both action edges.

Theorem 21 (Optimal Corresponding Bucket Plan). *Given a relaxed state progression sequence and a marking, we can construct an* optimal corresponding bucket plan *efficiently.*

Proof. We construct an *optimal corresponding bucket plan* by sorting action edges by earliest deploy time and regressively building the bucket plan starting from the action with the latest *deploy time*. At each step, we add $\mu(\langle s_i, a, s_j \rangle)$ copies of the action a into bucket $\mathcal{B}_{i+1}$ and reduce the marking of all action edges $\langle s'_i, a, s'_j \rangle$ whose deadline is *after* the current actions deploy time, i.e. $j' > i$, by the current repetition count. If an action edge's remaining repetition marking count reaches a number less or equal than zero, we can remove the action edge from the remaining scheduling task as the desired target value is already achieved from beneficial side effects. Then, we continue the regression with the action with the latest deploy time again.

The sketched procedure outputs a bucket plan corresponding to the marking, as the marking is valid and the repetition count of an action a is only reduced by the an amount that is included in sum $\sum_{a^r \in \mathcal{B}_k} r$.

The procedure yields an *optimal* corresponding plan as each as the action edge with the latest deploy time cannot be scheduled earlier. As there is no action edge that starts later, scheduling the action can not have more beneficial side-effects by scheduling at a later time point. While the actions whose *deadline* is after the current deploy time might be applied at an earlier time for a different optimal plan (with the same cost), the action has to be executed anyways so while we might not gain anything from scheduling the "free repetitions" from the side effect we reduce the number of repetitions by the largest number of repetitions possible. The remaining repetitions (if any repetitions remain) can have beneficial side effects if they are scheduled earlier, so we leave them in the remaining scheduling problem.

The procedure can be computed efficiently, as sorting the marked action's by deploy time and by deadline can be done efficiently. The bucket plan can be constructed in a single sweep without branching which is even linear in the input. □

Now that we know that an optimal corresponding plan can be constructed efficiently, it would be interesting to know whether an *optimal* marking of a relaxed state progression sequence can be found efficiently, where an *optimal* marking is now defined as marking where the corresponding optimal bucket plan has minimal cost among all markings opposed to minimizing the action cost of the marked actions. In Theorem 20 we restricted the bound β to the sum of costs of all marked actions. Now the optimal marking could be one that has to mark additional actions because the corresponding plan could be scheduled more efficiently as compared to another marking with lesser actions cost bound β. It seems highly unlikely that finding an optimal marking in respect to the cost of an optimal corresponding plan could be possible more efficiently, but we note that this is still an open research question.

4.5 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 approaches to restrict the number of facts considered during heuristic exploration: one motivated by the *planning graph*, another from a *priority queue* to build a relaxed state progres-

sion sequence. 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} which extracts valid relaxed plans out of the sequence of monotonically increasing state intervals by explicating target values.

In the next chapter we present the details of our implementation of 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.

Chapter 5

Numeric Fast Downward

In order to assess the quality of the interval based relaxation heuristics proposed in the previous chapter, we implemented a planning system able to deal with numerical state variables: *Numeric Fast Downward* (NFD). It is based on the Fast Downward Planning System [Hel06] which is a modular planning system based on heuristic search. Most successful planners at the International Planning Competition IPC nowadays are based on Fast Downward or at least on some of its components. In the next section, we briefly outline the planning domain definition language PDDL that is used to express planning problems. The aspects of Fast Downward that are relevant for the numeric extension are then described in Section 5.2. NFD also inherits parts of Temporal Fast Downward (TFD) [EMR09], another planning system which is also based on Fast Downward. TFD can handle numeric quantities as well as temporal aspects of planning. However, TFD deals with numeric aspects very roughly and focuses of concurrency issues of temporal planning. The relevant aspects are described in Section 5.3. We will then discuss the features we implemented in Section 5.4.

5.1 PDDL

The planning domain definition language PDDL [MGH+98] is most commonly used to express domain independent planning problems. PDDL is the language which is used for the International Planning Competition (IPC), [McD00]. As such, most benchmark instances for automated planning are formulated in PDDL.

PDDL uses a closed world representation, i.e. facts that are missing in a world description are assumed to be *false*. There is no need to model that a fact is false which implies that there is full observability.

The syntax of PDDL has a prefix notation similar to LISP. Operators are described in a schematic representation. In order to obtain a planning task description as in Section 2.1 the schematic operators have to be grounded by instantiating them.

Usually, planning tasks in PDDL are separated into a *planning domain* and into a *planning problem*. The former describes the commonalities of all instances of a planning problem, most notably the available actions. The latter describes the initial state and the objects that exist in a particular planning instance.

This allows to easily generate benchmark instances that share the same domain description and vary in the number of objects available in the world.

Among the various extensions that have been proposed for PDDL, Fox and Long [FL03] introduced PDDL 2.1 with five layers of increasing expressiveness. Classical planning resides in layer 1. Layer 2 allows to express numeric planning problems with instantaneous actions which is the area of automated planning that we focus on in this thesis. Layer 3 allows for durative actions and for concurrency where changes in the world happen at discrete moments (e.g. at the beginning or the end of an action). Actions with continuous change can be expressed by PDDL 2.1, layer 4 and layer 5 allows for exogenous events i.e. the world is not assumed to be static any more.

The numeric planning problems with instantaneous actions that we are interested in can be expressed with PDDL 2.1, layer 2.

5.2 Fast Downward

Fast Downward is a planning system based on heuristic forward search and hierarchical task decomposition. It takes a planning problem, usually described in PDDL, and outputs one or several *plans*. Fast Downward first performs several preprocessing steps (cf. Section 5.2.1) before performing a *heuristic search* (cf. Section 5.2.2) to output a plan or several plans of increasing quality if it is called in an anytime mode.

5.2.1 Preprocessing

The preprocessing components of Fast Downward translate the PDDL task into a grounded SAS^+ representation that is more suitable for the actual search. Preprocessing is split into a *translation* phase and a *knowledge compilation* phase. During translation, Boolean formulas in preconditions and effects are simplified. Then, mutex invariants are identified, i.e. variables that are *mutually exclusive*, i.e. they can never be *true* at the same time. After *grounding* the schematic PDDL operators, the mutex invariants can be used to translate Boolean to *multi-valued* variables.

In the *knowledge compilation* step, *domain transition graphs* and *causal graphs* are determined that are required for some of the heuristics used during *search*. Additionally, they allow for the detection of variables that can be eliminated beyond the simplifications performed during the *translation* phase. Moreover, a *successor generator* and an *axiom evaluator* are determined which allow the search to generate successors and evaluate axioms more efficiently.

5.2.2 Search

Fast Downward's search component allows for many different configurations. Most commonly, the core search algorithm is a variant of a *greedy best first search*. The search is *greedy* in the sense that it always expands the *best* node according to an *evaluation function*. The configuration of this evaluation function allows for many different search algorithms, e.g. $A^\star$ search [HNR68] evaluates nodes by summing the path cost from the initial node to the current node and the heuristic estimate of the current node to a goal. The evaluation function

is usually evaluated when search nodes are generated (*eager evaluation*). However, for search problems with a large branching factor, it is usually advisable to *defer* node evaluation to the time when they are expanded (*lazy* or *deferred evaluation*) and use the heuristic information of the parent node instead.

Moreover, Fast Downward allows for *multi-heuristic best first search* where several separate open lists for different *evaluation functions* are maintained.

5.3 TFD

Temporal Fast Downward [EMR09] is a Fast Downward based planning system for planning problems with temporal concurrency. TFD can also deal with numeric aspects of planning, although its main focus is the support of temporal actions.

As a temporal planner, TFD does not support *instantaneous actions* (actions without a duration) and can therefore not be applied on regular planning problems. TFD is not as optimized as Fast Downward and several simplifications and optimizations performed during preprocessing are skipped. TFD aims at solving temporal planning problems and implements one heuristic, the context enhanced additive heuristic (CEAH). While the heuristic deals with the propositional aspects of the planning task and gives an estimate on the cost of the actions that have to be applied in order to derive the propositional part, the support of numeric variables is very basic in nature. Numeric aspects are retrospectively added to the heuristic estimate, after the actions that are relevant for the propositional aspect have been determined. For each action that CEAH considers relevant, TFD checks the preconditions of this actions for numeric comparisons. If the comparison is not satisfied in the current state, the deviation of the current value of the variable to the required value contributes directly to the heuristic cost. The sum of all deviations is then used to establish basic guidance for the numeric aspect of the planning problem. Obviously, this difference does not reflect the actual effort that has to be spent to alter the value of a variable to its target value. Additionally, it can mislead the propositional aspect of the heuristic, as the deviation of a numeric variable from its target value can be orders of magnitude larger.

On the implementation side, TFD decomposes complex numeric expressions into a tree of binary expressions by inserting auxiliary variables. Thus,

Similarly, auxiliary propositional variables are introduced for the comparison result of numeric constraints. Numeric axioms evaluate these auxiliary variables on demand which reduces memory consumption during search.

5.4 NFD

The Numeric Fast Downward (NFD) planning system is a planner based on Fast Downward that has been developed during the doctoral process. NFD occupies the space between regular Fast Downward and TFD, namely numeric planning with instantaneous actions. NFD supports numeric planning from PDDL 2.1, layer 2 [FL03] as well as selected features from PDDL 3 such as global constraints. Classical planning benchmarks can be solved with NFD as well. The original Fast Downward does not support floating point numbers and

thus, major modifications had to be performed in preprocessing, state handling and search. In addition to the propositional variables that are compiled to multi-valued variables in Fast Downward, NFD also maintains a set of numeric variables. Opposed to TFD, NFD does not have to deal with temporal aspects. Thus, it can find much more accurate estimates for the numeric aspects of planning.

Structurally, NFD handles numeric expressions as TFD: it introduces auxiliary variables for all intermediate results and evaluates expressions with the help of binary arithmetic axioms. Similarly for numeric constraints an auxiliary propositional variable stores the result of a comparison axiom. These auxiliary variables facilitate *target value decompositions* (cf. Definition 13) and *regressions* (cf. Definition 15) as the sub-expressions are then present in the problem representation. Opposed to TFD, NFD performs all preprocessing optimizations on the propositional variables and can solve classical planning problems analogously. During preprocessing, numeric axioms are simplified (e.g., the sum of two constants is not evaluated by an axiom but replaced by a new constant whose value is computed once during preprocessing) and irrelevant variables are removed analogously to regular Fast Downward. Data structures like *domain transition graph* and *causal graph* are also generated for numeric variables, with the simplified assumption that all values (i.e. all values from $-\infty$ to ∞) are considered reachable if a variable is assigned a new value.

Fast Downward supports integer valued action costs (i.e. $\kappa(a) \in \mathbb{N}$) where NFD allows for 64-bit floating point numbers. This comes with a big disadvantage concerning compatibility to classical planning: a bucket based priority queue is usually the most efficient data structure for search nodes, but it becomes impractical when search nodes are rarely enqueued with the same priority. Thus, while classical planning problems can be solved with NFD as well, NFD can not always keep up with the performance of regular Fast Downward on classical benchmark problems. Another challenge with rational numbers is the representation problem, as numbers have to be rounded to the closest representable floating point number. For the interval based relaxations we round the upper interval bound up to the next representable number and for the lower bound we round down. This way we ensure that all reachable numbers are enclosed in the interval. However, if we compute target value decomposition or regression, we have to derive target values by applying inverse arithmetic operations. As such, we had to implement floating point rounding in a context-aware way. Critical situations appear often in practice, as locally promising target values are often found directly on the interval bounds for target value decomposition and regression.

The implementation of the priority queue based numeric heuristics follows instructions for classical planning presented by Röger and Helmert [RH13]: numeric effects of an action are evaluated in isolation. The "concurrency" of the effects of an action does not influence any of the heuristic computations, and therefore, actions are decomposed into unary actions: actions with only one effect. This also allows for easy handling of conditional effects, as the effect condition can be merged into the action precondition for such effects.

In order to ensure polynomial time complexity of the heuristic computation, we break cycles by the introduction of auxiliary variables as discussed in Section 4.2. A related issue of the priority queue based approach is discussed in Theorem 19: enqueuing actions respective to their cost does not necessarily

respect topological restrictions. As suggested in the section, we use a two phase computation for $h_{\max}$, where we identify maximally reachable intervals in a first phase, and use these intervals in the second phase. For h_{add}, we block variables from being enqueued in the priority queue if a topologically higher variable has not been processed yet. For h_{FF}, we base tie-breakers on the h_{add} estimate. At the time of implementing the heuristics, we were not aware that actions can be scheduled optimally in polynomial time in order to exploit beneficial side effects from marking actions (cf. Section 4.4.3). Instead we mark actions at the earliest possible time (close to s_i for marked action edges $\langle s_i, a, s_j \rangle$). Even without this optimization, h_{FF} with a priority queue based progression approach yields impressive results as we will see in the upcoming section.

5.4.1 Metric

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 [CCFL13], 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.

Chapter 6

Experiments

In this section we present the experiments we performed. First, we will introduce the JUMPBOT domain, a collection of benchmark instances that is particularly interesting as it uses numeric variables to model physical quantities. Then, we briefly discuss the other benchmark instances we used for our experiments. In Section 6.2, we will introduce the configurations that we tested before we show the performance of several NFD configurations in terms of *plan cost*, *coverage* and *algorithmic quality*. Afterwards, we compare the best NFD configurations to other state-of-the-art planners Metric FF [Hof03] and ENHSP [SHTR16; SHT16]. Finally, we discourse into Earth observation planning that was solved with Temporal Fast Downward with semantic attachments (TFD/M).

6.1 Jumpbot

The JUMPBOT domain is a planning domain for numeric planning with instantaneous actions which models physical properties in a dynamic world. We introduced the JUMPBOT domain in a technical report [AL16] on which this section is based. It features cyclic, non-linear effects for turning, accelerating or decelerating the robot, as well as classical preconditions. Therefore, it can neither be solved by control engineering [LEKN12] nor by planners requiring linear tasks such as Metric FF [Hof03].

JUMPBOT models a walking robot that has to reach a target region by jumping over water ditches. The kinematic of the robot is modeled by its current position and velocity vector. The planner has to reason about the correct accelerations, rotations, velocities, jump positions and space for the deceleration. States of the world are the *footprints* of the robot modeled by four numeric variables x, y, v_x, v_y describing the position vector and the velocity vector of the robot. A description of the scenario is depicted in Figure 6.1. The task is to plan a step trace from an initial pose (depicted by a red cross with a velocity vector depicting orientation and speed of the robot) to a pose where the robot is located inside the green *goal region* with a velocity close to zero. Water is depicted by blue waves and solid ground by white surface. The robot is only allowed to step on solid ground but it may step or jump over water ditches. The domain contains information about the robot's mass, velocity, momentum and acceleration capacities. There are six actions to steer the robot: `step`, `jump`,

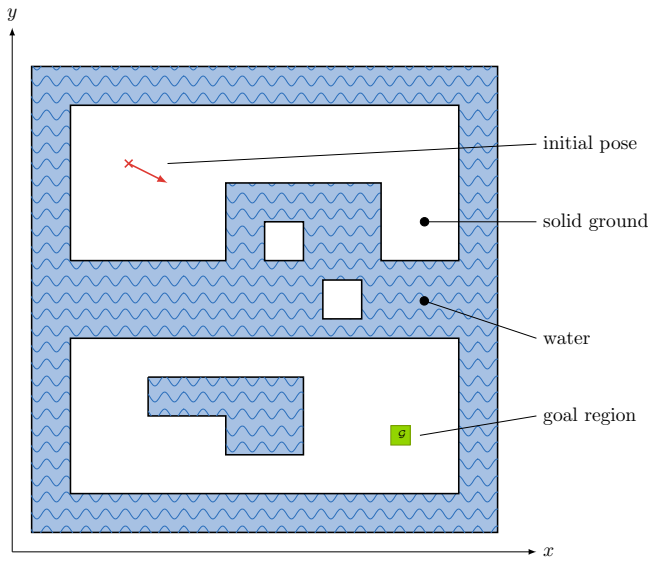

Figure 6.1: Scenario description of an exemplary JUMPBOT instance.

`accelerate`, `decelerate`, `steer left` and `steer right`.

The state evolution is obtained by state transition matrices considering the physical dynamics of the robot where actions discretize the world model by exerting the respective force for a duration of 0.5 seconds for most actions after which the robot "steps on the ground" again. The exception is the `jump` action where the robot progresses towards its current orientation for 1 second. Propositional preconditions prevent the robot from turning to alternating directions in consecutive actions and the `jump` action requires the robot to move at a minimum velocity in order to be executed.

JUMPBOT provides a set of benchmark instances for numeric planning in an area which is underrepresented by prevailing benchmarks: the use of numeric variables to model physical properties as opposed to their use for modeling resources. As such it offers challenges that go beyond little modifications of classical planning systems and is particularly interesting to benchmark numeric planning systems.

6.1.1 Benchmarks

Besides the JUMPBOT domain, we tested our NFD configurations as well as the other numeric planning algorithms on various numeric benchmark domains from the IPC 2002 [LF03], Francès and Geffner [FG15] and Scala, Haslum, and Thiébaux [SHT16].

- BLOCKGROUPING: blocks of different colors have to be moved in a grid so that all blocks of the same color end up in the same grid cell. The number of blocks of a certain color in a grid cell is modeled as resource that can be increased or decreased.
- COUNTERS: a set of counters that can be increased or decreased, have to be transferred into a state where the numbers of the counter are sorted

in increasing order. For COUNTERS-0, all counters are initialized to zero, for COUNTERS-INV, they are initially sorted in decreasing order and for COUNTERS-RND at random.

- DEPOTS: a combination of BLOCKS and LOGISTICS, where trucks transport crates that have to be stacked at the target locations. The numeric variant adds resources such as weight, capacity and fuel.
- DRIVERLOG: a LOGISTICS problem on two layers: trucks drive around to deliver packages, but the drivers have to walk between trucks on another map. The numeric variants have cost associated to driving and walking. In the *hard* variant this cost is non-linear.
- FARMLAND: several locations ("farms") contain a resource x which can be transferred from connected locations slowly $(-1/+1)$ or fast $(-4/+2)$. Farms have a weight factor, and the goal is to move the resource from farms with low weight to farms with high weight without depleting it (the option to move fast is a trap for relaxation approaches).
- GEO-ROVERS: a newer variant of the planetary ROVERS domain.
- HYDRAULIC-BLOCKS-WORLD: a BLOCKSWORLD problem where the blocks contain liquids and pistons that pose additional constraints.
- JUMPBOT: a robot has to jump over water ditches, see previous section for more details.
- PLANT-WATERING: an agent has to navigate in a grid world, moving water from a grid with a taps to grids with flowers.
- ROVERS: a planetary rover as to navigate a planet and collect soil samples etc. Energy is a resource that can be replenished by recharging in the sun.
- SAILING: a boat has to set sail in a continuous 2D world in order to save drowning people.
- SATELLITE: several satellite have to collect data like taking images from Earth. Satellites are equipped with non-replenishable fuel and capacity. In the *hard* variant the empty plan is a goal, but one that accumulates no reward.
- SETTLERS: a domain that heavily relies on resource management. Primary resources (timber, stone, ore) have to be gathered and refined to secondary resources (wood, coal, iron). Transport vehicles and refinement structures have to be built before they can be used.
- SOKOBAN: the famous puzzle in a version where agent and stones are modeled with numeric variables for x and y coordinates.
- UMT2: a complex numeric LOGISTICS domain using different vehicle types (plane, train, truck).
- ZENOTRAVEL: a LOGISTICS domain where people fly in a plane that can either run fast or slow. If it runs faster it consumes more fuel.

6.2 Numeric Planning Experiments

We ran the NFD planner on $3 \times 2 \times 2 \times 2 = 24$ configurations, all using greedy best first search. We compared the heuristics $h_{\max}$, h_{add} and h_{FF} in the two most promising combinations of relaxation and progression approach identified in Chapter 4.3: the *planning graph* approach in an *interval relaxation* (identified by the superscript h^{ig}) and the *priority queue* based approach in the *repetition relaxation* (identified by the superscript h^{rq}). We compared *eager search*, where heuristic estimates are computed for each search state as soon as the node is generated and put into the algorithm's priority queue to *lazy search* where successor nodes are enqueued with the cost estimate of their parents and the heuristic is only computed when nodes are dequeued and processed. Lazy search is also known as *deferred evaluation* in other contexts. Besides the regular action costs given in the planning task, we also performed experiments with a *unit cost* task transformation where actions are assumed to cost 1 during search and heuristic computation.

Experiments were run on an Oracle cluster with two Intel E5-2650v2 processors per node each with eight cores and a clock frequency of 2.6 GHz. Processes were given a timeout of 30 minutes CPU-time for each instance and a memory limit of 3 GB.

We used Lab [SPSH17] to perform the experiments and extract the experimental results in the form of tables and plots from the raw data. In our experiments we measure three quantities to asses the performance of an algorithm and its configuration: cost, coverage and quality. In the following, we will explain these measures in more detail.

The *cost* assesses the plan cost of the solutions returned by the algorithm. The plan cost measure can only be applied to instances where all algorithms have found a solution, so we usually show such differences with *scatter plots*. E.g., Figure 6.2 shows a scatter plot comparing an eager (E) and the lazy (L) configuration of the h^{ig}_{FF} heuristic. The coordinate axes are labeled with the algorithm's configuration and plan costs in the range from 0 to 50. In the example, the x-axis is the *eager* configuration and the y-axis the *lazy* configuration. As it is beneficial for a plan to have lower cost, values above the diagonal indicate that the eager configuration at the x-axis performed better for the given planning instance. The marks are colored depending on the domain and a legend is depicted to the right of the graph. We group domains with different variants such as COUNTERS-0 and COUNTERS-INV. Planning instances that could be solved by one configuration but not by the other are depicted at the very top or the very right respectively. As the plans of planning instances can become quite costly, depicting all instances would distort the coordinate range. We decided to crop plans with a cost greater than 50. If one configuration finds a plan with a cost less than 50 but the other configuration does not, the mark is depicted at the very top or right as well. Combinations where no configuration can find a plan with a cost less than 50 are excluded from the scatter plots.

Additionally, we compare *plan cost* in tables such as Table 6.1. Here, we compare configurations pairwise that only differ in the one feature that we want to assess. For each domain, we sum up the plan costs of all instances that both configurations could solve. Note that we can only compare the entries of adjacent rows in these tables. In particular, if one configuration has a smaller plan cost as compared to another one this does not necessarily mean that it

found a better plan, it can also mean that it solved fewer instances.

The *coverage* assesses the number of instances solved by a given planner configuration for a given domain. It can be either 0 (no solution found) or 1 (solution found) for each instance.

Finally, *quality* assesses the configurations quality as trade-off between *cost* and *coverage*. It is sometimes also referred to as *IPC-score* as the International Planning Competitions IPC [McD00] use that metric. Given the optimal solution of a certain benchmark instance, the IPC-score is defined as ratio between the cost of the optimal plan and the cost of the plan reported by the planning algorithm resulting in a number between 0 and 1. E.g., the planner receives an IPC-score of 1 if it finds the optimal plan, an IPC-score of 0.25 if it finds a plan that costs four times as much and an IPC-score of 0 if it cannot solve the problem. Unfortunately, the IPC-score can usually not be computed, as the cost of an optimal plan is unknown. As such, we approximate the cost of an optimal plan by the best plan cost that is known, which usually coincides with the plan cost reported by the planner that performed best on a given benchmark instance. An example where this approximation goes wrong can be found in the SatelliteHard domain where the objective is to maximize the data accumulated by the satellite. The empty plan is valid and found by all planners. As such all receive an IPC score of 1 even though they should receive a score of 0 if at least one planner that accumulates *some* reward would participate.

In our tables, we indicate the domains in the first row. The number in parentheses following the domain name indicates the number of instances that are available. For coverage and quality, this number is the theoretical maximum that a planning algorithm could receive for the domain. For all tables, the best configuration is marked in **bold** and domains where no instance could be solved are marked with a dash. For tables that evaluate plan cost, we compare configurations pairwise and as such the better configuration is indicated in bold.

6.2.1 Eager vs. Lazy Evaluation

In a first experiment, we evaluate the impact of using eager evaluation (E) compared to evaluating heuristic estimates lazily (L). Deferred (lazy) evaluation of search states allows the search algorithm to visit more states at the cost of a less informed heuristic estimate.

We first compare the plan length of the plans returned by both configurations. We compare configurations pairwise where all parameters except the evaluation method are fixed. Whenever both planners solve an instance of a planning domain, we record the cost of the resulting plan. The entry in Table 6.1, Table 6.2 and Table 6.3 is then the sum of all instances from the domain that both configurations could solve.

The pairwise comparisons of configurations using the $h_{\max}$ heuristic are presented in Table 6.1, starting with unit cost transformed heuristics with interval relaxation and planning graph based approach to generate the relaxed state progression $1h^{ig}_{\max}$, followed by unit cost transformed heuristics with repetition relaxation and a priority queue based approach $1h^{rq}_{\max}$. Then, the same configurations are shown in a setting where regular action costs are used, $h^{ig}_{\max}$ and $h^{rq}_{\max}$. The plans generated by *eager evaluation* are shorter (in the sense of summed up action cost) on average, though there are a few exceptions. The plan costs of all pairwise solved instances sum up to 1.52 million (abbreviated by

M in the following) for E$h_{\max}$ whereas lazy configurations L$h_{\max}$ return plans with a total length of $1.69M$, an increase in length by approximately 11%.

Configurations using the h_{add} heuristic, depicted in Table 6.2, show the increase in plan length from *eager evaluation* to *lazy evaluation* even more clearly. There is almost no domain (with SOKOBAN being the exception) where a lazy approach outperforms eager in terms of plan length. Eager instances sum up to $7.72M$ whereas lazy instances yield plans with a summed up cost of $10.23M$, a significant increase of approximately 32%. The increase in total cost as compared to $h_{\max}$ can be attributed to h_{add} finding more solutions especially on harder problems which require longer plans to be solved.

Finally, the pairwise comparison of plan cost of plans found by the h_{FF} heuristic is shown in Table 6.3. Again, the total plan cost sum of all eager instances, $5.56M$ is less than the respective sum of $5.79M$ for lazy instances, indicating an increase in plan length of 4% on average. However, this increase is far from significant and there is one configuration, lazy evaluation with a repetition relaxed priority queue approach Lh^{rq}_{FF}, where lazy can even find shorter plans compared to the corresponding eager configuration.

Summing up the costs over all instances from all heuristics $h_{\max}$, h_{add} and h_{FF}, the combined plan cost of eager instances is $14.800M$ whereas a cost of $17.704M$ is required for an increase of approximately 20%. Even though the increase can be mostly attributed to configurations using variants of the h_{add} heuristic, eager evaluation yields better plans averaged over all domains.

In addition to the tables, we depicted the cost of four combinations of algorithms in the scatter plots Figure 6.2, Figure 6.3, Figure 6.4 and Figure 6.5 all using the h_{FF} heuristic. In all plots we cropped plan cost to 50 and treated plans with a cost greater than 50 as unsolvable. We can see that there are more marks above the diagonal than below which supports our initial claim that eager evaluation usually yields slightly better plans, and that the result in the table are not distorted by single outliers.

Whereas eager evaluation performs better comparing plan cost, we experience the opposite concerning *coverage*. Again, we compare all configurations using eager heuristic evaluation to all combinations using deferred evaluation. Opposed to plan cost, we can compare the coverage of one particular configurations to all others. Note that in theory this would also be possible for plan costs if we restrict ourselves to instances that can be solved by every single configuration. However, this applies only to the most simple instances of the simpler domains and would therefore not be meaningful.

Again we start by $h_{\max}$ and the coverage of all configurations can be found in Table 6.4. This time, we aggregate the four eager configurations to the left, and the lazy configurations to the right. For $h_{\max}$, lazy configurations marginally outperform eager configurations by 2.3% in coverage averaged over all instances. A similar picture emerges for configurations based on h_{add} that are stated in Table 6.5. Again, the lazy approach achieves a marginal increase in coverage over eager by 0.7% on average. Finally, for Lh_{FF} there is an slight increase in coverage of 1.8% over eager configurations as well. Details are found in Table 6.6. Summarizing the coverage of all eager instances from all three heuristics, evaluating instances lazily increases coverage by a small amount of 1.6%. Even though this increase in coverage is not overly impactful, we can see that there is a trade-off between plan quality (eager) and coverage (lazy).

In order to relate plan cost and coverage, we use the IPC-score inspired

Plan Cost $h_{\max}$	1E$h^{ig}_{\max}$	1L$h^{ig}_{\max}$	1E$h^{rq}_{\max}$	1L$h^{rq}_{\max}$	E$h^{ig}_{\max}$	L$h^{ig}_{\max}$	E$h^{rq}_{\max}$	L$h^{rq}_{\max}$
block-grouping	**259.00**	**259.00**	**66.00**	**66.00**	**259.00**	**259.00**	**66.00**	**66.00**
counters-0	**35.00**	**35.00**	**7.00**	**7.00**	**35.00**	**35.00**	**7.00**	**7.00**
counters-inv	**15.00**	**15.00**	**15.00**	**15.00**	**15.00**	**15.00**	**15.00**	**15.00**
counters-rnd	**101.00**	104.00	**27.00**	**27.00**	**101.00**	104.00	**27.00**	**27.00**
depots	**296.00**	306.00	**296.00**	306.00	**185.00**	195.00	**184.00**	194.00
driverlog	**15 748.00**	17 166.00	**15 748.00**	17 166.00	**18 305.00**	19 983.00	**12 953.00**	14 127.00
driverlog-hard	**85 540.00**	94 290.00	**85 540.00**	94 290.00	**92 540.00**	103 690.00	**57 830.00**	63 730.00
farmland	**8 825.00**	8 864.00	**4 380.00**	**4 380.00**	**8 834.00**	8 876.00	**4 380.00**	**4 380.00**
geo-rovers	–	–	–	–	–	–	–	–
hydraulic-bw	**7 054.00**	7 110.00	**7 054.00**	7 110.00	**7 054.00**	7 110.00	**7 054.00**	7 110.00
jumpbot	**218.50**	**218.50**	–	–	**152.50**	159.00	–	–
plant-watering	–	–	**521.00**	**521.00**	–	–	**521.00**	**521.00**
rovers	**0.00**	**0.00**	**0.00**	**0.00**	–	–	**0.00**	**0.00**
sailing	**1 018.00**	1 019.00	**196.00**	**196.00**	**1 018.00**	1 019.00	**196.00**	**196.00**
satellite	–	–	406.74	**369.76**	**73.58**	**73.58**	225.48	**183.46**
satellite-hard	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
settlers	–	–	**9.00**	**9.00**	–	–	–	–
sokoban	2 771.00	**2 525.00**	699.00	**677.00**	2 777.00	**2 526.00**	751.00	**729.00**
zenotravel	169 303.00	**168 391.00**	169 303.00	**168 391.00**	**415 511.00**	523 182.00	**314 169.00**	337 988.00
Sum	**291 183.50**	300 302.50	**284 267.74**	293 530.76	**546 860.08**	667 226.58	**398 378.48**	429 273.46
		Sum eager:	**1 520 689.8**			**Sum lazy:**	1 690 333.3	

Table 6.1: Plan costs of *eager evaluation* (E) compared to *lazy evaluation* (L) with variants of $h_{\max}$.

Plan Cost $h_{\mathbf{add}}$	1Eh^{ig}_{add}	1Lh^{ig}_{add}	1Eh^{rq}_{add}	1Lh^{rq}_{add}	Eh^{ig}_{add}	Lh^{ig}_{add}	Eh^{rq}_{add}	Lh^{rq}_{add}
block-grouping	**567.00**	568.00	**436.00**	461.00	**567.00**	568.00	**369.00**	389.00
counters-0	107.00	**101.00**	**7.00**	9.00	107.00	**101.00**	**7.00**	9.00
counters-inv	**195.00**	**195.00**	**15.00**	**15.00**	**195.00**	**195.00**	**15.00**	**15.00**
counters-rnd	**556.00**	**556.00**	**39.00**	45.00	**556.00**	**556.00**	**39.00**	45.00
depots	**961.00**	1 151.00	**1 058.00**	1 209.00	**699.00**	759.00	**710.00**	808.00
driverlog	**54 736.00**	67 665.00	**65 067.00**	77 317.00	**35 318.00**	43 116.00	**32 790.00**	38 215.00
driverlog-hard	**254 340.00**	348 890.00	**353 020.00**	453 010.00	**171 940.00**	205 960.00	**185 830.00**	223 990.00
farmland	**6 131.00**	6 140.00	**4 410.00**	4 440.00	**6 130.00**	6 150.00	**4 410.00**	4 440.00
geo-rovers	**101.00**	119.00	–	–	**101.00**	119.00	–	–
hydraulic-bw	**7 852.00**	7 880.00	**7 852.00**	7 880.00	**7 852.00**	7 880.00	**7 852.00**	7 880.00
jumpbot	**207.00**	241.50	–	–	**181.50**	200.50	–	–
plant-watering	–	–	**1 052.00**	1 185.00	–	–	**1 052.00**	1 185.00
rovers	**0.00**	**0.00**	**2.00**	3.00	–	–	**0.00**	**0.00**
sailing	**17 266.00**	17 678.00	**546.00**	557.00	**17 266.00**	17 678.00	**1 391.00**	1 404.00
satellite	**662.32**	704.72	**1 793.69**	2 191.54	**510.84**	574.07	**803.71**	1 018.53
satellite-hard	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
settlers	150.00	**148.00**	**164.00**	**164.00**	–	–	–	–
sokoban	13 747.00	**13 387.00**	8 036.00	**7 610.00**	**13 661.00**	13 763.00	8 122.00	**7 472.00**
zenotravel	**2 131 002.00**	2 283 143.00	**2 193 692.00**	2 381 530.00	**1 030 809.00**	1 825 166.00	**1 068 743.00**	2 131 314.00
Sum	**2 488 580.32**	2 748 567.22	**2 637 189.69**	2 937 626.54	**1 285 893.34**	2 122 785.57	**1 312 133.71**	2 418 184.53
		Sum eager:	**7 723 797.06**			**Sum lazy:**	10 227 163.86	

Table 6.2: Costs of *eager evaluation* (E) compared to *lazy evaluation* (L) with different variants of h_{add}.

Plan Cost $h_{\mathbf{FF}}$	1Eh^{ig}_{FF}	1Lh^{ig}_{FF}	1Eh^{rq}_{FF}	1Lh^{rq}_{FF}	Eh^{ig}_{FF}	Lh^{ig}_{FF}	Eh^{rq}_{FF}	Lh^{rq}_{FF}
block-grouping	**730.00**	815.00	**984.00**	1 073.00	**566.00**	623.00	**990.00**	1 079.00
counters-0	**689.00**	691.00	**237.00**	269.00	691.00	**689.00**	**235.00**	263.00
counters-inv	**65.00**	71.00	**408.00**	507.00	**77.00**	79.00	**408.00**	507.00
counters-rnd	**308.00**	364.00	**2 273.00**	2 711.00	**296.00**	357.00	**2 273.00**	2 711.00
depots	**1 892.00**	1 914.00	**1 879.00**	2 593.00	**797.00**	840.00	**1 231.00**	1 524.00
driverlog	**89 387.00**	109 500.00	**85 239.00**	133 494.00	**22 740.00**	24 148.00	28 272.00	**28 005.00**
driverlog-hard	**442 540.00**	600 140.00	**470 210.00**	818 910.00	**140 420.00**	178 140.00	143 020.00	**134 260.00**
farmland	10 519.00	**8 951.00**	**8 836.00**	8 866.00	13 998.00	**11 631.00**	11 545.00	**11 188.00**
geo-rovers	–	–	**604.00**	705.00	–	–	1 075.00	**1 015.00**
hydraulic-bw	**7 630.00**	10 278.00	**7 416.00**	9 022.00	**7 630.00**	10 278.00	**7 416.00**	9 022.00
jumpbot	**201.50**	212.50	**95.00**	121.50	162.00	**156.00**	**85.50**	106.00
plant-watering	–	–	**1 945.00**	3 361.00	–	–	**1 911.00**	3 325.00
rovers	**0.00**	**0.00**	**23.00**	**23.00**	**0.00**	**0.00**	**0.00**	**0.00**
sailing	**999.00**	1 021.00	**1 974.00**	2 808.00	**999.00**	1 021.00	**1 974.00**	2 808.00
satellite	**1 148.99**	1 183.64	5 685.13	**5 212.93**	**371.98**	398.53	1 152.59	**496.86**
satellite-hard	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
settlers	**176.00**	**176.00**	2 002.00	**1 982.00**	–	–	**208.00**	212.00
sokoban	9 444.00	**7 759.00**	8 743.00	**8 405.00**	10 152.00	**9 431.00**	**9 402.00**	9 731.00
zenotravel	1 228 750.00	**1 059 713.00**	1 887 250.00	**1 693 988.00**	**478 169.00**	530 895.00	386 733.00	**314 392.00**
Sum	**1 794 479.49**	1 802 789.14	**2 485 803.13**	2 694 051.43	**677 068.98**	768 686.53	597 931.09	**520 644.86**
		Sum eager:	**5 555 282.69**			**Sum lazy:**	5 786 171.96	

Table 6.3: Plan costs of *eager evaluation* (E) compared to *lazy evaluation* (L) with different variants of h_{FF}.

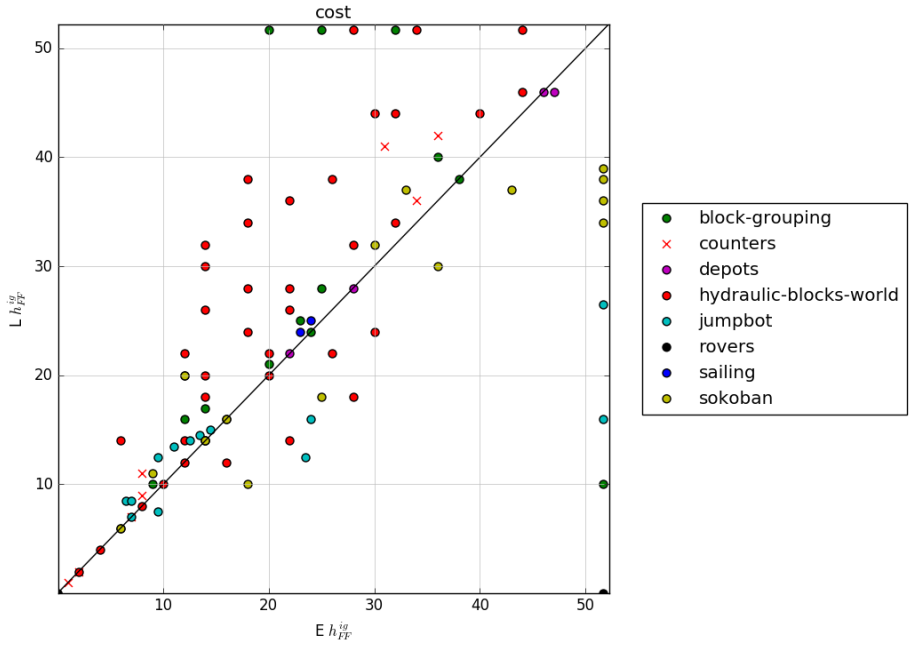

Figure 6.2: Comparison of *interval relaxation + planning graph* based h^{ig}_{FF} with lazy and eager evaluation in a original cost setting.

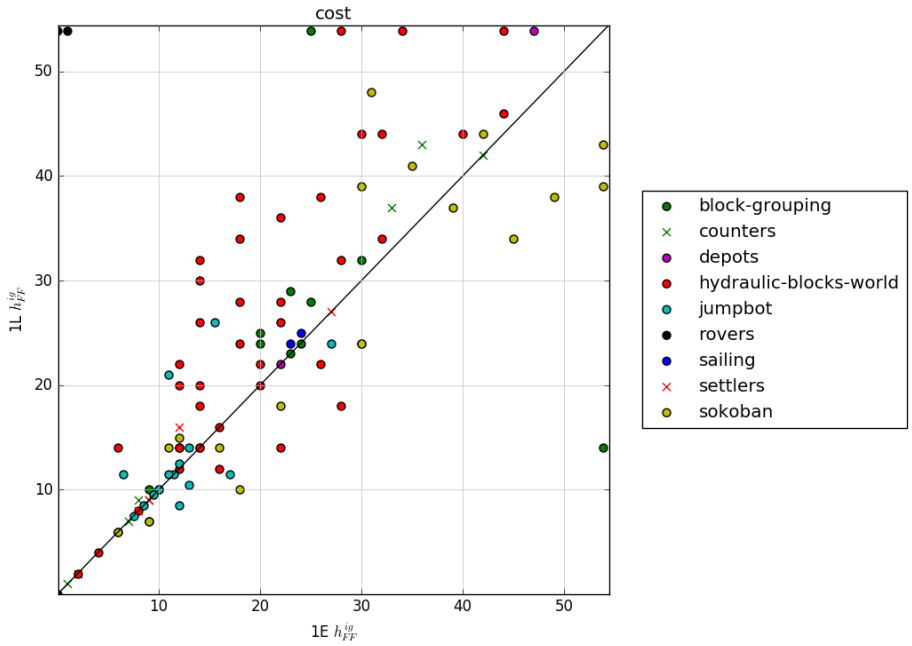

Figure 6.3: Comparison of *interval relaxation + planning graph* based $1h^{ig}_{\mathrm{FF}}$ with lazy and eager evaluation in a unit cost setting.

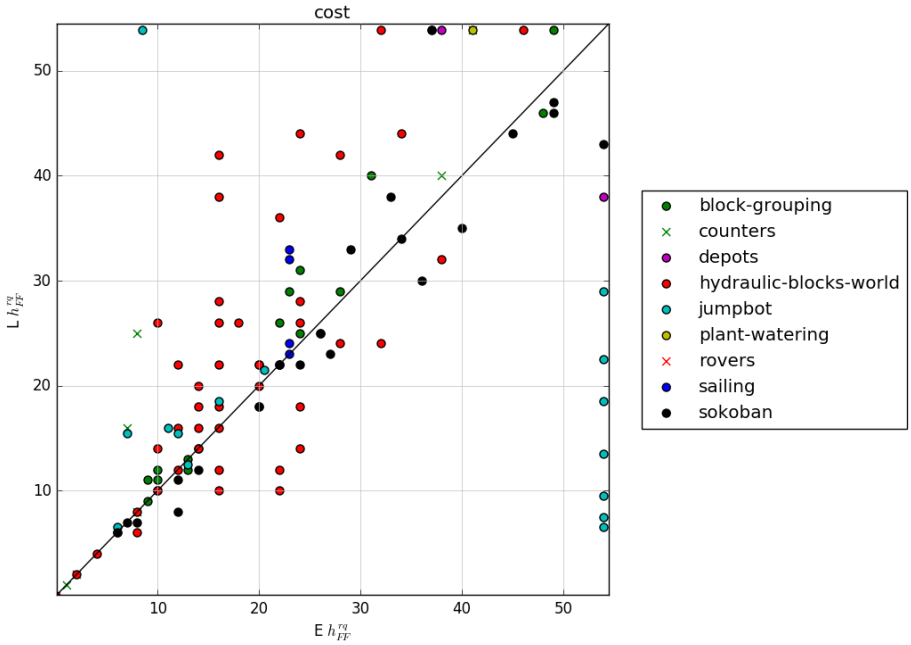

Figure 6.4: Comparison of *repetition relaxation + priority queue* based h^{rq}_{FF} with lazy and eager evaluation in a original cost setting.

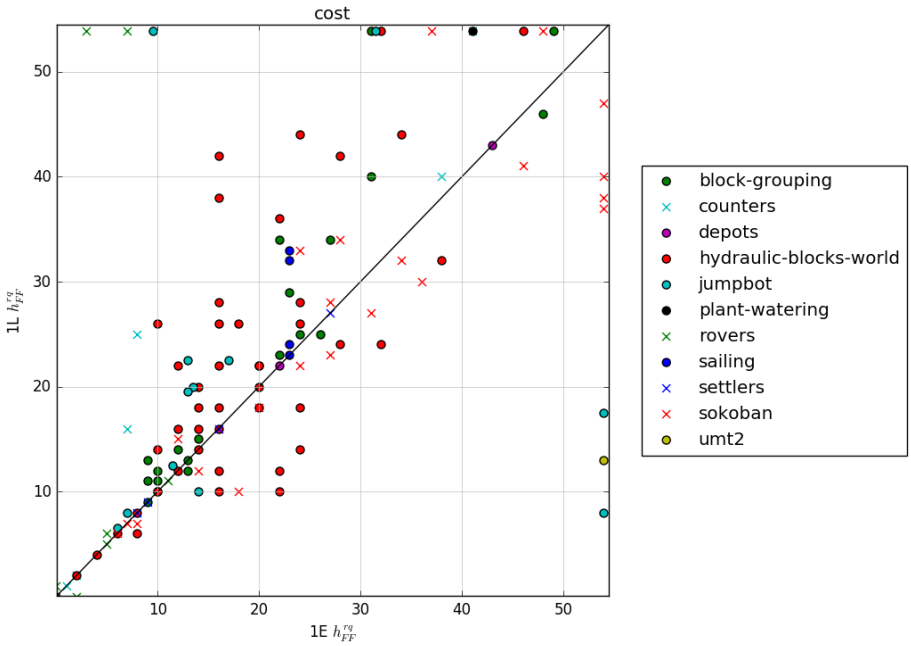

Figure 6.5: Comparison of *repetition relaxation + priority queue* based $1h^{rq}_{\mathrm{FF}}$ with lazy and eager evaluation in a unit cost setting.

Coverage $h_{\max}$	E$h^{ig}_{\max}$	1E$h^{ig}_{\max}$	E$h^{rq}_{\max}$	1E$h^{rq}_{\max}$	L$h^{ig}_{\max}$	1L$h^{ig}_{\max}$	L$h^{rq}_{\max}$	1L$h^{rq}_{\max}$
block-grouping (192)	**11**	**11**	8	8	**11**	**11**	7	7
counters-0 (34)	**3**	**3**	2	2	**3**	**3**	2	2
counters-inv (11)	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
counters-rnd (33)	8	8	6	6	**9**	8	6	6
depots (22)	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
driverlog (20)	13	13	10	13	**14**	12	10	12
driverlog-hard (20)	12	**13**	10	**13**	12	12	10	12
farmland (50)	**20**	**20**	10	10	**20**	**20**	10	10
geo-rovers (21)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
hydraulic-bw (562)	**541**	**541**	**541**	**541**	**541**	**541**	**541**	**541**
jumpbot (20)	15	**19**	0	0	15	**19**	0	0
plant-watering (51)	0	0	**11**	**11**	0	0	**11**	**11**
rovers (20)	0	1	3	**5**	0	1	3	4
sailing (45)	**12**	**12**	3	3	**12**	**12**	3	3
satellite (20)	1	0	2	**3**	1	0	2	**3**
satellite-hard (20)	9	10	14	**20**	10	10	**20**	**20**
settlers (20)	0	0	0	**1**	0	0	0	**1**
sokoban (325)	31	32	24	24	**59**	**59**	28	26
umt2 (15)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
zenotravel (20)	**11**	8	10	8	**11**	8	10	8
Sum (1 521)	694	698	661	675	**725**	723	670	673
	Sum eager: 2 728				**Sum lazy: 2 791**			

Table 6.4: Coverage of *eager evaluation* (E) compared to *lazy evaluation* (L) with different variants of $h_{\max}$.

Coverage $h_{\mathbf{add}}$	Eh^{ig}_{add}	1Eh^{ig}_{add}	Eh^{rq}_{add}	1Eh^{rq}_{add}	Lh^{ig}_{add}	1Lh^{ig}_{add}	Lh^{rq}_{add}	1Lh^{rq}_{add}
block-grouping (192)	15	15	17	**19**	15	15	17	**19**
counters-0 (34)	**5**	**5**	2	2	4	4	3	3
counters-inv (11)	**4**	**4**	2	2	**4**	**4**	2	2
counters-rnd (33)	**13**	**13**	6	6	**13**	**13**	6	6
depots (22)	7	7	9	7	7	7	**10**	7
driverlog (20)	15	17	16	18	15	17	16	**19**
driverlog-hard (20)	14	16	15	17	14	17	15	**18**
farmland (50)	**18**	17	10	10	17	17	10	10
geo-rovers (21)	**1**	**1**	0	0	**1**	**1**	**1**	**1**
hydraulic-bw (562)	**541**	**541**	**541**	**541**	**541**	**541**	**541**	**541**
jumpbot (20)	18	**19**	0	0	17	**19**	0	0
plant-watering (51)	0	0	**15**	**15**	0	0	**15**	**15**
rovers (20)	0	5	3	**9**	0	6	3	**9**
sailing (45)	**37**	**37**	6	6	36	36	6	5
satellite (20)	2	3	5	11	2	5	4	**13**
satellite-hard (20)	9	15	15	**20**	10	15	**20**	**20**
settlers (20)	0	3	0	**4**	0	3	0	**4**
sokoban (325)	67	67	68	**70**	**70**	**70**	**70**	**70**
umt2 (15)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
zenotravel (20)	15	**20**	15	**20**	16	**20**	16	**20**
Sum (1 521)	781	805	745	777	782	**810**	755	782
	Sum eager: 3 108				**Sum lazy: 3 129**			

Table 6.5: Coverage of *eager evaluation* (E) compared to *lazy evaluation* (L) with different variants of h_{add}.

Coverage $h_{\mathbf{FF}}$	Eh^{ig}_{FF}	1Eh^{ig}_{FF}	Eh^{rq}_{FF}	1Eh^{rq}_{FF}	Lh^{ig}_{FF}	1Lh^{ig}_{FF}	Lh^{rq}_{FF}	1Lh^{rq}_{FF}
block-grouping (192)	17	14	**25**	**25**	14	14	**25**	**25**
counters-0 (34)	**7**	**7**	6	6	**7**	**7**	5	5
counters-inv (11)	3	3	**5**	**5**	3	3	**5**	**5**
counters-rnd (33)	12	10	20	20	10	10	**21**	**21**
depots (22)	10	9	10	9	10	9	**11**	9
driverlog (20)	13	**20**	15	**20**	15	**20**	15	**20**
driverlog-hard (20)	15	19	15	**20**	15	19	15	**20**
farmland (50)	19	19	25	25	18	19	**27**	22
geo-rovers (21)	0	0	4	3	1	1	**5**	**5**
hydraulic-bw (562)	**541**	**541**	**541**	**541**	**541**	**541**	**541**	**541**
jumpbot (20)	13	**17**	8	10	15	**17**	14	10
plant-watering (51)	0	0	**15**	**15**	0	0	**15**	**15**
rovers (20)	1	6	4	**15**	2	2	4	13
sailing (45)	15	15	17	17	14	14	**18**	**18**
satellite (20)	2	4	7	**16**	2	8	8	**16**
satellite-hard (20)	10	15	15	**20**	**20**	**20**	**20**	**20**
settlers (20)	0	5	3	10	0	5	1	**13**
sokoban (325)	57	58	67	68	69	69	**70**	67
umt2 (15)	0	0	0	0	0	0	0	**1**
zenotravel (20)	15	**20**	14	**20**	14	**20**	17	**20**
Sum (1 521)	750	782	816	865	770	798	837	**866**
	Sum eager: 3 213				**Sum lazy: 3 271**			

Table 6.6: Coverage of *eager evaluation* (E) compared to *lazy evaluation* (L) with different variants of h_{FF}.

Quality $h_{\mathbf{max}}$	E$h^{ig}_{\max}$	1E$h^{ig}_{\max}$	E$h^{rq}_{\max}$	1E$h^{rq}_{\max}$	L$h^{ig}_{\max}$	1L$h^{ig}_{\max}$	L$h^{rq}_{\max}$	1L$h^{rq}_{\max}$
block-grouping (192)	**10.79**	**10.79**	8.00	8.00	**10.79**	**10.79**	7.00	7.00
counters-0 (34)	**3.00**	**3.00**	2.00	2.00	**3.00**	**3.00**	2.00	2.00
counters-inv (11)	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**
counters-rnd (33)	7.79	7.79	6.00	6.00	**8.34**	7.63	6.00	6.00
depots (22)	4.40	2.92	**4.42**	2.92	4.18	2.84	4.21	2.84
driverlog (20)	**11.36**	10.36	8.56	10.36	10.72	8.84	7.89	8.84
driverlog-hard (20)	0.05	**0.05**	0.05	**0.05**	0.04	0.04	0.04	0.04
farmland (50)	19.80	**19.84**	10.00	10.00	19.70	19.71	10.00	10.00
geo-rovers (21)	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
hydraulic-bw (562)	**497.55**	**497.55**	**497.55**	**497.55**	494.59	494.59	494.59	494.59
jumpbot (20)	12.64	16.24	0.00	0.00	12.15	**16.35**	0.00	0.00
plant-watering (51)	0.00	0.00	**11.00**	**11.00**	0.00	0.00	**11.00**	**11.00**
rovers (20)	0.00	1.00	3.00	**5.00**	0.00	1.00	3.00	4.00
sailing (45)	**10.07**	**10.07**	3.00	3.00	10.06	10.06	3.00	3.00
satellite (20)	0.39	0.00	1.24	2.15	0.39	0.00	1.38	**2.20**
satellite-hard (20)	9.00	10.00	14.00	**20.00**	10.00	10.00	**20.00**	**20.00**
settlers (20)	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	**1.00**
sokoban (325)	18.73	19.31	22.37	22.28	**38.94**	38.70	26.84	24.93
umt2 (15)	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
zenotravel (20)	**6.04**	5.23	5.24	5.23	4.99	5.25	4.77	5.25
Sum (1 521)	613.61	616.15	598.43	608.54	629.89	**630.82**	603.71	604.70
	Sum eager: 2 436.73				**Sum lazy: 2 469.12**			

Table 6.7: Quality of *eager evaluation* (E) compared to *lazy evaluation* (L) with different variants of $h_{\max}$.

Quality $h_{\mathbf{add}}$	Eh^{ig}_{add}	1Eh^{ig}_{add}	Eh^{rq}_{add}	1Eh^{rq}_{add}	Lh^{ig}_{add}	1Lh^{ig}_{add}	Lh^{rq}_{add}	1Lh^{rq}_{add}
block-grouping (192)	14.83	14.83	13.63	**15.07**	14.71	14.71	13.16	14.38
counters-0 (34)	**4.92**	**4.92**	2.00	2.00	4.00	4.00	2.75	2.75
counters-inv (11)	**3.73**	**3.73**	2.00	2.00	**3.73**	**3.73**	2.00	2.00
counters-rnd (33)	**12.46**	**12.46**	5.17	5.17	**12.46**	**12.46**	4.96	4.96
depots (22)	4.06	3.73	7.90	3.82	4.03	3.21	**8.26**	3.65
driverlog (20)	8.53	11.24	9.67	**12.24**	7.19	9.63	8.28	11.31
driverlog-hard (20)	0.04	1.04	0.04	1.04	0.03	0.57	0.04	**1.25**
farmland (50)	**17.68**	16.68	9.89	9.89	16.68	16.74	9.79	9.79
geo-rovers (21)	**1.00**	**1.00**	0.00	0.00	0.85	0.85	**1.00**	0.99
hydraulic-bw (562)	**476.01**	**476.01**	**476.01**	**476.01**	474.85	474.85	474.85	474.85
jumpbot (20)	15.73	**16.84**	0.00	0.00	14.01	15.96	0.00	0.00
plant-watering (51)	0.00	0.00	**13.06**	13.06	0.00	0.00	11.59	11.59
rovers (20)	0.00	5.00	3.00	**7.00**	0.00	5.00	3.00	6.00
sailing (45)	**35.10**	**35.10**	4.50	4.50	33.61	33.61	4.42	3.52
satellite (20)	0.65	1.11	3.14	7.73	0.48	2.23	2.21	**8.63**
satellite-hard (20)	9.00	15.00	15.00	**20.00**	10.00	15.00	**20.00**	**20.00**
settlers (20)	0.00	2.96	0.00	**3.73**	0.00	2.98	0.00	**3.73**
sokoban (325)	37.27	37.19	57.19	60.95	43.81	44.67	64.43	**64.72**
umt2 (15)	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
zenotravel (20)	7.21	**8.91**	7.11	8.90	5.42	8.63	5.22	8.46
Sum (1 521)	648.23	667.78	629.31	653.11	645.86	**668.85**	635.97	652.58
	Sum eager: 2 598.43				**Sum lazy: 2 603.18**			

Table 6.8: Quality of *eager evaluation* (E) compared to *lazy evaluation* (L) with different variants of h_{add}.

Quality $h_{\mathbf{FF}}$	Eh^{ig}_{FF}	1Eh^{ig}_{FF}	Eh^{rq}_{FF}	1Eh^{rq}_{FF}	Lh^{ig}_{FF}	1Lh^{ig}_{FF}	Lh^{rq}_{FF}	1Lh^{rq}_{FF}
block-grouping (192)	16.32	13.46	23.63	**23.69**	11.93	12.22	21.68	20.97
counters-0 (34)	6.96	**6.99**	5.73	5.70	6.93	6.98	4.52	4.51
counters-inv (11)	2.77	2.96	**4.65**	**4.65**	2.63	2.75	3.70	3.70
counters-rnd (33)	11.46	9.45	**18.45**	**18.45**	8.80	8.88	15.98	15.98
depots (22)	**8.66**	4.47	6.86	4.60	8.35	3.95	6.36	4.11
driverlog (20)	9.04	15.13	10.75	**15.82**	9.02	12.45	10.71	12.24
driverlog-hard (20)	0.05	1.83	0.06	**2.79**	0.04	1.34	0.05	1.96
farmland (50)	14.76	15.17	22.76	23.32	15.73	16.62	**24.84**	21.06
geo-rovers (21)	0.00	0.00	2.95	2.29	0.91	0.92	**3.16**	2.85
hydraulic-bw (562)	483.69	483.69	**487.39**	**487.39**	436.94	436.94	462.47	462.47
jumpbot (20)	9.47	**13.39**	6.58	7.29	10.44	13.23	9.89	6.55
plant-watering (51)	0.00	0.00	7.89	**7.97**	0.00	0.00	5.04	5.02
rovers (20)	1.00	3.00	4.00	**13.64**	2.00	2.00	4.00	11.47
sailing (45)	14.29	14.29	**14.32**	**14.32**	12.92	12.92	12.42	12.42
satellite (20)	0.67	1.72	4.70	10.45	0.41	4.18	8.00	**11.46**
satellite-hard (20)	10.00	15.00	15.00	**20.00**	**20.00**	**20.00**	**20.00**	**20.00**
settlers (20)	0.00	4.92	2.89	9.52	0.00	4.70	0.98	**12.53**
sokoban (325)	35.64	36.13	52.42	50.89	45.57	49.22	**58.27**	55.62
umt2 (15)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**1.00**
zenotravel (20)	12.22	14.39	10.25	11.32	10.64	**16.19**	13.46	11.91
Sum (1 521)	637.00	655.98	701.27	**734.11**	603.25	625.47	685.53	697.84
	Sum eager: 2 728.36				**Sum lazy:** 2 612.09			

Table 6.9: Quality of *eager evaluation* (E) compared to *lazy evaluation* (L) with different variants of h_{FF}.

Summary	eager	lazy
Cost - Sum	**14.800 M**	17.704 M
Coverage - Sum	9 049	**9 191**
Quality - Sum	**7 763.53**	7 684.47

Table 6.10: Summary comparison of *eager evaluation* (E) and *lazy evaluation* (L).

quality metric that weights the coverage with the length of the best known plan for a certain benchmark instance. For each solved instance, a configuration can achieve up to 1 quality point, proportional to the cost of the best known plan. Table 6.7 summarizes this algorithmic *quality* score for all configurations using the $h_{\max}$ heuristic. As the plan length of lazy L$h_{\max}$ configurations increased by 11% for the solved instances, an increase in only 2.3% coverage does not appear to be sufficient in order to catch up in terms of quality score. Surprisingly, L$h_{\max}$ configurations even outperform E$h_{\max}$ configurations in terms of quality, if only by 0.2%. Yet, if the lazy instances that can be solved exceeding beyond eager instances cannot be solved by other planners or if the plans are extraordinarily good, up to 1 quality point can be obtained per extra coverage, whereas a plan that is 11% more expensive can still yield a score of up to 0.9.

A similar phenomenon can be observed for the additive heuristic. The quality scores for all configurations using h_{add} can be found in Table 6.8. Again, we would expect that the plan cost increase of lazy Lh_{add} configurations by 32% cannot be compensated by an increase of 0.7% in coverage. However, the overall quality score indicates that lazy can equalize and even surpass the eager configurations by 0.2%.

A different picture emerges for h_{FF} based heuristics as shown in Table 6.9. While the increase in coverage of lazy instances by 1.8% with a mere increase of 4% in plan cost, and with Lh^{rq}_{FF} even producing better plans in a pairwise comparison, lazy could presumably reach a higher algorithmic quality score as well. However, deferred evaluation does not pay off for the h_{FF} heuristic and eager Eh_{FF} configuration achieve an average quality score that exceeds lazy by 4.5%. This is so much in the favor of eager Eh_{FF} that averaging over all three relaxation heuristics still outvalues lazy configurations by 1.0%.

Summarizing, as also shown in Table 6.10, evaluating heuristics at enqueue time (eager) increases the plan quality, as the search behaves a little bit more like breadth-first search whereas evaluating heuristics lazily allows the algorithm to explore more nodes and reach a larger search depth, resulting in an increase of coverage. Eager evaluation wins with a quality increase of 1.0% and while it suffers from 1.6% coverage decrease, it wins by having 20% better plans.

6.2.2 Unit Cost vs. Regular Cost

In our second set of experiments, we evaluate the impact of using the regular cost function as it appears in the planning task as opposed to using a unit cost transformation where all actions are considered to have a cost of 1 for the purpose of heuristic and search algorithm. Only in the resulting plan, the action costs of regular task are used again. A unit cost setting optimizes the *length* of a shortest plan which can be better if we are interested in coverage.

Again, we start by pairwise comparisons of all configurations that only differ in the action costs that are used in the heuristic and during search. We denote unit cost transformed heuristics by $1h_{\max}$ and omit a cost indicator for regular costs. The sum of all plan costs of instances that both configurations can solve in a pairwise comparison of $h_{\max}$ is given in Table 6.11. It is immediately apparent that both approaches return plans of exactly the same cost for all configurations of several domains. This comes at no surprise, as many domains have unit cost actions domains by default. However, if domains feature non-uniform action costs, the difference is quite significant. Overall, the regular cost tasks return plans that are 17.1% more costly. This comes quite surprisingly as heuristics that are closer to the true cost should be able to find plans of better cost, especially with a greedy search algorithm.

Pairwise comparisons of configurations based on h_{add} are shown in Table 6.12. Again, unit cost transformed tasks produce plans of better quality unless the domain has unit cost actions, this time even by 42.9%. We would have expected the opposite result and unfortunately, we fail to explain this clear advantage of the unit cost transformation.

For the h_{FF} heuristic, the comparisons in Table 6.13 of unit cost transformed tasks to regular tasks are less surprising, and the overall plan length is increased by 11.4% from using the unit cost transformation.

A possible explanation for the anomalies is provided by scatter plots that compare different configurations of h_{FF} comparing unit cost transformed configurations to regular ones. Figure 6.6 compares eager *interval relaxation + planning graph* based Eh^{ig}_{FF} heuristics, with the lazy variant Lh^{ig}_{FF} depicted in Figure 6.7. Additionally, Figure 6.8 shows the comparison of an eager *repetition relaxation + priority queue* based Eh^{rq}_{FF} combination, and the lazy variant Lh^{rq}_{FF} is shown in Figure 6.9. Marks of unit cost domains are usually directly on the diagonal with domains like JUMPBOT that are almost unit cost (actions cost either 0.5 or 1) scattered close to the diagonal as well. Apart from that, we see a large number of marks on the very top or the very right, indicating that plan costs differ heavily or that the coverage of both approaches has complementary strengths and one configuration can solve tasks the other cannot. This opens a vulnerability to outliers that can impact direct comparisons heavily. Still, this is not satisfactory to explain the strength of the unit cost transformation of the h_{add} configurations.

Whereas we did not expect configurations of *unit cost* transformed to yield plans with better cost, an increase in *coverage* comes expected. Table 6.14 compares $h_{\max}$ based heuristics with a regular cost setting (left) to unit cost transformed tasks (right). The average increase in coverage over all instances is with 0.7% rather small and, surprisingly, $\text{L}h^{ig}_{\max}$ achieves the highest coverage among all configurations even though it uses regular costs.

For h_{add}, the results are shown in Table 6.15 and here, the coverage of the best performing configuration, $1\text{L}h^{ig}_{\text{add}}$ uses the unit cost transformation as well. The total coverage increase of 3.6% averaged over all instances is also more pronounced.

Finally, in Table 6.16 we show the coverage results of comparing regular cost tasks to unit cost transformed tasks with h_{FF} based heuristics. Again, unit cost transformed tasks achieve 4.3% more coverage as compared to regular tasks.

With the *unit cost* variants achieving better plan quality (surprisingly) and better coverage (as expected) it comes at no surprise that unit cost transformed

Plan Cost $h_{\max}$	1E$h^{ig}_{\max}$	E$h^{ig}_{\max}$	1L$h^{ig}_{\max}$	L$h^{ig}_{\max}$	1E$h^{rq}_{\max}$	E$h^{rq}_{\max}$	1L$h^{rq}_{\max}$	L$h^{rq}_{\max}$
block-grouping	**259.00**	**259.00**	**259.00**	**259.00**	**76.00**	**76.00**	**66.00**	**66.00**
counters-0	**35.00**	**35.00**	**35.00**	**35.00**	**7.00**	**7.00**	**7.00**	**7.00**
counters-inv	**15.00**	**15.00**	**15.00**	**15.00**	**15.00**	**15.00**	**15.00**	**15.00**
counters-rnd	**101.00**	**101.00**	**104.00**	**104.00**	**27.00**	**27.00**	**27.00**	**27.00**
depots	296.00	**185.00**	306.00	**195.00**	296.00	**184.00**	306.00	**194.00**
driverlog	19 127.00	**18 305.00**	**17 166.00**	17 264.00	13 561.00	**12 953.00**	14 819.00	**14 127.00**
driverlog-hard	**88 980.00**	92 540.00	**76 570.00**	78 290.00	**55 050.00**	57 830.00	**58 300.00**	63 730.00
farmland	**8 825.00**	8 834.00	**8 864.00**	8 876.00	**4 380.00**	**4 380.00**	**4 380.00**	**4 380.00**
geo-rovers	–	–	–	–	–	–	–	–
hydraulic-bw	**7 054.00**	**7 054.00**	**7 110.00**	**7 110.00**	**7 054.00**	**7 054.00**	**7 110.00**	**7 110.00**
jumpbot	**150.00**	152.50	**147.50**	159.00	–	–	–	–
plant-watering	–	–	–	–	**521.00**	**521.00**	**521.00**	**521.00**
rovers	–	–	–	–	**0.00**	**0.00**	**0.00**	**0.00**
sailing	**1 018.00**	**1 018.00**	**1 019.00**	**1 019.00**	**196.00**	**196.00**	**196.00**	**196.00**
satellite	–	–	–	–	282.39	**225.48**	245.42	**183.46**
satellite-hard	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
settlers	–	–	–	–	–	–	–	–
sokoban	**2 771.00**	2 777.00	**9 329.00**	9 330.00	**699.00**	**699.00**	**827.00**	**827.00**
zenotravel	**169 303.00**	175 826.00	**168 391.00**	233 076.00	**169 303.00**	202 758.00	**168 391.00**	240 091.00
Sum	**297 934.00**	307 101.50	**289 315.50**	355 732.00	**251 467.40**	286 925.48	**255 210.42**	331 474.46
		Sum unit cost	**1 093 927.32**		**Sum regular metric cost**		1 281 233.44	

Table 6.11: Plan costs of *unit cost* (1) compared to *regular cost* (no additional indication) with different variants of $h_{\max}$.

Plan Cost $h_{\mathbf{add}}$	1Eh^{ig}_{add}	Eh^{ig}_{add}	1Lh^{ig}_{add}	Lh^{ig}_{add}	1Eh^{rq}_{add}	Eh^{rq}_{add}	1Lh^{rq}_{add}	Lh^{rq}_{add}
block-grouping	**616.00**	**616.00**	**577.00**	**577.00**	**366.00**	369.00	391.00	**389.00**
counters-0	**227.00**	**227.00**	**101.00**	**101.00**	**7.00**	**7.00**	**37.00**	**37.00**
counters-inv	**195.00**	**195.00**	**195.00**	**195.00**	**15.00**	**15.00**	**15.00**	**15.00**
counters-rnd	**556.00**	**556.00**	**556.00**	**556.00**	**39.00**	**39.00**	**45.00**	**45.00**
depots	961.00	**699.00**	1 151.00	**759.00**	1 058.00	**424.00**	1 209.00	**444.00**
driverlog	**29 246.00**	35 318.00	**34 330.00**	43 116.00	**29 246.00**	32 790.00	**33 696.00**	38 215.00
driverlog-hard	176 560.00	**171 940.00**	216 430.00	**205 960.00**	193 400.00	**185 830.00**	230 870.00	**223 990.00**
farmland	6 131.00	**6 130.00**	**6 140.00**	6 150.00	**4 410.00**	**4 410.00**	**4 440.00**	**4 440.00**
geo-rovers	**101.00**	**101.00**	**119.00**	**119.00**	–	–	257.00	**255.00**
hydraulic-bw	**7 852.00**	**7 852.00**	**7 880.00**	**7 880.00**	**7 852.00**	**7 852.00**	**7 880.00**	**7 880.00**
jumpbot	**192.00**	194.00	**185.00**	200.50	–	–	–	–
plant-watering	–	–	–	–	**1 052.00**	**1 052.00**	**1 185.00**	**1 185.00**
rovers	–	–	–	–	**0.00**	**0.00**	**0.00**	**0.00**
sailing	**17 843.00**	**17 843.00**	**17 678.00**	**17 678.00**	**1 391.00**	**1 391.00**	**557.00**	**557.00**
satellite	**429.94**	510.84	**377.18**	574.07	822.97	**803.71**	**968.16**	1 018.53
satellite-hard	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
settlers	–	–	–	–	–	–	–	–
sokoban	13 747.00	**13 661.00**	**14 093.00**	14 441.00	**7 815.00**	8 122.00	**7 610.00**	7 691.00
zenotravel	**1 009 731.00**	1 030 809.00	**1 093 914.00**	2 079 401.00	**989 210.00**	1 068 743.00	**1 149 805.00**	2 360 367.00
Sum	**1 264 387.94**	1 286 651.84	**1 393 726.18**	2 377 707.57	**1 236 683.97**	1 311 847.71	**1 438 965.16**	2 646 528.53
		Sum unit cost	**5 333 763.25**		**Sum regular metric cost**		7 622 735.65	

Table 6.12: Plan costs of *unit cost* (1) compared to *regular cost* (no additional indication) with different variants of h_{add}.

Plan Cost $h_{\mathbf{FF}}$	1Eh^{ig}_{FF}	Eh^{ig}_{FF}	1Lh^{ig}_{FF}	Lh^{ig}_{FF}	1Eh^{rq}_{FF}	Eh^{rq}_{FF}	1Lh^{rq}_{FF}	Lh^{rq}_{FF}
block-grouping	**755.00**	**755.00**	567.00	**545.00**	**984.00**	990.00	**1 073.00**	1 079.00
counters-0	**689.00**	691.00	691.00	**689.00**	453.00	**447.00**	269.00	**263.00**
counters-inv	**65.00**	77.00	**71.00**	79.00	**408.00**	**408.00**	**507.00**	**507.00**
counters-rnd	308.00	**296.00**	364.00	**357.00**	**2 273.00**	**2 273.00**	**3 135.00**	**3 135.00**
depots	1 574.00	**622.00**	1 914.00	**851.00**	1 879.00	**1 045.00**	2 593.00	**1 256.00**
driverlog	**20 409.00**	22 740.00	32 320.00	**32 280.00**	**24 842.00**	28 272.00	33 620.00	**28 005.00**
driverlog-hard	162 040.00	**140 420.00**	209 080.00	**178 140.00**	165 180.00	**143 020.00**	219 540.00	**134 260.00**
farmland	11 833.00	**10 605.00**	**10 936.00**	11 141.00	**9 958.00**	10 295.00	9 075.00	**9 066.00**
geo-rovers	–	–	**110.00**	111.00	**604.00**	680.00	1 662.00	**1 468.00**
hydraulic-bw	**7 630.00**	**7 630.00**	**10 278.00**	**10 278.00**	**7 416.00**	**7 416.00**	**9 022.00**	**9 022.00**
jumpbot	**143.50**	162.00	**193.00**	198.50	75.00	**65.00**	139.00	**135.00**
plant-watering	–	–	–	–	1 945.00	**1 911.00**	3 361.00	**3 325.00**
rovers	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
sailing	**1 096.00**	**1 096.00**	**1 021.00**	**1 021.00**	**2 280.00**	**2 280.00**	**3 336.00**	**3 336.00**
satellite	**610.09**	689.11	**426.16**	633.10	2 044.32	**1 668.53**	1 944.92	**828.39**
satellite-hard	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
settlers	–	–	–	–	**750.00**	768.00	228.00	**212.00**
sokoban	**8 407.00**	9 196.00	**10 292.00**	12 265.00	**8 938.00**	9 128.00	**8 713.00**	9 635.00
zenotravel	678 800.00	**538 424.00**	**409 138.00**	530 895.00	522 777.00	**445 728.00**	894 884.00	**793 102.00**
Sum	894 359.59	**733 403.11**	**687 401.16**	779 483.59	752 806.32	**656 394.53**	1 193 101.92	**998 634.39**
		Sum unit cost	3 527 668.99		**Sum regular metric cost**		**3 167 915.62**	

Table 6.13: Plan costs of *unit cost* (1) compared to *regular cost* (no additional indication) with different variants of h_{FF}.

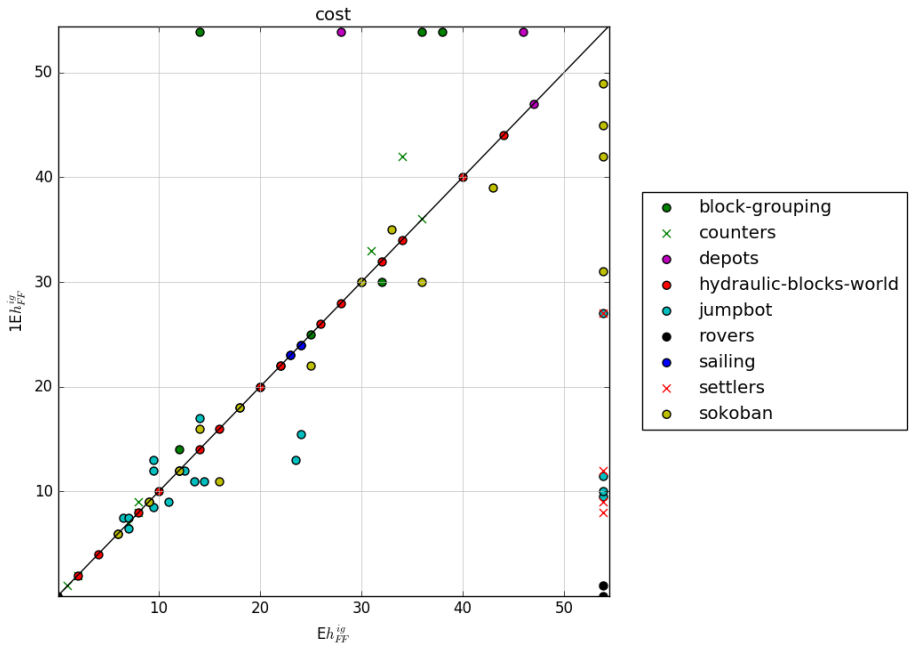

Figure 6.6: Comparison of *unit cost* transformed task to tasks with *regular* cost on an eager *interval relaxation + planning graph* based Eh^{ig}_{FF} setting.

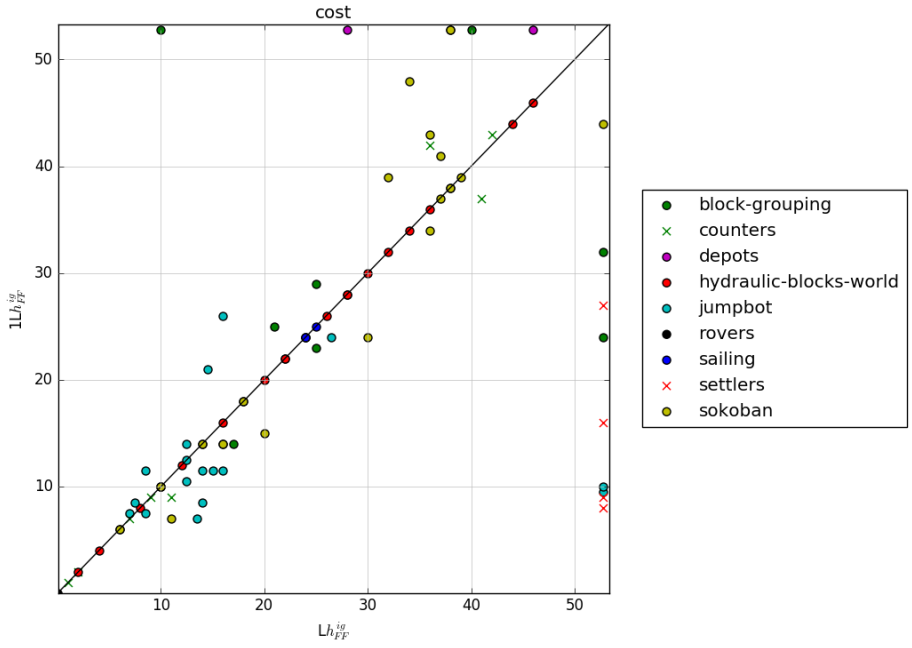

Figure 6.7: Comparison of *unit cost* transformed task to tasks with *regular* cost on an lazy *interval relaxation + planning graph* based Lh^{ig}_{FF} setting.

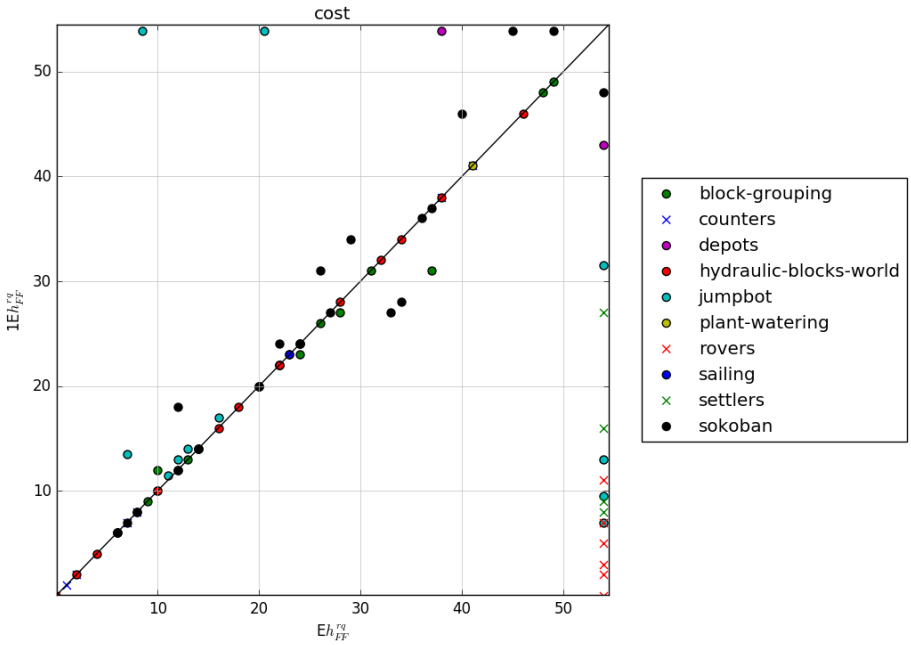

Figure 6.8: Comparison of *unit cost* transformed task to tasks with *regular* cost on an eager *repetition relaxation + priority queue* based Eh^{rq}_{FF} setting.

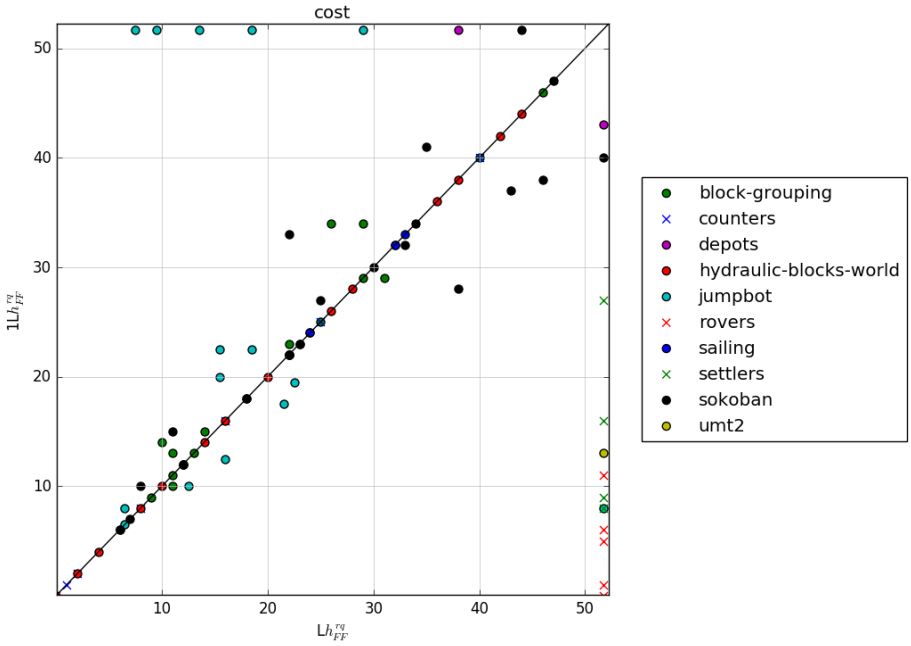

Figure 6.9: Comparison of *unit cost* transformed task to tasks with *regular* cost on an eager *repetition relaxation + priority queue* based Lh^{rq}_{FF} setting.

Coverage $h_{\max}$	E$h^{ig}_{\max}$	E$h^{rq}_{\max}$	L$h^{ig}_{\max}$	L$h^{rq}_{\max}$	1E$h^{ig}_{\max}$	1E$h^{rq}_{\max}$	1L$h^{ig}_{\max}$	1L$h^{rq}_{\max}$
block-grouping (192)	**11**	8	**11**	7	**11**	8	**11**	7
counters-0 (34)	**3**	2	**3**	2	**3**	2	**3**	2
counters-inv (11)	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
counters-rnd (33)	8	6	**9**	6	8	6	8	6
depots (22)	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
driverlog (20)	13	10	**14**	10	13	13	12	12
driverlog-hard (20)	12	10	12	10	**13**	**13**	12	12
farmland (50)	**20**	10	**20**	10	**20**	10	**20**	10
geo-rovers (21)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
hydraulic-bw (562)	**541**	**541**	**541**	**541**	**541**	**541**	**541**	**541**
jumpbot (20)	15	0	15	0	**19**	0	**19**	0
plant-watering (51)	0	**11**	0	**11**	0	**11**	0	**11**
rovers (20)	0	3	0	3	1	**5**	1	4
sailing (45)	**12**	3	**12**	3	**12**	3	**12**	3
satellite (20)	1	2	1	2	0	**3**	0	**3**
satellite-hard (20)	9	14	10	**20**	10	**20**	10	**20**
settlers (20)	0	0	0	0	0	**1**	0	**1**
sokoban (325)	31	24	**59**	28	32	24	**59**	26
umt2 (15)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
zenotravel (20)	**11**	10	**11**	10	8	8	8	8
Sum (1 521)	694	661	**725**	670	698	675	723	673
	Sum regular cost: 2 750				**Sum unit cost: 2 769**			

Table 6.14: Coverage of *unit cost* (1) compared to *regular cost* (no additional indication) with different variants of $h_{\max}$.

Coverage $h_{\mathbf{add}}$	Eh^{ig}_{add}	Eh^{rq}_{add}	Lh^{ig}_{add}	Lh^{rq}_{add}	1Eh^{ig}_{add}	1Eh^{rq}_{add}	1Lh^{ig}_{add}	1Lh^{rq}_{add}
block-grouping (192)	15	17	15	17	15	**19**	15	**19**
counters-0 (34)	**5**	2	4	3	**5**	2	4	3
counters-inv (11)	**4**	2	**4**	2	**4**	2	**4**	2
counters-rnd (33)	**13**	6	**13**	6	**13**	6	**13**	6
depots (22)	7	9	7	**10**	7	7	7	7
driverlog (20)	15	16	15	16	17	18	17	**19**
driverlog-hard (20)	14	15	14	15	16	17	17	**18**
farmland (50)	**18**	10	17	10	17	10	17	10
geo-rovers (21)	**1**	0	**1**	**1**	**1**	0	**1**	**1**
hydraulic-bw (562)	**541**	**541**	**541**	**541**	**541**	**541**	**541**	**541**
jumpbot (20)	18	0	17	0	**19**	0	**19**	0
plant-watering (51)	0	**15**	0	**15**	0	**15**	0	**15**
rovers (20)	0	3	0	3	5	**9**	6	**9**
sailing (45)	**37**	6	36	6	**37**	6	36	5
satellite (20)	2	5	2	4	3	11	5	**13**
satellite-hard (20)	9	15	10	**20**	15	**20**	15	**20**
settlers (20)	0	0	0	0	3	**4**	3	**4**
sokoban (325)	67	68	**70**	**70**	67	**70**	**70**	**70**
umt2 (15)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
zenotravel (20)	15	15	16	16	**20**	**20**	**20**	**20**
Sum (1 521)	781	745	782	755	805	777	**810**	782
	Sum regular cost: 3 063				**Sum unit cost: 3 174**			

Table 6.15: Coverage of *unit cost* (1) compared to *regular cost* (no additional indication) with different variants of h_{add}.

Coverage $h_{\mathbf{FF}}$	Eh^{ig}_{FF}	Eh^{rq}_{FF}	Lh^{ig}_{FF}	Lh^{rq}_{FF}	1Eh^{ig}_{FF}	1Eh^{rq}_{FF}	1Lh^{ig}_{FF}	1Lh^{rq}_{FF}
block-grouping (192)	17	**25**	14	**25**	14	**25**	14	**25**
counters-0 (34)	**7**	6	**7**	5	**7**	6	**7**	5
counters-inv (11)	3	**5**	3	**5**	3	**5**	3	**5**
counters-rnd (33)	12	20	10	**21**	10	20	10	**21**
depots (22)	10	10	10	**11**	9	9	9	9
driverlog (20)	13	15	15	15	**20**	**20**	**20**	**20**
driverlog-hard (20)	15	15	15	15	19	**20**	19	**20**
farmland (50)	19	25	18	**27**	19	25	19	22
geo-rovers (21)	0	4	1	**5**	0	3	1	**5**
hydraulic-bw (562)	**541**	**541**	**541**	**541**	**541**	**541**	**541**	**541**
jumpbot (20)	13	8	15	14	**17**	10	**17**	10
plant-watering (51)	0	**15**	0	**15**	0	**15**	0	**15**
rovers (20)	1	4	2	4	6	**15**	2	13
sailing (45)	15	17	14	**18**	15	17	14	**18**
satellite (20)	2	7	2	8	4	**16**	8	**16**
satellite-hard (20)	10	15	**20**	**20**	15	**20**	**20**	**20**
settlers (20)	0	3	0	1	5	10	5	**13**
sokoban (325)	57	67	69	**70**	58	68	69	67
umt2 (15)	0	0	0	0	0	0	0	**1**
zenotravel (20)	15	14	14	17	**20**	**20**	**20**	**20**
Sum (1 521)	750	816	770	837	782	865	798	**866**
	Sum regular cost: 3 173				**Sum unit cost: 3 311**			

Table 6.16: Coverage of *unit cost* (1) compared to *regular cost* (no additional indication) with different variants of h_{FF}.

Quality $h_{\mathbf{max}}$	Eh^{ig}_{max}	Eh^{rq}_{max}	Lh^{ig}_{max}	Lh^{rq}_{max}	1Eh^{ig}_{max}	1Eh^{rq}_{max}	1Lh^{ig}_{max}	1Lh^{rq}_{max}
block-grouping (192)	**10.79**	8.00	**10.79**	7.00	**10.79**	8.00	**10.79**	7.00
counters-0 (34)	**3.00**	2.00	**3.00**	2.00	**3.00**	2.00	**3.00**	2.00
counters-inv (11)	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**
counters-rnd (33)	7.79	6.00	**8.34**	6.00	7.79	6.00	7.63	6.00
depots (22)	4.40	**4.42**	4.18	4.21	2.92	2.92	2.84	2.84
driverlog (20)	**11.36**	8.56	10.72	7.89	10.36	10.36	8.84	8.84
driverlog-hard (20)	0.05	0.05	0.04	0.04	**0.05**	**0.05**	0.04	0.04
farmland (50)	19.80	10.00	19.70	10.00	**19.84**	10.00	19.71	10.00
geo-rovers (21)	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
hydraulic-bw (562)	**497.55**	**497.55**	494.59	494.59	**497.55**	**497.55**	494.59	494.59
jumpbot (20)	12.64	0.00	12.15	0.00	16.24	0.00	**16.35**	0.00
plant-watering (51)	0.00	**11.00**	0.00	**11.00**	0.00	**11.00**	0.00	**11.00**
rovers (20)	0.00	3.00	0.00	3.00	1.00	**5.00**	1.00	4.00
sailing (45)	**10.07**	3.00	10.06	3.00	**10.07**	3.00	10.06	3.00
satellite (20)	0.39	1.24	0.39	1.38	0.00	2.15	0.00	**2.20**
satellite-hard (20)	9.00	14.00	10.00	**20.00**	10.00	**20.00**	10.00	**20.00**
settlers (20)	0.00	0.00	0.00	0.00	0.00	**1.00**	0.00	**1.00**
sokoban (325)	18.73	22.37	**38.94**	26.84	19.31	22.28	38.70	24.93
umt2 (15)	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
zenotravel (20)	**6.04**	5.24	4.99	4.77	5.23	5.23	5.25	5.25
Sum (1 521)	613.61	598.43	629.89	603.71	616.15	608.54	**630.82**	604.70
	Sum regular cost: 2 445.64				**Sum unit cost: 2 460.21**			

Table 6.17: Quality of *unit cost* (1) compared to *regular cost* (no additional indication) with different variants of h_{max}.

Quality $h_{\mathbf{add}}$	Eh^{ig}_{add}	Eh^{rq}_{add}	Lh^{ig}_{add}	Lh^{rq}_{add}	1Eh^{ig}_{add}	1Eh^{rq}_{add}	1Lh^{ig}_{add}	1Lh^{rq}_{add}
block-grouping (192)	14.83	13.63	14.71	13.16	14.83	**15.07**	14.71	14.38
counters-0 (34)	**4.92**	2.00	4.00	2.75	**4.92**	2.00	4.00	2.75
counters-inv (11)	**3.73**	2.00	**3.73**	2.00	**3.73**	2.00	**3.73**	2.00
counters-rnd (33)	**12.46**	5.17	**12.46**	4.96	**12.46**	5.17	**12.46**	4.96
depots (22)	4.06	7.90	4.03	**8.26**	3.73	3.82	3.21	3.65
driverlog (20)	8.53	9.67	7.19	8.28	11.24	**12.24**	9.63	11.31
driverlog-hard (20)	0.04	0.04	0.03	0.04	1.04	1.04	0.57	**1.25**
farmland (50)	**17.68**	9.89	16.68	9.79	16.68	9.89	16.74	9.79
geo-rovers (21)	**1.00**	0.00	0.85	**1.00**	**1.00**	0.00	0.85	0.99
hydraulic-bw (562)	**476.01**	**476.01**	474.85	474.85	**476.01**	**476.01**	474.85	474.85
jumpbot (20)	15.73	0.00	14.01	0.00	**16.84**	0.00	15.96	0.00
plant-watering (51)	0.00	**13.06**	0.00	11.59	0.00	13.06	0.00	11.59
rovers (20)	0.00	3.00	0.00	3.00	5.00	**7.00**	5.00	6.00
sailing (45)	**35.10**	4.50	33.61	4.42	**35.10**	4.50	33.61	3.52
satellite (20)	0.65	3.14	0.48	2.21	1.11	7.73	2.23	**8.63**
satellite-hard (20)	9.00	15.00	10.00	**20.00**	15.00	**20.00**	15.00	**20.00**
settlers (20)	0.00	0.00	0.00	0.00	2.96	**3.73**	2.98	**3.73**
sokoban (325)	37.27	57.19	43.81	64.43	37.19	60.95	44.67	**64.72**
umt2 (15)	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
zenotravel (20)	7.21	7.11	5.42	5.22	**8.91**	8.90	8.63	8.46
Sum (1 521)	648.23	629.31	645.86	635.97	667.78	653.11	**668.85**	652.58
	Sum regular cost: 2 559.37				**Sum unit cost: 2 642.24**			

Table 6.18: Quality of *unit cost* (1) compared to *regular cost* (no additional indication) with different variants of h_{add}.

Quality $h_{\mathbf{FF}}$	Eh^{ig}_{FF}	Eh^{rq}_{FF}	Lh^{ig}_{FF}	Lh^{rq}_{FF}	1Eh^{ig}_{FF}	1Eh^{rq}_{FF}	1Lh^{ig}_{FF}	1Lh^{rq}_{FF}
block-grouping (192)	16.32	23.63	11.93	21.68	13.46	**23.69**	12.22	20.97
counters-0 (34)	6.96	5.73	6.93	4.52	**6.99**	5.70	6.98	4.51
counters-inv (11)	2.77	**4.65**	2.63	3.70	2.96	**4.65**	2.75	3.70
counters-rnd (33)	11.46	**18.45**	8.80	15.98	9.45	**18.45**	8.88	15.98
depots (22)	**8.66**	6.86	8.35	6.36	4.47	4.60	3.95	4.11
driverlog (20)	9.04	10.75	9.02	10.71	15.13	**15.82**	12.45	12.24
driverlog-hard (20)	0.05	0.06	0.04	0.05	1.83	**2.79**	1.34	1.96
farmland (50)	14.76	22.76	15.73	**24.84**	15.17	23.32	16.62	21.06
geo-rovers (21)	0.00	2.95	0.91	**3.16**	0.00	2.29	0.92	2.85
hydraulic-bw (562)	483.69	**487.39**	436.94	462.47	483.69	**487.39**	436.94	462.47
jumpbot (20)	9.47	6.58	10.44	9.89	**13.39**	7.29	13.23	6.55
plant-watering (51)	0.00	7.89	0.00	5.04	0.00	**7.97**	0.00	5.02
rovers (20)	1.00	4.00	2.00	4.00	3.00	**13.64**	2.00	11.47
sailing (45)	14.29	**14.32**	12.92	12.42	14.29	**14.32**	12.92	12.42
satellite (20)	0.67	4.70	0.41	8.00	1.72	10.45	4.18	**11.46**
satellite-hard (20)	10.00	15.00	**20.00**	**20.00**	15.00	**20.00**	**20.00**	**20.00**
settlers (20)	0.00	2.89	0.00	0.98	4.92	9.52	4.70	**12.53**
sokoban (325)	35.64	52.42	45.57	**58.27**	36.13	50.89	49.22	55.62
umt2 (15)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**1.00**
zenotravel (20)	12.22	10.25	10.64	13.46	14.39	11.32	**16.19**	11.91
Sum (1 521)	637.00	701.27	603.25	685.53	655.98	**734.11**	625.47	697.84
	Sum regular cost: 2 627.05				**Sum unit cost: 2 713.40**			

Table 6.19: Quality of *unit cost* (1) compared to *regular cost* (no additional indication) with different variants of h_{FF}.

Summary	unit cost	regular cost
Cost - Sum	**9.955 M**	12.207 M
Coverage - Sum	**9254**	8986
Quality - Sum	**7815.93**	7632.07

Table 6.20: Comparison summary of *unit cost* and *regular cost*.

configurations achieve a higher algorithmic *quality* score on average. For $h_{\max}$, quality scores are shown in Table 6.17 with an average increase in algorithmic quality of 0.6%. Similarly, Table 6.18 presents the results for h_{add} based configurations where unit cost transformed configurations achieve an average increase in algorithmic quality by 3.2%. The same pattern can be observed for h_{FF} in Table 6.19 where unit cost transformed tasks achieve an increase in quality by 3.3%. Nevertheless, if we look at the best algorithm for each instance, results are wide spread indicating that configurations using action based costs in the heuristics are beneficial, at least for some problem classes. If we compare the gains in coverage for all three heuristics to the gains in quality, we see that quality is mostly dependent on coverage.

Overall, ignoring action costs is beneficial for most instances, as we find shorter plans (in terms of the number of actions in the plan) and surprisingly also of better cost. A summary of comparisons between unit cost transformed configurations to regular costs is shown in Table 6.20.

6.2.3 Interval + Graph vs. Repetition + Queue Approach

In the third set of experiments we investigate the difference in using a *planning graph* based approach to generate the relaxed state progression sequence with an *interval relaxation* (abbreviated interval + graph or ig, cf. Section 4.3.2) and compare it to a *priority queue* based approach in the *repetition relaxation* (abbreviated repetition + queue or rq, cf. Section 4.3.3). We did not include a planning graph based approach in a repetition relaxation, as it only combines the downsides of both options and we did not include a priority queue based approach in an interval relaxation as the computation time is not bounded and the approach usually will not be able to compute heuristic estimates at all.

The interval relaxed planning graph based heuristics are denoted by h^{ig} and the repetition relaxed priority queue based heuristics by h^{rq}. Again we start by comparing the cost of pairwise configurations that only differ in the relaxation approach.

For $h_{\max}$ based configurations, shown in Table 6.21, the differences in plan cost are rather small for all instances for which both configurations that only differ in the relaxation approach find a plan. Whereas $h^{ig}_{\max}$ finds 1.7% shorter plans compared to $h^{rq}_{\max}$ by summing the plan costs of all instances, this increase can only be attributed to the eager E$h_{\max}$ configuration that uses regular costs. For all other configuration, the repetition relaxed priority queue approach produced better plans, although not significantly.

Table 6.22 shows the results for configurations based on h_{add}. Here, the interval based planning graph approaches h^{ig}_{add} produce better plans for all combinations that is also reflected by the overall increase in total plan cost by 4.7%.

A more differentiated picture emerges for h_{FF} based comparisons in Ta-

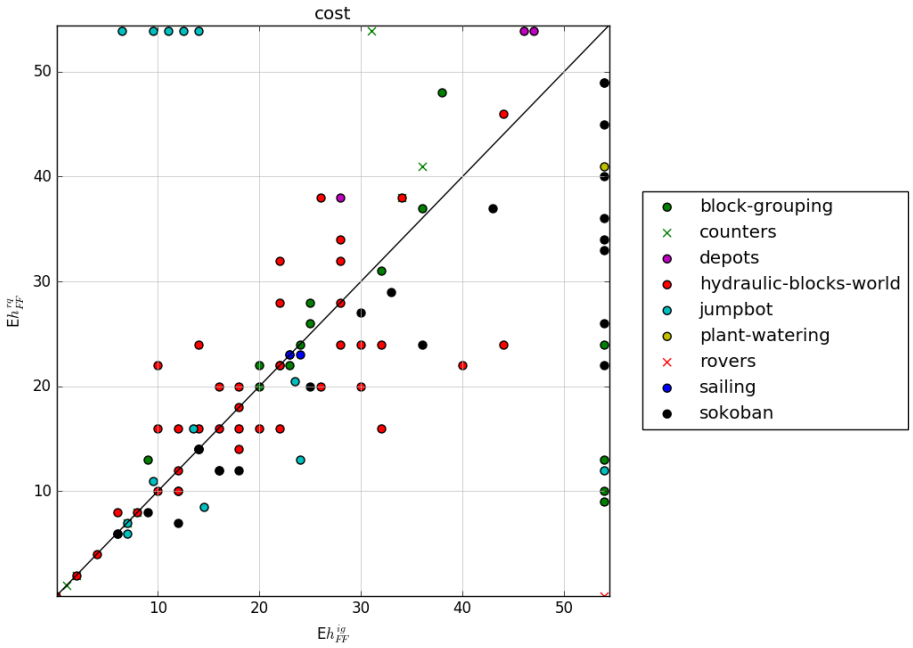

Figure 6.10: Comparison of eager $\mathrm{E}h^{ig}_{\mathrm{FF}}$ to $\mathrm{E}h^{rq}_{\mathrm{FF}}$ in a original cost setting.

ble 6.23. Whereas the total plan cost speaks for configurations based on the interval relaxed planing graph approach h^{ig}_{FF} again, with h^{rq}_{FF} reporting 29.4% more costly plans overall, there is one combination: lazy evaluation in a regular cost setting $\mathrm{L}h_{\mathrm{FF}}$ where the plan cost of $\mathrm{L}h^{rq}_{\mathrm{FF}}$ finds plans that are almost 10% shorter. If we investigate the domains more closely, we can see that much of this cost sum is dominated by ZenoTravel and excluding this domain results in a much more balanced picture.

Again we investigate the consistence of the aggregated results in the tables with scatter plots. We compare the relaxation approaches of eager $\mathrm{E}h_{\mathrm{FF}}$ heuristics in Figure 6.10 and the unit cost variants $1\mathrm{E}h_{\mathrm{FF}}$ in Figure 6.11. The lazy heuristics $\mathrm{L}h_{\mathrm{FF}}$ are compared in Figure 6.12 and the unit cost variants in Figure 6.13. With the exception of the Jumpbot domain, there tend to be more marks below the diagonal indicating that the repetition relaxed priority queue approach often performs better on the smaller instances with a plan cost of up to 50. Concluding, the relaxation approach has an impact on the expected plan cost, however, there is no clear winner over all domains and depending on the domain, one approach or the other can produce plans of lower costs.

The *coverage* of the different configurations using the $h_{\max}$ heuristic is shown in Table 6.24. The configurations using an interval relaxed planing graph approach to generate the relaxed state progression, $h^{ig}_{\max}$, are found at the left and heuristics using a repetition relaxed priority queue approach, $h^{qr}_{\max}$ are at the right. Aggregated over all configurations, the coverage of $h^{ig}_{\max}$ increases by 6.0%. However, this increase in coverage can mostly be attributed to three domains: Farmland, Sailing and Sokoban, with $h^{rq}_{\max}$-variants achieving a higher coverage in PlantWatering.

Plan Cost $h_{\max}$	1E$h^{ig}_{\max}$	1E$h^{rq}_{\max}$	E$h^{ig}_{\max}$	E$h^{rq}_{\max}$	1L$h^{ig}_{\max}$	1L$h^{rq}_{\max}$	L$h^{ig}_{\max}$	L$h^{rq}_{\max}$
block-grouping	**19.00**	**19.00**	**19.00**	**19.00**	**19.00**	**19.00**	**19.00**	**19.00**
counters-0	**7.00**	**7.00**	**7.00**	**7.00**	**7.00**	**7.00**	**7.00**	**7.00**
counters-inv	**15.00**	**15.00**	**15.00**	**15.00**	**15.00**	**15.00**	**15.00**	**15.00**
counters-rnd	**27.00**	**27.00**	**27.00**	**27.00**	28.00	**27.00**	28.00	**27.00**
depots	**296.00**	**296.00**	185.00	**184.00**	**306.00**	**306.00**	195.00	**194.00**
driverlog	**19 127.00**	**19 127.00**	**11 570.00**	12 953.00	**17 166.00**	**17 166.00**	**13 081.00**	14 127.00
driverlog-hard	**105 100.00**	**105 100.00**	60 930.00	**57 830.00**	**94 290.00**	**94 290.00**	66 260.00	**63 730.00**
farmland	4 410.00	**4 380.00**	4 410.00	**4 380.00**	4 440.00	**4 380.00**	4 440.00	**4 380.00**
geo-rovers	–	–	–	–	–	–	–	–
hydraulic-bw	**7 054.00**	**7 054.00**	**7 054.00**	**7 054.00**	**7 110.00**	**7 110.00**	**7 110.00**	**7 110.00**
jumpbot	–	–	–	–	–	–	–	–
rovers	**0.00**	**0.00**	–	–	**0.00**	**0.00**	–	–
sailing	275.00	**196.00**	275.00	**196.00**	275.00	**196.00**	275.00	**196.00**
satellite	–	–	**73.58**	115.60	–	–	**73.58**	**73.58**
satellite-hard	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
settlers	–	–	–	–	–	–	–	–
sokoban	1 021.00	**629.00**	1 130.00	**681.00**	1 273.00	**827.00**	1 465.00	**946.00**
zenotravel	**169 303.00**	**169 303.00**	**278 129.00**	314 169.00	**168 391.00**	**168 391.00**	345 394.00	**337 988.00**
Sum	306 654.00	**306 153.00**	**363 824.58**	397 630.60	293 320.00	**292 734.00**	438 362.58	**428 812.58**
		Sum $h^{ig}_{\max}$	**1 402 161.16**			**Sum $h^{rq}_{\max}$**	1 425 330.18	

Table 6.21: Plan costs of *interval relaxation + planning graph* ($h^{ig}_{\max}$) compared to *repetition relaxation + priority queue* ($h^{rq}_{\max}$).

Plan Cost $h_{\mathbf{add}}$	1Eh^{ig}_{add}	1Eh^{rq}_{add}	Eh^{ig}_{add}	Eh^{rq}_{add}	1Lh^{ig}_{add}	1Lh^{rq}_{add}	Lh^{ig}_{add}	Lh^{rq}_{add}
block-grouping	**215.00**	290.00	**190.00**	253.00	**225.00**	323.00	**200.00**	283.00
counters-0	**7.00**	**7.00**	**7.00**	**7.00**	**35.00**	37.00	**35.00**	37.00
counters-inv	**15.00**	**15.00**	**15.00**	**15.00**	**15.00**	**15.00**	**15.00**	**15.00**
counters-rnd	**27.00**	39.00	**27.00**	39.00	**27.00**	45.00	**27.00**	45.00
depots	**961.00**	1 058.00	699.00	**424.00**	**1 151.00**	1 209.00	759.00	**444.00**
driverlog	54 736.00	**53 650.00**	35 318.00	**32 790.00**	67 665.00	**66 695.00**	43 116.00	**38 215.00**
driverlog-hard	**254 340.00**	**254 340.00**	171 940.00	**169 890.00**	463 370.00	**453 010.00**	205 960.00	**201 790.00**
farmland	**4 410.00**	**4 410.00**	**4 410.00**	**4 410.00**	**4 440.00**	**4 440.00**	**4 440.00**	**4 440.00**
geo-rovers	–	–	–	–	–	–	–	–
hydraulic-bw	**7 852.00**	**7 852.00**	**7 852.00**	**7 852.00**	**7 880.00**	**7 880.00**	**7 880.00**	**7 880.00**
jumpbot	–	–	–	–	–	–	–	–
rovers	**0.00**	**0.00**	–	–	**1.00**	**1.00**	–	–
sailing	3 105.00	**1 391.00**	3 105.00	**1 391.00**	**490.00**	557.00	3 207.00	**1 404.00**
satellite	**662.32**	668.11	73.58	**72.52**	**1 088.73**	1 331.21	**126.11**	142.17
satellite-hard	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
settlers	150.00	**148.00**	–	–	**148.00**	**148.00**	–	–
sokoban	13 747.00	**7 486.00**	13 258.00	**7 572.00**	14 093.00	**7 610.00**	14 441.00	**7 691.00**
zenotravel	**2 131 002.00**	2 193 692.00	**1 030 809.00**	1 068 743.00	**2 283 143.00**	2 381 530.00	**2 079 401.00**	2 360 367.00
Sum	**2 471 229.32**	2 525 046.11	**1 267 703.58**	1 293 458.52	**2 843 771.73**	2 924 831.21	**2 359 607.11**	2 622 753.17
		Sum $h^{ig}_{\mathbf{add}}$	**8 942 311.74**			**Sum** $h^{rq}_{\mathbf{add}}$	9 366 089.01	

Table 6.22: Plan costs of *interval relaxation + planning graph* (h^{ig}_{add}) compared to *repetition relaxation + priority queue* (h^{rq}_{add}).

Plan Cost $h_{\mathbf{FF}}$	1Eh^{ig}_{FF}	1Eh^{rq}_{FF}	Eh^{ig}_{FF}	Eh^{rq}_{FF}	1Lh^{ig}_{FF}	1Lh^{rq}_{FF}	Lh^{ig}_{FF}	Lh^{rq}_{FF}
block-grouping	**755.00**	759.00	**843.00**	857.00	829.00	**790.00**	**633.00**	653.00
counters-0	**411.00**	453.00	**415.00**	447.00	**221.00**	269.00	**223.00**	263.00
counters-inv	**65.00**	75.00	77.00	**75.00**	**71.00**	97.00	**79.00**	97.00
counters-rnd	**308.00**	320.00	**530.00**	534.00	**364.00**	420.00	**357.00**	420.00
depots	1 892.00	**1 879.00**	**797.00**	986.00	**1 914.00**	2 593.00	**1 016.00**	1 524.00
driverlog	89 387.00	**85 239.00**	22 740.00	**22 097.00**	**109 500.00**	133 494.00	32 280.00	**28 005.00**
driverlog-hard	**442 540.00**	443 540.00	**140 420.00**	143 020.00	**600 140.00**	783 770.00	178 140.00	**134 260.00**
farmland	12 107.00	**7 545.00**	14 807.00	**8 764.00**	11 855.00	**8 377.00**	11 631.00	**7 957.00**
geo-rovers	–	–	–	–	**110.00**	143.00	**111.00**	120.00
hydraulic-bw	7 630.00	**7 416.00**	7 630.00	**7 416.00**	10 278.00	**9 022.00**	10 278.00	**9 022.00**
jumpbot	**111.00**	136.00	99.00	**82.00**	**128.00**	147.00	**162.50**	185.00
rovers	3.00	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
sailing	**922.00**	1 130.00	**922.00**	1 130.00	**844.00**	1 369.00	**844.00**	1 369.00
satellite	**1 148.99**	1 366.39	371.98	**370.86**	**2 857.35**	2 963.55	633.10	**142.07**
satellite-hard	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
settlers	**176.00**	178.00	–	–	**176.00**	178.00	–	–
sokoban	8 688.00	**6 856.00**	9 757.00	**7 060.00**	9 915.00	**8 392.00**	12 632.00	**10 062.00**
zenotravel	**1 228 750.00**	1 887 250.00	**388 212.00**	445 728.00	**1 059 713.00**	1 693 988.00	530 895.00	**508 688.00**
Sum	**1 794 893.99**	2 444 142.39	**587 620.98**	638 566.86	**1 808 915.35**	2 646 012.55	779 914.59	**702 767.07**
		Sum $h^{ig}_{\mathbf{FF}}$	**4 971 344.91**			**Sum $h^{rq}_{\mathbf{FF}}$**	6 431 488.87	

Table 6.23: Plan costs of *interval relaxation + planning graph* (h^{ig}_{FF}) compared to *repetition relaxation + priority queue* (h^{rq}_{FF}).

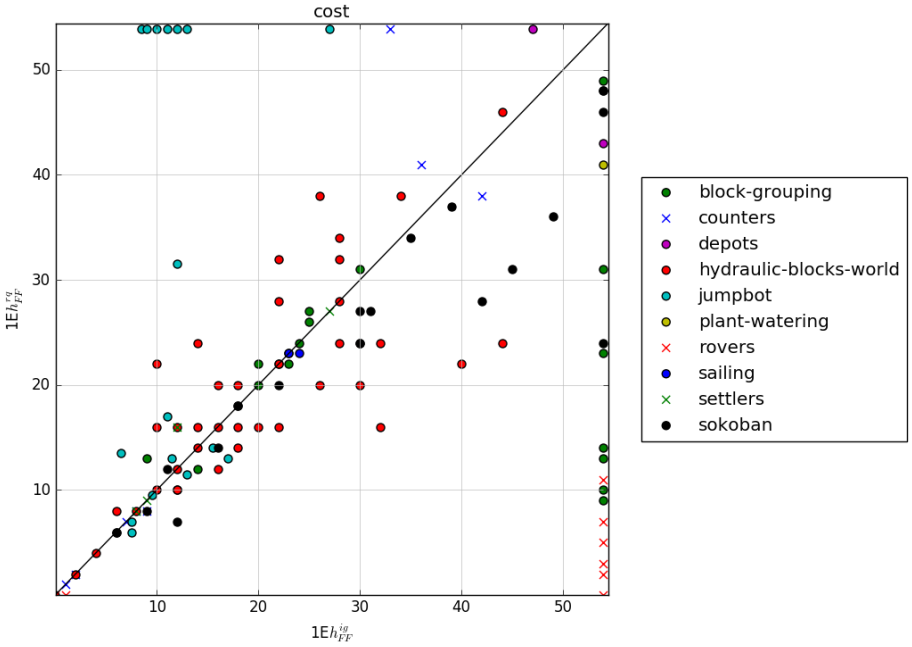

Figure 6.11: Comparison of eager $1\mathrm{E}h^{ig}_{\mathrm{FF}}$ to $1\mathrm{E}h^{rq}_{\mathrm{FF}}$ in a unit cost setting.

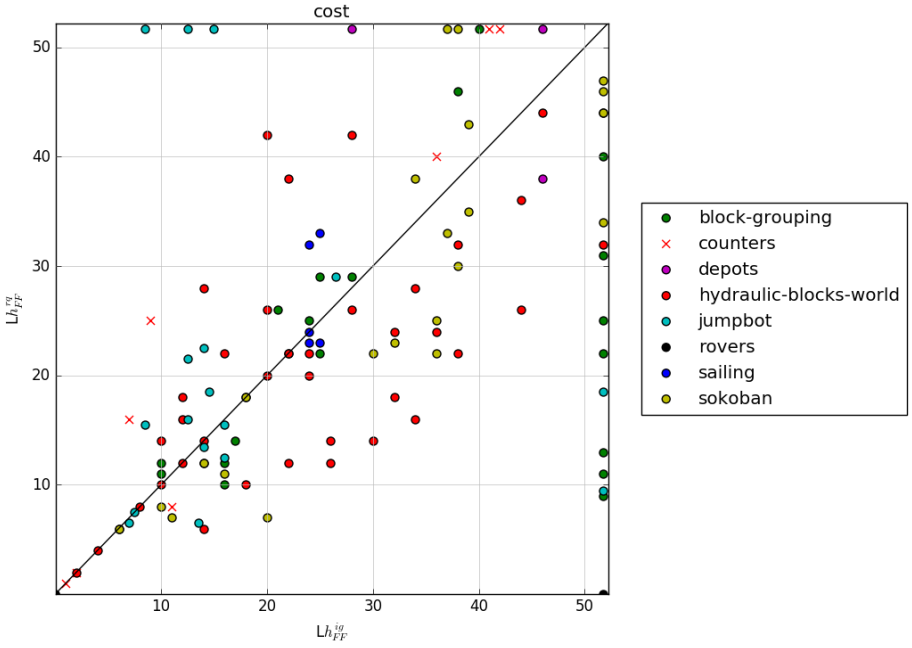

Figure 6.12: Comparison of eager $\mathrm{L}h^{ig}_{\mathrm{FF}}$ to $\mathrm{L}h^{rq}_{\mathrm{FF}}$ in a original cost setting.

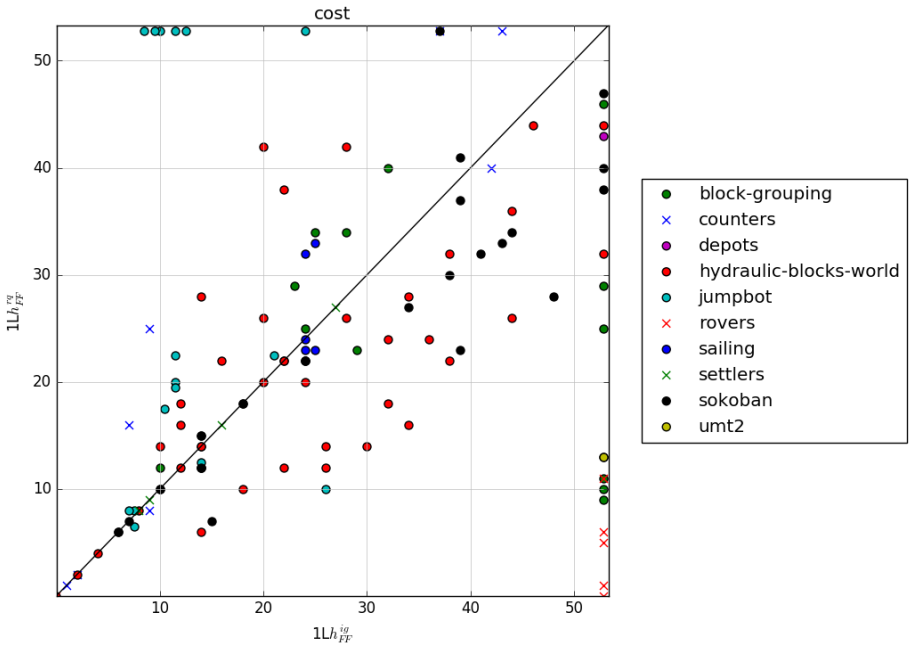

Figure 6.13: Comparison of eager $1\mathrm{L}h^{ig}_{\mathrm{FF}}$ to $1\mathrm{L}h^{rq}_{\mathrm{FF}}$ in a unit cost setting.

The coverage of h_{add} based configurations can be found in Table 6.25. Again, h^{ig}_{add} achieves an increase in coverage by 3.9% and again, this increase is not uniform but very domain dependent, with COUNTERS, FARMLAND, JUMPBOT and SAILING producing better coverage with h^{ig}_{add} and h^{rq}_{add} achieving higher coverage in PLANTWATERING and SATELLITE.

In Table 6.26 we compare the interval relaxed planing graph based h^{ig}_{FF} configurations to those using the repetition relaxed priority queue approach based approach h^{rq}_{FF}. With few exceptions (e.g. the JUMPBOT domain), using the repetition relaxation pays off and h^{rq}_{FF} increases coverage by 9.2%. This sudden advantage of the repetition relaxed approaches does not come surprisingly. During the progression phase, the repetition relaxation attributes the cost of one repetition to an interval that can be reached by applying the action arbitrarily often. For h^{rq}_{max} and h^{rq}_{add}, this repetitive use of actions is not compensated for. In contrast, h^{rq}_{FF} extracts an interval relaxed plan and explicates the required number of repetitions, compensating the biggest disadvantage of the repetition relaxed approach over the interval relaxation.

The *quality* score that relates coverage to plan cost highlights the importance of evening out the aggregation of several action to one step in the repetition relaxation again. The quality of h_{max}, shown in Table 6.27, is better for the interval relaxed planning graph approaches h^{ig}_{max}, with an average increase of coverage by 3.1%. Similarly, for h_{add}, quality increases by 2.3% if we use h^{ig}_{add}, as seen in Table 6.28. Quite in contrast are the quality results for h_{FF} which are shown in Table 6.29. With the extraction of an interval relaxed plan, h^{rq}_{FF} has the advantage of being able to account for the repeated application of action without impairing the estimated cost. On average, the quality of h^{rq}_{FF} based

Coverage $h_{\max}$	E$h^{ig}_{\max}$	1E$h^{ig}_{\max}$	L$h^{ig}_{\max}$	1L$h^{ig}_{\max}$	E$h^{rq}_{\max}$	1E$h^{rq}_{\max}$	L$h^{rq}_{\max}$	1L$h^{rq}_{\max}$
block-grouping (192)	**11**	**11**	**11**	**11**	8	8	7	7
counters-0 (34)	**3**	**3**	**3**	**3**	2	2	2	2
counters-inv (11)	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
counters-rnd (33)	8	8	**9**	8	6	6	6	6
depots (22)	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
driverlog (20)	13	13	**14**	12	10	13	10	12
driverlog-hard (20)	12	**13**	12	12	10	**13**	10	12
farmland (30)	**20**	**20**	**20**	**20**	10	10	10	10
geo-rovers (21)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
hydraulic-bw (562)	**541**	**541**	**541**	**541**	**541**	**541**	**541**	**541**
jumpbot (20)	15	**19**	15	**19**	0	0	0	0
plant-watering (51)	0	0	0	0	**11**	**11**	**11**	**11**
rovers (20)	0	1	0	1	3	**5**	3	4
sailing (45)	**12**	**12**	**12**	**12**	3	3	3	3
satellite (20)	1	0	1	0	2	**3**	2	**3**
satellite-hard (20)	9	10	10	10	14	**20**	**20**	**20**
settlers (20)	0	0	0	0	0	**1**	0	**1**
sokoban (325)	31	32	**59**	**59**	24	24	28	26
umt2 (15)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
zenotravel (20)	**11**	8	**11**	8	10	8	10	8
Sum (1 521)	694	698	**725**	723	661	675	670	673
	Sum $h^{ig}_{\max}$: 2 840				**Sum $h^{rq}_{\max}$:** 2 679			

Table 6.24: Coverage of *interval relaxation + planning graph* ($h^{ig}_{\max}$) compared to *repetition relaxation + priority queue* ($h^{rq}_{\max}$).

Coverage $h_{\mathbf{add}}$	Eh^{ig}_{add}	1Eh^{ig}_{add}	Lh^{ig}_{add}	1Lh^{ig}_{add}	Eh^{rq}_{add}	1Eh^{rq}_{add}	Lh^{rq}_{add}	1Lh^{rq}_{add}
block-grouping (192)	15	15	15	15	17	**19**	17	**19**
counters-0 (34)	**5**	**5**	4	4	2	2	3	3
counters-inv (11)	**4**	**4**	**4**	**4**	2	2	2	2
counters-rnd (33)	**13**	**13**	**13**	**13**	6	6	6	6
depots (22)	7	7	7	7	9	7	**10**	7
driverlog (20)	15	17	15	17	16	18	16	**19**
driverlog-hard (20)	14	16	14	17	15	17	15	**18**
farmland (50)	**18**	17	17	17	10	10	10	10
geo-rovers (21)	**1**	**1**	**1**	**1**	0	0	**1**	**1**
hydraulic-bw (562)	**541**	**541**	**541**	**541**	**541**	**541**	**541**	**541**
jumpbot (20)	18	**19**	17	**19**	0	0	0	0
plant-watering (51)	0	0	0	0	**15**	**15**	**15**	**15**
rovers (20)	0	5	0	6	3	**9**	3	**9**
sailing (45)	**37**	**37**	36	36	6	6	6	5
satellite (20)	2	3	2	5	5	11	4	**13**
satellite-hard (20)	9	15	10	15	15	**20**	**20**	**20**
settlers (20)	0	3	0	3	0	**4**	0	**4**
sokoban (325)	67	67	**70**	**70**	68	**70**	**70**	**70**
umt2 (15)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
zenotravel (20)	15	**20**	16	**20**	15	**20**	16	**20**
Sum (1 521)	781	805	782	**810**	745	777	755	782
	Sum h^{ig}_{add}: 3 178				**Sum h^{rq}_{add}:** 3 059			

Table 6.25: Coverage of *interval relaxation + planning graph* (h^{ig}_{add}) compared to *repetition relaxation + priority queue* (h^{rq}_{add}).

Coverage $h_{\mathbf{FF}}$	Eh^{ig}_{FF}	1Eh^{ig}_{FF}	Lh^{ig}_{FF}	1Lh^{ig}_{FF}	Eh^{rq}_{FF}	1Eh^{rq}_{FF}	Lh^{rq}_{FF}	1Lh^{rq}_{FF}
block-grouping (192)	17	14	14	14	**25**	**25**	**25**	**25**
counters-0 (34)	**7**	**7**	**7**	**7**	6	6	5	5
counters-inv (11)	3	3	3	3	**5**	**5**	**5**	**5**
counters-rnd (33)	12	10	10	10	20	20	**21**	**21**
depots (22)	10	9	10	9	10	9	**11**	9
driverlog (20)	13	**20**	15	**20**	15	**20**	15	**20**
driverlog-hard (20)	15	19	15	19	15	**20**	15	**20**
farmland (50)	19	19	18	19	25	25	**27**	22
geo-rovers (21)	0	0	1	1	4	3	**5**	**5**
hydraulic-bw (562)	**541**	**541**	**541**	**541**	**541**	**541**	**541**	**541**
jumpbot (20)	13	**17**	15	**17**	8	10	14	10
plant-watering (51)	0	0	0	0	**15**	**15**	**15**	**15**
rovers (20)	1	6	2	2	4	**15**	4	13
sailing (45)	15	15	14	14	17	17	**18**	**18**
satellite (20)	2	4	2	8	7	**16**	8	**16**
satellite-hard (20)	10	15	**20**	**20**	15	**20**	**20**	**20**
settlers (20)	0	5	0	5	3	10	1	**13**
sokoban (325)	57	58	69	69	67	68	**70**	67
umt2 (15)	0	0	0	0	0	0	0	**1**
zenotravel (20)	15	**20**	14	**20**	14	**20**	17	**20**
Sum (1 521)	750	782	770	798	816	865	837	**866**
		Sum $h^{ig}_{\mathbf{FF}}$: 3 100				**Sum $h^{rq}_{\mathbf{FF}}$: 3 384**		

Table 6.26: Coverage of *interval relaxation + planning graph* (h^{ig}_{FF}) compared to *repetition relaxation + priority queue* (h^{rq}_{FF}).

Quality $h_{\mathbf{max}}$	Eh^{ig}_{max}	1Eh^{ig}_{max}	Lh^{ig}_{max}	1Lh^{ig}_{max}	Eh^{rq}_{max}	1Eh^{rq}_{max}	Lh^{rq}_{max}	1Lh^{rq}_{max}
block-grouping (192)	**10.79**	**10.79**	**10.79**	**10.79**	8.00	8.00	7.00	7.00
counters-0 (34)	**3.00**	**3.00**	**3.00**	**3.00**	2.00	2.00	2.00	2.00
counters-inv (11)	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**
counters-rnd (33)	7.79	7.79	**8.34**	7.63	6.00	6.00	6.00	6.00
depots (22)	4.40	2.92	4.18	2.84	**4.42**	2.92	4.21	2.84
driverlog (20)	**11.36**	10.36	10.72	8.84	8.56	10.36	7.89	8.84
driverlog-hard (20)	0.05	**0.05**	0.04	0.04	0.05	**0.05**	0.04	0.04
farmland (50)	19.80	**19.84**	19.70	19.71	10.00	10.00	10.00	10.00
geo-rovers (21)	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
hydraulic-bw (562)	**497.55**	**497.55**	494.59	494.59	**497.55**	**497.55**	494.59	494.59
jumpbot (20)	12.64	16.24	12.15	**16.35**	0.00	0.00	0.00	0.00
plant-watering (51)	0.00	0.00	0.00	0.00	**11.00**	**11.00**	**11.00**	**11.00**
rovers (20)	0.00	1.00	0.00	1.00	3.00	**5.00**	3.00	4.00
sailing (45)	**10.07**	**10.07**	10.06	10.06	3.00	3.00	3.00	3.00
satellite (20)	0.39	0.00	0.39	0.00	1.24	2.15	1.38	**2.20**
satellite-hard (20)	9.00	10.00	10.00	10.00	14.00	**20.00**	**20.00**	**20.00**
settlers (20)	0.00	0.00	0.00	0.00	0.00	**1.00**	0.00	**1.00**
sokoban (325)	18.73	19.31	**38.94**	38.70	22.37	22.28	26.84	24.93
umt2 (15)	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
zenotravel (20)	**6.04**	5.23	4.99	5.25	5.24	5.23	4.77	5.25
Sum (1 521)	613.61	616.15	629.89	**630.82**	598.43	608.54	603.71	604.70
		Sum $h^{ig}_{\mathbf{max}}$: 2 490.47				**Sum $h^{rq}_{\mathbf{max}}$:** 2 415.38		

Table 6.27: Quality of *interval relaxation + planning graph* (h^{ig}_{max}) compared to *repetition relaxation + priority queue* (h^{rq}_{max}).

Quality $h_{\mathbf{add}}$	Eh^{ig}_{add}	1Eh^{ig}_{add}	Lh^{ig}_{add}	1Lh^{ig}_{add}	Eh^{rq}_{add}	1Eh^{rq}_{add}	Lh^{rq}_{add}	1Lh^{rq}_{add}
block-grouping (192)	14.83	14.83	14.71	14.71	13.63	**15.07**	13.16	14.38
counters-0 (34)	**4.92**	**4.92**	4.00	4.00	2.00	2.00	2.75	2.75
counters-inv (11)	**3.73**	**3.73**	**3.73**	**3.73**	2.00	2.00	2.00	2.00
counters-rnd (33)	**12.46**	**12.46**	**12.46**	**12.46**	5.17	5.17	4.96	4.96
depots (22)	4.06	3.73	4.03	3.21	7.90	3.82	**8.26**	3.65
driverlog (20)	8.53	11.24	7.19	9.63	9.67	**12.24**	8.28	11.31
driverlog-hard (20)	0.04	1.04	0.03	0.57	0.04	1.04	0.04	**1.25**
farmland (50)	**17.68**	16.68	16.68	16.74	9.89	9.89	9.79	9.79
geo-rovers (21)	**1.00**	**1.00**	0.85	0.85	0.00	0.00	**1.00**	0.99
hydraulic-bw (562)	**476.01**	**476.01**	474.85	474.85	**476.01**	**476.01**	474.85	474.85
jumpbot (20)	15.73	**16.84**	14.01	15.96	0.00	0.00	0.00	0.00
plant-watering (51)	0.00	0.00	0.00	0.00	**13.06**	13.06	11.59	11.59
rovers (20)	0.00	5.00	0.00	5.00	3.00	**7.00**	3.00	6.00
sailing (45)	**35.10**	**35.10**	33.61	33.61	4.50	4.50	4.42	3.52
satellite (20)	0.65	1.11	0.48	2.23	3.14	7.73	2.21	**8.63**
satellite-hard (20)	9.00	15.00	10.00	15.00	15.00	**20.00**	**20.00**	**20.00**
settlers (20)	0.00	2.96	0.00	2.98	0.00	**3.73**	0.00	**3.73**
sokoban (325)	37.27	37.19	43.81	44.67	57.19	60.95	64.43	**64.72**
umt2 (15)	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
zenotravel (20)	7.21	**8.91**	5.42	8.63	7.11	8.90	5.22	8.46
Sum (1 521)	648.23	667.78	645.86	**668.85**	629.31	653.11	635.97	652.58
	Sum $h^{ig}_{\mathbf{add}}$: 2 630.72				**Sum $h^{rq}_{\mathbf{add}}$:** 2 570.97			

Table 6.28: Quality of *interval relaxation + planning graph* (h^{ig}_{add}) compared to *repetition relaxation + priority queue* (h^{rq}_{add}).

Quality $h_{\mathbf{FF}}$	Eh^{ig}_{FF}	1Eh^{ig}_{FF}	Lh^{ig}_{FF}	1Lh^{ig}_{FF}	Eh^{rq}_{FF}	1Eh^{rq}_{FF}	Lh^{rq}_{FF}	1Lh^{rq}_{FF}
block-grouping (192)	16.32	13.46	11.93	12.22	23.63	**23.69**	21.68	20.97
counters-0 (34)	6.96	**6.99**	6.93	6.98	5.73	5.70	4.52	4.51
counters-inv (11)	2.77	2.96	2.63	2.75	**4.65**	**4.65**	3.70	3.70
counters-rnd (33)	11.46	9.45	8.80	8.88	**18.45**	**18.45**	15.98	15.98
depots (22)	**8.66**	4.47	8.35	3.95	6.86	4.60	6.36	4.11
driverlog (20)	9.04	15.13	9.02	12.45	10.75	**15.82**	10.71	12.24
driverlog-hard (20)	0.05	1.83	0.04	1.34	0.06	**2.79**	0.05	1.96
farmland (50)	14.76	15.17	15.73	16.62	22.76	23.32	**24.84**	21.06
geo-rovers (21)	0.00	0.00	0.91	0.92	2.95	2.29	**3.16**	2.85
hydraulic-bw (562)	483.69	483.69	436.94	436.94	**487.39**	**487.39**	462.47	462.47
jumpbot (20)	9.47	**13.39**	10.44	13.23	6.58	7.29	9.89	6.55
plant-watering (51)	0.00	0.00	0.00	0.00	7.89	**7.97**	5.04	5.02
rovers (20)	1.00	3.00	2.00	2.00	4.00	**13.64**	4.00	11.47
sailing (45)	14.29	14.29	12.92	12.92	**14.32**	**14.32**	12.42	12.42
satellite (20)	0.67	1.72	0.41	4.18	4.70	10.45	8.00	**11.46**
satellite-hard (20)	10.00	15.00	**20.00**	**20.00**	15.00	**20.00**	**20.00**	**20.00**
settlers (20)	0.00	4.92	0.00	4.70	2.89	9.52	0.98	**12.53**
sokoban (325)	35.64	36.13	45.57	49.22	52.42	50.89	**58.27**	55.62
umt2 (15)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**1.00**
zenotravel (20)	12.22	14.39	10.64	**16.19**	10.25	11.32	13.46	11.91
Sum (1 521)	637.00	655.98	603.25	625.47	701.27	**734.11**	685.53	697.84
	Sum $h^{ig}_{\mathbf{FF}}$: 2 521.70				**Sum $h^{rq}_{\mathbf{FF}}$: 2 818.75**			

Table 6.29: Quality of *interval relaxation + planning graph* (h^{ig}_{FF}) compared to *repetition relaxation + priority queue* (h^{rq}_{FF}).

Summary	**interval+graph**	**repetition+queue**
Cost - Sum	**15.316 M**	17.223 M
Coverage - Sum	9118	**9122**
Quality - Sum	7642.90	**7805.10**

Table 6.30: Comparison of *interval relaxation + planning graph* (h^{ig}_{FF}) and *repetition relaxation + priority queue* (h^{rq}_{FF}).

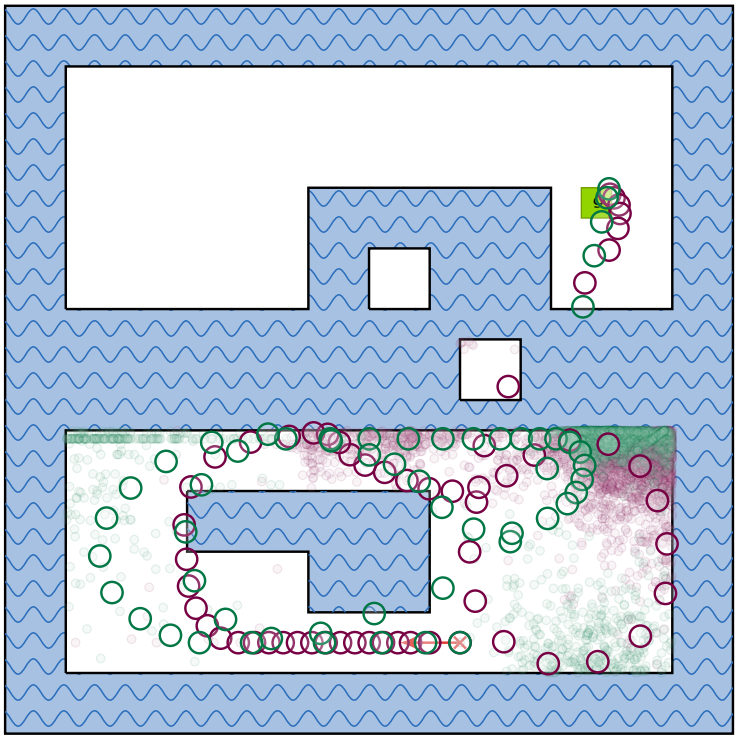

Figure 6.14: JUMPBOT scenario 14 (islands 4), step trace of Lh^{ig}_{FF} (ruby) and Lh^{rq}_{FF} (emerald)

relaxations increases by 11.8% as compared to h^{ig}_{FF}. The results are summarized in Table 6.30 and show that the repetition relaxed priority queue approach outperforms the interval relaxed planing graph variants even by aggregating all heuristics. Nevertheless, the repetition relaxed priority queue based approach should only be favored when the heuristic is able to extract an interval relaxed plan.

Ultimately, we also visualized the behavior of different relaxation approaches in the JUMPBOT domain as shown exemplarily in Figure 6.14. Initially, the robot is located at the bottom of the lower island facing west. The goal region is indicated by a green square on the upper island. We recorded the x and y position of the robot during search which allows us to plot the visited states. Finally, large circles show the step positions of the robot in the final plan. The step trace produced by the Lh^{ig}_{FF} heuristic is depicted in (ruby) whereas the Lh^{rq}_{FF} heuristic is depicted in (emerald). We can see that both configurations spent time in different parts of the search space. The relatively long plans demonstrate that we use a greedy best first search.

6.2.4 Comparison to Other Relaxation-Based Planners

In order to asses our implementation of the relaxation heuristics in relation to the state of the art, we compared NFD to Metric FF [Hof03] and two configurations of ENHSP: subgoaling with redundant constraints $\hat{h}^{radd}_{hbd+}$ [SHT16] and h_{AIBR} [SHTR16].

In a first experiment, we compared the plan cost of the three heuristics to the configuration 1Eh^{rq}_{FF}, NFD with h_{FF} computed on a state progression generated by a repetition relaxed and priority queue based approach with eager evaluation of search nodes and a unit cost task transformation. Plan costs can only be compared if all planning algorithms solve the same instances and thus, we show the results in a scatter plot again. We clamped costs at 120 and treat plans of higher cost as unsolvable. Again, we aggregated domains with several variants such as COUNTERS.

Figure 6.15 depicts the plan costs of pairwise compared instances from 1Eh^{rq}_{FF} to Metric FF (abbreviated Mh_{FF} in the figure). Overall, We can see that both planners have complementary strengths. For easier instances of easier domains, most marks are close to the diagonal, whereas for harder domains, there is usually one planner producing a significantly better plan than the other, up to the extreme that one planner can solve certain domains that the other cannot. Mh_{FF} finds shorter plans for SAILING and COUNTERS and can find plans for DEPOTS where 1Eh^{rq}_{FF} cannot. On the other hand, NFD achieves better plan costs for PLANTWATERING and can solve many instances in domains where Metric FF can only solve the easiest instances.

A similar picture emerges in the comparison of 1Eh^{rq}_{FF} to ENHSP with the h_{AIBR} heuristic which is depicted in Figure 6.16. Here, h_{AIBR} is particularly strong in PLANTWATERING whereas 1Eh^{rq}_{FF} finds shorter plans for SOKOBAN. Again, there are many marks at the top or to the right indicating that there are many domains where one algorithm can only solve only the easier instances while the other algorithm can also find plans for harder instances.

Finally, a comparison of plan costs of NFD with the 1Eh^{rq}_{FF} heuristic to ENHSP with the $\hat{h}^{radd}_{hbd+}$ heuristic is depicted in Figure 6.17. There is no domain

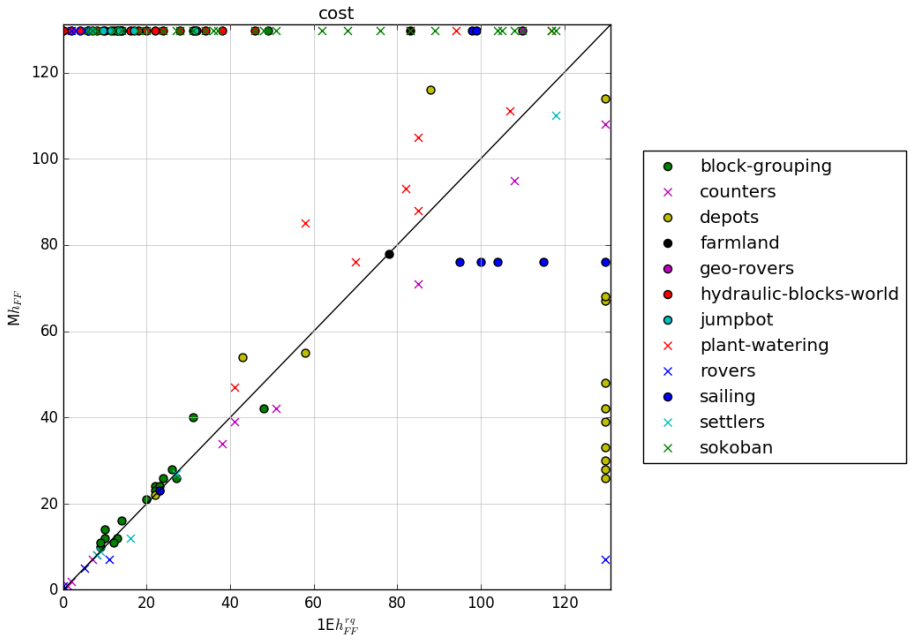

Figure 6.15: Plan cost comparison $1\mathrm{E}h^{rq}_{\mathrm{FF}}$ to Metric-h_{FF}.

where one heuristic produces significantly better plans than the other. SOKOBAN instances are more or less uniformly scattered around the diagonal, and in particular, we observe complementary coverage with one of the combinations only solving the easier instances of the domain.

The pairwise comparisons of the different algorithms already indicate that the planning algorithms have complementary strengths. This is also supported by the *coverage* results shown in Table 6.31. Metric FF achieves a very high coverage on COUNTERS and DEPOTS, ENHSP can solve particularly many instances in FARMLAND, GEOROVERS, PLANTWATERING, SAILING and SOKOBAN and NFD achieves a noteworthy coverage at JUMPBOT, SATELLITE and SETTLERS.

The pairwise comparisons of plan cost indicate that there are only few instances where one planner can find a good plan whereas the other algorithm finds a plan of with significantly higher cost. Basically, the algorithms either find plans of similar quality, or, one algorithm can find a solution while the other cannot. As such, it comes at no surprise that the algorithmic *quality*, shown in Table 6.32 does not differ significantly from the coverage table.

We did not address the heterogeneity of the number of instances per domain of our benchmark suite yet. BLOCKGROUPING, HYDRAULICBLOCKSWORLD and SOKOBAN have significantly more instances which deteriorates the results to the performance of algorithmic configurations in these domains. In order to mitigate for the effect, we aggregated domains with several variants and normalized the quality scores from Table 6.32 by dividing the result by the number of instances of each domain. As such, an algorithm can achieve up to 1 point in combined quality over all domains, if can solve every instance and returns

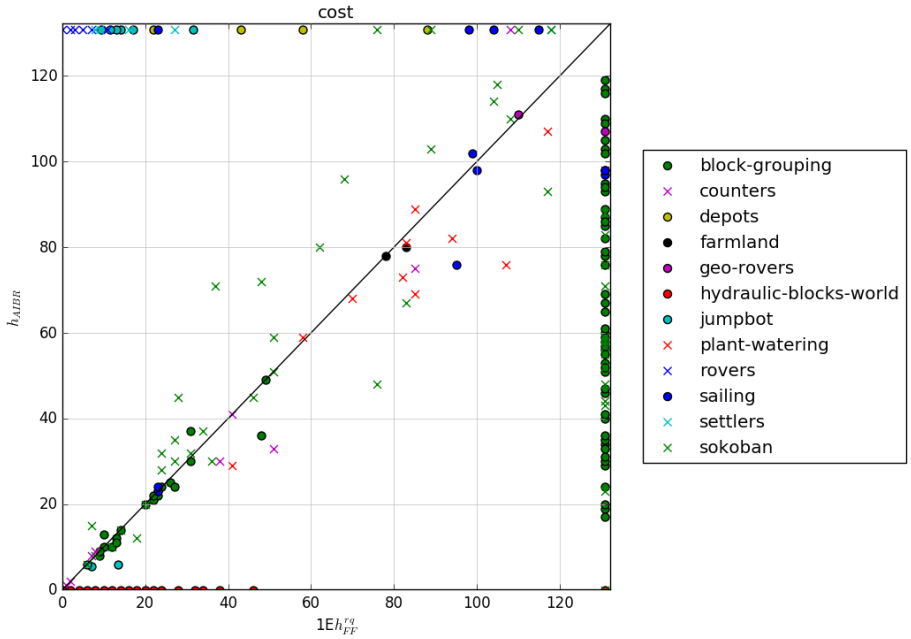

Figure 6.16: Plan cost comparison $1\mathrm{E}h^{rq}_{\mathrm{FF}}$ to h_{AIBR}.

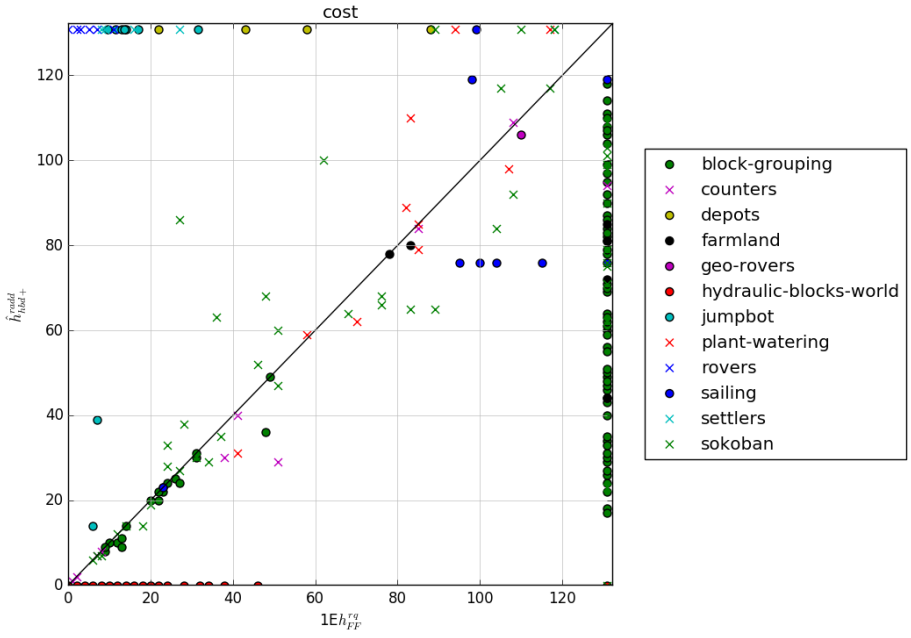

Figure 6.17: Plan cost comparison $1\mathrm{E}h^{rq}_{\mathrm{FF}}$ to $\hat{h}^{radd}_{hbd+}$.

Coverage	M-FF	ENHSP		NFD				
	h_{FF}	h_{AIBR}	$\hat{h}^{radd}_{hbd+}$	$\mathrm{E}h^{rq}_{\mathrm{FF}}$	$1\mathrm{E}h^{rq}_{\mathrm{FF}}$	$1\mathrm{L}h^{rq}_{\mathrm{FF}}$	$1\mathrm{L}h^{ig}_{\mathrm{FF}}$	$1\mathrm{L}h^{ig}_{\mathrm{add}}$
block-grouping (192)	22	**157**	152	25	25	25	14	15
counters-0 (34)	**8**	3	6	6	6	5	7	4
counters-inv (11)	**5**	2	**5**	**5**	**5**	**5**	3	4
counters-rnd (33)	**24**	10	21	20	20	21	10	13
depots (22)	**20**	–	–	10	9	9	9	7
driverlog (20)	17	–	–	15	**20**	**20**	**20**	17
driverlog-hard (20)	17	–	–	15	**20**	**20**	19	17
farmland (50)	9	11	**50**	25	25	22	19	17
geo-rovers (21)	–	7	**18**	4	3	5	1	1
hydraulic-bw (562)	–	–	–	**541**	**541**	**541**	**541**	**541**
jumpbot (20)	–	3	2	8	10	10	17	**19**
plant-watering (51)	22	17	**51**	15	15	15	–	–
rovers (20)	12	–	–	4	**15**	13	2	6
sailing (45)	18	11	**41**	17	17	18	14	36
satellite (20)	11	1	8	7	**16**	**16**	8	5
satellite-hard (20)	15	11	**20**	15	**20**	**20**	**20**	15
settlers (20)	9	–	–	3	10	**13**	5	3
sokoban (325)	–	47	**100**	67	68	67	69	70
umt2 (15)	–	–	–	–	–	**1**	–	–
zenotravel (20)	**20**	–	–	14	**20**	**20**	**20**	**20**
Sum (1 521)	229	280	474	816	865	**866**	798	810

Table 6.31: Coverage comparison of Metric FF, ENHSP and NFD.

Quality	M-FF	ENHSP		NFD				
	h_{FF}	h_{AIBR}	$\hat{h}^{radd}_{hbd+}$	$\mathrm{E}h^{rq}_{\mathrm{FF}}$	$1\mathrm{E}h^{rq}_{\mathrm{FF}}$	$1\mathrm{L}h^{rq}_{\mathrm{FF}}$	$1\mathrm{L}h^{ig}_{\mathrm{FF}}$	$1\mathrm{L}h^{ig}_{\mathrm{add}}$
block-grouping (192)	18.96	145.23	**152.00**	23.63	23.69	20.97	12.22	14.71
counters-0 (34)	**8.00**	3.00	5.80	5.73	5.70	4.51	6.98	4.00
counters-inv (11)	4.76	2.00	**4.95**	4.65	4.65	3.70	2.75	3.73
counters-rnd (33)	**22.84**	9.36	19.58	18.45	18.45	15.98	8.88	12.46
depots (22)	**17.16**	–	–	6.86	4.60	4.11	3.95	3.21
driverlog (20)	11.96	–	–	10.75	**15.82**	12.24	12.45	9.63
driverlog-hard (20)	**17.00**	–	–	0.06	2.79	1.96	1.34	0.57
farmland (50)	8.26	11.00	**50.00**	22.76	23.32	21.06	16.62	16.74
geo-rovers (21)	–	6.74	**16.96**	2.95	2.29	2.85	0.92	0.85
hydraulic-bw (562)	–	–	–	**487.39**	**487.39**	462.47	436.94	474.85
jumpbot (20)	–	3.00	0.57	6.58	7.29	6.55	13.23	**15.96**
plant-watering (51)	12.61	12.25	**43.09**	7.89	7.97	5.02	–	–
rovers (20)	11.00	–	–	4.00	**13.64**	11.47	2.00	5.00
sailing (45)	16.81	10.32	**37.03**	14.32	14.32	12.42	12.92	33.61
satellite (20)	6.98	1.00	4.74	4.70	10.45	**11.46**	4.18	2.23
satellite-hard (20)	0.00	11.00	**20.00**	15.00	**20.00**	**20.00**	**20.00**	15.00
settlers (20)	8.72	–	–	2.89	9.52	**12.53**	4.70	2.98
sokoban (325)	–	36.81	**77.07**	52.42	50.89	55.62	49.22	44.67
umt2 (15)	–	–	–	–	–	**1.00**	–	–
zenotravel (20)	13.87	–	–	10.25	11.32	11.91	**16.19**	8.63
Sum (1 521)	178.95	251.71	431.78	701.27	**734.11**	697.84	625.47	668.85

Table 6.32: Quality comparison of Metric FF, ENHSP and NFD.

Quality Normalized	M-FF	ENHSP		NFD				
	h_{FF}	h_{AIBR}	$\hat{h}^{radd}_{hbd+}$	$\text{E}h^{rq}_{\text{FF}}$	$1\text{E}h^{rq}_{\text{FF}}$	$1\text{L}h^{rq}_{\text{FF}}$	$1\text{L}h^{ig}_{\text{FF}}$	$1\text{L}h^{ig}_{\text{add}}$
block-grouping (192)	0.099	0.756	**0.792**	0.123	0.123	0.109	0.064	0.077
counters (78)	**0.456**	0.184	0.389	0.370	0.369	0.310	0.239	0.259
depots (22)	**0.780**	0.000	0.000	0.312	0.209	0.187	0.180	0.146
driverlog (40)	**0.724**	0.000	0.000	0.270	0.465	0.355	0.345	0.255
farmland (50)	0.165	0.220	**1.000**	0.455	0.466	0.421	0.332	0.335
geo-rovers (21)	0.000	0.321	**0.808**	0.140	0.109	0.136	0.044	0.040
hydraulic-bw (562)	0.000	0.000	0.000	**0.867**	**0.867**	0.823	0.777	0.845
jumpbot (20)	0.000	0.150	0.029	0.329	0.365	0.328	0.662	**0.798**
plant-watering (51)	0.247	0.240	**0.845**	0.155	0.156	0.098	0.000	0.000
rovers (20)	0.550	0.000	0.000	0.200	**0.682**	0.574	0.100	0.250
sailing (45)	0.374	0.229	**0.823**	0.318	0.318	0.276	0.287	0.747
satellite (40)	0.175	0.300	0.619	0.493	0.761	**0.787**	0.605	0.000
settlers (20)	0.436	0.000	0.000	0.145	0.476	**0.627**	0.235	0.149
sokoban (325)	0.000	0.113	**0.237**	0.161	0.157	0.171	0.151	0.137
umt2 (15)	0.000	0.000	0.000	0.000	0.000	**0.067**	0.000	0.000
zenotravel (20)	0.694	0.000	0.000	0.513	0.566	0.596	**0.810**	0.432
Sum (16) domains	4.699	2.514	5.540	4.850	**6.090**	5.863	4.829	4.469

Table 6.33: Quality comparison of Metric FF, ENHSP and NFD, normalized.

the best known plan for that instance as well. The normalized quality score is found in Table 6.33. Notably, the ENHSP $\hat{h}^{radd}_{hbd+}$ heuristic achieves this perfect score for the FARMLAND domain. With this normalized quality table, we can assess the performance of the different planners and the different configurations much more fairly. Overall, The eagerly evaluated $1\text{E}h^{rq}_{\text{FF}}$ heuristic using a repetition relaxed priority queue approach on a unit cost transformed task achieves the highest trade-off between coverage and plan quality of 6.09 from 16 possible quality points, a score mostly achieved by having the highest coverage among the algorithms. Nevertheless, we can see that all presented configurations perform best or are tied for best performance in at least one domain. Metric FF can only solve problems that can be compiled to linear tasks. As such $\text{M}h_{\text{FF}}$ cannot find solutions for nonlinear problems such as GEOROVERS or JUMPBOT. Whereas the h_{FF} heuristic used by Metric FF is similar to h^{ig}_{FF} in spirit, differences in the generation of the relaxed state progression, a different search algorithm and a different implementation result in a planner with different strengths. ENHSP, particularly with the subgoaling heuristic $\hat{h}^{radd}_{hbd+}$ achieves very high coverage on many domains and given that it lacks solving some of the domains because of parsing errors, it usually outperform NFD based relaxation heuristics. NFD proved to be able to solve instances in every domain, even in UMT2, a domain that was intended to pose a challenge for handcrafted planners at the IPC 2002. Many domains feature traps for relaxation heuristics or relaxation challenge such as the *cyclic resource transfer problem* where resources can be generated in the heuristic when resources are transferred from one location to the other. Nevertheless, the h_{FF} based relaxation heuristic allow for basic guidance, computable in polynomial time, also in domains that include non-linear effects.

6.3 Agile Earth Observation: an Application

In this section we will discuss an application example of a numeric planning problem: the *agile Earth observation* scenario that was first presented at the

"Planning in Continuous Domains" workshop at the International Conference on Automated Planning and Scheduling 2013 [AL13] on which this section is based upon.

Agile Earth observation satellites are satellites orbiting Earth with the purpose to gather information of the Earth's surface by slewing the satellite toward regions of interest, straight stripes referred to as *patches*, during the flyover. Constraints arise not only from dynamical and kinematic aspects of the satellite and its sensors. Regions of interest change over time and bad weather can conceal important observation targets. This results in a constant need to re-plan the satellite's tasks and raises the desire to automatize this planning process. A task in the context of an Earth observation mission is to select and to schedule a sequence of observation patches. Determining the sequence of patches is a rather simple discrete planning problem for current automated planning systems. For satellites carrying firmly fixed heavy instruments, the satellite has to be *agile* and slew towards the *patch* that has to be scanned. Complex numerical calculations have to be performed to determine such slew maneuvers. The feasibility of the discrete plan tightly depends on the continuous aspects since it has to consider the satellite's orbital motion, its attitude and angular rate as well as its torque capability in realistic scenarios. Instrument alignment and required scan velocity pose additional constraints. The feasibility of slews between two successive scans depends on the satellite's attitude, angular rate and position and is varying in time. Any decision to scan a certain patch at a certain position in orbit may affect the feasibility of future scan maneuvers.

Testing slew maneuver feasibility goes beyond the capabilities of current state-of-the-art planning systems including NFD. However, it is possible to decouple the deterministic planning task, form the numeric calculations which makes Earth observation an appealing task for modular planning systems e.g. Temporal Fast Downward with Modules (TFD/M) [DEK+09]. As the planning problem is sequential in nature (there is only one satellite), the agile Earth observation scenario is a motivation for future planning systems that incorporate external semantic attachments into more informed sequential planning system such as NFD.

6.3.1 Temporal Fast Downward with Modules

The TFD planning system [EMR09] that we introduced in Section 5.3 is a temporal and numeric planning system. The modules extension TFD/M [DEK+09] allows to access "semantic attachments", modules that outsource the numerical calculations from the propositional planning level. During the planning process, external modules are called. These modules process the required computations and return the result to the planning process. The interface for "semantic attachments" in PDDL allows the addition of three types of modules to the planning domain: `conditionchecker` modules, `cost` modules and `effect` modules. `Conditionchecker` modules evaluate to propositional variables and occur in the precondition of a planning action. Similarly, `cost` modules represent numeric variables. Even though the name suggests they can only be used to compute action *costs* they can appear in numeric expressions of an action precondition as well. Finally, `effect` modules modify a set of variables in the planner state. The modified variables can be either propositional or numeric variables.

TFD/M interleaves the causal planning problem "what to do" with the nu-

merical task "how to achieve it". Neither a top-down nor a bottom-up approach can satisfy the interdependency between low level calculations and high level plan structure, and we therefore rely on an interleaved approach. A classical top-down decomposition of the planning task solves the problem on an abstract symbolic domain, and then refines that symbolic plan. In the case of Earth observation the planner would first schedule the sequence of patches to be scanned, while the maneuvers to follow this sequence would then be calculated in a refinement step. The drawback of a top-down approach is that high-level solutions can be incorrect or pose contingencies for low-level planners. Even if the maneuvers are feasible, the resulting plan is unlikely to be good. The opposite approach, a bottom-up decomposition, precomputes *all* refined solutions, so that a higher level symbolic planner can then draw on the lower level plans. While the resulting plans are usually optimal, precomputing all low level solutions requires excessive memory and run-time. In continuous settings there are infinitely many low level plans. In the Earth observation scenario, all possible maneuvers would have to be precomputed which is intractable even for coarse discretizations. The semantic attachments of TFD evaluate the decomposition of a symbolic action on demand and can thus involve the interdependency between high level symbolic actions and low level numeric calculations.

6.3.2 Planning the Earth Observation Task

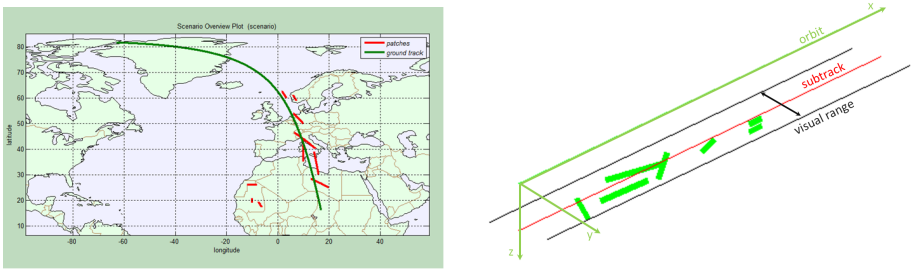

Figure 6.18: Earth observation scenario with subtrack of the satellite and observation sites to be scanned (left) and planning problem obtained after preprocessing (right).

In order to solve Earth observation problems with TFD/M, the continuous world has to be discretized. Our general framework is a three step process. At first, we precompile the real world problem into a planning task, at second solve this planning task with TFD/M and finally verify the planned results with a physics simulator. The preprocessing step reduces some of the numerical complexity from the planning problem. We consider the subtrack obtained by projecting the satellite perpendicular to the earth surface and peel off a stripe of the Earth's surface following the subtrack. The width of the peeled off stripe includes all patches that are within the satellite's visual range determined by its maximal angular deflection. For an example, consider the Earth observation scenario depicted on the left in Figure 6.18. The ground track of the satellite is depicted by a green line. The patches to be scanned are depicted in red. Some of the patches in north west Africa are out of range of the satellite's sensors.

The planning problem that arises after preprocessing is depicted on the right in Figure 6.18. The green patches on the right correspond to the red observation sites in the scenario on the left. In the planning problem we omit Earth's curvature and treat the surface of the precompiled problem as long plane with a satellite flying over it on an orbit depicted in olive green (cf. Figure 6.18).

If the problem horizon entails multiple satellite orbits, the same *patch* can be visible from different orbit positions. This results in multiple instances of the same patch in the precompiled stripe, usually in different orientation. To distinguish such patches, we use the term *observation site* for a site on Earth that has to be scanned, and use *patch* for a concrete instance observed from the current orbit.

Scanning patches corresponds to achieving soft goals because it is not always possible to scan all patches in the planning problem. Following Keyder and Geffner [KG09] we introduce an action to `ignore` an observation site, which results in a high penalty cost. By modifying the penalty for ignoring a patch, the observation sites can be given different priorities. The goal of the planning problem is to *deal* with all observation sites, which can be done by either scanning a patch, or by actively ignoring it. While scanning occurs at a cost depending on the optimization criterion of the planning problem (e.g. available time or energy consumption) ignoring an observation site occurs at a much larger penalty cost.

6.3.3 Discretization

To model the Earth observation scenario, each state of the Earth observation planning problem describes a *snapshot* of the continuous world. Usually, the satellite has just scanned a patch and the numeric state variables describe the attitude and rate of the satellite in this position with line of sight toward the end of the patch. Discrete planning decisions are made between these snapshot states. The available actions at such a state are to either scan or to ignore one of the remaining patches. While ignoring a patch results in a discrete successor state that differs to the previous one only in propositional variables, determining the successor snapshot state of the planner after applying a `scan` action is not obvious. As the world is continuous, deciding for the next patch to scan could result in infinitely many possible successor states, since it is possible to scan a patch from different positions in the orbit. To commit to one discrete successor state after deciding for a patch to scan, we make the assumption that it is always best to scan a patch as soon as possible. This assumption leaves a wider scope for future actions and implies that it is more important to scan many patches than to scan them with a good image quality which is usually obtained when the angular deflection of the satellite's line of sight is minimal.

In order to determine the *earliest* orbit position for scanning a certain patch, we first look at the extreme positions and omit the constraints posed from other patches. Then we will propose a method based on interval nesting to determine the earliest possible orbit position to start scanning the selected patch. At first we consider the case, where the patch to scan is far away from the satellite's current orbit position. The earliest possible approach configuration of a satellite is obtained by deflecting the satellite as far possible. The maximal deflection is limited by the maximal angle under which the sensor can operate.

An example is illustrated in Figure 6.19. The subtrack of the satellite is

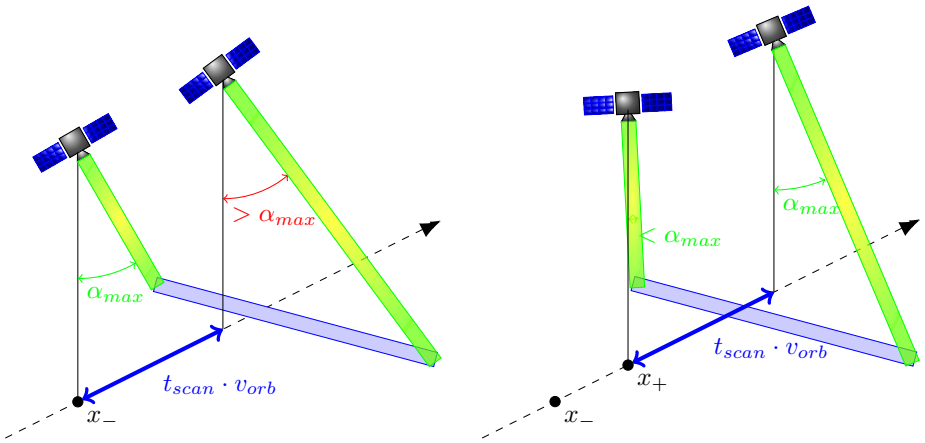

Figure 6.19: Satellite states to determine the earliest possible position to scan the patch

depicted by the dashed line. The left satellite in the left graphic depicts the state of the satellite at the earliest orbit position x_- to scan the patch. There, the satellite is deflected with the maximal angle $\alpha_{\max}$ towards the patch. However, the attitude dynamic constraints of the satellite could be violated by scanning the patch starting form x_-. A scan maneuver starting at x_- would end at the infeasible state depicted by the right satellite on the left in Figure 6.19, where the deflection exceeds $\alpha_{\max}$. Instead, we can determine a different *earliest orbit position* where the satellite is at its maximal deflection $\alpha_{\max}$ pointed towards the *end* of the patch as depicted by the right satellite in Figure 6.19 on the right. The orbit position x_+, depicted by the left satellite in the right of Figure 6.19 is then obtained from regressing this orbit position by the time needed for scanning the patch t_{scan} and the orbital velocity v_{orb}. Depending on the scanning speed and the orientation of the patch relative to the satellite's subtrack position, either x_- or x_+ can be the critical *earliest possible orbit position* to scan the patch. The satellite state s_{first} is then the state where the satellite is at x_- if the maneuver is feasible and at x_+ otherwise.

The last possible orbit position to scan the patch can be computed by minimizing over the latest attitude under which the satellite can start or end a scan analogously and the resulting state is denoted by s_{last}. All feasible scan maneuvers are in the interval between the orbit positions at s_{first} and s_{last}. In real planning problems with multiple patches it is not likely that all patches can be scanned as early as s_{first}. Instead, scanning a patch should start as soon as possible from the satellite's current state. Unfortunately, neither the satellite's orbit position nor its attitude after executing this best slew maneuver are known in advance. The principal problem of finding the earliest orbit position to start the next scan builds a non-linear equation system for which no closed form solutions methods are known to us[1]. We therefore approximate the satellite state with the help of interval nesting.

The flow chart in Figure 6.20 illustrates the mathematical calculations needed inside a `scan` planning operator, which approximates the earliest orbit position

[1]This is one of the reasons for implementing the problem in TFD/M instead of NFD.

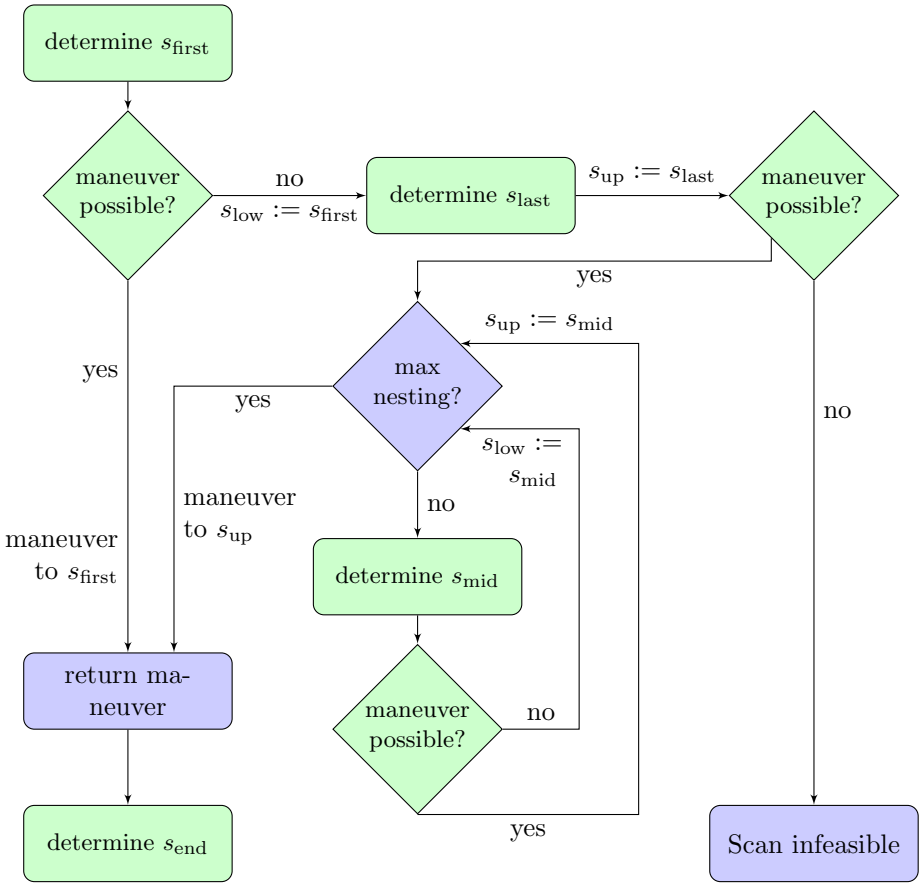

Figure 6.20: Inside a `scan-patch` planning operator. Green boxes can be calculated by Function 1 while green diamonds are calculated by Function 2.

to scan the patch as well as the corresponding slew maneuver. Two types of mathematical calculations have to be performed frequently in an Earth observation planning task:

Function 1 (State from Orbit Position). Determine the satellite's state (attitude and angular rates) from a given orbit position when pointing towards a patch, with angular rates satisfying the vectorial velocity for scanning the patch.

Function 2 (Maneuver). Determine the feasibility of a maneuver given two satellite states.

The green boxes in the flow chart in Figure 6.20 are all instances of Function 1 while the green diamonds can be calculated with Function 2. We are interested in the earliest orbit position to start scanning the next patch, the *approach state*, given a current state where the satellite is pointing towards the end of another patch that it just finished scanning. The orbit position after an optimal slew maneuver from the current state of the satellite to the best approach state lies in the interval between the orbit position at s_{first} and s_{last}. If a maneuver from the current satellite state s_{current} to the earliest possible scan configuration s_{first} is possible, we can return it immediately. After scanning the patch the satellite adopts state s_{end}, with an orbit position depending on orbital velocity and the scanning time, both of which are given in the domain description. The deliberations made for s_{first} and depicted in Figure 6.19 ensure that s_{end} is valid. If the maneuver from s_{current} to s_{first} is infeasible, it is used as lower bound s_{low} of an interval nesting, and s_{last} is calculated as latest possible orbit position to scan the patch. If the maneuver to s_{last} is infeasible, the patch can not be scanned at all. Otherwise, the satellite's state of a time optimal maneuver is found between the unreachable lower bound s_{low} and the reachable but not time optimal upper bound satellite state $s_{\text{up}} = s_{\text{last}}$. Unless the maximal nesting depth is exceeded, we determine the satellite state s_{mid} in the middle of the boundaries with the help of Function 1, and check if the maneuver from the current satellite configuration to the middle configuration is feasible with the help of Function 2. If the maneuver is feasible, a better upper bound has been found, while the s_{mid} is used as lower bound if the time slew time is exceeded.

In our implementation we limit the depth of the interval nesting to 10 which offers a good trade-off between run-time (less than $1ms$) of the operation and precision (approximately $10m$ deviation). We note that more sophisticated interval nesting methods to determine the orbit position of the most promising middle configuration could be used.

While we calculate the satellite's attitude and angular rates s_{end} after scanning, we do not check if the maneuver from s_{up} to s_{end} is feasible. However, the assumption that such a maneuver exists is justified to the best of our knowledge.

As mentioned earlier, the extension of the PDDL planning language allows planning operators to contain three types of modules: `conditionchecker` modules, `cost` modules and `effect` modules. In our implementation, scanning a patch is decoupled into two actions `approach-patch` and `scan-patch`. This is mostly done for technical reasons, since it is easier to determine the satellite state after each operator execution during planning which makes it easier to verify the feasibility of the plan during post processing. Within these two operations, we use five modules, which all calculate parts of the flow chart of

Figure 6.20. To avoid the recalculation of the same function, a database stores all calculations performed by Function 1 and Function 2. The time intensive interval nesting is therefore only computed once for each configuration.

The modules executed by the `approach-patch` operator all follow the flow chart diagram (Figure 6.20). A `conditionchecker`-module tests, whether approaching a patch is possible, a `cost`-module determines the maneuver time to approach the patch, and finally an `effect`-module modifies the planning state and sets the planning variables of the satellite to s_{up}. The modules used in the `scan-patch` operator are a rather simple `cost` module that calculates the time needed for scanning. An `effect` module sets the planner state of the variables concerning the satellite to s_{end}. Additionally the planning operator sets the corresponding observation site `dealt` variable to *true*.

The physical foundations of the *slew feasibility* given the *satellite attitude dynamics* is discussed in more detail in the original paper [AL13].

6.3.4 Experimental Results

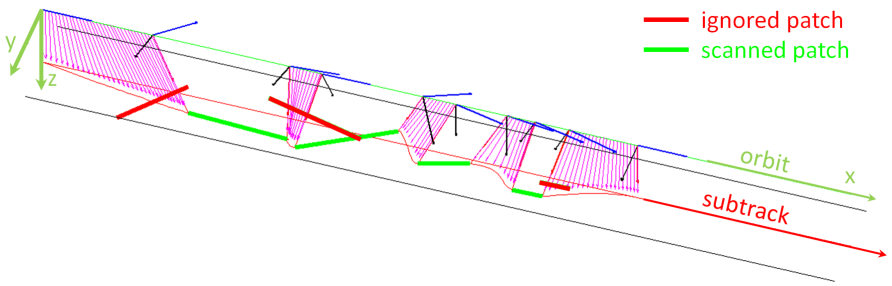

Figure 6.21: Earth observation scenario with subtrack of the satellite and observation sites to be scanned. The magenta colored arrows depict the line of sight of the instrument during the slew maneuver.

To test the feasibility of our approach, we implemented the precompiled planning problem from Figure 6.18 in PDDL and solved it with TFD/M. Figure 6.21 shows a visualization of the intermediate states of the satellite extracted from the resulting plan. The satellite's attitude is depicted by the current body fixed frame in blue black and magenta.

The satellite slews towards the first patch to scan it at its earliest possible state. The start of the scan maneuver of all other patches is constrained by the attitude of the satellite after scanning the previous patch, so all other maneuvers have to be calculated by interval nesting.

The resulting plan happens to be optimal for the tested planning instance given our assumptions and the satellite parameters used. The leftmost patch cannot be scanned because the satellite's state to start the scan maneuver is infeasible. In additional experiments we investigated the influence of increasing the maximal angular rates of the satellite in the planning problem. In this case the slew maneuver from the last patch to the ignored rightmost patch becomes feasible. With more angular scope also the first patch is in range and TFD/M finds a plan scanning six of the patches.

Although our approach is promising and seems to work well in practice, optimality cannot be guaranteed, even regarding the inaccuracies of the model such as plain Earth surface, circular orbits, etc. The interval nesting approach is only iterated to a certain depth so that each scan action is started at a minimally later orbit position than theoretically possible. Now it is easy to construct a problem that will not be solved optimally by our approach by adding a new patch that is reachable by a slew maneuver from the orbit position after scanning the previous patch from the theoretical earliest orbit position but not from the orbit position found by interval nesting. Completeness can be achieved with the trivial plan, but in scenarios where all patches could be scanned it is not guaranteed that TFD/M will unnecessarily ignore some observation sites with the analogous argument as for optimality.

6.3.5 Conclusion

We have presented an agile Earth observation task and an automated planning system capable of solving it. Preliminary experiments show the feasibility of our approach.

The automated planning system TFD/M can be successfully applied to our Earth observation scenario. Nevertheless, it would be beneficial if the planning system would have a better handling of numeric variables internally. As modeled, the Earth observation problem does not involve temporal concurrency, and the planning problem is even serialized artificially with the help of propositional `idle` variables. While numerical variables are required for the Earth observation scenario, the temporal aspect of TFD is not. An extension of NFD that outsources numeric calculations that go beyond the arithmetic base operations to *semantic attachments* would be even better suited for the Earth observation problem at hand.

Chapter 7

Conclusion

In this thesis, we discussed relaxation heuristics to approach numeric planning problems. We identified interval based relaxations to be best suitable for generalizing the concept of the delete relaxation to numeric planning, namely that facts that are achieved once remain achieved.

We introduced two relaxations for numeric planning, both utilizing interval algebra: the interval relaxation and the repetition relaxation. We used the repetition relaxation as a means to prove that the interval relaxed plan existence problem can be decided in polynomial time for *acyclic dependency tasks*, tasks where the expression assigned to a variable by a numeric effect does not depend on the affected variable. The repetition relaxation accounts for the repeated application of actions in one step. We showed that the values that become reachable by applying a numeric action arbitrary often can be computed efficiently by analyzing the behavior of its effects. This makes actions in the repetition relaxation pseudo-idempotent so that the repetition relaxation becomes useful to be used in heuristics on its own.

In order to compute heuristics efficiently in numeric planning, we have to restrict the number of facts that we consider by generating a relaxed state progression sequence. Given such a progression structure, we can apply the relaxation heuristics from classical planning in a similar fashion. We presented two approaches to generate a relaxed state progression sequence, one based on the concept of a planning graph and the other based on priority queues. We discussed additional requirements to ensure that the state progression sequence can be generated efficiently and we overcome the problem of cycles in the planning task by relaxing the values of cyclic variables even further. We identified the combination of the interval relaxation with the planning graph based progression approach and the repetition relaxation with the priority queue based progression approach as most promising candidates.

We implemented these heuristics in the NFD planning system, a numeric planner based on Fast Downward that has been developed over the course of the doctoral process. Afterwards, we compared various configurations of our heuristics in terms of plan cost, coverage and algorithmic quality. The priority queue based state progression approach combined with a repetition relaxation is particularly useful in combination with the h_{FF} heuristic. Explicating the number of repetitions that is actually needed exploits the full potential of the strengths of a repetition relaxation and compensates for its weakness. However,

for $h_{\max}$ and h_{add} that determine the estimates progressively, the more traditional planning graph based state progression approach yields better results.

Finally, we compared our approach to other state-of-the-are planning systems. Besides having the highest overall quality score, our NFD planner was the only planning systems that could solve at least one instance in every domain proving that the approach is not only applicable to a wide range of numeric planning problems, but also that it offers basic guidance even in problems that pose traps for relaxation heuristics. However, the experiments also revealed that other approaches can achieve a significantly larger coverage in some of the domains and have complementary strengths.

The relaxation heuristics are a first step toward solving numeric planning problems more efficiently. Other approaches that look promising include linear programs which are not only a very obvious tool to solve linear planning tasks, but could also be used to approximate non-linear dependencies. Furthermore, our discourse into Earth observation planning revealed that many real world problems are too complex to tackle inside a planning system. A hybrid planner with an interface to an external solver, analogously to TFD/M for temporal planning, would certainly increase the applicability of NFD in particular and numeric planning in general.

List of Figures

List of Tables

Bibliography

[AL13] Johannes Aldinger and Johannes Löhr. "Planning for Agile Earth Observation Satellites." In: *Proceedings of the ICAPS-2013 Workshop on Planning in Continuous Domains (PCD 2013)*. 2013, pp. 9–17.

[AL16] Johannes Aldinger and Johannes Löhr. *The Jumpbot Domain for Numeric Planning*. Tech. rep. 279. University of Freiburg, Apr. 2016.

[AMG15a] Johannes Aldinger, Robert Mattmüller, and Moritz Göbelbecker. "Complexity Issues of Interval Relaxed Numeric Planning". In: *Proceedings of the ICAPS-2015 Workshop on Heuristics and Search for Domain-independent Planning (HSDIP 2015)*. 2015, pp. 4–12.

[AMG15b] Johannes Aldinger, Robert Mattmüller, and Moritz Göbelbecker. "Complexity of Interval Relaxed Numeric Planning". In: *Proceedings of the 38th German Conference on Artificial Intelligence (KI 2015)*. Ed. by Steffen Hölldobler, Markus Krötzsch, Rafael Peñaloza, and Sebastian Rudolph. Vol. 9324. LNAI. Springer-Verlag, 2015, pp. 19–31.

[AN17a] Johannes Aldinger and Bernhard Nebel. *Addendum to 'Interval Based Relaxation Heuristics for Numeric Planning with Action Costs'*. Tech. rep. 280. University of Freiburg, Oct. 2017.

[AN17b] Johannes Aldinger and Bernhard Nebel. "Extended Abstract: Interval Based Relaxation Heuristics for Numeric Planning with Action Costs". In: *Proceedings of the 10th International Symposium on Combinatorial Search (SoCS 2017)*. 2017, pp. 155–156.

[AN17c] Johannes Aldinger and Bernhard Nebel. "Interval Based Relaxation Heuristics for Numeric Planning with Action Costs". In: *Proceedings of the 40th German Conference on Artificial Intelligence (KI 2017)*. Ed. by Gabriele Kern-Isberner, Johannes Fürnkranz, and Matthias Thimm. Vol. 10505. LNAI. Springer-Verlag, 2017, pp. 15–28.

[BAV15] Miquel Bofill, Joan Espasa Arxer, and Mateu Villaret. "The RANTANPLAN Planner: System Description". In: *Proceedings of the ICAPS-15 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAPS 2015)*. 2015, pp. 1–10.

[BFK+17] Felix Burget, Lukas D.J. Fiederer, Daniel Kuhner, Martin Völker, Johannes Aldinger, Robin T. Schirrmeister, Chau Do, Joschka Boedecker, Bernhard Nebel, Tonio Ball, and Wolfram Burgard. "Acting Thoughts: Towards a Mobile Robotic Service Assistant for Users with Limited Communication Skills". In: *Proceedings of the European Conference on Robotics (ECMR 2017)*. 2017, pp. 385–390.

[BG01] Blai Bonet and Héctor Geffner. "Planning as Heuristic Search". In: *Artificial Intelligence* 129.1–2 (2001), pp. 5–33.

[BG99] Blai Bonet and Héctor Geffner. "Planning as Heuristic Search: New Results". In: *Proceedings of the 5th European Conference on Planning (ECP 1999)*. 1999, pp. 360–372.

[BLG97] Blai Bonet, Gábor Loerincs, and Héctor Geffner. "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)*. July 1997, pp. 27–31.

[BN93] C. Bäckström and Bernhard Nebel. "Complexity results for SAS^+ planning". In: *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI 1993)*. 1993, pp. 1430–1435.

[Byl94] Tom Bylander. "The Computational Complexity of Propositional STRIPS Planning". In: *Artificial Intelligence* 69.1–2 (1994), pp. 165–204.

[CCFL13] Amanda Coles, Andrew Coles, Maria Fox, and Derek Long. "A Hybrid LP-RPG Heuristic for Modelling Numeric Ressource Flows in Planning". In: *Journal of Artificial Intelligence Research (JAIR)* 46 (2013), pp. 343–412.

[CFLS08] Andrew Coles, Maria Fox, Derek Long, and Amanda Smith. "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, pp. 52–59.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2009.

[DEK+09] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. "Semantic Attachments for Domain-independent Planning Systems". In: *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*. 2009.

[Dij59] Edsger W. Dijkstra. "A Note on Two Problems in Connexion with Graphs". In: *Numerische Mathematik* 1.1 (1959), pp. 269–271.

[Ede04] Stefan Edelkamp. "Generalizing the Relaxed Planning Heuristic to Non-Linear Tasks". In: *Proceedings of the 27th German Conference on Artificial Intelligence (KI 2004)*. Springer-Verlag, 2004.

[EH01] Stefan Edelkamp and Malte Helmert. "The Model Checking Integrated Planning System MIPS". In: *AI Magazine* (2001), pp. 67–71.

[EMR09] 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)*. AAAI Press, Sept. 2009, pp. 130–137.

[FG15] Guillem Francès and Hector Geffner. "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.

[FL03] Maria Fox and Derek Long. "PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains". In: *Journal of Artificial Intelligence Research (JAIR)* 20 (2003), pp. 61–124.

[FN71] Richard E. Fikes and Nils J. Nilsson. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". In: *Artificial Intelligence* 2.3–4 (1971), pp. 189–208.

[Gol91] David Goldberg. "What Every Computer Scientist Should Know About Floating Point Arithmetic". In: *ACM Computing Surveys* 23.1 (1991), pp. 5–48.

[GS02] Alfonso Gerevini and Ivan Serina. "LPG: A Planner Based on Local Search for Planning Graphs with Action Costs". In: *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*. 2002.

[Hel02] Malte Helmert. "Decidability and Undecidability Results for Planning with Numerical State Variables". In: *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*. 2002, pp. 303–312.

[Hel06] Malte Helmert. "The Fast Downward Planning System". In: *Journal of Artificial Intelligence Research (JAIR)* 26 (2006), pp. 191–246.

[HN01] Jörg Hoffmann and Bernhard Nebel. "The FF Planning System: Fast Plan Generation Through Heuristic Search". In: *Journal of Artificial Intelligence Research* 14 (2001), pp. 253–302.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[Hof03] Jörg Hoffmann. "The Metric-FF Planning System: Translating 'Ignoring Delete Lists' to Numeric State Variables". In: *Journal of Artificial Intelligence Research (JAIR)* 20 (2003), pp. 291–341.

[KAB+18] Daniel Kuhner, Johannes Aldinger, Felix Burget, Mara Göbelbecker, Wolfram Burgard, and Bernhard Nebel. "Closed-Loop Robot Task Planning Based on Referring Expressions". In: *Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2018)*. 2018.

[KFB+18] Daniel Kuhner, Lukas Fiederer, Felix Burget, Martin Völker, Johannes Aldinger, Robin Schirrmeister, Chau Do, Wolfram Burgard, and Bernhard Nebel. "A Deep Learning Framework for BCI Control of a Robotic Service Assistant using Intelligent Goal Formulation". In: *Journal of Robots and Autonomous Systems (RAS)* (2018), to appear.

[KG09] Emil Keyder and Héctor Geffner. "Soft Goals can be Compiled Away". In: *Journal of Artificial Intelligence Research (JAIR)* 36 (2009), pp. 547–556.

[KS92] Henry Kautz and Bart Selman. "Planning as Satisfiability". In: *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI 1992)*. Wiley, 1992, pp. 359–363.

[LAWW13] Johannes Löhr, Johannes Aldinger, Stefan Winkler, and Georg Willich. "Automated Planning for Earth Observation Spacecraft under Attitude Dynamical Constraints". In: *Jahrbuch der Deutschen Gesellschaft für Luft- und Raumfahrt (DGLR2013)*. 2013.

[LEKN12] Johannes Löhr, Patrick Eyerich, Thomas Keller, and Bernhard Nebel. "A Planning Based Framework for Controlling Hybrid Systems". In: *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*. 2012.

[LF03] Derek Long and Maria Fox. "An Overview and Analysis of the Results of the 3rd International Planning Competition". In: *Journal of Artificial Intelligence Research (JAIR)* 20 (2003), pp. 1–59.

[McD00] Drew McDermott. "The 1998 AI Planning Systems Competition". In: *AI Magazine* 21 (2000), pp. 35–55.

[MGH+98] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. *PDDL - The Planning Domain Definition Language*. Tech. rep. CVC TR-98-003. Yale University, Center for Computational Vision and Control, 1998.

[MKC09] Ramon E. Moore, R. Baker Kearfott, and Michael J. Cloud. *Introduction to Interval Analysis*. Society for Industrial and Applied Mathematics, 2009.

[Nil80] Nils J. Nilsson. *Principles of Artificial Intelligence*. Springer, 1980.

[RH13] Gabriele Röger and Malte Helmert. *Engineering a Heuristic Search Planner*. `http://icaps13.icaps-conference.org/wp-content/uploads/2013/06/helmert-roger.pdf`. ICAPS 2013 Tutorial. 2013.

[SHT16] Enrico Scala, Patrik Haslum, and Sylvie Thiébaux. "Heuristics for Numeric Planning via Subgoaling". In: *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*. 2016, pp. 655–663.

[SHTR16] Enrico Scala, Patrik Haslum, Sylvie Thiébaux, and Miquel Ramírez. "Interval-Based Relaxation for General Numeric Planning". In: *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016)*. 2016, pp. 655–663.

[SPSH17] Jendrik Seipp, Florian Pommerening, Silvan Sievers, and Malte Helmert. *Downward Lab*. 2017. DOI: 10.5281/zenodo.790461. URL: https://doi.org/10.5281/zenodo.790461.

[SRHT16] Enrico Scala, Miquel Ramírez, Patrik Haslum, and Sylvie Thiébaux. "Numeric Planning with Disjunctive Global Constraints via SMT". In: *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS 2016)*. 2016.

[Tar72] Robert Tarjan. "Depth-First Search and Linear Graph Algorithms". In: *SIAM Journal on Computing* (1972), pp. 146–160.

[You31] Rosalind Cecily Young. "The Algebra of Many-valued Quantities". In: *Mathematische Annalen 104* (1931), pp. 260–290.