Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Experimental Analysis of Critical Path Heuristics

Evžen Šírek

Supervisor: Ing. Daniel Fišer
Field of study: Computer and Information Science
May 2018

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Šírek Evžen**	Personal ID number:	**434672**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Branch of study:	**Computer and Information Science**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Experimental Analysis of Critical Path Heuristics

Bachelor’s thesis title in Czech:

Experimentální analýza heuristik kritických cest

Guidelines:

The goal of the thesis is to implement the general hm heuristic functions for classical planning and compare them with the state-of-the-art. The analysis will show the strenghts and weaknesses of the implemented heuristics in standard benchmarks. Efficiency will be improved using strategies for the selection of a subset of meta-facts while keeping the estimates as high as possible.
1) Study literature in the area of the classical planning, in particular the literature related to the critical path heuristics.
2) Create an efficient implementation of h2, h3, and general hm heuristics.
3) Compare the implemented heuristics with the state-of-the-art heuristics in terms of the heuristic value in the initial state, the coverage on the standard benchmark set, and the number of evaluated states per second.
4) Propose, implement and experimentally evaluate a strategy for selecting a subset of meta-facts allowing faster evaluation of hm heuristics while preserving high heuristic estimates.

Bibliography / sources:

[1] Haslum, P. (2009). hm(P) = h1(Pm): Alternative characterisations of the generalisation from hmax to hm. In Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS), pp. 354-357.
[2] Haslum, P. (2012). Incremental lower bounds for additive cost planning problems. In Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS).
[3] Haslum, P., Bonet, B., & Geffner, H. (2005). New admissible heuristics for domain-independent planning. In Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, pp. 1163-1168.
[4] Haslum, P., & Geffner, H. (2000). Admissible heuristics for optimal planning. In Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS), pp. 140-149.
[5] Keyder, E. R., Hoffmann, J., & Haslum, P. (2014). Improving delete relaxation heuristics through explicitly represented conjunctions. J. Artif. Intell. Res. (JAIR), 50, 487-533.

Name and workplace of bachelor’s thesis supervisor:

Ing. Daniel Fišer, Department of Computer Science and Engineering, FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **04.01.2018** Deadline for bachelor thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

Ing. Daniel Fišer	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I would like to thank my supervisor Ing. Daniel Fišer for the valuable comments and remarks he has given me during the creation of this work.

Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures".

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 25. May 2018

Abstract

The critical path heuristics are well studied in the area of classical planning. The critical path heuristics are denoted by h^m, where m corresponds to the maximal size of sets of facts used in the computation. This thesis describes and provides effective implementations of the h^2 and h^3 heuristics. For that, we utilize the alternative characterization Π^m. Analysis of the h^2, h^3 and other state-of-the-art heuristics is made. We compare the heuristics in terms of heuristic values in initial states, the number of solved problems in the International Planning Competition datasets and the number of evaluated states per second. Moreover, a new characterization of the task, Π^m_r, is introduced. This characterization allows for the choice of a set of facts r excluded from the meta-fact creation, reducing the size of the Π^m task. Finally, a strategy for choosing a set of facts, which will not lower the heuristical estimates too much, is proposed, implemented, and evaluated, showing promising results in *pegsol* domain from IPC dataset from 2011.

Keywords: planning, heuristics, STRIPS

Supervisor: Ing. Daniel Fišer

Abstrakt

Heuristiky kritických cest jsou dobře prozkoumanou oblastí z oboru klasického plánování. Tyto heuristiky jsou označovány jako h^m, kde m je maximální velikost množiny faktů použitých při výpočtu. Tato práce poskytuje efektivní implementaci heuristik h^2 a h^3. Toho je dosáhnuto vyjádřením plánovacího problému Π pomocí alternativní reprezentace problému Π^m. Byla provedena analýza těchto heuristik ve srovnání s ostaními *state-of-the-art* heuristikami. Heuristiky porovnáváme na základě heuristických hodnot v počítečních stavech, počtu vyřešených problémů v datasetech z International Planning Competition a počtu vyhodnocených stavů za vteřinu. Dále byla navržena nová charakterizace plánovacího problému, Π^m_r. Tato charakterizace umožňuje volbu množiny faktů, které nemohou být použity pro vytváření meta faktů v Π^m, zmenšujíc tak velikost tohoto problému. Nakonec byla navržena strategie pro výběr takové množiny faktů, že její použití příliš nesníží hodnoty heuristických odhadů. Tato strategie byla naimplemenována a vyhodnocena, se slibnými výsledky v doméně *pegsol* z IPC datasetu z roku 2011.

Klíčová slova: plánování, heuristika, STRIPS

Překlad názvu: Experimentální analýza heuristik kritických cest

Contents

Figures

Tables

Chapter 1
Introduction

The goal of this thesis is to provide effective implementations of h^2 and h^3 from the h^m family of heuristics, along with the general h^m heuristic. h^m, introduced by Haslum and Geffner [10], is a generalization of the standard h^{max} heuristic. Instead of considering reachability of single facts, h^m works with combinations of facts with the size of at most m. The computational complexity of h^m is exponential in m and h^m is thus rarely used for $m \geq 2$. However, the h^2 and h^3 (and h^m in general) are not bounded by h^+, that is, the cost of the optimal plan in the relaxed problem. For sufficiently large m, h^m even equals the cost of an optimal plan. This is the motivation for an effective implementation of these heuristics.

We create the implementation utilizing the alternative characterization of the planning task Π^m, introduced by Haslum [7]. This allows computing h^m heuristics as h^1 of the modified planning task Π^m. We experimentally evaluate h^2 and h^3 on datasets used in International Planning Competition from years 2011 and 2014. We compare the heuristics with other *state-of-the-art* heuristics, the LM-Cut heuristic [12], flow-based heuristic [1, 2] and potential heuristic [16].

We compare the heuristics in terms of heuristic values in initial states, the number of solved problems in the International Planning Competition datasets (coverage) and the number of evaluated states per second. We point out the strengths and weaknesses of these heuristics.

Based on the idea of Π^m [7] and Π^C [8], we propose a new characterization of the planning task, Π^m_r. It acts as a restriction of the regular Π^m characterization of the task, as it allows for a choice of a set of facts which cannot be used in combinations of facts represented by meta-facts. We experimentally show that a proper choice of the restriction set r for Π^2_r keeps the same heuristical estimates as for regular Π^2.

Finally, we propose a strategy for choosing a set of facts, which will keep the heuristical estimates reasonably high. The strategy is implemented and evaluated, showing promising results in *pegsol* domain from IPC dataset from 2011, but not performing very well in other domains.

The structure of this thesis is following: in Chapter 2 we introduce definitions and establish the background necessary for this thesis, presenting several existing characterizations of the planning task Π. In Chapter 3 the implementations of h^2, h^3 and h^m are described. Two different approaches are presented. The implementations of heuristics from Chapter 3 are then experimentally evaluated in Chapter 4. In Chapter 5 we propose the new characterization Π_r^m. Finally, in Chapter 6 we summarize the work.

Chapter 2
Background

2.1 Domain-independent Planning

Domain independent planning is a field which focuses on techniques used for solving planning problems without any specific knowledge about the particular domain of the problem. In the following sections we establish necessary background needed for this thesis.

2.2 Definitions

2.2.1 STRIPS Planning Task

Definition 2.1. A STRIPS [3] planning task Π is a tuple $\langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$, where $\mathcal{F} = \{f_1, f_2, \dots, f_n\}$ is a set of facts and $\mathcal{O}$ is a set of operators. State $s \subseteq \mathcal{F}$ is a set of facts. We say that fact f holds or is true in a state s, if $f \in s$. $s_{init} \subseteq \mathcal{F}$ is the initial state and $s_{goal} \subseteq \mathcal{F}$ is a goal specification.

Operator $o \in \mathcal{O}$ is a quadruple $\langle \mathrm{pre}(o), \mathrm{add}(o), \mathrm{del}(o), \mathrm{cost(o)} \rangle$, where $\mathrm{pre}(o) \subseteq \mathcal{F}$ is a set of preconditions, $\mathrm{add}(o) \subseteq \mathcal{F}$ are add effects and $\mathrm{del}(o) \subseteq \mathcal{F}$ are delete effects. $\mathrm{cost}(o) \in \mathbb{R}_0^+$ is a cost of applying the operator o. All operators are well-formed, i.e., $pre(o) \cap add(o) = \emptyset$ and $add(o) \cap del(o) = \emptyset$.

Operator o is applicable in a state s if $\mathrm{pre}(o) \subseteq s$. The resulting state of application of o on s is $o[s] = (s \setminus \mathrm{del}(o)) \cup \mathrm{add}(o)$. State s is called a goal state iff $s_{goal} \subseteq s$. A sequence of operators $\pi = \langle o_1, \dots, o_n \rangle$ is applicable in s_0 if there are states $s_1, \dots, s_n$ such that o_i is applicable in s_{i-1} and $s_i = o[s_{i-1}]$ for $1 \leq i \leq n$. $\pi[s_0] = s_n$ is then the resulting state of applying the sequence on s_0. Sequence of operators π is called a plan iff $s_{goal} \subseteq \pi[s_{init}]$. Cost of the plan π is a sum of all its operators' costs, i.e., $\mathrm{cost}(\pi) = \sum_{o \in \pi} \mathrm{cost}(o)$. The optimal plan is the plan with the minimal cost over all plans. A state s is called reachable if there exists an applicable operator sequence π such $\pi[s_{init}] = s$. A set of all reachable states is denoted by $\mathcal{R}$. A state s is called a dead-end state iff $s \not\supseteq s_{goal}$ and there exists no sequence of operators π applicable in s such that $\pi[s] \supseteq s_{goal}$.

A simple example of a STRIPS planning task is shown in Example 2.2.

Example 2.2. Let $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$, where $\mathcal{F} = \{i, 1, 2, 3, 4, g\}$, $s_{init} = \{i\}$, $s_{goal} = \{g\}$ and $\mathcal{O}$ is given by the following table:

	pre	add	del	cost
op_1	{i}	{1,2}	{i}	1
op_2	{1,2}	{3}	{1}	1
op_3	{1,2}	{4}	{2}	2
op_4	{1}	{2}	$\emptyset$	3
op_5	{2}	{1}	$\emptyset$	3
op_6	{3,4}	{g}	$\emptyset$	4

We can see that only operator op_1 is applicable in the initial state. Sequence of operators $\pi = \langle op_1, op_2, op_5, op_3, op_4, op_6 \rangle$ is a plan, as $\pi[s_{init}] = \{1, 2, 3, 4, g\}$ and $s_{goal} \subseteq \pi[s_{init}]$ holds. The cost of the plan is 14. However, this plan is not optimal, as there are plans with lower costs, e.g. plan $\pi_{opt} = \langle op_1, op_2, op_5, op_3, op_6 \rangle$ is the optimal plan with $cost(\pi_{opt}) = 11$.

2.2.2 Heuristic Functions

A heuristic function h is function $h : \mathcal{R} \mapsto \mathbb{R}_0^+ \cup \infty$ mapping each reachable state to a positive number or infinity.

Definition 2.3. We say that h is an admissible heuristic function, if it holds for every state $s \in \mathcal{R}$ that $h(s) \leq h_{opt}(s)$, where h_{opt} is the cost of optimal plan from the state s to a goal state.

In other words, admissible heuristics are optimistic — they never overestimate the cost of reaching a goal state. This is an important property for the optimality of informed search algorithms using the heuristic function, such as A* algorithm.

Definition 2.4. Let h_1 and h_2 be admissible heuristic functions. We say that h_1 dominates h_2 if it holds for all states $s \in \mathcal{R}$ that $h_1(s) \geq h_2(s)$.

One of the consequences of h_1 dominating h_2 is a possible improvement of performance of A^* search algorithm in terms of the number of visited states [17].

2.2.3 h^m Heuristics

Let $R(\Pi)$ be a set of transitions corresponding to the backward search in planning task Π. It holds that for every transition $(s, o, s') \in R(\Pi)$ there exists an operator o in Π such that s regressed through o yields s', i.e., $s \cap \mathrm{del}(s) = \emptyset$ and $s' = (s \setminus \mathrm{add}(o)) \cup \mathrm{pre}(o)$. The cost of this transition is $\mathrm{cost}(o)$. Lets define $h^*(s)$ to be the minimum cost of any path in $R(\Pi)$ from s to any state contained in s_{init} ($h^*(s) = \infty$ if no such path exist), i.e., the cost of the optimal plan, and $h^+(s)$ to be the cost of the optimal plan in the corresponding delete-relaxed problem.

Definition 2.5. The $\mathrm{h}^m (m = 1, 2, \ldots)$ is a family of heuristics defined [9] as follows:

$$\mathrm{h}^m(s) = \begin{cases} 0 & if s \subseteq s_{init}, \\ \min_{(s,o,s') \in R(\Pi)} (\mathrm{h}^m(s') + cost(o)) & if |s| \leq m, \\ \max_{s' \subseteq s, |s'| \leq m} \mathrm{h}^m(s') & otherwise. \end{cases}$$

It holds for sufficiently high m that $\mathrm{h}^m(s) = h^*(s)$, i.e., the heuristic value equals the cost of optimal path. It also holds that for every $m_1 \geq m_2$, h^{m_1} dominates h^{m_2}.

2.2.4 Alternative Characterization of h^m

Haslum [7] proposed an alternative characterization of h^m using modified planning task:

Definition 2.6. Let Π be a planning task $\langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$. Planning task Π^m is a tuple $\langle \Phi, \Omega, \phi_{init}, \phi_{goal} \rangle$, where Φ is a set of meta-facts (meta-atoms), $\Phi = \{\phi_c \mid c \subseteq \mathcal{F}, |c| \leq m\}$, i.e., each meta-fact corresponds to a set of facts from Π of size at most m. The inital state $\phi_{init} = \{\phi_c \mid c \subseteq s_{init}, |c| \leq m\}$ and goal specification $\phi_{goal} = \{\phi_c \mid c \subseteq s_{goal}, |c| \leq m\}$ are defined analogously.

For each operator $o \subseteq \mathcal{O}$ and for each set of facts $f \subseteq \mathcal{F}$, $|f| \leq m - 1$ and $f \nsubseteq \mathrm{add}(o) \cup \mathrm{del}(o)$, Π^m contains a meta-operator $\omega_{o,f} \in \Omega$:

$\mathrm{pre}(\omega_{o,f}) = \{\phi_c \mid c \subseteq (\mathrm{pre}(o) \cup f),\ |c| \leq m\}$,
$\mathrm{add}(\omega_{o,f}) = \{\phi_c \mid c \subseteq (\mathrm{add}(o) \cup f),\ c \cap \mathrm{add}(o) \neq \emptyset,\ |c| \leq m\}$
$\mathrm{del}(\omega_{o,f}) = \emptyset$, and
$\mathrm{cost}(\omega_{o,f}) = \mathrm{cost}(o)$.

It holds that $h^1(\Pi^m) = h^m(\Pi^1)$, which allows to compute the h^m value as h^1 of the compiled task. However, $h^*(\Pi^m) \neq h^*(\Pi^1)$, which means that applying an arbitrary admissible heuristic to Π^m does not necessarily yield an admissible estimate for Π. This is shown in Example 2.8. In the Example 2.7 we show a principle of Π^m construction.

Example 2.7. Recall Example 2.2. We will show some steps of Π^m construction on that example, in this case for $m = 2$. In the original planning task Π, $\mathcal{F} = \{i, 1, 2, 3, 4, g\}$. In the corresponding problem Π^2, Φ consists of meta-facts corresponding to all subsets of $\mathcal{F}$ of size 2 and smaller, i.e., $\mathcal{F} = \left\{\phi_{\{i\}}, \phi_{\{1\}}, \ldots, \phi_{\{i,1\}}, \phi_{\{i,2\}}, \ldots, \phi_{\{4,g\}}\right\}$. The same applies for $\phi_{init} = \left\{\phi_{\{i\}}\right\}$ and $\phi_{goal} = \left\{\phi_{\{g\}}\right\}$.

We will demonstrate the construction of meta operators in Π^2 on operator op_2. This is the operator from original task:

	pre	**add**	**del**	**cost**
op_2	{1,2}	{3}	{1}	1

We create a new meta operator for every $f \subseteq \mathcal{F}$ satisfying conditions from definition 2.7. Here for $f = \emptyset$:

	pre	add	del	cost
$\omega_{op_2,\emptyset}$	$\{\phi_{\{1\}}, \phi_{\{2\}}, \phi_{\{1,2\}}, \}$	$\phi_{\{3\}}$	$\emptyset$	1

And for $f = \{4\}$:

	pre	add	del	cost
$\omega_{op_2,\{4\}}$	$\{\phi_{\{1\}}, \phi_{\{2\}}, \phi_{\{4\}}, \phi_{\{1,2\}}, \phi_{\{1,4\}}, \phi_{\{2,4\}}\}$	$\phi_{\{3\}}, \phi_{\{3,4\}}$	$\emptyset$	1

Note that the meaning of applying this operator could be understood as simultaneously making the effect of operator ω_{op_2} true while also preserving the truth of fact f. This observation is closely related to Example 2.8.

Example 2.8. Consider the task Π^2, constructed in Example 2.7, and delete-relaxed task Π from Example 2.2. Consider state $s = \{1, 4\}$ in task Π. In this state, the operator op_4 is applicable, with resulting state $s' = \{1, 2, 4\}$. To achieve the same effect in Π^2, two applications of operators are needed:

State $t = \{\phi_{\{1\}}, \phi_{\{4\}}, \phi_{\{1,4\}}\}$ corresponds to state s. To achieve state $t' = \{\phi_{\{1\}}, \phi_{\{2\}}, \phi_{\{4\}}, \phi_{\{1,2\}}, \phi_{\{1,4\}}, \phi_{\{2,4\}}\}$ corresponding to state s', application of meta-operators $\omega_{op_4,\{1\}}$, adding $\phi_{\{1,2\}}$, and $\omega_{op_4,\{4\}}$, adding $\phi_{\{1,4\}}$ are needed.

This can cause non-admissibility of heuristics (e.g., some types of additive heuristics, as they take into account the amount of actions needed to reach the goal) applied to the compiled task. This is however not the case for h^1, which computes the heuristic value for state s as the most expensive fact from s, and is thus not affected by this non-admissibility problem.

It is also necessary to note that the characterization itself does not simplify the complexity of computing the heuristic value [7]:

> "The new characterisation does not directly lead to a practical way of generalising an arbitrary admissible heuristic from 1 to m. Nor is it a more efficient way to compute h^m : computing $h^1([\Pi]^m)$ typically requires more time and memory than computing $h^m([\Pi])$."

The problem of non-admissibility of Π^m led to a new compilation Π^C, which solves this problem by allowing operators to make true subsets of explicitly expressed conjunctions, specified in C.

2.2.5 Π^C

There are several different definitions of Π^C, e.g., in [14] or in [8]. In this thesis a slightly adjusted definition from [8] is used, as the construction is similar to the already defined Π^m.

Definition 2.9. Let $C = \{c_1, \ldots, c_n\}$, where $|c_i| > 1$, be a set of sets of facts in planning task Π and o be an operator in Π. We define following partition of C:

$$\begin{aligned}
C^t(o) &= \{c \in C \mid c \subseteq ((pre(o) \setminus del(o)) \cup add(o)) \text{ and } c \cap add(o) \neq \emptyset\},\\
C^f(o) &= \{c \in C \mid c \cap del(o) \neq \emptyset\},\\
C^n(o) &= \{c \in C \mid c \cap del(o) = c \cap add(o) = \emptyset\},\\
C^p(o) &= \{c \in C \mid c \cap del(o) = \emptyset, c \cap add(o) \neq \emptyset \text{ and}\\
&\qquad\qquad c \nsubseteq ((pre(o) \setminus del(o)) \cup add(o))\}.
\end{aligned}$$

The $C^t(o)$ is a set of sets of facts necessarily made true by o, the $C^t(o)$ is a set of those made false by o, $C^n(o)$ are those facts on which o has no effect and finally $C^p(o)$ is a set of sets of facts possibly made true by o, depending on the state where o is applied.

Definition 2.10. Let $X \subseteq C^p(o)$. The X is then called *downward closed* iff for all $c \in X$ and $c' \in C^p(o)$ such that $c' \subseteq c$, $c' \ \in X$.

Example 2.11. Let $C = \{\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$ and $X = \{\{1,2,3\}\}$, $X \subseteq C$. The X is **not** *downward closed*, as for example $c' = \{1,2\}$, $c' \subseteq c$, $c \in C^p(o)$, but $c' \notin X$.

Definition 2.12. Let Π be a planning task $\langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal}\rangle$. Π^C has all facts of Π, and for each $c \in C$ it has a meta-fact ϕ_c. ϕ_c is initially true iff c holds in the initial state s_{init} of Π, and is in a goal iff $c \subseteq s_{goal}$. For any set of facts $X \subseteq \mathcal{F}$, let $X^C = X \cup \{\phi_c \mid c \in C, c \subseteq X\}$.

For each operator o in Π and for each set $X \subseteq C^p(o)$ that is *downward closed*, Π^C has an operator $\alpha_{o,X}$ with

$$\begin{aligned}
pre(\alpha_{o,X}) &= \left(pre(o) \cup \bigcup_{c \in X} (c \setminus add(o))\right)^C\\
add(\alpha_{o,X}) &= add(a) \cup \{\phi_c \mid c \in C^t(o) \cup X\}\\
del(\alpha_{o,X}) &= \emptyset\\
cost(\alpha_{o,X}) &= cost(o).
\end{aligned}$$

The operator o is called the *original operator* of all $\alpha_{o,X}$. Operator $\alpha_{o,X}$ is called *representative of o* of o is *original operator* of $\alpha_{o,X}$.

The adjustment, which differentiates our definition from the original one [8], is made in the construction of delete effects of representatives of operators—as it is dealt only with delete-relaxed problems in this thesis, it is unnecessary to define them. This approach of empty delete effects is also used in the definition in [14].

The Π^C grows potentially exponentially in $|C|$ (that is, in number of conjunctions, not in their size), as it creates new operators for each *downward closed* subset of C. There exists another compilation Π^C_{ce} [8] which introduces conditional effects to Π^C, resulting in linear growth in $|C|$. This is however out of the scope of this thesis. In Example 2.13 we show how Π^C deals with the non-admissibility problem of Π^m shown in Example 2.8.

Example 2.13. Recall Example 2.8. Only one application of an operator was needed to achieve state s' from s in the original planning task Π. However, two applications of operators were necessary to achieve the same situation in corresponding states t' and t in the planning task Π^2. We will now show the same situation in the Π^C compilation.

Let Π be a delete-relaxed planning task from Example 2.2. Let $C = \{\{f_1, f_2\} \mid f_1, f_2 \in \mathcal{F}, f_1 \neq f_2\}$ and let Π^C be a properly formed planning task according to the Definition 2.12. The C contains all pairs of facts from the original task and each is represented by its own meta-fact $\phi_{\{f_1,f_2\}}$. This resembles the Π^2 compilation, but the main difference is in the definition of operator's representatives.

Consider states $s = \{1, 4\}$ and $s' = \{1, 2, 4\}$ from Example 2.8. Corresponding states in Π^C would be $u = \{1, 4, \phi_{\{1,4\}}\}$ and $u' = \{1, 2, 4, \phi_{\{1,2\}}, \phi_{\{1,4\}}, \phi_{\{2,4\}}\}$, respectively. To achieve the state u', only one application of operator's representative is needed, and that is the application of $\alpha_{op_4,\{\phi_{\{1,2\}},\phi_{\{2,4\}}\}}$, adding facts $2, \phi_{\{1,2\}}$ and $\phi_{\{2,4\}}$ at the same time.

Chapter 3

Implementation

3.1 Overview

Several versions of algorithms were implemented for this thesis. As the idea of h^1 (or, alternatively, h^{max}) is important for the computation of $h^1(\Pi^m)$, it was implemented first. Two versions of h^2 were implemented, both using the alternative characterization Π^2 of the task. The first one uses brute force approach, the second one uses more efficient approach of using a priority queue. Two versions of h^3 were implemented in the same manner. Finally, the general h^m was implemented, which allows for computation of h^m value for any $m \geq 1$, although its practical usage is limited, as the computation times and memory requirements are too high.

3.1.1 General Implementation Information

The implementation was written in C language and integrated into the MAPlan planner [6], which provides the problem specification in FDR representation [11], so it is necessary to translate it to STRIPS first. This is easily done, as illustrated in the following example:

Example 3.1. Consider simple problem specified in FDR with one variable v with domain $d(v) = \{1, 2, 3\}$. This means the variable v can have three different values, one of $\{1, 2, 3\}$. To translate this to STRIPS facts, we create three facts $\{v_1, v_2, v_3\}$, which corresponds to the variable holding specific value, e.g., v_2 corresponds to the variable v having value 2. It is clear that each assignment $(variable, value)$ corresponds to exactly one STRIPS fact. This allows for easy translation of operators too.

It is also necessary to take into account the fact, that only one fact from $\{v_1, v_2, v_3\}$ can hold at the same time. This is solved by altering the delete effects of operators, e.g., when an operator has v_1 as precondition and v_2 as effect, the v_1 has to be present in delete effects of this operator. However, as we deal with delete-relaxed problems, this can be omitted. In the following sections it is assumed that the problem is already translated from FDR to STRIPS in the way described above.

The implementation also uses the Boruvka library [4], from which the implementation of priority queue and memory management functions were used.

3.1.2 Project Structure

The MAPlan planner offers a simple way of implementing new heuristic by defining the interface of methods which need to be implemented in order for the heuristic to be used in the planner.

Following list shortly describes all source files created and used for this thesis.

fact_conv.c and *fact_conv.h*	— functions used for translation between STRIPS and FDR representation
hmax.c	— implementation of h^1
simpleh2.c	— brute force implementation of h^2
h2.c	— optimized implementation of h^2
simpleh3.c	— brute force implementation of h^3
h3.c	— optimized implementation of h^3
hm.c	— implementation of general h^m heuristics
hmtable.c and *hmtable.h*	— implementation of hash table used for storing facts along with their values

These files are located in *src* directory of the *maplan* project.

3.2 h^1 Heuristic

h^1 could be implemented just by using the recursive equations from Definition 2.5. This would however be very inefficient, as this recursive definition leads to unnecessarily many repeated computations. Therefore, different approach was used.

One of the basic approaches to the implementation of h^1 would be to take the state s for which we want to compute the heuristic value and gradually try to apply all operators, adding new facts to this state. This is done until reaching a fixpoint, where no operators can be applied anymore. A value, whose meaning is the cost of achieving this fact, is stored for each fact. Initially, facts in the s get the initial value of 0, others get infinity or some flag of a not visited state. Every time a operator is applied, values of added facts from operator's effects are updated if the cost of operator plus the maximal value of facts from preconditions of operator is smaller then the current value of added fact. The $h^1(s)$ value is then computed as the maximum of all values of the facts in s_{goal}. The pseudo-code is shown in Algorithm 1.

The following Algorithm 2 is taken from [5] and it is the one used for the improved implementation. It uses priority queue for ordering the facts based on their value, with lowest values having higher priority. It requires some

Algorithm 1 h^1 simple

```
Input: $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$, state $s$
Output: $h^1(s)$
 1: Initialize values for all facts $f \in \mathcal{F}$: $V(f) \leftarrow 0$ iff $f \in s$ and $V(f) \leftarrow \infty$
        otherwise
 2: $currState \leftarrow s$
 3: $changed \leftarrow True$
 4: while $changed$ do
 5:     $changed \leftarrow False$
 6:     for all $o \in \mathcal{O}$ do
 7:         if $o$ is applicable in $currState$ then
 8:             for all $f \in \mathrm{add}(o)$ do
 9:                 /** Get the maximal value of precondition of $o$ **/
10:                 $maxPre \leftarrow \max_{f_{pre} \in pre(o)}(V(f_{pre}))$
11:                 if $V(f) > maxPre + cost(o)$ then
12:                     $V(f) \leftarrow maxPre + cost(o)$
13:                     $currState \leftarrow currState \cup f$
14:                     $changed \leftarrow True$
15:                 end if
16:             end for
17:         end if
18:     end for
19: end while
20: return $\max_{f \in s_{goal}}(V(f))$
```

pre-computations—instead of set of precondition facts, each operator stores only the number of unsatisfied precondition facts. Also every fact stores list of operators, in whose preconditions it is in. Finally, new fact *goal* and new operator op_{goal} adding this fact is introduced, with preconditions being the facts from s_{goal} and cost zero. This allows for representation of the goal specification by a single fact while not changing any plan.

Initially, facts from the state s are inserted into the queue with the value 0. When a fact is popped from the priority queue, all operators, whose preconditions contain this fact, have their counter of unsatisfied preconditions decreased by 1. This is an important point of the algorithm (line 24 in Algorithm 2)—when an operator achieves 0 of unsatisfied preconditions, the value of the last fact that satisfied operator's preconditions, is the maximum value of all operator's preconditions. This is thanks to the priority queue, as all the facts, which have previously decreased the counter, must have had lower value, otherwise they would not be popped from the priority queue. This is an advantage compared to Algorithm 1, because we do not need to repeatedly check for the applicability of the operators, as the applicable operators are determined by the zero value of the counter.

After achieving zero in the counter of unsatisfied preconditions, values of added effects are updated in the priority queue. The values can only be decreased. When the *goal* fact is popped from the priority queue, the algorithm terminates and returns the value of the *goal* fact. This value is h^1 value of the input state s. If at no point of the algorithm the *goal* fact is popped, the input state s is a dead-end state and ∞ is returned.

Note that the algorithm resembles Dijkstra's shortest path algorithm.

Algorithm 2 h^1 (h^{max})

```
Input: $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$, state $s$
Output: $h^1(f)$
 1: Initialize min prio. queue PQ.init($\{(f, 0) \mid f \in s\} \cup \{(f, \infty) \mid f \in \mathcal{F} \setminus s\}$);
 2: Initialize number of unsatisfied preconditions $U(o) \leftarrow |pre(o)|, \forall o \in \mathcal{O}$;
 3: while not PQ.empty do
 4:     /** Pops the element(fact) with lowest $h^1(f)$ **/
 5:     $(f, h^1(f)) \leftarrow$ PQ.pop()
 6:     if $f$ is goal then
 7:         return $h^1(f)$
 8:     end if
 9:     for all $o \in \mathcal{O}, f \in pre(o)$ do
10:         $U(o) \leftarrow U(o) - 1$
11:         if $U(o) = 0$ then
12:             for all $g \in add(o)$ do
13:                 if $(h^1(f) + cost(o)) <$ PQ.getValue($g$) then
14:                     PQ.update($(g, h^1(f) + cost(o))$)
15:                 end if
16:             end for
17:         end if
18:     end for
19: end while
20: /** The goal fact was not achieved from $s$ **/
21: return $\infty$
```

3.3 h^2 Heuristic

For the h^2 heuristic the alternative characterization described in 2.2.4 is used, along with the fact that $h^1(\Pi^m) = h^m(\Pi^1)$. The planning task Π is expanded to Π^2 and then the h^1 heuristic is applied. It became clear that it is not necessary to create complete Π^2, as some set of facts are unreachable—for those sets are the corresponding meta-facts useless and their elimination can improve the performance of the algorithm. Before this, the brute force solution was implemented, which served as a baseline for performance testing and a basis for improvement.

3.3.1 Brute Force Implementation

The brute force implementation uses the same idea as Algorithm 1, which is extended to Π^2. We, however, avoid the explicit construction of meta-operators, as the preconditions and effects of them are determined during the run of the algorithm.

Algorithm 3 Brute force h^2

```
Input: $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$, state $s$
Output: $h^2(s)$
1: Init doubles (fact combination, values): $(\{(\{c\}, 0) \mid c \subseteq s, 1 \leq |c| \leq 2\} \cup \{(\{c\}, \infty) \mid c \subseteq \mathcal{F} \setminus s, 1 \leq |c| \leq 2\})$
2: $changed \leftarrow True$
3: while $changed$ do
4:     $changed \leftarrow False$
5:     for all $o \in \mathcal{O}$ do
6:         $pres \leftarrow \{\{c\} \mid c \subseteq pre(o), 1 \leq |c| \leq 2\}$
7:         if all $comb$ in $pres$ have value set then
8:             $maxPre \leftarrow max_{c \in pres}$ getValue($c$)
9:             /**update can only lower the existing value**/
10:            update($\{(\{c\}, maxPre + cost(o)) \mid c \subseteq add(o), 1 \leq |c| \leq 2\}$)
11:            if some $value$ was changed then
12:                $changed \leftarrow True$
13:            end if
14:        end if
15:        /**Get facts available for extension of the operator $o$**/
16:        $avail \leftarrow \{f \mid f \in \mathcal{F}, (f \cap add(o)) = \emptyset\}$
17:        for all $f$ in $avail$ do
18:            $fPres \leftarrow \{c \mid c \subseteq (pre(o) \cup f), 1 \leq |c| \leq 2\}$
19:            if all $fPre$ in $fPres$ have $value$ set then
20:                $maxfPre \leftarrow max_{fPre \in fPres}$ getValue($fPre$)
21:                update($\{(\{c\}, maxfPre + cost(o)) \mid c \subseteq (add(o) \cup fSet), 1 \leq |c| \leq 2, c \cap add(o) \neq \emptyset\}$
22:                if some $value$ was lowered then
23:                    $changed \leftarrow True$
24:                end if
25:            end if
26:        end for
27:    end for
28: end while
29: return $\max_{c \subseteq s_{goal}, 1 \leq |c| \leq 2}$ getValue($c$)
```

3.3.2 Conversion to Π^2

Conversion to Π^2 requires to create a meta-fact for every single fact and every pair of facts from $\mathcal{F}$. This is simply done by enumerating and creating a representative for every subset of $\mathcal{F}$ of size one and two. It, however, proved very useful to preserve the information about the underlying combination being represented, as it, for example, allows to iterate through meta-facts representing combinations of size 1, which can speed up the algorithm.

For every operator o from the original task the *add(o)* and *pre(o)* sets are replaced with sets of meta-facts representing every single fact and every pair of facts from those sets, e.g., $\{1, 2\}$ becomes $\{\phi_1, \phi_2, \phi_{1,2}\}$. This corresponds to the meta-operator from Definition 2.2.4 with $f = \emptyset$, i.e., $\omega_{o,\emptyset}$. We refer to this meta-operator's precondition and effects as *simple preconditions* and *simple effects* of the operator o.

Instead of creating all the new meta-operators as in Definition 2.2.4, it is enough to extend *simple preconditions* and *simple effects* for every $o \in \mathcal{O}$. Note that the fulfillment of *simple preconditions* of o is necessary condition for application of any meta-operator that extends the original operator o with non-empty f. There are only two possible sizes of f in Π^2, 0 and 1. The case of $|f| = 0$ is already covered by *simple preconditions* and *simple effects*. We therefore need to represent n additional meta-operators, where $n = |\mathcal{F}|$, i.e., the number of meta-facts representing single facts from the original Π task.

For that we need to identify the meta-facts unique to preconditions of meta-operator, i.e., $\{pre(\omega_{o,\{f\}}) \setminus pre(\omega_{o,\emptyset})\}$, where $f \neq \emptyset$, as the *simple preconditions* of o are a subset of precondition set of every meta-operator extending corresponding operator o with $f \neq \emptyset$. This is easily done, as those unique meta-facts are the ones representing pairs of f and every single fact precondition of *simple preconditions* of o, and meta-fact representing the fact f itself. These sets of meta-facts are further referred to as *extended preconditions* for f of operator o. Analogously, the sets $\{add(\omega_{o,\{f\}}) \setminus add(\omega_{o,\emptyset})\}$, where $f \neq \emptyset$, are called *extended effects* for f of operator o.

We can now put together the previously mentioned sets of *simple preconditions and effects* and *extended preconditions and effects* into one *operator representative*, whose logic would work as shown in Algorithm 4.

Algorithm 4 Operator's representative satisfaction logic in Π^2

```
Input: Operator $o$, task $\Pi^2$
1: $op \leftarrow$ operator representative of $o$
2: if $op.simplePreconditions$ are $satisfied$ then
3:     apply $\omega_{o,\emptyset}$
4:     for all $f \in \mathcal{F}$ do
5:         if $op.extendedPreconditions(f)$ are $satisfied$ then
6:             apply $\omega_{o,\{f\}}$
7:         end if
8:     end for
9: end if
```

It is now possible to express the preconditions as the size of the precondition set and move the information to meta-fact representatives, as in the h^1 implementation. The meta-fact representative, however, has to remember its role in the *operator representative's* unsatisfied counters, as the meta-fact representative can point to the extended *operator representative's simple precondition's* counter or to the *extended precondition's* counter. In the second case it is also necessary to remember, for which f was the meta-fact created.

This representation helps to keep the number of operators in the task low and also saves the memory, as the introduction of all meta-operators would bring a redundancy of $pre(\omega_{o,\emptyset})$ being the subset of $pre(\omega_{o,\{f\}})$ for every $o \in \mathcal{O}$ and $f \in \mathcal{F}, |f| = 1$.

For a better illustration of the idea let us show the structure of the actual C structure from implementation representing an *operator representative* (note that in the implementation single fact representatives are identified by ids from 0 to $n = (|\mathcal{F}| - 1)$):

field	**type**	**meaning**
pre_size	int	number of *simple preconditions*
pre_unsat	int	number of unsatisfied *simple preconditions*
pre_size2	int array	pre_size2 [f] = size of *extended preconditions* for f
pre_unsat2	int array	pre_unsat2[f] = number of unsatisfied *extended preconditions* for f

The C structure of a meta-fact representative looks as follows:

field	**type**	**meaning**
pre_op	pointer to operator r.	operator r. having this fact in its *simple preconditions*
pre_extop	list of pointers to op r.	list of operator r. having this fact in *extended preconditions*
pre_extop_f	list of fact ids	list of ids of single fact representatives identifying f set defining *extended precondition*

Also note that it is not necessary to create *extended preconditions and effects* for every fact $f \in \mathcal{F}$, as some facts cannot be used for a creation of meta-operators due to breaking the constraints in Definition 2.7. The complete h^2 algorithm is shown in Algorithm 5.

Algorithm 5 h^2

```
Input: $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$, state $s$
Output: $h^2(s)$
1: Construct $\Pi^2 = \langle \Phi, \emptyset, \phi_{init}, \phi_{goal} \rangle$, $\phi_s = \{\phi_c \mid c \subseteq s, |c| \leq 2\}$
2: Construct set of *operator representatives* $\mathcal{OR}$
3: Initialize number of unsatisfied *simple preconditions* $UnsatSP(or)$ and
       *extended preconditions* $UnsatEP(or)$, $\forall or \in \mathcal{OR}$
4: Initialize lists of pointers to counters of $or$, $SP(\phi), EP(\phi)$ ,$\forall \phi \in \Phi$
5: Initialize min prio. queue PQ.init($\{(\phi, 0) \mid \phi \in \phi_s\} \cup \{(\phi, \infty) \mid \phi \in$
       $\Phi \setminus \phi_s\}$);
6: while not PQ.empty do
7:     $(\phi, h^2(\phi)) \leftarrow$ PQ.pop()
8:     if $\phi$ is *goal* then
9:         return $h^2(\phi)$
10:    end if
11:    for all $or \in SP(\phi)$ do
12:        $UnsatSP(or) \leftarrow UnsatSP(or) - 1$
13:        if $UnsatSP(or) = 0$ then
14:            update values of $or$.*simpleEffects* in PQ
15:            for all $f \in \mathcal{F}$ do
16:                if $UnsatEP(or) = 0$ then
17:                    update values of $or$.*extendedEffects(f)* in PQ
18:                end if
19:            end for
20:        end if
21:    end for
22:    for all $or \in EP(\phi)$ do
23:        $UnsatEP(or) \leftarrow UnsatEP(or) - 1$
24:        if $UnsatEP(or) = 0$ then
25:            /**Get $f$, for which the $\phi$ was ext. precond. of $or$**/
26:            $fSetFact \leftarrow getfSetFact(\phi, or)$
27:            update values of $or$.*extendedEffects($fSetFact$)* in PQ
28:        end if
29:    end for
30: end while
31: return $\infty$
```

3.3.3 Mutex Elimination

Mutexes are sets of facts that are mutually exclusive, i.e., facts, which cannot hold at the same time. The Π^2 construction creates meta-facts representing pairs of facts, from which some are clearly unreachable. This follows from the underlying FDR representation of the problem. As shown in Example 3.1, FDR variable translates to r STRIPS facts, where r is the size of the variable's domain. From those r facts only one can hold at the same time, which means that any meta-fact that corresponds to a pair of facts from those r facts is unreachable and can be completely omitted from the Π^2 task. This can significantly reduce the size of the Π^2 task, as each meta-fact representative can possibly contain significant amount of information.

Mutexes and the impact of their elimination on the planning task are furthermore explored in Chapter 4.

3.4 h^3 Heuristic

Two implementations of h^3 were created, the first with a brute force approach (used as a baseline for performance testing) and the second one with adjusted algorithm using priority queue, a variation of the already implemented algorithm from h^1 and h^2 implementation.

3.4.1 Brute Force Implementation

At first the brute force force solution was implemented. The idea is the same as in the brute force h^2. Instead of explicitly constructing the Π^3 task, we compute the heuristic on the original planning task Π, while constructing the Π^3 meta-operators and meta-facts on the go, i.e., instead of explicitly creating a meta-fact representative for a set of facts beforehand and working with it as a regular facts, we only work with facts from Π and we create combinations of these facts when needed.

However, a problem arises with the need to store values along with the facts and fact combinations, as the number of all combinations is $n(n-1)(n-2)/6$, where n is the size of $\mathcal{F}$ in Π. For a lot of problems in the IPC domains used for experiments in Chapter 4 it was not possible to store this amount of values in a simple table within the memory limits. For this reason, the values of combinations are being stored in a hash table. Initially, only the combinations of facts from initial state are inserted. Other combinations are inserted gradually, as they are encountered for the first time during the computation. This proved quite useful. Usually, not all combinations are encountered during the computation of heuristic.

We do not list the pseudo-code for this algorithm, as it is basically the same as Algorithm 7 for $m = 3$.

3.4.2 Improved Implementation

The same idea with using the priority queue as in Algorithm 2 and Algorithm 5 was used. We expand the task to the Π^3 compilation and apply the h^1 heuristic to it. All combinations of facts from the original task of size at most 3 are enumerated a represented by meta-facts. From these then the mutexes are eliminated, as described in Section 3.3.3, only with the extension to triplets of facts. We apply the same transformations as in Section 3.3.2, that means introduction of a new meta-fact representing the goal specification, which is reachable only by a new goal meta-operator, whose preconditions are the meta-facts of the original goal state.

The Π^3 is constructed in analogous way to Section 3.3.2. The idea of keeping all the newly created meta-operators from one operator in one *operator representative*, with hierarchically checked preconditions can by applied here too. Consider the following observation.

Let $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$ denote a planning task. Let $\Pi^m = \langle \Phi, \Omega, \phi_{init}, \phi_{goal} \rangle$ denote a compilation of Π from Definition 2.6. It holds for every pair of facts $f_1, f_2 \in \mathcal{F}, f_1 \neq f_2$ and for every $o \in \mathcal{O}$ that $pre(\omega_{o,\emptyset}) \subseteq pre(\omega_{o,\{f_1\}})$ and $pre(\omega_{o,\emptyset}) \subseteq pre(\omega_{o,\{f_2\}})$. It also holds that $pre(\omega_{o,\{f_1\}}) \subseteq pre(\omega_{o,\{f_1,f_2\}})$, $pre(\omega_{o,\{f_2\}}) \subseteq pre(\omega_{o,\{f_1,f_2\}})$ and $(pre(\omega_{o,\{f_1,f_2\}}) \setminus pre(\omega_{o,\{f_1\}})) \setminus pre(\omega_{o,\{f_2\}}) = \{\phi_{\{f_1,f_2\}}\} \cup \{\phi_{\{f_1,f_2,p\}} \mid p \in \mathrm{pre}(o), p \neq f_1, p \neq f_2\}$.

It follows from this observation that it is again possible to create one *operator representative* for each operator in the original task. Let $o \in \mathcal{O}$ and $f_1, f_2 \in \mathcal{F}, f_1 \neq f_2$. By *simple preconditions* of o we then mean the set $pre(\omega_{o,\emptyset})$, by *extended preconditions* for o and f_1 the set $pre(\omega_{o,\{f_1\}}) \setminus pre(\omega_{o,\emptyset})$ and by the *double extended preconditions* for o and facts f_1, f_2 the set $pre(\omega_{o,\{f_1,f_2\}}) \setminus (pre(\omega_{o,\{f_1\}}) \cup pre(\omega_{o,\{f_2\}}))$.

This *operator representative* then consists of *simple preconditions*, n *extended preconditions* and $\binom{n}{2}$ *double extended preconditions*, where $n = |\mathcal{F}|$. The logic of applying the *operator representative* is illustrated in Algorithm 6.

Algorithm 6 Operator's representative satisfaction logic in Π^3

```
Input: Operator $o$, task $\Pi^3$
 1: $op \leftarrow$ operator representative of $o$
 2: if $op.simplePreconditions$ are $satisfied$ then
 3:     apply $\omega_{o,\emptyset}$
 4:     for all $f_1 \in \mathcal{F}$ do
 5:         if $op.extendedPreconditions(f_1)$ are $satisfied$ then
 6:             apply $\omega_{o,\{f_1\}}$
 7:             for all $f_2 \in \mathcal{F}, f_1 \neq f_2$ do
 8:                 if $op.extendedPreconditions(f_2)$ are $satisfied$ and
                       $op.doubleExtendedPreconditions(f_1, f_2)$ are
                       $satisfied$ then
 9:                     apply $\omega_{o,\{f_1,f_2\}}$
10:                 end if
11:             end for
12:         end if
13:     end for
14: end if
```

During the construction of *operator representatives* for all operators of the original task, the same transformation as in Section 3.3.2 is applied, that means that instead of each set of *preconditions*, only the size of the set is stored in the *operator representative.* Every meta-fact from those sets then contains pointers to those *operator representative precondition's* counters.

However, because of the need of construction of the task and the way of doing it, it is not possible to use the same method of storing the meta-fact's values in a hash table gradually (as in 3.4.1). All of the meta-facts (except for the meta-facts representing mutexes) are needed at the time of construction, as they carry references to the *operator representative's* counters. This brings problems with the memory requirements, as for many problems it is not possible to hold all meta-facts representatives in the memory at the same time.

The actual algorithm is very similar to Algorithm 5, but instead of checking and decreasing of counters of unsatisfied preconditions only for *simple preconditions* and *extended preconditions*, the counter for *double extended preconditions* is introduced. The conditions for applying an *operator representative* are then analogous to the principle introduced in Algorithm 6.

3.5 h^m Heuristic

For the implementation of h^m, the brute force approach analogous to Algorithm 1 was used, along with the Π^m representation of the task. As the algorithm has to be general for any m, it is difficult to generalize any of the improvements introduced in the implementations of h^2 and h^3. However, this general implementation was not meant for efficient and fast computation of h^m values, but rather as a tool for obtaining information about h^m behaviour for values of $m > 3$.

For the storage of facts along with their values the hash table was used, and facts are inserted into the hash table gradually during the run of the algorithm. This lowers memory requirements (it is not necessary to hold all fact values in the memory at once from the beginning of computation), while keeping the access time to the stored values reasonable. Similarly, meta-facts and meta-operators of the Π^m representation are not explicitly stored in the memory, but are constructed "on the fly" from the Π task. The algorithm is described in pseudo-code in Algorithm 7.

Algorithm 7 h^m

```
Input: $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$, state $s$
Output: $h^m(s)$
 1: Insert combinations of facts into hash table HTable.insert($\{(\{c\}, 0) \mid c \subseteq s, 1 \leq |c| \leq m\}$)
 2: $changed \leftarrow True$
 3: while $changed$ do
 4:     $changed \leftarrow False$
 5:     for all $o \in \mathcal{O}$ do
 6:         $combinations \leftarrow \{\{c\} \mid c \subseteq pre(o), 1 \leq |c| \leq m\}$
 7:         if all $comb$ in $combinations$ are in HTable then
 8:             $maxPre \leftarrow max_{c \in combinations}$ HTable.getValue($c$)
 9:             HTable.updateOrInsert($\{(\{c\}, maxPre + cost(o)) \mid c \subseteq add(o), 1 \leq |c| \leq m\}$)
10:             if HTable was updated then
11:                 $changed \leftarrow True$
12:             end if
13:         end if
14:         /**Get facts available for extension of the operator $o$**/
15:         $avail \leftarrow \{f \mid f \in \mathcal{F}, (f \cap add(o)) = \emptyset\}$
16:         $fSets \leftarrow \{f \mid f \subseteq avail, 1 \leq |f| \leq (m-1)\}$
17:         for $fSet$ in $fSets$ do
18:             $fPres \leftarrow \{c \mid c \subseteq (pre(o) \cup fSet), 1 \leq |c| \leq m\}$
19:             if all $fPre$ in $fPres$ are in HTable then
20:                 $maxfPre \leftarrow max_{fPre \in fPres}$ HTable.getValue($fPre$)
21:                 HTable.updateOrInsert($\{(\{c\}, maxfPre + cost(o)) \mid c \subseteq (add(o) \cup fSet), 1 \leq |c| \leq m, c \cap add(o) \neq \emptyset\}$)
22:                 if HTable was updated then
23:                     $changed \leftarrow True$
24:                 end if
25:             end if
26:         end for
27:     end for
28: end while
29: return $\max_{c \subseteq s_{goal}, 1 \leq |c| \leq m}$(HTable.getValue($c$))
```

Chapter 4

Experiments

Experiments were performed on datasets from International Planning Competition (IPC) from years 2011 and 2014. Both datasets consists of 16 domains, each containing 20 problems, that being 560 problems in total. The goal of these experiments is to compare the implemented heuristics h^2 and h^3 with state-of-the-art heuristics. For these heuristics, the MAPLAN's [6] implementations of following heuristics were used:

- *lm-cut*: LM-Cut heuristic [12],
- *flow*: flow-based heuristic [1, 2], and
- *pot*: potential heuristic [16].

We also compare the h^2 and h^3 with h^1 heuristic, as the former two are an "extension" of h^1 and it is, therefore, useful to see the performance difference between them.

The performance of heuristics is evaluated by comparison of heuristic values in initial states, the number of evaluated states per second and the number of solved problems from given domains (coverage).

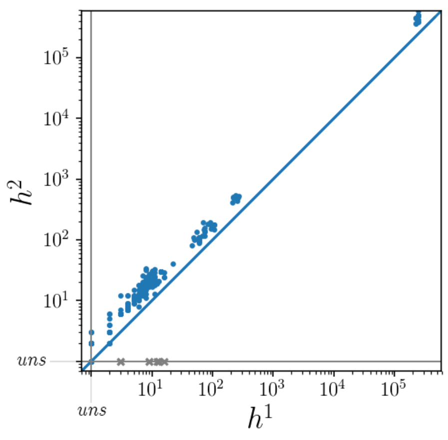
4.1 Experiment Setup

The experiments were performed using the MetaCentrum [15], which provides computational resources for remote computing. All experiments were performed on a machine with an Intel E5-2670 2.6GHz CPU, except for the h^m experiments, which were performed on a machine with an Intel Xeon E5-4617 2.9GHz CPU because of higher memory requirements of the h^m, which the first machine was not able to satisfy. Memory and time limits were chosen individually depending on the type of experiment, and are always listed in that experiment's description.

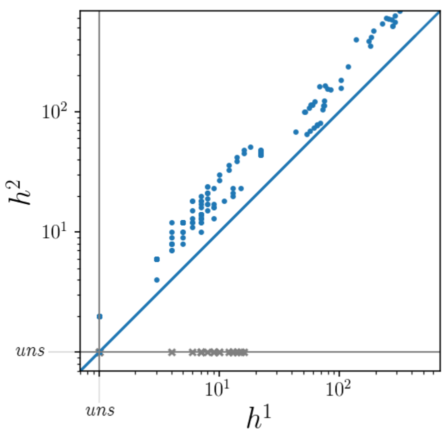
4.2 Heuristic Values in Initial States

In this experiment, only heuristic values were computed for the initial states of each problem. The experiment was carried out with the memory limit of 5 GB and the time limit of 60 minutes for each problem. Figures are shown separately for datasets from years 2011 and 2014. This is useful for the testing of the dominance of heuristics. Once computed, the values are plotted in a scatter plot in the following way. Each point in the scatter plot corresponds to one planning problem, axes corresponds to heuristics, and axis coordinates of a point correspond to the values of heuristics in the initial state. For example, with x-axis corresponding to h^2 and y-axis to h^3, a point with coordinates $[3; 5]$ corresponds to the problem, for which the values of heuristic were 3 for h^2 and 5 for h^3.

This allows for an easy observation of the heuristic dominance. Points, which are above the diagonal, have higher y coordinate and thus higher value of the heuristic corresponding to the y axis. Analogously for the points below the diagonal, for which the x coordinate is greater that the y one. Problems, for which the value in initial state could not be computed (because of time or memory limits), are plotted as gray "x" points with the coordinate of value 1 on the axis corresponding to the failed heuristic. The lines, on which the unsolved problems are plotted, are marked with the "uns" label. Since the problems within domains scale exponentially, a logarithmic scale was used.



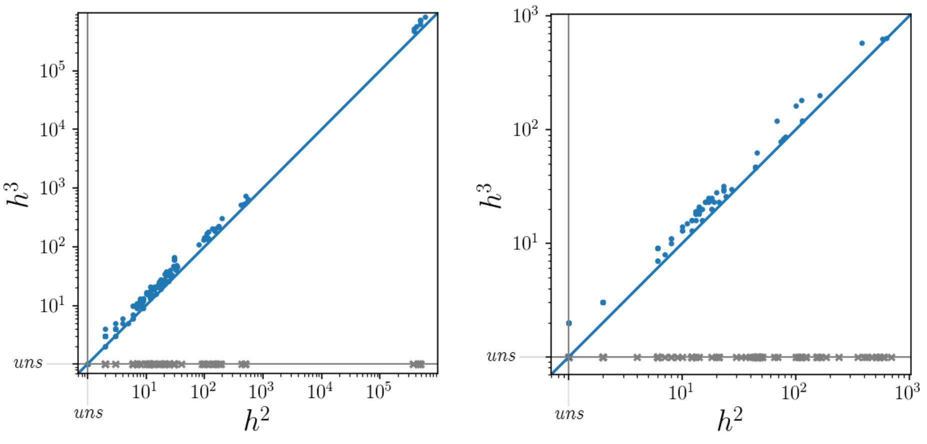
(a) : IPC 2011



(b) : IPC 2014

Figure 4.1: h^1 vs h^2 scatter plot of heuristic values in initial states

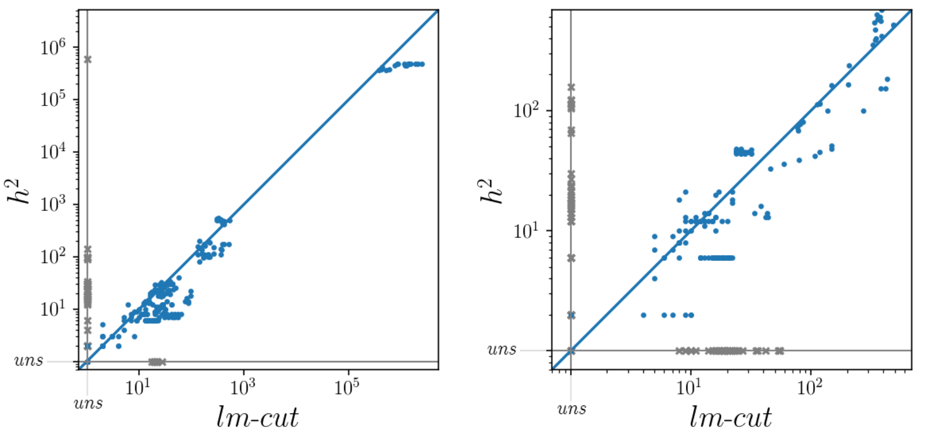
We can observe clear dominance of h^2 over h^1 in Figure 4.1. That complies with the theoretical properties of h^m. It can be however noticed, that h^1 was able to compute initial values for more problems than h^2. The h^1 computed all values in the IPC 2014 dataset and all but one in the IPC 2011 dataset. On the other hand, the h^2 failed to compute 10 and 38 values in the limits in the 2011 and 2014 datasets, respectively. This result is also expected, as h^2 is computationally more demanding then h^1.



(a) : IPC 2011 **(b) :** IPC 2014

Figure 4.2: h^2 vs h^3 scatter plot of heuristic values in initial states

Similar situation can be observed in Figure 4.2. h^3 dominates h^2, but was not able to compute as many problems (116 and 176 unsolved for h^3, 10 and 38 unsolved for h^2).



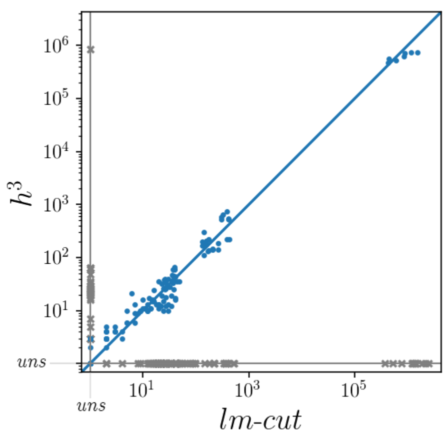
(a) : IPC 2011 **(b) :** IPC 2014

Figure 4.3: h^2 vs *lm-cut* scatter plot of heuristic values in initial states

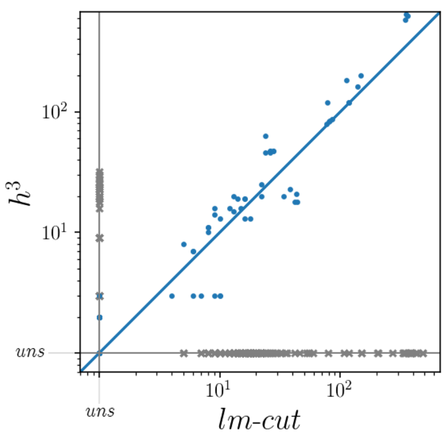
Figures 4.3 and 4.4 depict the comparison between h^2 and *lm-cut*. There is no clear dominance in neither case.

4.3 Coverage

In the coverage experiment, we test how many problems is the MAPlan planner able to solve with given heuristics and with given time and memory limits using the A^* algorithm. The time limts and memory limits for one problem are 60 minutes and 5 GB, respectively. We test the coverage for h^2,



(a) : IPC 2011



(b) : IPC 2014

Figure 4.4: h^3 vs *lm-cut* scatter plot of heuristic values in initial states

h^3, *lm-cut*, *pot* and *flow* heuristics. The results are shown in Table 4.1 and Table 4.2 for IPC 2011 and IPC 2014 datasets, respectively.

	# of solved problems					
domain(#)	**h^1**	**h^2**	**h^3**	**lm-cut**	**pot**	**flow**
barman (20)	4	0	0	4	4	4
elevators (20)	9	11	2	17	7	7
floortile (20)	4	4	2	6	0	2
nomystery (20)	8	8	7	16	8	10
openstacks (20)	12	7	1	12	11	10
parcprinter (20)	10	9	2	17	15	20
parking (20)	0	0	0	4	1	1
pegsol (20)	17	17	10	17	17	19
scanalyzer (20)	6	3	3	13	9	11
sokoban (20)	20	17	1	20	15	16
tidybot (20)	14	6	0	14	13	6
transport (20)	6	6	5	7	6	6
visitall (20)	9	9	8	10	16	17
woodworking (20)	3	2	0	12	2	6
$\sum$ (280)	122	99	41	169	124	135
% (100.00)	43.57	35.36	14.64	60.36	44.29	48.21

Table 4.1: Coverage in IPC 2011 dataset

We can see that h^2 and h^3 are not able to keep up with *lm-cut*, *pot* and *flow* heuristics in most domains and its total coverages are lower in both datasets, in the case of h^3 significantly. In the IPC 2014 dataset, the h^3 was able to solve only 22 out of 280 problems, failing on either memory or time limits in the rest of the problems. However, there are some exceptions. In the *elevators*,

	# of solved problems					
domain(#)	**h^1**	**h^2**	**h^3**	**lm-cut**	**pot**	**flow**
barman (20)	6	0	0	6	3	3
cavediving (20)	7	7	0	7	6	7
childsnack (20)	0	0	0	0	0	0
citycar (20)	16	10	2	16	9	9
floortile (20)	2	2	0	5	0	0
ged (20)	15	13	9	15	15	13
hiking (20)	11	8	3	10	9	10
maintenance (20)	5	5	3	17	5	5
openstacks (20)	1	1	0	1	1	1
parking (20)	0	0	0	4	3	2
tetris (20)	7	2	0	9	11	11
tidybot (20)	10	0	0	10	5	0
transport (20)	6	6	3	6	4	4
visitall (20)	3	4	2	4	13	13
$\sum$ (280)	89	58	22	110	84	78
% (100.00)	31.79	20.71	7.86	39.29	30.00	27.86

Table 4.2: Coverage in IPC 2014 dataset

floortile and *sokoban* domains from IPC 2011 dataset, h^2 outperformed both *pot* and *flow* heuristics. The same holds for the *transport* domain from IPC 2014. In both datasets, h^3 outperformed only the *pot* heuristic in *floortile* domain int IPC 2011. Also note that the h^m heuristics did not solve any problem in both *parking* domains, while the remaining heuristics were able to solve at least one problem.

Two coverage experiments were also run for h^m heuristic, one for $m = 4$ and the second one for $m = 5$. The memory limit was set to 100 GB and the time limit to 5 hours. However, not a single problem was solved, with most of them failing due to the time limit.

4.4 Evaluated States Per Second

This experiment was focused on the speed, with which is the planner able to evaluate states with given heuristics. The same heuristics as in the previous coverage experiment were used. The experiment setup is similar to the coverage experiment. Time limits and memory limits for one problem are 60 minutes and 5 GB, respectively. We compute the number of evaluated states per second for each problem. These values are then averaged over each domain. A “dash” sign in a table means, that the heuristic was not able to compute even the initial state for any problem in the domain. The results are shown in Table 4.3 and Table 4.4.

It is clear that the calculation of h^3 is very slow. For example, in the IPC

2014 domain, the h^3 was not able to finish a initial state evaluation for any problem in 7 domains.

	# evaluated states per second					
domain	**h^1**	**h^2**	**h^3**	**lm-cut**	**pot**	**flow**
barman	56935.3	95.8	0.1	4156.4	731321.1	3396.2
elevators	99715.8	609.7	2.7	8589.4	1119131.7	5807.5
floortile	95323.9	829.4	8.0	9539.2	1262216.7	4681.8
nomystery	73900.6	2285.6	132.6	7797.2	1556514.6	3004.2
openstacks	90600.4	771.6	19.8	56784.9	1275567.3	3182.4
parcprinter	63648.3	776.4	7.1	12416.1	867593.7	3877.2
parking	4705.0	-	-	293.0	760372.9	459.4
pegsol	121417.0	1615.9	6.2	40344.2	1324796.9	3013.1
scanalyzer	23199.4	1929.2	299.2	3482.0	922557.2	1112.5
sokoban	89583.5	329.7	5.4	13022.5	997807.0	3088.4
tidybot	5841.6	5.1	-	417.3	54817.2	154.3
transport	46207.6	309.6	4.5	1175.7	1089358.9	3413.1
visitall	157583.0	7303.5	2024.4	21032.0	1233533.6	6011.0
woodworking	58706.8	266.0	-	7228.9	853696.8	3804.5
average	70526.3	1317.5	228.2	13305.6	1003520.4	3214.7

Table 4.3: States per second in IPC 2011 dataset

	# evaluated states per second					
domain	**h^1**	**h^2**	**h^3**	**lm-cut**	**pot**	**flow**
barman	55772.8	88.5	0.1	4008.0	740269.8	3184.3
cavediving	55880.9	385.4	-	10528.6	704039.5	2354.8
childsnack	44719.7	497.7	-	8716.8	994637.3	2455.8
citycar	67561.1	336.7	0.8	3205.9	671806.8	3410.2
floortile	99603.8	559.5	-	8875.3	1224168.0	4917.4
ged	66095.5	1380.5	57.3	26845.1	1659684.0	911.4
hiking	49151.2	1952.8	233.7	6234.9	768559.9	3178.0
maintenance	227348.6	9765.2	154.0	153521.7	909329.2	10495.9
openstacks	30798.5	99.1	-	14091.6	1048073.2	1064.4
parking	5622.6	-	-	371.1	765490.6	594.2
tetris	12464.4	8.3	-	414.8	129058.9	273.1
tidybot	2142.6	-	-	43.5	17812.7	50.3
transport	34499.3	332.6	6.1	976.5	958408.9	2609.0
visitall	101164.8	2001.0	44.9	11326.2	945642.6	3419.5
average	60916.1	1450.6	71.0	17797.1	824070.1	2779.9

Table 4.4: States per second in IPC 2014 dataset

Chapter 5

Proposed Improvement

Based on the idea of Π^C [8] and Π^m [7], we propose a new representation of planning task Π.

Definition 5.1. Let Π be a planning task $\langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$ and let $r \subseteq \mathcal{F}$ be a set of facts from Π. Planning task Π^m_r is a tuple $\langle \Phi, \Omega, \phi_{init}, \phi_{goal} \rangle$, where Φ is a set of meta-facts (meta-atoms), $\Phi = \{\phi_c \mid c \subseteq \mathcal{F}, |c| \leq 1\} \cup \{\phi_c \mid c \subseteq \mathcal{F}, 1 < |c| \leq m, c \cap r = \emptyset\}$. Initial state and goal specification are defined analogously:

$\phi_{init} = \{\phi_c \mid c \subseteq s_{init}, |c| \leq 1\} \cup \{\phi_c \mid c \subseteq s_{init}, 1 < |c| \leq m, c \cap r = \emptyset\}$
$\phi_{goal} = \{\phi_c \mid c \subseteq s_{goal}, |c| \leq 1\} \cup \{\phi_c \mid c \subseteq s_{goal}, 1 < |c| \leq m, c \cap r = \emptyset\}$.

For each operator $o \subseteq \mathcal{O}$ and for each set of facts $f \subseteq \mathcal{F} \setminus r$, $|f| \leq m - 1$ and $f \not\subseteq \text{add}(o) \cup \text{del}(o)$, Π^m contains such a meta-operator $\omega_{o,f} \in \Omega$:

$$\text{pre}(\omega_{o,f}) = \{\phi_c \mid c \subseteq (\text{pre}(o) \cup f),\ |c| = 1\} \cup \{\phi_c \mid c \subseteq (\text{add}(o) \cup f),\ c \cap r = \emptyset,\ 1 < |c| \leq m\}$$

$$\text{add}(\omega_{o,f}) = \{\phi_c \mid c \subseteq \text{add}(o),\ |c| = 1\} \cup \{\phi_c \mid c \subseteq (\text{add}(o) \cup f),\ c \cap \text{add}(o) \neq \emptyset, c \cap r = \emptyset,\ 1 < |c| \leq m\}$$

$\text{del}(\omega_{o,f}) = \emptyset$, and
$\text{cost}(\omega_{o,f}) = \text{cost}(o)$.

This is, in fact, a Π^m task, with the restriction on a set of combinations for which meta-facts are created. This is done by introducing set $r \subseteq \mathcal{F}$ of facts from the original task Π. All combinations of size bigger than 1, which would contain fact f from r, have not their corresponding meta-fact created. This is similar to the Π^C characterization of the task, as the same effect could be achieved with an appropriate choice of C, however, the main difference is in the definition of meta-operators. Meta-operators in Π^m_r follow the same logic as in the Π^m in Definition 2.5. That means that meta-operator is created for every subset of $\mathcal{F}$ of a size smaller than m. Recall that in the Π^C task from Definition 2.12, the meta-operator was created for every subset of C. This puts the Π^m_r task somewhere in between the Π^m and Π^C task. The heuristic value is computed as h^1 of to the Π^m_r task, same as in Definition 2.5.

Notice, that the choice of restricted facts r allows for a transition between Π^m tasks for different m values. Let $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$. The Π^2_r task, where $r = \mathcal{F}$ is then equal to the original Π task. However, with $r = \emptyset$, the Π^2_r tasks corresponds to the Π^2 task.

5.1 Initial Experiments

For the experiments, the basic implementation of $h^1(\Pi^2_r)$ was created. It is built upon the brute force implementation of h^2, shown in Algorithm 3. The implementation takes a list of facts, that cannot be used in any combinations of facts throughout the whole task.

Two sample problems included with the MAPlan planner [6] were used in this experiment, *rovers-p01.proto* and *rovers-p02.proto*. They were chosen due to their relatively small size, which allows for an easier observation of their initial structure, as opposed to the problems from the IPC datasets.

The first experiment was focused on studying the impact of the r set selection on the planning task. This was done be generating all combinations of facts from $\mathcal{F}$. Regular h^1 and h^2 heuristics were computed on the two tested problems. The heuristic value at the initial state and the total number of evaluated states were noted. After that, the $h^1(\Pi^2_r)$ were computed with r iterating through all the generated combinations of facts.

Part of the results are shown in Table 5.2. Note that the first row of the table corresponds to results obtained by h^2, while the last one to the ones by h^1 heuristic. For a more clear way of describing the used facts, the names of single facts derived from the FDR representation are included in the table.

r set (ids)	r set (names)	init heur	eval states
$\emptyset$	-	5	124
...	...	...	...
$\{f_3\}$	{at(rover0, waypoint3)}	5	124
$\{f_4\}$	{at_rock_sample(waypoint0)}	5	124
$\{f_5\}$	{have_rock_analysis(rover0, waypoint0)}	5	171
$\{f_6\}$	{at_soil_sample(waypoint0)}	5	124
$\{f_7\}$	{have_soil_analysis(rover0, waypoint0)}	5	171
$\{f_8\}$	{calibrated(camera0, rover0)}	5	148
...	...	...	...
$\{f_3, f_4\}$	-	5	124
...	...	...	...
$\{f_5, f_7\}$	-	4	206
...	...	...	...
$\{f_3, f_4, f_6\}$	-	5	124
...	...	...	...
$\mathcal{F}$	-	3	259

Table 5.1: Impact of r set selection on Π^2_r

We can see that restricting some facts has no effect on the number of evaluated states and the search with this heuristic overall. Consider for example r *sets* of $\{f_3\}$, $\{f_4\}$ and $\{f_6\}$. The heuristic values and number of evaluated states are the same as for unrestricted h^2. It could also be noticed, that combining those single facts, that have no effect on the number of evaluated states into one set, which was then passed to the heuristic as r set, had also no effect on the number of evaluated states.On the other hand, combining those single facts, that have no effect on the heuristic value at the initial state, did not produce such good results. It, however, still performed better than the h^1 heuristic by itself.

Nevertheless, this experiment showed, that it is possible to select a set of facts r in such way, that the $h^1(\Pi^2_r)$ produces the same heuristic values as h^2. In the following section, we discuss the strategy for selecting set of facts, that would keep the heuristic estimates as high as possible.

5.2 Selection of Restricted Facts

As described in the previous section, two ways of selecting the set of facts to restrict were proposed. The better performing one selected those facts, for which the number of evaluated states is the same or smaller than in the case of h^2. However, this is not very practical, as for this we need to actually solve the problem completely. Because of that, a strategy based on heuristic values in initial states is proposed.

5.2.1 Proposed Strategy

Let $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$ and let $\Pi^2_r = \langle \Phi, \Omega, \phi_{init}, \phi_{goal} \rangle$ constructed as in Definition 5.1. Let us apply the following strategy for selecting the set of facts r_{set}. Compute the value of $h^1(\Pi^2)$. Lets call this value *original init value*. For every fact $f \in \mathcal{F}$ compute $h^1(\Pi^2_r)$ for $r = f$. If this value is the same as *original init value*, add fact f to r_{set}. Finally, compute $h^1(\Pi^2_{r_{set}})$.

5.2.2 Strategy Evaluation

Let us show the results on *pegsol* domain from IPC 2011 dataset. For each problem, the brute force h^2 was used and the problem was solved, with the total search time shown in the second column of Table 5.2. After that, the strategy from Section 5.2.1 is applied to the $h^1(\Pi^2_r)$ for r selection. The total time for the selection process plus solving the task itself is in the third column.

We can see that the strategy produces the solution to the problems in *pegsol* faster than regular brute force h^2. However, this is the best performing domain from the dataset and the strategy itself does not guarantee any theoretical properties. The relative success in this domain can thus be attributed to the structure of this domain.

	time in seconds		
problem	h^2	$h^1(\Pi^2_r)$ + **strategy**	**diff**
p1	0.07	0.49	-0.42
p2	489.14	162.60	326.54
p3	26.05	20.72	5.33
p4	192.63	134.07	58.56
p5	36.52	22.34	14.18
p6	223.30	124.29	99.01
p7	126.05	73.35	52.70
p8	363.25	172.51	190.74
p9	357.45	206.10	151.35
p10	627.80	237.14	390.66
p11	30.83	28.79	2.03
p12	26.54	9.85	16.69
p13	963.83	493.40	470.43
p14	883.42	572.60	310.82
p15	897.52	518.71	378.81
p16	863.62	904.16	-40.53
p17	977.36	916.34	61.02
p18	894.69	919.36	-24.67
p19	550.00	948.05	-398.05
p20	652.69	960.86	-308.17
$\sum$	9182.76	7425.74	1757.02

Table 5.2: Search time of h^2 vs $h1(\Pi^2_r)$ with strategy for r selection

5.2.3 Justification Graph

Some analytical approach of selecting the set of facts to restrict, which would produce results with some theoretical guarantees, would be much more convenient. With the hope of discovering some structural connection or relationships of the task with the facts not affecting when restricted with, we tried to construct justification graphs [13] of the investigated problems. Justification graph is defined [13] as follows:

Definition 5.2. Let $\Pi = \langle \mathcal{F}, \mathcal{O}, s_{init}, s_{goal} \rangle$. A *precondition choice function* is a function $p : \mathcal{O} \to \mathcal{F}$, which maps every operator $o \in \mathcal{O}$ to one of its precondition.

Justification graph JG for *precondition choice function* p is a directed graph with annotated edges. The *JG* contains a vertex for every fact $f \in \mathcal{F}$ and edge $p(o) \xrightarrow{o} e$ for every $o \in \mathcal{O}$, $e \in add(o)$.

We constructed the justification graph for the Π^2 with *precondition choice function* being the h^1, i.e., for each operator selecting the hardest to reach precondition. The idea was to look for some properties of the graph, which would explain why some facts can be restricted from the meta-fact creation while maintaining the same heuristic values. However, this turned up to be nearly impossible, as the justification graph represents every fact in the task with a node. For example, in the Π^2 representation of the *rovers-p02* problem that means several hundred of meta-facts. The graph gets very large and it is thus difficult to observe any rules or regularities.

Chapter 6 Conclusion

One of the goals of this thesis was to efficiently implement h^2 and h^3 heuristics. This was done by using the alternative characterization of the planning task Π, Π^m, and the fact, that $h^1(\Pi^m) = h^m(\Pi)$ [7]. A brute force solutions were implemented first. Then we gradually improved them by introducing optimizations to the algorithms. The optimized algorithms for h^2 and h^3 were experimentally evaluated on datasets used in International Planning Competition from years 2011 and 2014. We compared the heuristics with other *state-of-the-art* heuristics, namely the LM-Cut heuristic [12], flow-based heuristic [1, 2] and potential heuristic [16].

In the analysis, heuristics were compared in terms of heuristic values in initial states, the number of solved problems in the International Planning Competition datasets (coverage) and the number of evaluated states per second. We used the heuristic values in initial states for showing the dominance between heuristics. We obtained results in accordance with the theoretical properties of h^m, i.e., h^3 dominating h^2 and h^2 dominating h^1. There was no dominance found between any of the other heuristics. The coverage experiment showed that h^3 is not able to keep up with other *state-of-the-art*, with the exception for one domain. The h^2 was, however, able to solve more problems than the LM-Cut and flow-based heuristics in several of domains, while in general performing worse than the *state-of-the-art*. Similar results were obtained when comparing the number of evaluated states per second, with h^2 and h^3 being outperformed by *state-of-the-art* heuristics.

General h^m heuristic was also implemented, however, its usage was very limited and it performed poorly in experiments, completely failing in the coverage experiment.

We proposed a new characterization of the planning task, Π^m_r. It acts as a restriction of the regular Π^m characterization of the task, as it allows for a choice of a set of facts which cannot be used in combinations of facts represented by meta-facts. It was experimentally shown that a proper choice of the restriction set for Π^2_r keeps the same heuristical estimates as for regular Π^2.

Finally, a strategy for choosing a set of facts, which will not lower the heuristical estimates too much, was proposed, implemented, and evaluated, showing promising results in *pegsol* domain from IPC dataset from 2011, but not performing very well in other domains. The problem of selecting the set of restricted facts will be a subject of a future work.

Bibliography

[1] Blai Bonet. "An Admissible Heuristic for SAS$^+$ Planning Obtained from the State Equation". In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*. 2013, pp. 2268–2274. URL: `http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6927`.

[2] Blai Bonet and Menkes van den Briel. "Flow-Based Heuristics for Optimal Planning: Landmarks and Merges". In: *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS)*. 2014, pp. 47–55. URL: `http://www.aaai.org/ocs/index.php/ICAPS/ICAPS14/paper/view/7933`.

[3] Richard Fikes and Nils J. Nilsson. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". In: *Artif. Intell.* 2.3/4 (1971), pp. 189–208. DOI: `10.1016/0004-3702(71)90010-5`. URL: `http://dx.doi.org/10.1016/0004-3702(71)90010-5`.

[4] Daniel Fišer. *Boruvka*. `https://github.com/danfis/boruvka`. 2018.

[5] Daniel Fišer. *LM-Cut Heuristic*. URL: `http://cw.fel.cvut.cz/wiki/_media/courses/a4m36pah/t4-lmcut-fiser.pdf`. Slide 4. Mar. 2017.

[6] Daniel Fišer, Michal Štolba, and Antonin Komenda. "MAPlan". In: *Competition of Distributed and Multi-Agent Planners (CoDMAP)*. 2015, pp. 8–10.

[7] Patrik Haslum. "$h^m(P) = h^1(P^m)$: Alternative Characterisations of the Generalisation From $h^{\max}$ To h^m". In: *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*. 2009, pp. 354–357. URL: `http://aaai.org/ocs/index.php/ICAPS/ICAPS09/paper/view/713`.

[8] Patrik Haslum. "Incremental Lower Bounds for Additive Cost Planning Problems". In: *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*. 2012. URL: `http://www.aaai.org/ocs/index.php/ICAPS/ICAPS12/paper/view/4703`.

[9] Patrik Haslum, Blai Bonet, and Hector Geffner. “New Admissible Heuristics for Domain-Independent Planning”. In: *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*. 2005, pp. 1163–1168. URL: `http://www.aaai.org/Library/AAAI/2005/aaai05-184.php`.

[10] Patrik Haslum and Hector Geffner. “Admissible Heuristics for Optimal Planning”. In: *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS)*. 2000, pp. 140–149. URL: `http://www.aaai.org/Library/AIPS/2000/aips00-015.php`.

[11] Malte Helmert. “Concise finite-domain representations for PDDL planning tasks”. In: *Artif. Intell.* 173.5–6 (2009), pp. 503–535. DOI: `10.1016/j.artint.2008.10.013`. URL: `http://dx.doi.org/10.1016/j.artint.2008.10.013`.

[12] Malte Helmert and Carmel Domshlak. “Landmarks, Critical Paths and Abstractions: What’s the Difference Anyway?” In: *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*. 2009.

[13] Malte Helmert and Gabriele Roger. *A Beginner’s Introduction to Heuristic Search Planning - 6. Delete Relaxation and Landmarks*. URL: `http://ai.cs.unibas.ch/misc/tutorial_aaai2015/planning06.pdf`. Slide 30. Jan. 2015.

[14] Emil Keyder, Jörg Hoffmann, and Patrik Haslum. “Improving delete relaxation heuristics through explicitly represented conjunctions”. In: *Journal of Artificial Intelligence Research* 50 (2014), pp. 487–533.

[15] *MetaCentrum VO*. `https://metavo.metacentrum.cz/cs/`. Accessed: 2018-05-20.

[16] Florian Pommerening et al. “From Non-Negative to General Operator Cost Partitioning”. In: *Proceedings of the Twenty-Ninth Conference on Artificial Intelligence (AAAI)*. 2015, pp. 3335–3341. URL: `http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9983`.

[17] S. Russell and P. Norvig. “Artificial Intelligence: A Modern Approach”. In: Third. Series in Artificial Intelligence. Upper Saddle River, NJ: Prentice Hall, 2010. Chap. 3, pp. 103–104. URL: `http://aima.cs.berkeley.edu/`.