Researching Heuristic Functions to Detect and Avoid Dead Ends in Action Planning

Author: Christian Reißner

Supervisor: Prof. Dr. Ute Schmid
with Michael Siebers

A thesis submitted in fulfilment of the requirements for the degree of Master of Science in the degree program

Applied Computer Science

Cognitive Systems Group
Faculty of Information Systems and Applied Computer Sciences

December 2013
Abstract

Faculty of Information Systems and Applied Computer Sciences
Bamberg

Master of Science

Researching Heuristic Functions to Detect and Avoid Dead Ends in Action Planning

by Christian Reißner

One approach to reduce the runtime in learning is to find dead ends by planning and use this information to avoid them efficiently. The heuristic search is a well performing planning approach. The target of this paper is the modification of the heuristic search. I will present methods to detect dead ends and run them in different planning domains. A comparison of the results decides the best method to detect dead ends in planning. I will show that the Wisp method delivers the best results and that abstraction heuristics are the best for detecting dead ends.
# Contents

Abstract i

List of Figures iv

List of Tables vi

Abbreviations vii

Symbols viii

1 Introduction 1
  1.1 Avoid Dead Ends .................................................. 2
  1.2 Overview of this Thesis .......................................... 3

2 Heuristic Search Action Planning 4
  2.1 Heuristic Search Planning ........................................ 6
    2.1.1 What are Heuristics ........................................ 7
  2.2 Properties of Heuristic Functions ............................... 7
    2.2.1 Admissibility .............................................. 7
    2.2.2 Monotony .................................................. 9
    2.2.3 Safety .................................................. 10
  2.3 Fast Downward Planner ........................................... 10
  2.4 Modern heuristic approaches .................................... 11
    2.4.1 A heuristic overview ...................................... 11
    2.4.2 A Categorization of Heuristic Functions ................. 11
      2.4.2.1 Delete Relaxations Heuristics ........................ 12
      2.4.2.2 Critical Path Heuristics ............................ 18
      2.4.2.3 Abstraction Heuristics ............................... 20
      2.4.2.4 Landmark Heuristics .................................. 26

3 Methods to Detect and Avoid Dead Ends in Action Planning 31
  3.1 Detect Dead Ends .................................................. 32
    3.1.1 Heuristic Touched Diligent Search ........................ 32
    3.1.2 Uniform Cost Search with Heuristic Tendency ............ 32
    3.1.3 Modify Known Heuristics .................................... 33
    3.1.4 Wisp ...................................................... 34
Contents

4 Implementation 35
  4.1 Run the Fast Downward Planner 35
  4.2 First Steps 36
    4.2.1 Choose the Right Domains 38
    4.2.2 Data Recovery 38
    4.2.3 Used Hardware 40
  4.3 Add Some Heuristic Tendency 40
  4.4 Start Breadth Become Faster 41
  4.5 Modify Existing Heuristics 41
  4.6 Integrate Wisp Into Heuristics 42

5 Evaluation 43
  5.1 Results of Research 47
    5.1.1 Heuristic Complete Search 47
    5.1.2 Uniform Cost Search with Heuristic Tendency 48
    5.1.3 Modified Heuristic Search 49
    5.1.4 Wisp 49
  5.2 Evaluation of Methods 51
  5.3 Evaluation of Heuristics 56

6 Conclusion 60

A Time Tables of Methods 64

B Bar Charts of Methods 68

C Charts of Direct Comparison 74

Bibliography 79

Declaration of Authorship 79
List of Figures

1.1 Example of a simple search graph ........................................ 1
1.2 Search graph with marked deadlines ................................. 2
2.1 Example of an A* application for reaching I from W ............ 8
2.2 Four categorizations of Domain-independent Planning .......... 13
2.3 Relaxed Planning Graph Keyder and Richter (2011) ............ 17
2.4 Plan computed by $h^{FF}$ Keyder and Richter (2011) ............. 17
2.5 Logistics problem with one package, two trucks, two locations Keyder and Richter (2011) ........................................... 21
2.6 Transition System for package Keyder and Richter (2011) .... 21
2.7 Abstraction Transition System for package and truck A Keyder and Richter (2011) .................................................. 22
2.8 Merging step of two atomic projections Keyder and Richter (2011) ................................................................. 23
2.9 Shrinking step of the $h^{MS}$ heuristic Keyder and Richter (2011) ................................................................. 24
2.10 Some rules and their and-or dag Charniak and Husain (1991) 25
2.11 Transport object from A to B, truck or by teleportation Keyder and Richter (2011) .................................................. 28
2.12 The cut of the $h^{LM-cut}$ heuristic Keyder and Richter (2011) ................................................................. 29
2.13 The justification graph after extraction Keyder and Richter (2011) ................................................. 29
3.1 Principle of modified heuristics ........................................... 34
4.1 Small Sokuban domain ......................................................... 39
5.1 Complete state space search example .................................. 45
5.2 Deadlines by domains for a normal search with Fast Downward 51
5.3 Deadlines by domains for a heuristic complete search ........... 52
5.4 Normal Run vs. Modify 1 .................................................... 53
5.5 Normal Run vs. Modify 2 .................................................... 54
5.6 Bar chart for normal search ............................................... 55
5.7 Deadlines by domains for wisp = 10 ................................. 56
5.8 Normal Run vs. Wisp ......................................................... 57
5.9 Normal Run vs. Wisp ......................................................... 58
B.1 Deadlines by domains for a normal search with Fast Downward 68
B.2 Deadlines by domains for a Breadth First search .................. 69
B.3 Deadlines by domains for a heuristic complete search ........... 69
B.4 Deadlines by domains for a uniform cost search with heuristic tendency 70
B.5 Deadlines by domains for a modified heuristic search .......... 71
List of Figures

B.6 Deadlines by domains for wisp = 1 ........................................... 72
B.7 Deadlines by domains for wisp = 10 ....................................... 73
C.1 Comparison of normal run and modified heuristics method 1 ........... 75
C.2 Comparison of normal run and modified heuristics method 2 .......... 76
C.3 Comparison of normal run and wisp=10 integrated method 1 .......... 77
C.4 Comparison of normal run and wisp=10 integrated method 2 .......... 78
List of Tables

2.1 Overview of Heuristic Functions ............................................. 12
5.1 Complete State Space ................................................................. 45
5.2 Normal Run ................................................................................. 47
5.3 Complete Search with Heuristic Tendency ...................................... 48
5.4 Uniform Cost Search with Heuristic Tendency .................................. 48
5.5 Modified Heuristic Search ........................................................... 49
5.6 Heuristic Search with Integrated Wisp = 1 ...................................... 50
5.7 Heuristic Search with Integrated Wisp = 10 ................................... 50
A.1 Times of a Normal Run ................................................................. 64
A.2 Times of Complete Search with Heuristic Tendency ......................... 65
A.3 Times of Uniform Cost Search with Heuristic Tendency .................... 65
A.4 Times of Results of Modified Heuristic Search ................................. 66
A.5 Times of Results Integrated Wisp=1 ............................................. 66
A.6 Times of Results Integrated Wisp=10 .......................................... 67
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>Additive</td>
</tr>
<tr>
<td>CEA</td>
<td>Context-enhanced Additive</td>
</tr>
<tr>
<td>CG</td>
<td>Causal Graph</td>
</tr>
<tr>
<td>CS</td>
<td>Cost Sharing</td>
</tr>
<tr>
<td>d</td>
<td>Distance</td>
</tr>
<tr>
<td>DTG</td>
<td>Domain Transition Graph</td>
</tr>
<tr>
<td>FF</td>
<td>Fast Forward</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>G</td>
<td>Graphplan</td>
</tr>
<tr>
<td>HD</td>
<td>Hamming Distance</td>
</tr>
<tr>
<td>L</td>
<td>Landmark</td>
</tr>
<tr>
<td>LA</td>
<td>Landmark Admissible</td>
</tr>
<tr>
<td>LC</td>
<td>Landmark Counting</td>
</tr>
<tr>
<td>max</td>
<td>Maximum</td>
</tr>
<tr>
<td>MD</td>
<td>Manhatten Distance</td>
</tr>
<tr>
<td>MS</td>
<td>Merge Shrink</td>
</tr>
<tr>
<td>PDB</td>
<td>Pattern Data base</td>
</tr>
<tr>
<td>pmax</td>
<td>Pairwise maximum</td>
</tr>
<tr>
<td>S</td>
<td>Structural</td>
</tr>
<tr>
<td>SA</td>
<td>Set Additive</td>
</tr>
<tr>
<td>SAS</td>
<td>Simplified Action Structures</td>
</tr>
</tbody>
</table>
## Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Σ</td>
<td>Non-deterministic state-transition system</td>
</tr>
<tr>
<td>ℙ</td>
<td>Planning problem</td>
</tr>
<tr>
<td>ℛ</td>
<td>Transition system</td>
</tr>
<tr>
<td>γ</td>
<td>State-transition function</td>
</tr>
<tr>
<td>Γ(s)</td>
<td>Successor states of s</td>
</tr>
<tr>
<td>ˆΓ(s)</td>
<td>Transitive reachable states for s</td>
</tr>
<tr>
<td>h*</td>
<td>Optimal heuristic estimate</td>
</tr>
<tr>
<td>G</td>
<td>Graph</td>
</tr>
<tr>
<td>h(s)</td>
<td>Heuristic function of state s</td>
</tr>
<tr>
<td>e</td>
<td>Edge between two states</td>
</tr>
<tr>
<td>E</td>
<td>Set of edges</td>
</tr>
<tr>
<td>l</td>
<td>Landmark</td>
</tr>
<tr>
<td>s</td>
<td>State</td>
</tr>
<tr>
<td>s'</td>
<td>Successor state</td>
</tr>
<tr>
<td>s₀</td>
<td>Initial state</td>
</tr>
<tr>
<td>g</td>
<td>Goal state</td>
</tr>
<tr>
<td>G</td>
<td>Set of goals</td>
</tr>
<tr>
<td>c</td>
<td>Costs</td>
</tr>
<tr>
<td>p</td>
<td>Fluent</td>
</tr>
<tr>
<td>ℙ</td>
<td>Set of fluents</td>
</tr>
<tr>
<td>o</td>
<td>Operator</td>
</tr>
<tr>
<td>O</td>
<td>Set of operators</td>
</tr>
<tr>
<td>v</td>
<td>Variables</td>
</tr>
<tr>
<td>V</td>
<td>Set of Variables</td>
</tr>
<tr>
<td>⊓</td>
<td>Planning task</td>
</tr>
<tr>
<td>⊓⁺</td>
<td>Relaxed planning task</td>
</tr>
<tr>
<td>z</td>
<td>Precondition of a state</td>
</tr>
<tr>
<td>α</td>
<td>Homomorphism function</td>
</tr>
<tr>
<td>φ</td>
<td>Propositional landmark formula</td>
</tr>
<tr>
<td>ψ</td>
<td>Action landmark formula</td>
</tr>
<tr>
<td>π</td>
<td>Plan</td>
</tr>
<tr>
<td>π⁺</td>
<td>Explicit plan</td>
</tr>
<tr>
<td>λ</td>
<td>Jump point</td>
</tr>
<tr>
<td>θ</td>
<td>Deadline</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

One state of the art approach in planning is planning as heuristic search. This approach combines a search with the estimate of a heuristic function. While searching the heuristic is calculating for each search node. This influence of the heuristic increases the planning speed. The search is designed to run as optimal as possible. A heuristic calculates just an estimate where it is useful to go further while searching. The target of a planning task is to find a solution for a problem in a domain. This problem may be defined as a state space graph.

An approach to be more efficient in planning is to use the information of dead ends. A dead end in the state space of a planning task is a state where no way leads to a target. This information may be used to avoid them effectively. I will pursue this approach and will try to find more dead ends to avoid them afterwards. The task is to analyse the heuristic search and to modify it to find dead ends.

I show an example of a possible planning graph in figure 1.1. We can see some states that are more or less connected to other states. The marked state I is the initial state, the state with the G is the goal state and states with a D are dead end states. Further it is not important for us to know the dead end state itself for the later learning.

![Figure 1.1: Example of a simple search graph](image-url)
experience it is more important to know the state before a dead end and which operator is leading to the dead end. This means that the information we want to get by detecting a dead end is a pair of a pre state before dead end state and the operator that leads to the dead end. We call this pair a deadline to a dead end. Generally we may say such a combination of state and outgoing operator is called a jump point. If it does not lead to a dead end we say it is a normal jump point. In figure 1.2 these states and the associated transitions are marked.

It is also interesting to know which actions of a state do not lead to a dead end. We may use this information later in the context of learning to have more information about the state space of a problem task. This means I will try to detect every jump point I visit at the search and differentiate them by separating the jump points to dead end states from the other ones.

All methods I will present are constructed to find jump points but I will just compare the methods based on the number of deadlines found.

### 1.1 Avoid Dead Ends

The next step after detecting dead ends is to avoid them. This is not part of this thesis but I will give an insight in this very interesting chapter of learning in action planning. Correctly we are looking for the deadlines of state space. This information of the states before dead ends in combination with the operator that leads to the dead end can be used to learn a problem more efficiently. Target is to reduce the runtime in planning as heuristic search by reduce the number of actions applicable to a state. The approach is to use the deadline information to create action annulments. More about this topic can be found in (Siebers, 2012).
Chapter 1. Introduction

1.2 Overview of this Thesis

In this master thesis I will start by describing the history of heuristics and the different fields heuristics are used in.

After this general introduction I will give a look at the current state of the research. Subsequently I will demonstrate the benefit of heuristic functions and the properties they may have. A list of existing and used heuristic functions will simplify the comprehension of the chapter ”Heuristic Search Action Planning”.

In the next chapter ”Methods to Detect and Avoid Dead Ends in Action Planning” I will address the methodology of the actual topic of the thesis. I will examine the ability of different heuristic functions to detect dead ends in the problem search space. Based on these results I will optimize them to find the maximum of possible dead ends. At the end of this chapter I will give a preview how this result can be used to avoid dead ends in learning for action planning.

In chapter four I will describe the ”Implementation” of the methods I presented before. I will give an overview of the planner I used and how I modified the source code in order to obtain the desired results.

The next chapter will deal with the ”Evaluation” of my research including a comparison of the different heuristics in there ability of finding dead ends.

Finally I will summarize the key aspects of my thesis in the last chapter ”Conclusion”.
Chapter 2

Heuristic Search Action Planning

In this chapter I give definitions for the area of heuristic search planning by (Ghallab et al., 2004). With this definitions I start giving an introduction into heuristic search planning. I also will describe the admissibility of heuristics and involve the Fast Downward planner the fist time. I will show a possibility to split up heuristic functions and how they are defined.

The target of planning is to find a solution or a planning problem. A planning problem is a special condition of a domain where exists a initial state and a set of goal states. Formal we may say:

Let \( P = (\Sigma, s_0, S_g) \) be a set-theoretic planning problem, where

- \( \Sigma \) is a non-deterministic state-transition system
- \( s_0 \) is the initial state and \( s_0 \in S \)
- \( g \) is a list of propositions that must be satisfied by a state in order to be a goal state
- \( S_g \) is the set of goal states for \( S_g = \{ s \in S | g \subseteq s \} \)

The set-theoretic representation relies on a finite set of proposition symbols that are intended to represent various propositions about the world (Ghallab et al., 2004). The solution for this planning problem is defined as a sequence of actions \( a_1, ..., a_n \), where \( k \geq 0 \). This sequence lead from the initial state to the goal state and has the length \( k = |\pi| \). Formally is \( \pi \) a solution for \( P \) if

\[
    g \subseteq \gamma(s_0, \pi)
\]

where \( \gamma \) is the state-transition function and is formalized as follows:

\[
    \gamma(s, \pi) = \begin{cases} 
        s & \text{if } k = 0 \text{ (i.e., } \pi \text{ is empty)} \\
        \gamma(\gamma(s, a_1), (a_2, ..., a_k)) & \text{if } k > 0 \text{ and } a_1 \text{ is applicable to } s \\
        \text{undefind} & \text{otherwise}
    \end{cases}
\]
This function applies the actions of $\pi$ to $s$ and produces a state $s$ that is produced by applying $\pi$ in the given order.

A state is a set of ground atoms of first order logic. Each of them represents a condition of the domain of them. All states of a domain together represent the whole state space of the domain. We can say the state space may mapped by a set of states $S$. The ground atoms are the conditions of a state that define them.

Planning operators are for the transitions between states. A planning operator is a triple $o=(\text{name}(o), \text{precond}(o), \text{effects}(o))$ (Ghallab et al., 2004) where:

- $\text{name}(o)$ is a unique syntactic expression
- $\text{precond}(o)$ is a set of expressions on state variables and relations
- $\text{effects}(o)$ is a set of assignments of values to state variables

$\text{precond}$ are the negative ($o^-)$ and positive ($o^+$) literals for $o$ in a state $s$. The literals are the preconditions for an operator. The effects are the literals that will be changed by an operator.

If a state $s$ fulfilled the terms $\text{precond}^+(a) \subseteq$ and $\text{precond}^-(a) \cap s = \emptyset$ then $a$ is an applicable action for $s$.

For a definition of a dead end in action planning we look at first for the reachability of a state. A set of all successors of a state $s$ is defined as

$$\Gamma(s) = \{ \gamma(s,a) | a \in A \text{ and } a \text{ is appicatable to } s \}$$

If we now search for all states that are reachable from state $s$ we get the transitive formula

$$\hat{\Gamma}(s) = \Gamma(s) \cup \Gamma^2(s) \cup ...$$

Where $\Gamma^2(s)$ are the successors of the successors of $s$. Whit this formalisation we may say that a planning problem $P = (\sum, s_0, g)$ has a solution iff $S_g \cap \hat{\Gamma}(s_0) \neq \emptyset$.

If a solution is possible for a state, we also know when it is not. This means we can define a dead end state $s_{\text{dead}}$ by

$$S_g \cap \hat{\Gamma}(s_{\text{dead}}) = \emptyset$$

This formula says that a state is a dead end if there is no more possibility to reach a goal state $s_g \in S_g$ from the current state.

A combination of states of a domain linked by the operator may be create a planning graph. This created graph represents a planning domain. The above given explanation may be applied on this graph.

With such graphs we can start to run a heuristic search to find a solution.
2.1 Heuristic Search Planning

It is important to know what heuristics are used for in planning and how they are defined (Keyder and Richter, 2011). At first we have to discuss the environment of a planning task. So this is given as a directed graph $G = \langle S, E \rangle$, where

- $S$ is a finite set of vertices
- $E$ is a set of directed edges $\langle t, h \rangle, t, h \in S$

The planning can be characterized as a search problem $P$. Search problem $P$ is defined by:

- An initial state $s_0 \in S$
- Goal states $S_G \subseteq S$
- Cost function : $E \rightarrow \mathbb{R}^+_0$

If there is a problem there is probably also a solution for it. This solution is given in the form of a sequence of edges $\pi = \langle e_0, \ldots, e_n \rangle$ this sequence reflect a path from $s_0$ to some $s_g \in S_G$.

After this we can define that an optimal solution is a path of edges where the cost is minimal when the cost of the path is given by the sum of its edge costs

$$\text{cost}(\pi) = \sum_{e \in \pi} \text{cost}(e)$$

Next step is to define a directed graph such as we discussed above. There are some sub elements of such a graph that are defined as follows:

- Set of states S the vertices of the graph
- Initial state $s_0 \in S$
- A function $G(s)$ that indicates whether a state is a goal or not
- Planning operators O the edges of the graph
- Applicable operators $A(s) \subseteq O$ in state $s$
- Transition function $app : S \times O \rightarrow S$, defined for $s \in S, o \in A(s)$
- Non-negative operator costs $\text{cost}(o) \in \mathbb{R}^+_0$
- A fluent $p$ is a condition of a state $s$ that may change over time
To solve a search problem there are also approaches which do not use heuristics. The simplest approach is to systematically explore the whole graph which is called the brute force approach. Other searches are the uniform cost search and the Dijkstra algorithm. These methods use special techniques to explore reachable vertices beginning at $s_0$ until $s_0 \in G$ is found. The adding of heuristics to these approaches omits the exploration of unpromising regions of the graph and instead is guiding the search towards promising regions.

### 2.1.1 What are Heuristics

Heuristics are functions that estimate the cost of a path to a goal node. Formally, this means

$$h: S \mapsto \mathbb{R}_0^+$$

There is an optimal heuristic function $h^*(s)$ which stands for the lowest cost path from $s$ to some $s' \in S_G$. And $h^*(s)$ stands also for the optimal solution in linear time. The target of heuristic functions is to get as close as possible to $h^*$.

### 2.2 Properties of Heuristic Functions

Today there are many heuristic functions in computer science. It exists properties that characterize them. These properties are presented in the following:

#### 2.2.1 Admissibility

One of the significant properties of a heuristic function is the admissibility. An admissible heuristic is one that never overestimates the cost to reach the goal. Admissible heuristics are by nature optimistic because they think that the cost of solving the problem is less than it actually is (Russell, 2009). A heuristic $h$ is admissible for all $s \in S$:

$$h(s) \leq h^*(s)$$

Figure 2.1 shows a small example of an A* application that uses an admissible heuristic to reach the target. The real cost of each branch is a black number in the middle of the branch it is called $g(s)$. The estimated cost of each state is green at the top at each state is called $h(s)$. The aim of the search is to find the shortest way from the initial state W to the target state I. The algorithm uses a priority queue for the storing of the states that will be explored next. The state with the lowest estimate will be explored first. For doing that the algorithm uses the costs of each state and a heuristic estimate. A* chooses his next step by calculating $f(s) = g(s) + h(s)$ for every branch from the current state by adding the known real cost and the costs estimate by a heuristic function of
the current state. The algorithm then chooses the branch with the lowest costs. In other words this means that the estimate of a state (by example at $h(E) = 110$) must be smaller than the real distance cost between the target and the current state (in the example $g(E \text{ to } I) = 120$). A comparison shows:

$$h(E) = 110 \leq g(E \text{ to } I) = 120$$

The heuristic function that was used to estimate the distances from each state is admissible because the estimated cost of each state in the graph is lower than the real costs. A good example for understanding is the heuristic of linear distance between two geographic points. Heuristics are called optimistic if they never overestimates.
Maximum of Admissible Heuristics

Based on the knowledge from the above paragraph about the admissibility of heuristics we can go one step further. If we have more than one admissible heuristics, we can compare the quality of them. Among admissible estimates $h_1$ and $h_2$, the higher the better. That means the optimal estimate heuristic $h^*$ is an upper limit, so larger estimates are more exact. In the little $A^*$ example above fewer nodes need to be expanded when using higher admissible estimates.

Now we know how to compare admissible heuristics. The next step is to find the maximum of admissible heuristics. A definition for the maximum of admissible heuristics is given next. Let $h_1$ and $h_2$ be two admissible heuristics. Note that

$$h_1(s) \leq h^*(s) \land h_2(s) \leq h^*(s) \implies \max(h_1(s), h_2(s)) \leq h^*(s)$$

With the newly acquired knowledge the maximum of two admissible heuristics is a more informed admissible heuristic than before (Keyder and Richter, 2011).

Sum of Admissible Heuristics

Admissible heuristics can also be added. The sum of two admissible heuristics is not necessarily admissible itself. However, under certain conditions the sum of two admissible heuristics may be an admissible heuristic too. If there is a set of such heuristics with certain conditions it is called an additive set (Keyder and Richter, 2011).

2.2.2 Monotony

In mathematics a function is monotonically increasing if it is getting bigger by an increasing function argument. A monotone heuristic shows a similar behaviour. A heuristic must satisfy two conditions to be monotonic

- The cost may not overestimate
- $h(s') + c(s, s') \geq h(s)$ for all states $s$ connected to states $s'$ by an action with cost $c(s, s')$

The first point demonstrates that only an admissible heuristic may be monotonic. The second point shows that the estimated cost of a node $s$ must be smaller or equal than the real cost of reaching a node for a predecessor state $s'$ plus the estimated cost of the node. A monotonic heuristic is also called a consistent heuristic (Keyder and Richter, 2011).
2.2.3 Safety

The last property I will present in this thesis is the safety of heuristic functions. A heuristic is safe if it marks a state with infinity in the case there is no way from this state to a goal state. As a formula we can say:

\[ h(s) = \infty \iff d(s, s_g) = 0 \]

If a heuristic function satisfies this property, we can say the heuristic is a good one for finding dead ends.

2.3 Fast Downward Planner

The Fast Downward Planner is a planning system based on heuristic search. The current version is a merge of three different projects. The original version of Fast Downward was developed by Malte Helmert and Silvia Richter. The LAMA planner was developed by Silvia Richter and Matthias Westphal and based on the original Fast Downward. The FD-Tech is a modified version of Fast Downward developed by (Hel, 2012).

The planner is a software platform that uses heuristics to solve problems. For this thesis it will be used to implement and run the researched heuristics in different domains. The results of the implementations can be found in section 5.

At this time I will just give a short information about the functionality of the planner. Fast Downward is a progression planner, searching the space of world states of a planning task in the forward direction. An example for a forward search planning algorithm by (Ghallab et al., 2004):

\[
\text{Forward-search}(O, s_0, g) \\
\quad s \leftarrow s_0 \\
\quad \pi \leftarrow \text{the empty plan} \\
\quad \text{loop} \\
\quad \quad \text{if } s \text{ satisfies } g \text{ then return } \pi \\
\quad \quad \quad \text{applicable} \leftarrow \{ a \mid a \text{ is a ground instance of an operator in } O, \text{and precond}(a) \text{ is true in } s \} \\
\quad \quad \quad \text{if } \text{applicable} = \emptyset \text{ then return failure} \\
\quad \quad \quad \text{nondeterministically choose an action } a \in \text{applicable} \\
\quad \quad \quad s \leftarrow \gamma(s, a) \\
\quad \quad \quad \pi \leftarrow \pi, a
\]

Listing 2.1: A forward search planning algorithm

The planning system is based on heuristic state space search and hierarchical problem decomposition. The system creates a causal graph and a domain transition graph from a domain that components are used to optimize the preconditions for the planning task.
Chapter 2. Heuristic Search Planning

The heuristic evaluator proceeds downward in so far as it tries to solve planning tasks in the hierarchical fashion outlined at the beginning. Starting from top-level goals, the algorithm recurses further and further down the causal graph until all remaining subproblems are basic graph search tasks. Now may begin the search part and a chosen search heuristic combination run until a solution is found or not.

The search algorithm uses the heuristic estimate of a heuristic function of every successor state to find a preferably way to the goal state. Furthermore gives the heuristic function the possibility to detect dead end states. That mean we need the heuristic functions to detect dead end states with the Fast Downward planner.

A deeper explanation of the Fast Downward Planner and how it works can be found in (Helmert and Richter, 2004).

2.4 Modern heuristic approaches

2.4.1 A heuristic overview

Since the beginning of computing heuristic functions there are a lot of new developments. Not all of these functions are useful for this master thesis. For the completeness the most of the nowadays known heuristics are shown in table 2.1.

2.4.2 A Categorization of Heuristic Functions

Today there are lot of heuristic functions for planning. To give a better overview about the wide field of heuristic function it is helpful to split them up into different base categorizations. All the heuristics can be summarized in four base categorizations for domain independent planning (Helmert and Domshlak, 2009). These categorizations are delete relaxations which contain all the heuristics that ignore deletes to find a estimate. The next categorization contains critical path heuristics where the heuristic estimate is based on using critical branches. Next ones are the abstraction heuristics which abstract problems into a better space for problem solving. The fourth of the groups are the landmark heuristics which are used to create landmarks and use them for a estimate. A visual overview about the categorizations for domain independent planning are given in figure 2.2. This grouping is only one possibility there can be other categorizations for heuristics. The classification in this thesis is a suggestion. The next step is to give a view of the heuristic functions which are contained in each categorization.
Table 2.1: Overview of Heuristic Functions

<table>
<thead>
<tr>
<th>Known heuristic functions</th>
<th>Admissible Heuristics</th>
<th>Inadmissible Heuristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blind</td>
<td>Simple goal state heuristic</td>
<td>goal count heuristic</td>
</tr>
<tr>
<td>$h^{X+Y}$</td>
<td>Addition of two admissible heuristics</td>
<td>relaxation-based additive heuristic</td>
</tr>
<tr>
<td>$h^{MD}$</td>
<td>Manhattan distance heuristic</td>
<td>relaxed plan heuristic</td>
</tr>
<tr>
<td>$h^+$</td>
<td>Idealized cost heuristic</td>
<td>causal graph heuristic</td>
</tr>
<tr>
<td>$h^{max}$</td>
<td>maximum of the accumulated costs</td>
<td>context enhanced additive heuristic</td>
</tr>
<tr>
<td>$h^m(m = 1, 2, \ldots)$</td>
<td>$h^m$ heuristic family</td>
<td>cost-sharing heuristic</td>
</tr>
<tr>
<td>$h^{PDB}$</td>
<td>pattern database heuristic</td>
<td>set additive heuristic</td>
</tr>
<tr>
<td>$h^{MS}$</td>
<td>merge-and-shrink heuristic</td>
<td>pairwise max heuristic</td>
</tr>
<tr>
<td>$h^{S}$</td>
<td>structural abstraction heuristics</td>
<td>pairwise max heuristic</td>
</tr>
<tr>
<td>$h^{LA}$</td>
<td>admissible landmark heuristic</td>
<td>graph-plan heuristic</td>
</tr>
<tr>
<td>$h^L$</td>
<td>a simple landmark heuristic</td>
<td>Finite state machine distance heuristic</td>
</tr>
<tr>
<td>$h^{LM-cut}$</td>
<td>landmark cut heuristic</td>
<td>Hamming distance heuristic</td>
</tr>
<tr>
<td>$h^{LC}$</td>
<td>landmark count heuristic</td>
<td></td>
</tr>
</tbody>
</table>

2.4.2.1 Delete Relaxations Heuristics

"Relaxation heuristics estimate the cost of reaching a goal state by considering a relaxed task derived from the actual planning task by ignoring all delete effects of operators i.e., replacing each operator by a new operator with the same preconditions, add effects and costs" (Helmert and Domshlak, 2009).

In other words a relaxation of a problem is a less constrained version of it. It is called relaxed because the new version of the problem is easier to solve then the original one. The target is now to solve the relaxed problem and use the cost of the solution as a heuristic estimate for the original problem. If this is an optimal solution for the relaxation it guarantees admissibility.

An example: After a step in planning the list of conditions change. Effects may be added or deleted and also preconditions change. In the group of delete relaxation heuristics the entries of negative conditions will be ignored. This gives more possible ways which can be used as an estimate (Helmert and Domshlak, 2009).
Chapter 2. Heuristic Search Planning

Figure 2.2: Four categorizations of Domain-independent Planning

$h^+$

$h^+$ is not a real own heuristic itself its more a heuristic principle. Definition of $h^+$

$$h^+ = h^*(\sqcap)$$

In other words $h^+$ is the abstraction of the best estimate of all fluents for a delete relaxation task $\sqcap$. It is admissible and any solution to normal task $\sqcap$ is a solution to a relaxed task $\sqcap$ as well. But this can be said only in one direction. The opposite is not true because in $\sqcap$, a fluent may be verified more than once (Hoffmann, 2001; Mirkis and Domshlak, 2007).

$h^{\text{max}}$

The estimate of the $h^{\text{max}}$ heuristic is the maximum of the accumulated costs of the paths to the goal propositions in the relaxed problem (a delete-free version of the problem). It is an admissible heuristic that gives a lower limit on $h^+$. The idea is to estimate the cost of a set of fluents as the cost of the most expensive fluent in set:

$$h^{\text{max}}(P, s) = \max_{p \in P} h^{\text{max}}(p, s)$$

The disadvantages of this heuristic are that it is kind of uninformative and it ignores lower limits.

Let us have a closer look at the above formula.

$$h^{\text{max}}(s) = h^{\text{max}}(G, s)$$

The estimate of $h^{\text{max}}$ for a state $s$ is the estimate for the goal from $s$. When considering individual fluents we get

$$h^{\text{max}}(P, s) = \max_{p \in P} h^{\text{max}}(p, s)$$
The estimate for a set of fluents $P$ is the maximum of estimates for all fluents $p$ of state $s$. Now it is time to give a look at the calculation part of the heuristic

$$h_{\text{max}}(p, s) = \begin{cases} 0 & \text{if } p \in s \\ \min_{\{o | p \in \text{add}(o)\}} [\text{cost}(o) + h_{\text{max}}(\text{pre}(o), s)] & \text{otherwise} \end{cases}$$

The estimate is zero if the fluent is an element of the state. In all other cases the estimate is the minimum of the addition of the cost of the operation $\text{cost}(o)$ and the estimate of the precondition from the operation $\text{pre}(o)$ and state $s$. This is for all operators $o$ where $p \in \text{add}(o)$ (Keyder and Richter, 2011; Bonet et al., 1997; Bonet and Geffner, 2001).

$h_{\text{add}}$

The $h_{\text{add}}$ heuristic (Bonet and Geffner, 2001) estimates cost of a set of elements as the sum of the costs of this elements. Let us take a closer look at the components. The total cost of a goal $G$ are therefore all estimates of sub-elements from $G$.

$$h_{\text{add}}(s) = h_{\text{add}}(G, s)$$

The heuristic estimate for a single state $s$ is the heuristic estimate for a goal $G$ from $s$. At this point it is already the same situation as for the $h_{\text{max}}$ heuristic I talked before. The next steps shows the different. We now estimate for a set of fluents $P$ of the state $s$

$$h_{\text{add}}(P, s) = \sum_{p \in P} h_{\text{add}}(p, s)$$

The difference to $h_{\text{max}}$ is obviously we now take the sum of estimates and not more the maximum of them. The calculation remains the same.

$$h_{\text{add}}(p, s) = \begin{cases} 0 & \text{if } p \in s \\ \min_{\{o | p \in \text{add}(o)\}} [\text{cost}(o) + h_{\text{add}}(\text{pre}(o), s)] & \text{otherwise} \end{cases}$$

The $h_{\text{add}}$ heuristic was the first domain-independent planning heuristic for satisfying planning. The properties of the heuristic are that it is not admissible and a not consistent heuristic (Anderson, 2010; Bonet et al., 1997; Bonet and Geffner, 2001).
The set-additive heuristic is a slightly modified version of the additive heuristic. The main message is that the set-additive heuristic estimates the cost of a set of actions. The set-additive heuristic for a state is defined by (Helmert and Domshlak, 2009; Keyder and Geffner, 2007, 2008) as

\[ h^{SA}(s) = \text{Cost}(\pi(G, s)) \]

The cost will be required to achieve a goal for a state with a set of actions. \( \pi(G; s) \) actually represents a set of actions with no action duplicates in the corresponding relaxed plan.

Remember the additive heuristic produces relaxed plans by taking the cost information into account. A rewritten definition of the additive heuristic is

\[
h(p, s) = \begin{cases} 0 & \text{if } p \in s \\ h(o_p(s), s) & \text{otherwise} \end{cases}
\]

where

\[ o_p = \arg \min_{o \in O(p)} h(o, s) \]

\( o_p \in O \) represents the best supporting operator from an set of operators of an atom \( p \) in a state \( s \). \( h(o, s) \) stand for \( h(o, s) = \text{cost}(o) + \sum_{q \in \text{Pre}(o)} h(q, s) \) which is the same as for the additive heuristic. We recapitulate that for the additive heuristic the value of the best operator is propagated and gives the heuristic value for \( h(p, s) \). In the set-additive heuristic the best supporter itself is propagated. This is results in a function that represents a set of actions. This function is called \( \pi(p, s) \) and can be defined similarly to \( h(p, s) \).

\[
\pi(p; s) = \begin{cases} \{\} & \text{if } p \in s \\ \pi(o_p, s) & \text{otherwise} \end{cases}
\]

where

\[ o_p = \arg \min_{o \in O(p)} \text{Cost}(\pi(o, s)) \]

\[ \pi(o, s) = \{o\} \bigcup \{\cup_{q \in \text{Pre}(o)} \pi(q, s)\} \]

\[ \text{Cost}(\pi(o, s)) = \sum_{o' \in \pi(o, s)} \text{cost}(o') \]

\( o_p \) is the best operator of \( p \) and is propagated to compute \( \pi(p, s) \). The set of operators \( \pi(p, s) \) stands actually for a relaxed plan that achieve \( p \) from \( s \) while \( \pi(o, s) \) is a relaxed plan that achieves each of the preconditions of \( o \) and applies then \( o \). The set additive heuristic is not admissible.
Chapter 2. Heuristic Search Planning

$h^{CG}$

The causal graph heuristic $h^{CG}$ estimates the number of operators that are needed to reach the goal from a state. The heuristic is not admissible and also not consistent. The estimated cost of changing the value of each variable that appears in the goal from its value in the state to its value in the goal are taken into account. As the formula says

$$h^{CG}(s) \equiv \sum_{v \in \text{dom}(s^*)} \text{cost}_v(s(v), s^*(v))$$

In the formula above is $s$ a state. $v$ is a variable and $s(v)$ is the variable in $s$. $s^*(v)$ is the value in the goal. The cost function $\text{cost}_v(d, d')$ determine the heuristic $h^{CG}(s)$ and gives us the cost of a plan $\pi$ that solves a sub-problem with an initial state $(s[v = d])$ and a goal state $(s[v = d'])$. The costs then will be compute in topological causal order starting with the variables with no parents in the causal graph.

The definition of the cost-function consists of two structures. The first is the domain transition graph DTG and the second one is the causal graph CG. A domain transition graph is a labelled directed graph with a set of edges $D_v$ and edges $(d, d')$. For a variable $v \in V$ are the edges labelled with the conditions $z$ for rules $v = d, z \rightarrow v = d'$ in a Problem $\Pi$. The causal graph is a directed graph with a set of edges $V$ and $\text{arcs}(v, V')$ for $v \neq v'$.

The causal graph heuristic was created by Helmert in 2004. A new extension of the causal graph heuristic is shown in (Helmert and Geffner, 2008). In this paper is shown that the heuristic is compatible to the additive heuristic $h^{add}$ a combination of them creates the context-enhanced additive heuristic $h^{CEA}$ which is also presented later(Helmert, 2004).

$h^{FF}$

The $h^{FF}$ heuristic (Hoffmann, 2001) used at first a relaxed plan. The idea of a relaxed plan is to find an explicit plan $\pi^+$ for a delete relaxation task $\ominus^+$. The heuristic estimate is then the costs of $\pi^+$

$$h = \text{cost}(\pi^+)$$

This relaxed plan heuristic has the advantages not to count over and to construct incrementally a plan with no duplicates. The disadvantages are its impermissibility and therefore optimality is no longer possible.

Let us take a look at an example relaxed planning graph in figure 2.3. On the left side of the picture we see a relaxed planning graph consisting of states $a$ to $g$ with $g$ as goal state. On the other side a table is shown which represent the graph and each possible way per column.

The $h^{FF}$ heuristic starts at the goal and construct a relaxed plan backwards. This plan choose operators for each fluent at first layer in that it appears. In figure 2.4 is shown a plan for the graph computed by $h^{FF}$. The heuristic calculate the value "10". This value is calculated as follows
Chapter 2. Heuristic Search Planning

Figure 2.3: Relaxed Planning Graph Keyder and Richter (2011)

Figure 2.4: Plan computed by $h^{FF}$ Keyder and Richter (2011)

For comparison the $h^{add}$ heuristic would calculate an estimate of "12" because it takes $\langle a \rightarrow b \rangle$ twice. We don not have the properties of admissibility and consistency but it is nevertheless a good heuristic (Keyder and Richter, 2011; Hoffmann and Nebel, 2011).

$h^{CEA}$

The Context-enhanced additive heuristic $h^{CEA}$ is a combination of the $h^{add}$ heuristic (Bonet et al., 1997) and the $h^{CG}$ heuristic (Helmert, 2004). The heuristic based on the idea of $h^{add}$ and is supplemented by assumptions of $h^{CG}$. For the beginning remember the construction of $h^{add}$. We may also write it in the following form

$$h^{add}(G; s) = \sum_{x \in G} h^{add}(x; x_s)$$

$$h^{add}(x; x_s) = \begin{cases} 0 & \text{if } x \in x_s \\ \min_{o: z \rightarrow x} \left[ \text{cost}(o) + \sum_{x_i \in z} h^{add}(x_i|x_i) \right] & \text{if } if x \neq x_s \end{cases}$$

In this notation is $o : z \rightarrow x$ where $o$ is an action that assigns $x$ and has $z$ as a precondition. $x_s$ is the value of the variable of $x$ in $s$.

The heuristic is defined in terms of finite-domain variables ($SAS^+$, (Bäckström and Nebel, 1995)). The assumption of the $h^{CEA}$ heuristic is considering side effects by evaluate preconditions. In other words create a new state $s'$ that is different from $s$ because of the considered precondition $x_i$. To achieve values for a set of variables achieve one of them first. Afterwards compute the cost of the other variables achieve the first one. The formula is changed by this approach into
following

\[ h^{CEA}(x; x') = \begin{cases} 
0 & \text{if } x \in x' \\
\min_{o:x''\rightarrow x}[\text{cost}(o) + h^{CEA}(x''|x') + \sum_{x_i \in z} h^{CEA}(x_i|x_i')] & \text{if } x \neq x_s
\end{cases} \]

We may recognize the different to the formula of \( h^{add} \) that we discussed before. Now we consider the preconditions in two parts:

- The value \( x'' \) has been added, the value is defined on same variable as \( x \)
- and \( z \), which is the rest of the preconditions

The formula gives us the decision for the heuristic cost. If the values are the same the estimated costs are zero. If they are not equal the costs are the minimum of the sum calculated by the cost of the operation. That are the heuristic costs of achieving \( x'' \) and the heuristic cost of achieving the other preconditions from their values \( x'_i \) that result from achieving \( x'' \). The last part are the other values \( x'_i \) in the projected state \( s(x''|x') \) these are resulting from achieving \( x'' \). Like described before is \( x'' \) the value that will be achieved at first. That is called the pivot condition.

In a comparison the heuristic may be more informative than the described \( h^+ \) heuristic but they are not in general comparable. In return of the higher information content we lost the properties of admissibility and consistency (Keyder and Richter, 2011; Helmert and Geffner, 2008).

### 2.4.2.2 Critical Path Heuristics

Critical path heuristics estimate goal distances by computing lower-bound estimates on the cost of achieving sets of cardinality facts. Roughly speaking, the underlying simplifying assumption is that a set of facts is reachable with cost if all of its \( m \)-subsets are reachable with same cost. Computing \( h^m(s) \) requires polynomial time for fixed \( m \), but exponential time in \( m \) (Helmert and Domshlak, 2009). The reason why they are called critical path heuristics is because of the estimate based by the length of the most expensive branch in an search graph.

\[ h^m(m = 1, 2, ..) \]

The family of the \( h^m \) heuristics estimate costs from sets of fluents of size \( m \). It is similar to the \( h^{max} \) heuristic. Remember how the \( h^{max} \) heuristic estimates. It calculates the costs of a set of fluents \( P \) as the max cost of any \( p \in P \).

\[ h^{max}(P, s) = \max_{p \in P} h^{max}(p, s) \]
Chapter 2. Heuristic Search Planning

The estimate of a set of fluents is the maximum estimate from the fluents of a state. Alternatively we can say

\[ h^1(P, s) = \max_{P' \subseteq P \land |P'| \leq 1} h^1(P', s) \]

Now the idea of the \( h^m \) heuristic family is to generalize to arbitrary \( m \).

We now consider subsets of \( P \) of size \( m \). Then we can say the estimate of a subset of size \( m \) is the max of the fluents of this subset.

\[ h^m(P, s) = \max_{P' \subseteq P \land |P'| \leq m} h^m(P', s) \]

Now the cost of \( P \) will be estimate as most expansive subset of size \( m \) or less (Geffner and Haslum, 2000). The change is, that the \( h^m \) heuristics take into account delete information.

We can identify the main decision that create the estimate by formalizing \( h^m \).

\[ h^m(s) = h^m(G, s) \]

The estimate for a state \( s \) is the estimate to the goal of this state. The computation is made for a set \( P \) of a state \( s \).

\[ h^m(P, s) = \begin{cases} 
0 & \text{if } P \subseteq s \\
\min_o \left( \text{cost}(o) + h^m(R(P, o), s) \right) & \text{if } |P| \leq m \\
\max_{P' \subseteq P \land |P'| \leq m} h^m(P', s) & \text{otherwise}
\end{cases} \]

Further is for a set of fluents \( P \) with \(|P| \land m\), the regression \( P \) through \( o \): where \( R(P, o) \) is the regression for a set of fluents \( P \) s.t. \(|P| \leq m\).

\[ R(P, o) = (P \setminus \text{add}(o)) \cup \text{pre}(o) \]

This is defined when:

- \( P \cap \text{del}(o) = \emptyset \)
- \( P \cap \text{add}(o) \neq \emptyset \)

Let us analyse the calculation. If \( P \) is a subset of state \( s \) the estimate is 0. This is logical because it is itself. If the value of \( P \) is smaller or equal than \( m \), the estimate is the minimum cost with respect to operator \( o \) and the estimate of the regression from state \( s \). That case is similar to the second case from the \( h^\text{max} \) heuristic. In all other cases is the estimate the maximum in the area of \( P' \cup P \land |P'| \leq m \) of the estimates from \( P' \) in \( s \).

The \( h^m \) heuristics return admissible estimates that are obtained by considering critical paths but just if the task has no conditional effects or axioms. They take deletes into account and are incomparable to \( h^+ \), despite the similarities.
Chapter 2. Heuristic Search Planning

For a fixed $m$ the computation of the heuristic is polynomial, exponential in $m$ in general. The computation in polynomial can still be very expensive. That leaves us conclude that a compute in every state is too slow. But the heuristic can be used in backwards search and to compute mutexes (Geffner and Haslum, 2000; Domshlak et al., 2010).

### 2.4.2.3 Abstraction Heuristics

Abstraction heuristics map each state of a planning task to an abstract state $\alpha(s)$ through a homomorphism function $\alpha$. The heuristic value $h^{\alpha}(s)$ is then the distance from $\alpha(s)$ to the nearest abstract goal state in the transition system induced by $\alpha$ on the transition system of the planning task (Helmert and Domshlak, 2009). Abstraction heuristics are always admissible because each plan the planning task has a corresponding abstract plan that has the same cost. Similar to the delete-relaxation heuristics is that target of abstraction heuristics derived from a simplification of the problem task. But now we do not simplify by the operation rather we simplify the search space directly.

Before talking about the heuristics in this area we have to give a little definition of what is a transition system. A transition system is a 5-tupel $T = \langle S, L, T, I, G \rangle$ (Keyder and Richter, 2011)

- $S$ is a finite set of states (the state space),
- $L$ is a finite set of (transition) labels,
- $T \subseteq S \times L \times S$ is the transition relation,
- $I \in S$ is the initial state, and
- $G \subseteq S$ is the set of goal states.

The transition system is important for abstraction heuristics as it is used to abstract to a simpler version of it. This is called the abstraction transition system $T'$.

$h^{PDB}$

A pattern database heuristic ($h^{PDB}$) is an abstraction heuristic that is based on abstract aspects. The heuristic distinguishes between aspects of the task that are represented with precision after abstraction and all the other aspects of the task that are not represented at all. In short words: it is a way to define a problem task much simpler than the original.

Let us make an example. In figure 2.5 we can see a transition system within two locations $L, R$, two trucks $A, B$ and one package $p$. The initial state is that the package is at location $L$ and both trucks are at $R$. The goal is to bring the package to location $R$. The state $LRR$ means that the package is at $L$, truck $A$ is at $R$, and $B$ is at $R$.

Let us check the heuristic estimate of the pattern database heuristic. The heuristic estimate for goal distances in a transition system are the abstract goal distances. In
Chapter 2. Heuristic Search Planning

Figure 2.5: Logistics problem with one package, two trucks, two locations Keyder and Richter (2011)

The formula:

\[ h^\alpha(s, T) = h^*(\alpha(s), T') \]

Where the \( \alpha \) stands for the abstracted state estimate and \( T \) is the transition system. The formula shows us that the estimate for a state of the transition system is the optimal estimate of the abstracted state of an abstract transition system \( T' \).

Let us switch back to our example. We can abstract the various units. In figure 2.6 the abstraction induced by package \( \pi_{\text{package}} \). In this example the abstraction is the estimate \( h^\text{package}(LRR) = 2 \).

The abstraction transition system can also be abstracted by inducing more than one unit. An example is shown in figure 2.7 where next to the package also truck A is an unit for abstraction. Now we get the heuristic estimate \( h^\text{package, truckA}(LRR) = 2 \) for both units.
Chapter 2. Heuristic Search Planning

Figure 2.7: Abstraction Transition System for package and truck AKeyder and Richter (2011)

\( h^{MS} \)

The Merge and Shrink abstraction heuristic \( (h^{MS}) \) follows the idea to reflect all state variables in a more potentially lossy way. This means that a few state variables will not be perfectly reflected. The idea was created by (Dräger et al., 2009). This works because the heuristic performs two steps sequentially.

- Merge step: add a new step variable to the abstraction
- Shrink step: reduce abstraction size by fusing abstraction states

Let us take a look at the first step. The merge step combines two separate transition systems for example \( A \) and \( A' \). This may be done without loss by a graph-theoretic operation. That operation is the synchronized product \( A \otimes A' \). Only atomic projections must be recovered to get the complete state space. Then we may build fine-grained abstractions from coarser ones.

\[ \otimes_{v \in V} \pi_v \text{ is isomorphic to } \pi_V \]

A variable as \( v \) in and a set of variables \( V \). After the merge step we get a new transition graph with both of the original atomic projections. The principle is illustrated in figure 2.8. Next step is to shrink the created abstraction. The shrinkage starts when intermediate results become too big. Then the algorithm shrinks them by fusing some abstract states. This means that a new state with the properties of of two contiguous states will be created. The principle is shown on an example in figure 2.9.

This heuristic is similar to the pattern database heuristic. The difference is that the pdb heuristic only use the merge abstraction and not the shrink abstraction. This causes that the representation power is at least as large as that of the pdb heuristics. It is in fact strictly greater. Further is this heuristic admissible and also consistent (Helmert et al., 2008, 2007).
The third one of the abstraction heuristics is the structural abstraction heuristic. Here is the main idea to reflect many state variables simple instead of perfectly reflecting a few state variables. The good thing about this heuristic is the guarantee that the abstract space can be searched (implicitly) in poly-time. That is possible because of the abstraction of a task by an instance of a tractable fragment of cost-optimal planning. The causal graph of task will be decomposed into fragments with single sink variable. More about this topic may be found in (Katz and Domshlak, 2008).
Chapter 2. Heuristic Search Planning

Figure 2.9: Shrinking step of the $h^{MS}$ heuristic Keyder and Richter (2011)

$h^{CS}$

The cost-sharing heuristic ($h^{CS}$) is applied in a so called Waodag graph. Let us take a look at the definition of a waodag graph. The components of a waodag graph

- An and-dag $\delta$ is a directed acyclic graph
- with nodes $N_\delta(n \in N_\delta)$, edges $E_\delta(e \in E_\delta)$ and leaves $L_\delta(L_\delta \subset N_\delta, l \in L_\delta)$, and a root $r_\delta \in N_\delta$
- The cost of an and-dag is naturally defined as the sum of the cost of the leaves
- An and-dag (from $\delta$) is a dag obtained from $\delta$ by removing all but one of the descendants from every node
- $\mathcal{A}(\delta)$ is the set of such dags
- A cost function is called $\mathcal{L}: L_\delta \rightarrow \Re$ (the non-negative reals)
A weighted and-or dag (waodag) is a pair of $\delta$ and $L$

After this definitions for a waodag graph we add one more definition for the compute of the cost of $\delta$ as follows

$$L_\alpha = \sum_{l \in L_\alpha} L_l$$

$$L_\delta = \min_{\alpha \in A(\delta)} L_\alpha$$

Now we come to the actual heuristic cost estimation function $h^{CS}$. The target is to get the estimates from underlying leaves and not from leaves from above. An example of a waodag graph is shown in figure 2.10. The cost of a node can be shared, we divide the cost by how many parents a node has. This shared costs are passed down to the parents. Next step is to define an estimator $\$\$ for the heuristic function. The estimator $\$: $N_\delta \cup E_\delta \cup 2^{E_\delta} \rightarrow \mathbb{R}$ is defined as follows

$$\$n = \begin{cases} L_n & n \in L_\delta \\ \$E_n & n \text{ is an and-node} \\ \min_{e \in E_n} \$e & n \text{ is an or-node} \end{cases}$$

$$\$e^n = \frac{\$n}{|E^n|}$$

$$\$E = \sum_{e \in E} \$e$$

The heuristic estimation cost is then the sum of a set of edges. This heuristic is a formalism to find the best explanations of minimal cost proofs. It is an older heuristic from 1991 by (Charniak and Husain, 1991).
Chapter 2. Heuristic Search Planning

**GoalCount**

The goal count heuristic is one of the simplest heuristics. It is neither admissible nor consistent. This heuristic counts the number of unachieved goals. The information that we can get from this heuristic is not much at all but it can be used in combination with other heuristics. If we want to categorize the heuristic in one of our four categorizations it is most likely an extremely abstraction heuristic. It is extreme because we abstract to the destination node itself. A formula of this heuristic might be as follows:

\[
h_{\text{GoalCount}}(s) = \sum_{\eta \in G} 1
\]

With \( \eta \) as goal states that are not reached. These are elements of a set of goal states \( G \).

**Blind**

The blind heuristic returns a heuristic estimate based on the achievement of goal states. The estimation value can be represent binary with a "0" for a goal state and a "1" for non-goal states. For non-goal states may also choose the cheapest action cost of non-unit-cost tasks. The arrangement is similar to the goal heuristic from above. Positive aspects of the blind heuristic are the properties of available admissibility and consistency.

2.4.2.4 **Landmark Heuristics**

A fact landmark for a state \( s \) is a fact that is true at some point in every plan of \( s \). Landmarks can be (partially) ordered according to the order in which they must be achieved. Some of these landmarks and orderings can be discovered automatically. Current approaches consider only landmarks that are facts or disjunctions of facts (Keyder and Richter, 2011).

Before the presentation of the heuristics begins I will clarify what exactly landmarks are. Landmarks represent a property of every plan for a planning task. Landmarks are part of the today’s state of the art of heuristics. A general distinction are the following two types (Keyder and Richter, 2011):

1. **Propositional landmarks**: a formula \( \phi \) over the set of fluents
   \( \models \phi \) is made true in some state during the execution of any plan

2. **Action landmark**: a formula \( \psi \) over the set of actions
   \( \models \psi \) is made true by any plan interpreted as a truth assignment to its set of action

Another characteristic of landmarks is that they can be (partially) arranged. Some of both landmarks may be discovered automatically. This is dependent on the order of the landmarks. This is primarily used when it comes to single fact/action landmarks or
simple formulas. To order landmarks there are two types of orders. The first one called sound landmarks ordering. This one guarantees to hold and not to prune the search space because it is automatically satisfied. The second one is called unsound landmark ordering. These ordered landmarks have additional constraint plans they may rule out valid solutions.

Landmarks may connect to other landmarks. That can be a propositional landmark that implies action landmarks or the opposite that an action landmark implies propositional landmarks. Formally it is as follows.

A single fact landmark \( p \) implies the disjunctive action landmark

\[
\bigvee \{ a \mid p \in \text{add}(a) \}
\]

A single action landmark \( a \) implies the propositional landmarks

\[
\bigwedge_{p \in \text{pre}(a)} p \text{ and } \bigwedge_{p \in \text{add}(a)} p
\]

There are some more facts about landmarks about which I will not talk here (Helmert and Domshlak, 2009).

First step is to find these landmarks, afterwards they can be used for different tasks. To find these landmarks is another task that we will not discuss here (more informations about finding landmarks in (Richter et al., 2008)). Interesting for this thesis is to derive heuristic estimates from landmarks. If we find the number of landmarks that still need to be achieved, we may use this number as a heuristic estimate (Richter and Westphal, 2010).

Landmark heuristics are usually path-dependent and can be used to derive admissible and inadmissible landmark-counting heuristics. The landmarks describe the implicit structure of a planning task (Keyder and Richter, 2011).

\( h^{LC} \)

The landmark count heuristic or \( h^{LC} \) is a path-dependent heuristic which is using landmarks to estimates goal distance. It is used in the LAMA planner as a pseudo-heuristic. The estimate is the landmarks that still need to be achieved after reaching state \( s \) via path \( \pi \). We get this landmarks by

\[
L(s, \pi) = (L \setminus \text{Accepted}(s, \pi)) \cup \text{ReqAgain}(s, \pi)
\]

In this formula we can see the \( L \) which stands for all (discovered) landmarks. Now we want the difference of the set of all landmarks and the intersection of all accepted landmarks and the required again landmarks.

The set of accepted landmarks is given by \( \text{Accepted}(s, \pi) \). A landmark is called accepted
by a path in a state if all predecessors of the landmark have been accepted in the landmark graph and the landmark comes true in the state at some time along \( \pi \). Was a landmark once accepted this status is stored and maintained.

The required again landmarks are a subset of the accepted landmarks \( \text{ReqAgain}(s, \pi) \subseteq \text{Accepted}(s, \pi) \). That can be the case for a landmark that is required again by a path in a state if the landmark is false in the state and is a goal. That situation is called a false-goal. When the landmark is false in the state and is a predecessor of some other landmark that is not accepted. Than it is called a open-prerequisite situation. There also exists the doomed-goal which is if a landmark is true in a state and is also a goal, but one of its predecessors was not accepted and is inconsistent with the landmark. The heuristic is admissible under special conditions but it is not consistent (Helmert and Domshlak, 2009; Richter et al., 2008; Richter and Westphal, 2010).

\[ h^{LM-cut} \]

Normally landmark heuristics are path-dependent but the landmark cut heuristic is not. This heuristic is part of a new approach for finding landmarks. It computes landmarks for each state rather than once and this gives a very accurate admissible approximation to \( h^+ \). The first idea of it is by (Helmert and Domshlak, 2009). They use the critical paths from the \( h^{max} \) computation for finding disjunctive action landmarks. Next they extract landmarks iteratively and subtracting their cost of the \( h^{max} \) estimate. Afterwards they determine the cost of each landmark as the cost of cheapest action in landmark. Further they define a heuristic value as the sum over all landmark costs.

The following example by (Keyder and Richter, 2011) simplifies the understanding for the procedure. In figure 2.11 is shown up the transportation of an object from A to B. The given possibilities to do this are by truck or by teleportation. The \( h^{LM-cut} \) heuristic starts by computing the \( h^{max} \) costs for all facts. Afterwards the goals and preconditions will reduce to singleton sets in a way that preserves \( h^{max} \). Now all multi-effect operators will be replaced by a set of unary operators in a way that preserves \( h^{max} \). The next step is to compute the justification graph for the resulting task with the values from \( h^{max} \) as the shortest distances. The created justification graph is shown in figure 2.12. Now the
cut comes. This will be computed in the justification graph that separates the before-goal zone (red circles) from the goal zone (green circle). The made cut corresponds to a disjunctive action landmark in the result $\Rightarrow$ teleport $-o \lor unload -o$. It has to be extracted through cost partitioning. From now they must start by the beginning until each $h_{\text{max}}$ value is 0. Extract next $\Rightarrow$ teleport $-o \lor load -o \lor teleport -t$ and $\Rightarrow$ teleport $-o \lor drive -t$. The graph after extractions is shown in figure 2.13. Now we get a set of landmarks

$\text{Landmarks} = \{teleport - o \lor unload - o, teleport - o \lor load - o \lor teleport - t, teleport - o \lor drive - t\}$

and a landmark-cut heuristic value

$$h_{LM - \text{cut}} = 3$$

The landmark cut heuristic is more expensive than other landmark heuristics but approximates $h^+$ closely. The landmarks are not computed per task rather per state (Helmert and Domshlak, 2009).
This is a simple heuristic that shows a way to get an estimate of the goal distance $h^*(s)$ for landmark heuristics. The trick is to differ from actions that can possibly be used to directly achieve landmark $\phi$ along a goal-achieving suffix of $\pi$. The cost of a landmark must be smaller or equal as the minimal cost of an action to this landmark. The following formula reflect this definition

$$\forall \phi \in L(s, \pi) : \text{cost}(\phi) \leq \min_{a \in \text{ach}(\phi|s, \pi)} \text{cost}(a, \phi)$$

where $\text{ach}(\phi|s, \pi) \subseteq A$ is an action subset that contains all actions that can possibly be used to directly achieve a landmark along a goal-achieving suffix of $\pi$.

With this definition for action costs we may define the formula to get an estimate of the goal distance (Karpas and Domshlak, 2009; Keyder et al., 2010).

$$h^L(s, \pi) = \text{cost}(L(s, \pi)) = \sum_{\phi \in L(s, \pi)} \text{cost}(\phi)$$

$h^{LA}$

The admissible landmark heuristic or $h^{LA}$ is similar to the above landmark heuristic. The admissible landmark heuristic as the name is saying returns an admissible estimate. This is possible by reducing all the pre-discovered action landmarks that were not used along the path $\pi$ to evaluated state $s$. This sum of pre-discovered action landmarks are represented by $U(s, \pi)$. First step is to sum up the cost of all the unused action landmarks. Then remove from the landmark set $L(s, \pi)$ all the landmarks achievable by the actions in $U(s, \pi)$. Now perform only the regular action cost sharing over the remaining landmarks. Doing that we get the formula for the heuristic estimate of $h^{LA}$

$$h^{LA}(s, \pi) = \text{cost}(L_U(s, \pi)) + \sum_{a \in U(s, \pi)} C(a),$$

where

$$L_U(s, \pi) = L(s, \pi) \setminus \bigcup_{a \in U(s, \pi)} L(a|s, \pi)$$

and

$$\text{cost}(L_U(s, \pi)).$$

In other words the heuristic estimates are the cost of the edited landmark subset $(L_U(s, \pi))$ plus the sum of the non negative cost of each action $(C(a))$ that are a subset of $U(s, \pi)$ (Karpas and Domshlak, 2009; Keyder et al., 2010).
Chapter 3

Methods to Detect and Avoid Dead Ends in Action Planning

In this chapter I will introduce a formal definition for elements that are related with dead ends. In this chapter I will present methods that I will examine for detecting dead end states in action planning. At first I define facts around dead ends formally.

Defines for the dead end environment:

- A jump point \( \lambda \) is a pair of one state \( s \) and an action \( a \) from this states
- A deadline \( \theta \) is a jump point that leads to a dead end state
- A normal jump point \( \lambda \) is a jump point that do not leads to a dead end state

With the definitions we can create a clear structured dead end detection. Now we can say a dead end is defined by the formula

\[
\neg \exists \pi(s, s') \ g \subseteq s
\]

For all jump points we may separate them as

\[
\forall \lambda = \begin{cases} 
\theta & \text{if } \lambda \Rightarrow s_{\text{dead}} \\
\lambda_{\text{normal}} & \text{otherwise}
\end{cases}
\]

All methods I will present are constructed to find all jump points but I will just compare the methods based on the number of deadlines found.
3.1 Detect Dead Ends

3.1.1 Heuristic Touched Diligent Search

After I searched the complete search space I want to search for deadlines in a more performant way. The Fast Downward planner has a decision inside when a solution is found. I will modify that decision. That means that at any place of the source code must be a simple question if all targets are satisfied or not. I run a heuristic search with the conditions of never satisfiable solutions. Every domain problem has a solution for which the planner is searching. I may change this solution decision. The assumption is that the planner will again search the whole state space but uses the trend of the heuristic function. The assumption is that not every state must be visited. The search will be much faster than the diligent search but we find more deadlines than the normal search. The heuristic functions will prune a lot of states that the search does not have to visit. On the other hand the search will not end even if all goals are found. I will run this setting with the heuristic functions the fast downward planner provides. The result of this experiment may be found in chapter 5.1.1.

3.1.2 Uniform Cost Search with Heuristic Tendency

The next approach is to obtain more of the performance from the normal heuristic search. I will use the heuristic function in the normal way but change the first cost evaluation of every state visit first. Normally a successor state gets an evaluated value by

\[ f(x) = g(s) + g(o) \]

I will change the first evaluation value for new states too

\[ f(x) = g(o) \]

that means only the operator cost is the estimate for the successor state. Because the value of the operators are zero for the most domains we simulate an uniform cost search. And for the whole search it is

\[ f(x) = \begin{cases} g(o) & \text{if } s \text{ visited for the first time} \\ g(s) + h(o) & \text{if } s \text{ already visited} \end{cases} \]

If the state is visited a second time, the heuristic value of the state will be evaluated normally. This procedure force the algorithm to search extensive for new states but with a better tendency if it comes again to an already visited state. I show the results of this approach in chapter 5.1.2.
3.1.3 Modify Known Heuristics

A further and more efficient approach is to modify known heuristic functions. In the second chapter I present some heuristic functions that have the target to find a preferably good solution of a problem task. Now I will use this preferences of the given heuristic functions to detect dead ends. If a heuristic function give a tendency for reaching a target it must be possible to modify the heuristic in such a way it does the opposite. I will modify a few of the heuristic functions I presented above and switch the heuristic decision that they tended to dead ends. That means I do not have to visit the whole state space which gives us a more smart and useful method. The main approach is to find the estimate decision of the heuristic function and to manipulate it. After the manipulation the heuristic does not give any longer a tendency for the goal direction. The heuristic functions I will research for this method are the following ones

- Max heuristic \( h^{\text{max}} \)
- Additive heuristic \( h^{\text{add}} \)
- Causal graph heuristic \( h^{\text{cg}} \)
- Context-enhanced additive heuristic \( h^{\text{cea}} \)
- FF heuristic \( h^{\text{ff}} \)
- Landmark cut heuristic \( h^{\text{lm-cut}} \)
- Merge and shrink heuristic \( h^{\text{MS}} \)

I choose this heuristics because they cover the most of the four heuristic distributions I presented in caption two. There is no critical path heuristic because the \( h^{m} \) heuristic family is too slow for any comparison that is also proposed by the developer of the Fast Downward planner. That is because this heuristic is better used in backwards search. But with the other heuristics I may cover the most differences between the most heuristics. There are delete relaxations with the \( h^{\text{max}}, h^{\text{add}}, h^{\text{cg}}, h^{\text{cea}} \) and \( h^{\text{ff}} \) heuristics, an abstraction heuristic \( h^{\text{MS}} \) and also a landmark heuristic \( h^{\text{lm-cut}} \). With five delete relaxation heuristics it is the only group with more than one heuristic. These heuristics are often used and may give a good additional comparison for this subgroup of heuristics.

The big innovation point is to change the output for found dead ends. For normal a heuristic returns smaller values if a goal state is near. I will change this behaviour of the planner by inverting these values. I may do this by calculate the maximum possible value minus the original heuristic estimate. I will not change the return of the heuristic function if a dead end a goal state is found. This force the planner to tend much more in the direction of dead ends. Figure 3.1 explains the principle. It is important not to change the return if a dead end state or a goal state is found because I need to recognize dead ends and also if the search founds a solution.
The left side of the picture shows the normal diagram for the estimation values. Zero for goal states and infinity for dead ends the values for all other states are from low to high between them. On the right side you may see the estimation values for the new principle. Now the values between goal states and dead end states are from high to low.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure31.png}
\caption{Principle of modified heuristics}
\end{figure}

### 3.1.4 Wisp

For this approach I will change the heuristic function by integrating a new variable. This value deceive the algorithm. The approach is that every time a state is evaluated with zero I will change the heuristic value for this state. If the state evaluation is zero that means the algorithm has found a goal state. This variable may be seen as a threshold for misleading of the search algorithm. I suspect to find more dead ends if the algorithm is more distract from the original goals of the search. I call this variable \textit{Wisp} and will research the results for different values.
Chapter 4

Implementation

I give at first a introduction in the functionality of the used Fast Downward planner and how to use it. I give a short overview about the domains that are useful for detect deadlines and how the data may be recovered from the planner. Afterwards I show the changes of the planner to implement the methods presented in the last chapter.

4.1 Run the Fast Downward Planner

For the implementation of the methods I use the Fast Downward planning framework, a state of the art planner. I already showed some facts about the planner in chapter 2. The planner provide some benchmarks of problem domains, different heuristics and search algorithms.

The planner is available at the developer homepage Hel (2012). There are many informations and instruction how to run the planner at your system. To run the planner we have to execute the three phases of the fast downward planner. These I will describe now.

The three fast downward phases of execution (Helmert (2006))

- Translation
- Knowledge Compilation
- Search

**Translation** is the component where the PDDL input is transformed to a non-binary form which is easier to handle for further hierarchical planning approaches. There is a number of normalizations. These are needed to compile away syntactic constructs like disjunctions. An important feature is the use of invariant synthesis methods for finding groups of related propositions. This can be encoded as a single multi-valued variable.
The execution creates a multi-valued planning task as output. In other words: the phase has the main jobs of handling of invariants and groundings.

**Knowledge compilation** is the component that generates a data structure consisting of four kinds which plays a central role during search. First part is the *Domain Transition Graph* that encodes how and under which conditions state variables change their values. The hierarchical dependence between the different state variables are represented by a *Causal Graph*. The set of applicable operators in a given state can be determined by an efficient data structure called the *Successor Generator*. The last of the four kinds is the axiom evaluator. It is an efficient data structure for computing the values of derived variables.

**Search** is the last component and contains three different implementations of search algorithms that can be used for planning. The first algorithm is the *greedy best-first search*, the second one is an extension of it which tries to combine several heuristics the heuristic best-first search and the third algorithm is called *focused iterative-broadening search*. The first two algorithms are using heuristic evaluation functions the last one just use the encoded information to estimate the usefulness of operators towards satisfying the goals of the task.

I am only interested in the search part of the planner. To run the planner you have to understand the other components, too.

This first steps are easy and except of a first small compatibility issue with my operation system I had no problems to run the planner for the first time. The whole code is mainly implemented in Python and C++ and the planners directory structure is very complex.

### 4.2 First Steps

The Fast Downward planning system has a huge framework for planning and at first I had to familiarize myself with the planner. As search algorithm I chose the eager search for the further work.

I also have to know how the planner is working while running. I have analysed these procedures of the planner. To start the planning task we need to run the translation part. In the command line I just have to type in

```
./translate DOMAIN PROBLEM
```

In the first steps to get the translation for a domain problem. Now I have to move the output to the pre-process directory and type in the follow command

```
./preprocess < OUTPUT
```
After this steps I am ready to start a search. I must move the created output into the search directory. To run a search with a special heuristic function the command is

```
./downward [OPTION] --search SEARCH < OUTPUT
```

We may see there is a field for options this is for adding heuristic predefinitions or (and) random seed to the search. The SEARCH field stands for one of the implemented search algorithms and integrates here also the applied heuristic that is called in the search algorithm. The OUTPUT stands for the file of the preprocessing. For example a call of an A* algorithms that is using a lm-cut heuristic is

```
./downward --search "astar(lmcut())" < output
```

While running the search we get some information about the run in the command line. How much time the search needed, the expanded nodes, the reopened nodes, the length of the founded plan if one exists, the cost of this plan and some other information. This output information can be customized at some points in the source code. It is a good thing to customize the output to a format that is helpful for the own tasks.

I am searching for the deadlines like I have presented it in the previous chapter. To get this I searched the location of the source code where states are explored for the first time. I found the part where for a current state all applicable operations and the following successor states are considered. If the successor state is a new one it is evaluated. After this evaluation I check if it is a dead end and if it is a dead end I catch the current state and the operator that leads to the dead end. With that I get all deadlines that are found while searching.

Because I search for the deadlines I had to do further work. When the search comes to a state that was visited before, it checks if it is a dead end or not. If it is not a dead end the search will reopen the state and take a new evaluation of it. If it is a dead end the state will just be ignored and the search continue but this combination of state and operator may be a new deadline. What I have done was to catch also the deadline if a dead end is reached again because a dead end state can have more than one deadlines.

Now I have the data of every state and operator that lead to a dead end. I also catch the normal jump points of states because these are also interesting for a learning task. One big problem I had was that the decision whether a state is a dead end or not is made by the heuristic function. If the heuristic functions do not need dead ends for creating an estimate there are no dead ends and also no deadlines I can catch.

After these first preliminary steps it is also important to make a good choice of the planning domains for running the search tasks.
4.2.1 Choose the Right Domains

In the context of finding dead ends it is not unimportant to choose domains for the search task. Not all domains make sense. Some domains do not contain dead ends and a search for dead ends is pointless. This for example is the case for the blocks world domain. The domain is about blocks at a table and the possibility to lift them up and down. The blocks can be at the table or at another block. The operators can lift them up and down as often as possible but there is never a state that can not be reached.

The fast downward planner supplies a lot of domains and also some problems for each of them. They can be found in the benchmarks directory of the planner. There are small domains and also some huge domains. For a better comparison I choose more smaller domains because some of the methods I will implement need much time and space and will probably fast lead to a memory limit exception if the domain is too big. That is also interesting but not useful here.

I choose four domains that all contain dead ends and I also choose more than one problem for each of them. That gives the additional opportunity for comparison. The four domains are

- Sokoban
- Mystery
- Parcprinter
- Pathways

For each domain I examine three problems except the Sokoban domain where I examine five problems. This choice contains domains in different sizes and allows a comparison of the method effectiveness under different conditions. I chose more problems for the Sokoban domain because it is a good representation for building a search graph.

4.2.2 Data Recovery

As example I will explain one of the domains in detail. The Sokoban domain is a good domain for planning. It is an old game easy to understand and represent the state space of a planning task. At first I will give a short explanation of the game rules. A example of a Sokoban field is given in figure 4.1. The game contains a player, a box or stone and a target this components may seen in the figure from down to up. Aim of the game is to push the box at the target.

A state in the state space is a reflection of this Sokoban field described by the condition of every field and every object. This mean a position for example $\text{var}(2/2)$ can be two condition. In our case it is clear or negated. Also the object player can have some conditions for every position he could stand. Now I will give an example of a state output where we may see the condition allocation:
Chapter 4. Implementation and Evaluation

Figure 4.1: Small Sokoban domain

\begin{verbatim}
Atom clear(pos-6-2) -> 0
Atom clear(pos-5-2) -> 0
Atom clear(pos-5-3) -> 0
Atom clear(pos-6-3) -> 0
Atom clear(pos-4-2) -> 0
Atom clear(pos-6-5) -> 0
Atom clear(pos-6-6) -> 0
Atom clear(pos-5-5) -> 0
Atom clear(pos-5-6) -> 0
Atom clear(pos-2-5) -> 0
NegatedAtom clear(pos-3-4) -> 1
Atom clear(pos-3-5) -> 0
Atom clear(pos-4-3) -> 0
Atom clear(pos-2-4) -> 0
Atom clear(pos-2-6) -> 0
Atom clear(pos-3-6) -> 0
NegatedAtom clear(pos-4-4) -> 1
Atom at(stone-01, pos-3-4) -> 3
Atom clear(pos-4-6) -> 0
NegatedAtom clear(pos-4-5) -> 1
Atom at(player-01, pos-4-4) -> 8
Atom at(stone-02, pos-4-5) -> 9
NegatedAtom at-goal(stone-02) -> 1
NegatedAtom at-goal(stone-01) -> 1
\end{verbatim}

Listing 4.1: Output of a planning state

The first lines correspond to the fields of the Sokoban Domain. Their conditions may be "clear" if nothing is standing on the field or it is a negated "clear" if the player or a stone is on the field. The last two lines are the goal conditions. Until they are "1" they are unsatisfied if they are "0" a stone is on a goal field and the goal condition is satisfied.

An operation is defined by the changes of these conditions that are made if one state is left and an other is reached and the action that has triggered the changes. In our case
this may be actions like *move* if just the player is moved or *push* if the player pushes
the box in front of him. As example I will show such an action:

```
push-to-nongoal
player-01 stone-02 pos-3-5 pos-4-5 pos-5-5 dir-right:
[ var0: 4 => 9 ] [ var2: 9 => 12 ] [ var4: -1 => 1 ]
[ var9: -1 => 0 ] [ var18: 0 => 1 ]
```

Listing 4.2: Output of a planning state

This is an example of a founded deadline in a Sokoban domain. The player pushes a
stone from the location "4/5" to the location "5/5" in the right direction. After the
action we may see variables with some numbers that are changed. This variables are
the objects of the current state that change their conditions with the action.

### 4.2.3 Used Hardware

To perform the experiments I used following environment:

- Operating system: Ubuntu 13.10
- Type of OS: 64 Bit
- Processor: Intel Core i7-2640M CPU @ 2.80GHz x 4
- Memory: 8 GiB

With that components I run all methods except the breadth first search.

### 4.3 Add Some Heuristic Tendency

One approach I presented in chapter 3 Approach is to visit every state of the planning
graph, so we have to find every dead end.
The Fast Downward planner normally run until all given goals are reached, an error
occurs or the state space is completely explored. To check if all goals are satisfied the
planner check the list of goals a which condition they have. If every goal have the
condition satisfied a solution was found and the planner stop searching. Now I change
this part of the planner, so the query whether a goal is found or not response every time
*false*.

With this configuration I force the planner to search until the complete state space is
explored. The results of this experiment can be found in chapter 5.
4.4 Start Breadth Become Faster

The implementation of this approach takes place in the source code of the search. Every time the search algorithm reach a new state I change the state evaluation from

```c
search(){
    ...
    state_estimate = estimate(node_cost + operator_cost);
    ...
}
```

Listing 4.3: Output of a planning state

```c
search(){
    ...
    state_estimate = assign_estimate(operator_cost);
    ...
}
```

with this change every new state get only the cost of the operation as evaluation value. The search expand at first all nearest states and with a second visit he continue the search with a normal calculated heuristic estimate.

4.5 Modify Existing Heuristics

The modification of existing heuristic is the main approach of the thesis. As I write in the last chapter I want to modify the heuristic decision that gives the search algorithm the heuristic value. The method call for the heuristic value is similar for the most heuristics. Just a few smaller changes for each heuristic is necessary. A function is called by the search that returns an integer value as heuristic value. Next I will give a short pseudo source code fragment of this method:

```c
compute_heuristic(State){
    h = compute_heuristic_value()
    ...
    return h
}
```

Like the principle presented in chapter 3.1.3 I invert every evaluation value except for dead end states and goal states. I do this by subtracting the current calculated heuristic value from the maximal possible value. The next pseudo code shows the changes:
Now the method returns the invert heuristic value. If the heuristic value is zero it does not touch because it is a goal state. I need to continue recognize them for finding a solution and also because it would be seen as a dead end.

The results of this approach are presented in the next chapter.

### 4.6 Integrate Wisp Into Heuristics

The implementation of this approach is quite similar to the last section of the implementation of modified heuristics. I change the source code at the same function but this time I only act if a goal state is found by the heuristic. If this happen I return a higher value instead of the original estimate. With this change the search algorithm have to search further. The change of the source code is show in the following pseudo code fragment:

```c
compute_heuristic(State){
    h = compute_heuristic_value()
    ...
    if(h != DEAD_END && h != 0)
        h = highest_possible_value - h
    return h
}
```

The modification can be vary by the Wisp value. I run this method two times for all problems and heuristics with the values $Wisp = 1$ and $Wisp = 10$. The results may be presented in the evaluation chapter 5.1.4.
Chapter 5

Evaluation

In this section at first I will introduce a limit for the search of deadlines. Afterwards I present the results of the explored methods.

Explore the Complete State Space

If we visit every state of the graph we will find all deadlines. The big disadvantage is the bad performance to visit the whole graph. It needs a lot of time and in some bigger domains it is almost impossible to get a result. But it is a first approach which gives all deadlines for domains under the certain circumstances of enough calculation power and memory. In other words we make a breadth first search over the whole state space. A formal definition by (Ghallab et al., 2004):

\[
\text{State-space-search}(O, s_0, g)
\]

\[
\begin{align*}
\text{if } g(s) \text{ then return } s \\
\text{applicable } &\leftarrow \{\text{all operators applicable to } s\} \\
\text{if applicable} &\neq \emptyset \text{ then return failure} \\
\text{nondeterministically choose an action } o &\in \text{applicable} \\
\text{s' } &\leftarrow o(s) \\
\text{return State-space-search(s', g, O)}
\end{align*}
\]

Listing 5.1: Nondeterministic state-space procedure

There is no heuristic used in this search but this approach will give an upper limit. The results of this experiment may be found in table 5.1.
Implement of the Breadth First Search

To get a search for the whole state space I use another implementation than the Fast Downward planner. I use again the output of the preprocessing but now I search the state space with a breadth first search. In our case this means to visit every state and make sure that it is a dead end. This will happen by taking a search from every current state for every operator until no target was found. In pseudo code we may say as following:

```plaintext
Extract(s_0)
insert.(s_0)
while(!all states are visited){
  for(all operators){
    if(lead to dead end)
      add as negativ
    else{
      add as positiv
      add state to queue
    }
  }
}
```

Listing 5.2: Output of a planning state

with the function for detecting dead ends "lead to dead":

```plaintext
Is_dead
init nodestack
while(!nodestack is empty){
  if(goal)
    return false
  else
    push node to stack
}
return true
```

Listing 5.3: Output of a planning state

This is a cost intensive procedure and needs a lot of time to be done but is very helpful to know how many deadlines exists in a domain.
I give an example at the small domain from chapter 4.1. When we run a complete state space search the algorithm finds 23 positive examples and only 2 negative examples. Positive examples are normal jump points and negative examples are deadlines. The figure 5.1 shows the described domain with all deadlines that are found by the complete
state space search. In this really small domain the complete search is not a problem but for only a bit bigger domains the search needs at least some days for finding a solution. Therefore I run just the smaller domains with this approach. This is enough for a comparison because we are not much interested in these results because of the bad performance.

![Figure 5.1: Complete state space search example](image)

**Results Explore the Complete State Space**

The following results are from the experiment to run a breadth first search over the whole state space to detect all deadlines but with the loss of performance. Target is to get as near as possible to each value with the modification methods. Because of the complexity I just analyse the smaller domains for this approach. The results are shown in table 5.1

<table>
<thead>
<tr>
<th><strong>Domain</strong></th>
<th><strong>Deadlines</strong></th>
<th><strong>Normal jump points</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sokoban:P1</td>
<td>2</td>
<td>23</td>
</tr>
<tr>
<td>Sokoban:P2</td>
<td>7</td>
<td>83</td>
</tr>
<tr>
<td>Sokoban:P3</td>
<td>23</td>
<td>394</td>
</tr>
<tr>
<td>Sokoban:P4</td>
<td>34</td>
<td>713</td>
</tr>
<tr>
<td>Parcprinter:P1</td>
<td>3</td>
<td>17</td>
</tr>
<tr>
<td>Pathways:P1</td>
<td>1002</td>
<td>1203</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1071</td>
<td>2433</td>
</tr>
</tbody>
</table>

The definition of the search algorithm allows duplicated jump points. That means the number of normal jump points may be higher than it actually is. The number of deadlines is the true one.
Get The Basic

To find out how many deadlines must be found to get a better result than a normal run of a heuristic search it is important to know how many deadlines can be found by a normal run. I run every heuristic function in combination with the eager search in every domain in which I will apply my developed methods. This step gives a lower limit. The target is to get more deadlines than this limit. I can check my results on this boundary to know if the results are better or worse. The formula of the eager search is

\[ f(x) = g(x) + h(x) \]

\(g(x)\) are the current cost of a state and \(h(x)\) returns the heuristic estimate that is calculated by a heuristic function. I will take a look at the results of all heuristic functions of the downward planner in some domains. The results can be found in 5.2.

Normal Run

For getting the lower limit I run the planner without any modification. This means I made the translation and the preprocessing step for every domain. This must be done only once the output file can be used later for the modified searches too. These heuristic functions are

- Max heuristic \(h^{\max}\)
- Additive heuristic \(h^{add}\)
- Causal graph heuristic \(h^{cg}\)
- Context-enhanced additive heuristic \(h^{cea}\)
- FF heuristic \(h^{ff}\)
- Landmark cut heuristic \(h^{lm-cut}\)
- Merge and shrink heuristic \(h^{MS}\)

These heuristics will be also used for the other methods.

Number of Dead Ends in Normal Run

Now I will show the deadlines that are found by the planner without modification of any setting. I just run the planner in some different domains with different heuristic functions and give a view at the resulting deadlines. The first results of running the planner without changes can be found in table 5.2. These values can be seen as a lower limit of searchable deadlines. Time table of the results found in table A.1 of the appendix.
5.1 Results of Research

Now I will present the results of detecting deadlines by the new methods. Not every search delivers a solution. If the search was aborted a cell is marked in the table. Other runs do not abort by themselves but were aborted by the operator after they run too long. The results of these runs are the current number of deadlines after abort.

5.1.1 Heuristic Complete Search

In this section I present the results of running the search algorithm for the complete state space but with the tendency of heuristic functions. The results can be seen in table 5.3. In the table you will see this method perform very well for small problems like the Sokoban problem P1 - P2 and the Parcprinter problem P1. Each heuristic finds all available deadlines. You may also obtain that this method will not give a solution for bigger problems. As described in the captions before the method needs too much resources to solve these problems.

Another observation is that some heuristics give more deadlines than the breadth first search in the paragraph before. That is not an error of the method it is due to the principles of the heuristic functions. Some of them search further also if one dead end is reached. For example the $h_{max}$ heuristic return $-1$ if it finds a dead end state. Further it calculates the maximum of the ways to different goal criteria. If one of these ways is a dead end the heuristic function looks at first at the other paths. That is why they may find more then theoretical possible. It would be nice to avoid these deadlines because we do not need them for learning. Time table of the results found in table A.2 of the appendix.
Table 5.3: Complete Search with Heuristic Tendency

<table>
<thead>
<tr>
<th>Domain</th>
<th>$h_{max}$</th>
<th>$h_{add}$</th>
<th>$h_{cg}$</th>
<th>$h_{cea}$</th>
<th>$h_{ff}$</th>
<th>$h_{lm-cut}$</th>
<th>$h_{MS}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sokoban:P1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Sokoban:P2</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Sokoban:P3</td>
<td>42</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>Sokoban:P4</td>
<td>71</td>
<td>45</td>
<td>39</td>
<td>3</td>
<td>44</td>
<td>44</td>
<td>34</td>
</tr>
<tr>
<td>Sokoban:P5</td>
<td>94</td>
<td>108</td>
<td>89</td>
<td>94</td>
<td>105</td>
<td>105</td>
<td>62</td>
</tr>
<tr>
<td>Sokoban:P6</td>
<td>38</td>
<td>20</td>
<td>19</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Mystery:P1</td>
<td>482</td>
<td>7</td>
<td>16</td>
<td>16</td>
<td>7</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>Mystery:P2</td>
<td>4475</td>
<td>45139</td>
<td>37564</td>
<td>37564</td>
<td>44793</td>
<td>4475</td>
<td>3102</td>
</tr>
<tr>
<td>Mystery:P3</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Parcprinter:P1</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Parcprinter:P2</td>
<td>50</td>
<td>108</td>
<td>108</td>
<td>108</td>
<td>108</td>
<td>108</td>
<td>76</td>
</tr>
<tr>
<td>Parcprinter:P3</td>
<td>4032</td>
<td>494</td>
<td>463</td>
<td>494</td>
<td>500</td>
<td>494</td>
<td>138</td>
</tr>
<tr>
<td>Pathways:P1</td>
<td>1104</td>
<td>1104</td>
<td>1104</td>
<td>1104</td>
<td>1104</td>
<td>1104</td>
<td>1002</td>
</tr>
<tr>
<td>Pathways:P2</td>
<td>3732</td>
<td>3732</td>
<td>3612</td>
<td>3732</td>
<td>3732</td>
<td>3732</td>
<td>1592</td>
</tr>
<tr>
<td>Pathways:P3</td>
<td>5469</td>
<td>199167</td>
<td>199167</td>
<td>199167</td>
<td>199167</td>
<td>199167</td>
<td>3774</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>59802</td>
<td>249959</td>
<td>242215</td>
<td>242337</td>
<td>249615</td>
<td>249489</td>
<td>9835</td>
</tr>
</tbody>
</table>

*Abort after a memory exception: not enough memory

5.1.2 Uniform Cost Search with Heuristic Tendency

The next results are presenting the approach to evaluate every newly found state just with the cost of the previous state. I assume to get more deadlines if the search starts to expand to the breadth first and than carry on normal. The results are shown in table 5.4. Time table of the results found in table A.2 of the appendix.

Table 5.4: Uniform Cost Search with Heuristic Tendency

<table>
<thead>
<tr>
<th>Domain</th>
<th>$h_{max}$</th>
<th>$h_{add}$</th>
<th>$h_{cg}$</th>
<th>$h_{cea}$</th>
<th>$h_{ff}$</th>
<th>$h_{lm-cut}$</th>
<th>$h_{MS}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sokoban:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P3</td>
<td>19</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Sokoban:P4</td>
<td>25</td>
<td>22</td>
<td>15</td>
<td>3</td>
<td>22</td>
<td>21</td>
<td>18</td>
</tr>
<tr>
<td>Sokoban:P5</td>
<td>36</td>
<td>26</td>
<td>22</td>
<td>23</td>
<td>25</td>
<td>26</td>
<td>15</td>
</tr>
<tr>
<td>Sokoban:P6</td>
<td>19</td>
<td>14</td>
<td>11</td>
<td>14</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Mystery:P1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mystery:P2</td>
<td>22</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Mystery:P3</td>
<td>3522</td>
<td>7</td>
<td>16</td>
<td>16</td>
<td>7</td>
<td>34</td>
<td>*</td>
</tr>
<tr>
<td>Parcprinter:P1</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Parcprinter:P2</td>
<td>20</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>13</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>Parcprinter:P3</td>
<td>728</td>
<td>10</td>
<td>8</td>
<td>10</td>
<td>13</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Pathways:P1</td>
<td>1074</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pathways:P2</td>
<td>3545</td>
<td>66</td>
<td>66</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>Pathways:P3</td>
<td>5469</td>
<td>216</td>
<td>112</td>
<td>216</td>
<td>112</td>
<td>36</td>
<td>658</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>14481</td>
<td>376</td>
<td>200</td>
<td>365</td>
<td>237</td>
<td>176</td>
<td>761</td>
</tr>
</tbody>
</table>

*Abort after a memory exception: not enough memory
5.1.3 Modified Heuristic Search

This section presents the results of using modified heuristics for a heuristic search. The results may be seen in table 5.5. Time table of the results found in table A.4 of the appendix.

<table>
<thead>
<tr>
<th>Domain</th>
<th>( h^{\text{max}} )</th>
<th>( h^{\text{add}} )</th>
<th>( h^{\text{cg}} )</th>
<th>( h^{\text{cea}} )</th>
<th>( h^{\text{ff}} )</th>
<th>( h^{\text{lm}} - \text{cut} )</th>
<th>( h^{\text{MS}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sokoban:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Sokoban:P3</td>
<td>13</td>
<td>20</td>
<td>15</td>
<td>15</td>
<td>17</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Sokoban:P4</td>
<td>41</td>
<td>33</td>
<td>39</td>
<td>3</td>
<td>44</td>
<td>44</td>
<td>33</td>
</tr>
<tr>
<td>Sokoban:P5</td>
<td>65</td>
<td>92</td>
<td>81</td>
<td>51</td>
<td>80</td>
<td>94</td>
<td>35</td>
</tr>
<tr>
<td>Sokoban:P6</td>
<td>37</td>
<td>18</td>
<td>16</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>Mystery:P1</td>
<td>44313</td>
<td>3936</td>
<td>700</td>
<td>235</td>
<td>540</td>
<td>1630</td>
<td>43</td>
</tr>
<tr>
<td>Mystery:P2</td>
<td>29623640*</td>
<td>5194268</td>
<td>54689</td>
<td>57911</td>
<td>1380746</td>
<td>3775985</td>
<td>1440967</td>
</tr>
<tr>
<td>Mystery:P3</td>
<td>23111813†</td>
<td>*</td>
<td>*</td>
<td>21546324†</td>
<td>24389403†</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Parcprinter:P1</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Parcprinter:P2</td>
<td>33</td>
<td>108</td>
<td>43</td>
<td>43</td>
<td>48</td>
<td>48</td>
<td>36</td>
</tr>
<tr>
<td>Parcprinter:P3</td>
<td>285</td>
<td>257</td>
<td>252</td>
<td>269</td>
<td>269</td>
<td>254</td>
<td>50</td>
</tr>
<tr>
<td>Pathways:P1</td>
<td>328</td>
<td>1104</td>
<td>715</td>
<td>715</td>
<td>715</td>
<td>715</td>
<td>613</td>
</tr>
<tr>
<td>Pathways:P2</td>
<td>1312</td>
<td>3732</td>
<td>3095</td>
<td>3215</td>
<td>3446</td>
<td>3446</td>
<td>1523</td>
</tr>
<tr>
<td>Pathways:P3</td>
<td>2051</td>
<td>29391</td>
<td>28815</td>
<td>29391</td>
<td>29391</td>
<td>29391</td>
<td>3774</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>52783868</strong></td>
<td><strong>41232724</strong></td>
<td><strong>88464</strong></td>
<td><strong>21580275</strong></td>
<td><strong>25804718</strong></td>
<td><strong>3811645</strong></td>
<td><strong>1447112</strong></td>
</tr>
</tbody>
</table>

5.1.4 Wisp

The following results are from the Wisp integrated approach. The method was run with a Wisp value of "1" (see results in the next table 5.6) and with a Wisp value of "10" (see results in table 5.7).

Wisp = 1

Time table of the results found in table A.5 of the appendix.

Wisp = 10

Time table of the results found in table A.6 of the appendix.

* Abort after a memory exception: not enough memory † Operator initiated termination after an run over 24h
### Table 5.6: Heuristic Search with Integrated Wisp = 1

<table>
<thead>
<tr>
<th>Domain</th>
<th>$h_{max}$</th>
<th>$h_{add}$</th>
<th>$h_{cg}$</th>
<th>$h_{cea}$</th>
<th>$h_{ff}$</th>
<th>$h_{lm-cut}$</th>
<th>$h^{MS}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sokoban:P1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Sokoban:P2</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Sokoban:P3</td>
<td>18</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>14</td>
</tr>
<tr>
<td>Sokoban:P4</td>
<td>18</td>
<td>24</td>
<td>17</td>
<td>3</td>
<td>18</td>
<td>18</td>
<td>21</td>
</tr>
<tr>
<td>Sokoban:P5</td>
<td>22</td>
<td>32</td>
<td>15</td>
<td>38</td>
<td>18</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>Sokoban:P6</td>
<td>18</td>
<td>17</td>
<td>17</td>
<td>18</td>
<td>18</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>Mystery:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mystery:P2</td>
<td>24</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Mystery:P3</td>
<td>2885</td>
<td>14</td>
<td>34</td>
<td>42</td>
<td>14</td>
<td>53</td>
<td>0</td>
</tr>
<tr>
<td>Parcprinter:P1</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Parcprinter:P2</td>
<td>62</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>36</td>
<td>36</td>
<td>54</td>
</tr>
<tr>
<td>Parcprinter:P3</td>
<td>57</td>
<td>13</td>
<td>9</td>
<td>13</td>
<td>15</td>
<td>7</td>
<td>74</td>
</tr>
<tr>
<td>Pathways:P1</td>
<td>320</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pathways:P2</td>
<td>3533</td>
<td>88</td>
<td>110</td>
<td>88</td>
<td>198</td>
<td>198</td>
<td>110</td>
</tr>
<tr>
<td>Pathways:P3</td>
<td>37419</td>
<td>1332</td>
<td>1987</td>
<td>1332</td>
<td>112</td>
<td>4761</td>
<td>3250</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>44386</td>
<td>1542</td>
<td>2223</td>
<td>1557</td>
<td>445</td>
<td>5136</td>
<td>3571</td>
</tr>
</tbody>
</table>

### Table 5.7: Heuristic Search with Integrated Wisp = 10

<table>
<thead>
<tr>
<th>Domain</th>
<th>$h_{max}$</th>
<th>$h_{add}$</th>
<th>$h_{cg}$</th>
<th>$h_{cea}$</th>
<th>$h_{ff}$</th>
<th>$h_{lm-cut}$</th>
<th>$h^{MS}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sokoban:P1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Sokoban:P2</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Sokoban:P3</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>Sokoban:P4</td>
<td>44</td>
<td>45</td>
<td>39</td>
<td>3</td>
<td>44</td>
<td>44</td>
<td>33</td>
</tr>
<tr>
<td>Sokoban:P5</td>
<td>97</td>
<td>90</td>
<td>85</td>
<td>83</td>
<td>94</td>
<td>94</td>
<td>54</td>
</tr>
<tr>
<td>Sokoban:P6</td>
<td>18</td>
<td>18</td>
<td>17</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>Mystery:P1</td>
<td>3309</td>
<td>1717</td>
<td>1145</td>
<td>1791</td>
<td>2068</td>
<td>2625</td>
<td>432</td>
</tr>
<tr>
<td>Mystery:P2</td>
<td>480834</td>
<td>35654</td>
<td>32773</td>
<td>37162</td>
<td>63677</td>
<td>95999</td>
<td>21693</td>
</tr>
<tr>
<td>Mystery:P3</td>
<td>22035712*</td>
<td>4993269</td>
<td>12652930</td>
<td>6284367</td>
<td>6696762</td>
<td>17832316*</td>
<td>205056*</td>
</tr>
<tr>
<td>Parcprinter:P1</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Parcprinter:P2</td>
<td>62</td>
<td>45</td>
<td>4</td>
<td>45</td>
<td>34</td>
<td>36</td>
<td>54</td>
</tr>
<tr>
<td>Parcprinter:P3</td>
<td>60</td>
<td>13</td>
<td>9</td>
<td>13</td>
<td>15</td>
<td>7</td>
<td>74</td>
</tr>
<tr>
<td>Pathways:P1</td>
<td>1104</td>
<td>1104</td>
<td>1104</td>
<td>1104</td>
<td>1104</td>
<td>1104</td>
<td>1002</td>
</tr>
<tr>
<td>Pathways:P2</td>
<td>13732</td>
<td>3624</td>
<td>3612</td>
<td>3624</td>
<td>3732</td>
<td>3732</td>
<td>1592</td>
</tr>
<tr>
<td>Pathways:P3</td>
<td>177351</td>
<td>153501</td>
<td>155895</td>
<td>153501</td>
<td>155895</td>
<td>155895</td>
<td>3774</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>22712354</td>
<td>5189111</td>
<td>12847643</td>
<td>6481742</td>
<td>6923474</td>
<td>18091901</td>
<td>28757</td>
</tr>
</tbody>
</table>

* Abort after a memory exception: not enough memory
5.2 Evaluation of Methods

In this section I will make a comparison of the implemented methods to show which one is preferable for finding deadlines. Further the performance will be analysed for this comparison. The diagram in figure 5.2 shows the total results of each method in comparison. In most cases there is a similar relationship between time and the number of deadlines found.

![Figure 5.2: Deadlines by domains for a normal search with Fast Downward](image)

[The vertical axis represents the search time and the horizontal axis represents the deadlines found.]

**The Normal** run returns fast results and finds only few deadlines. As a lower limit I will compare the following method results with the results of this standard method.

**The Heuristic Complete** method found a lot more deadlines than the normal method but the performance is nearly as bad as the complete state space search. For larger problems the search will abort by a memory exception because the memory is fully used and is not sufficient for the search. On the other hand it may be used as an alternative approximation to the complete state space search which is only possible for very small domains. If the method finds a solution the results for each problem are quite constant. This is shown in the bar chart of the heuristic complete search 5.3.
The **Low Cost** method values in the diagram are similar to the normal run of the planner. The method just finds a few more deadlines but also needs nearly as little resources as the normal run.

The **Modify** method has partially good results that may be seen in the diagram. The largest number of deadlines is made by this method but the method needs a lot of time to get the results. Runs had to be cancelled for the largest domain but the method provided a lot of deadlines in this time. For smaller problems it performs quite well, nearly similar to the normal search times but with more hits. It is a good method for detecting deadlines but if the domains are too large it needs too long to return the results. A time limit may be a good solution when using this method. There is a visual direct comparison of the method and the normal method in the appendix that shows the direct difference in deadlines found. See figure 5.4 and 5.5 or in the appendix. In this figures we may see the better result in comparison to the normal run.
Figure 5.4: Normal Run vs. Modify 1
[Comparison of the domain problems Sokoban 1 - 6 and Mystery 1 - 2]
Figure 5.5: Normal Run vs. Modify 2

[Comparison of the domain problems Mystery 3, Parcprinter 1 - 3 and Pathways 1 - 3]
The Wisp approach has very good results and looks like the best method for finding deadlines. If a small Wisp value is chosen the performance is effectively the same as for the normal search but more deadlines are found. A comparison of the bar charts 5.6 and 5.7 show the similarity.

For a given Wisp value of ”10” the method returns very good results. Further the results of this method are almost constant. The bar chart B.7 gives a nice view on this. The method needs more time than the original algorithm but returns in exchange a huge number of deadlines. In smaller problems the performance is similar to the normal performance and also large problems return good results after a while. Additional there is a direct comparison with the normal run. See figure 5.8 and 5.9 or in the appendix. In this figures we may see the better result in comparison to the normal run.
5.3 Evaluation of Heuristics

In this section I give a comparison about using heuristic functions for detecting deadlines. The obtained heuristics follow different procedures to estimate the path to the target. Some of these procedures lead to the problem that a heuristic search continues after a dead end was found. That means that a heuristic may return jump points as deadlines that are no deadlines. That are false positives. That is not good for the intention of this thesis and also the first criterion for detecting deadlines by heuristics. Like I described in 5.1.1 the $h_{max}$ heuristic is one of these heuristics that "oversearch" a problem. That may be demonstrated by a comparison of the complete state space search results and the heuristic tendency complete search results. In stability $h_{max}$ is the most fluctuating one. For some problems the heuristic is the only one that finds deadlines at all finds a lot more deadlines than the other ones. We may obtain this for the normal run also for the low cost search and the Wisp approach. For a few other situations the $h_{max}$ heuristic returns less deadlines than the other ones like for the three problems in the Pathways domain with the modified heuristic search. That shows that the results of this heuristic are depending on the method that we use for search. The merge and shrink heuristic find only the deadlines I am looking for. In other words: the heuristic does
Figure 5.8: Normal Run vs. Wisp
[Comparison of the domain problems Sokoban 1 - 6 and Mystery 1 - 2]
Figure 5.9: Normal Run vs. Wisp
[Comparison of the domain problems Mystery 3, Parcprinter 1 - 3 and Pathways 1 - 3]
not "oversearch". Compare to the complete search for the smaller domain the heuristic finds exactly the same number of deadlines. The problem is, it has poor performance. No other heuristic needs more time to find a solution and no other heuristic aborts as many searches as the merge and shrink heuristic. The stability is good but in most cases it found less deadlines than the other heuristics.

Additive, causal graph, content enhanced additive and ff heuristics are similar in behaviour. They mostly return a good number of deadlines and also perform quiet well. This may be because they are similar in there functionality as delete relaxation heuristics. There are small differences to the others in just a few search runs. For example the normal search for the second problem of the pathways domain $h^{cg}$ does not find any deadline. In the modified heuristic search the $h^{add}$ heuristic finds the highest count of deadlines for most of the problems 5.3. Further they return constant results with a low fluctuation.

The last heuristic is the landmark cut heuristic. It is also similar to the last group of compared heuristics but tends to be a bit worse except for a few problems where this heuristic found more deadlines than the group before. It is also a slow heuristic that needs long time for most of the problems and methods but it is quite constant.

We may also observe that the results of the heuristic functions are depending on domains and problems applied. For small problems all heuristics return similar results but if the problem is a bigger one like the Mystery domain problems the results differ. We may observe this in the Mystery domain for the Modified heuristic search and the Wisp integrated search where Wisp=10.
Chapter 6

Conclusion

The main target of this thesis is to find a method that detects as many as possible deadlines for different domains and problems. Under consideration of the results from the researched methods I come to the decision that the Wisp method is the best method for detecting deadlines. With this method I find a lot of deadlines. The number of deadlines is constant over all domains for every heuristic and the performance is good over all tests that were made. The configuration of the Wisp-value allows an adaptation of the requested quality of results.

When choosing a suitable heuristic there are several possibilities. These are the additive heuristic $h^{add}$, the context enhanced additive heuristic $h^{cea}$ and the ff heuristic $h^{ff}$. They give good constant results with a good performance.

The next steps must be to recognize false jump points within the heuristics. That means that only real deadlines will be found with each heuristic. It may be one more further approach to integrate a dead end recognition in heuristics that can not recognize them now. There are also possibilities to combine the featured methods to get better results.
Bibliography


Appendix A

Time Tables of Methods

In this appendix are the time results of running the methods from chapter 3. All times are presented in seconds and if a task have a zero it mean that the problem was solved under 0.01 seconds. For the total time the fastest one is the best one. If there was a search abort in a run it is not any more a candidate for the overall best time except all others have the same search abort. For an abort by user the time until the abort is notice and mark with ">". This mean it runs over 24 hours. If the run abort by a memory exception you find a ∅ instead of a time.

Table A.1: Times of a Normal Run

<table>
<thead>
<tr>
<th>Domain</th>
<th>$h^{max}$</th>
<th>$h^{add}$</th>
<th>$h^{cg}$</th>
<th>$h^{cea}$</th>
<th>$h^{ff}$</th>
<th>$h^{ln-cut}$</th>
<th>$h^{MS}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sokoban:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P4</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0</td>
<td>0.02</td>
<td>0.06</td>
<td>0.06</td>
</tr>
<tr>
<td>Sokoban:P5</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0.06</td>
<td>0.06</td>
</tr>
<tr>
<td>Sokoban:P6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>Mystery:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.36</td>
<td>0.36</td>
</tr>
<tr>
<td>Mystery:P2</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>1.26</td>
<td>1.26</td>
</tr>
<tr>
<td>Mystery:P3</td>
<td>0.68</td>
<td>0.06</td>
<td>0.04</td>
<td>0.06</td>
<td>0.06</td>
<td>0.18</td>
<td>18.68</td>
</tr>
<tr>
<td>Parcprinter:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Parcprinter:P2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>Parcprinter:P3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pathways:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>Pathways:P2</td>
<td>0.06</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3.18</td>
<td>3.18</td>
</tr>
<tr>
<td>Pathways:P3</td>
<td>0.68</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>5.12</td>
<td>4.26</td>
<td>27.96</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1.44</td>
<td>0.08</td>
<td>0.06</td>
<td>0.1</td>
<td>0.08</td>
<td>5.32</td>
<td>27.96</td>
</tr>
</tbody>
</table>
### Table A.2: Times of Complete Search with Heuristic Tendency

<table>
<thead>
<tr>
<th>Domain</th>
<th>$h_{\text{max}}$</th>
<th>$h_{\text{add}}$</th>
<th>$h_{cg}$</th>
<th>$h_{cea}$</th>
<th>$h_{ff}$</th>
<th>$h_{lm-cut}$</th>
<th>$h_{MS}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sokoban:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P4</td>
<td>0.02</td>
<td>0</td>
<td>0.02</td>
<td>0</td>
<td>0.02</td>
<td>0.8</td>
<td>0.04</td>
</tr>
<tr>
<td>Sokoban:P5</td>
<td>0.02</td>
<td>0.02</td>
<td>0.04</td>
<td>0.04</td>
<td>0.02</td>
<td>0.04</td>
<td>0.06</td>
</tr>
<tr>
<td>Sokoban:P6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.04</td>
</tr>
<tr>
<td>Mystery:P1</td>
<td>0.72</td>
<td>0.06</td>
<td>0.04</td>
<td>0.06</td>
<td>0.06</td>
<td>0.18</td>
<td>0*</td>
</tr>
<tr>
<td>Mystery:P2</td>
<td>1.4</td>
<td>1.46</td>
<td>1.1</td>
<td>1.8</td>
<td>1.52</td>
<td>2.84</td>
<td>0.8</td>
</tr>
<tr>
<td>Mystery:P3</td>
<td>$\emptyset^*$</td>
<td>$\emptyset^*$</td>
<td>$\emptyset^*$</td>
<td>$\emptyset^*$</td>
<td>$\emptyset^*$</td>
<td>$\emptyset^*$</td>
<td>$\emptyset^*$</td>
</tr>
<tr>
<td>Parcprinter:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Parcprinter:P2</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>0.02</td>
<td>0</td>
</tr>
<tr>
<td>Parcprinter:P3</td>
<td>0.03</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pathways:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.04</td>
</tr>
<tr>
<td>Pathways:P2</td>
<td>0.06</td>
<td>0.08</td>
<td>0.06</td>
<td>0.06</td>
<td>0.08</td>
<td>1.1</td>
<td>3.5</td>
</tr>
<tr>
<td>Pathways:P3</td>
<td>3.64</td>
<td>6.54</td>
<td>6.26</td>
<td>6.34</td>
<td>6.58</td>
<td>7.14</td>
<td>9.14</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>5.91</strong></td>
<td><strong>8.16</strong></td>
<td><strong>7.52</strong></td>
<td><strong>8.3</strong></td>
<td><strong>8.28</strong></td>
<td><strong>11.36</strong></td>
<td><strong>14.4</strong></td>
</tr>
</tbody>
</table>

* Abort after a memory exception: not enough memory

### Table A.3: Times of Uniform Cost Search with Heuristic Tendency

<table>
<thead>
<tr>
<th>Domain</th>
<th>$h_{\text{max}}$</th>
<th>$h_{\text{add}}$</th>
<th>$h_{cg}$</th>
<th>$h_{cea}$</th>
<th>$h_{ff}$</th>
<th>$h_{lm-cut}$</th>
<th>$h_{MS}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sokoban:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.08</td>
</tr>
<tr>
<td>Sokoban:P5</td>
<td>0</td>
<td>0.02</td>
<td>0</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0.08</td>
</tr>
<tr>
<td>Sokoban:P6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
</tr>
<tr>
<td>Mystery:P1</td>
<td>0.04</td>
<td>0</td>
<td>0.02</td>
<td>0</td>
<td>0.02</td>
<td>0</td>
<td>1.3</td>
</tr>
<tr>
<td>Mystery:P3</td>
<td>1.46</td>
<td>0.06</td>
<td>0.04</td>
<td>0.06</td>
<td>0.18</td>
<td>$\emptyset^*$</td>
<td>$\emptyset^*$</td>
</tr>
<tr>
<td>Parcprinter:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Parcprinter:P2</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.04</td>
</tr>
<tr>
<td>Parcprinter:P3</td>
<td>0.08</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pathways:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pathways:P2</td>
<td>0.04</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3.26</td>
</tr>
<tr>
<td>Pathways:P3</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5.9</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1.94</strong></td>
<td><strong>0.08</strong></td>
<td><strong>0.06</strong></td>
<td><strong>0.06</strong></td>
<td><strong>0.18</strong></td>
<td><strong>11.02</strong></td>
<td></td>
</tr>
</tbody>
</table>
### Table A.4: Times of Results of Modified Heuristic Search

<table>
<thead>
<tr>
<th>Domain</th>
<th>$h_{\text{max}}$</th>
<th>$h_{\text{add}}$</th>
<th>$h_{\text{cg}}$</th>
<th>$h_{\text{cea}}$</th>
<th>$h_{\text{ff}}$</th>
<th>$h_{\text{lm-cut}}$</th>
<th>$h_{\text{MS}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sokoban:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>0.02</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P4</td>
<td>0.04</td>
<td>0</td>
<td>0.04</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>0.06</td>
</tr>
<tr>
<td>Sokoban:P5</td>
<td>0.02</td>
<td>0.06</td>
<td>0.04</td>
<td>0.02</td>
<td>0.02</td>
<td>0.04</td>
<td>0.08</td>
</tr>
<tr>
<td>Sokoban:P6</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>0</td>
<td>0.02</td>
<td>0</td>
</tr>
<tr>
<td>Mystery:P1</td>
<td>3</td>
<td>3.06</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.12</td>
<td>0.34</td>
</tr>
<tr>
<td>Mystery:P2</td>
<td>∅*</td>
<td>∅*</td>
<td>9</td>
<td>15.62</td>
<td>220.12</td>
<td>993.96</td>
<td>452.38</td>
</tr>
<tr>
<td>Mystery:P3</td>
<td>&gt;†</td>
<td>∅*</td>
<td>&gt;†</td>
<td>&gt;†</td>
<td>∅*</td>
<td>∅*</td>
<td></td>
</tr>
<tr>
<td>Parcprinter:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Parcprinter:P2</td>
<td>0.06</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Parcprinter:P3</td>
<td>0.08</td>
<td>0.08</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0</td>
</tr>
<tr>
<td>Pathways:P1</td>
<td>0.02</td>
<td>0.06</td>
<td>0.04</td>
<td>0.04</td>
<td>0.02</td>
<td>0.02</td>
<td>0.1</td>
</tr>
<tr>
<td>Pathways:P2</td>
<td>0.16</td>
<td>0.46</td>
<td>0.38</td>
<td>0.42</td>
<td>0.44</td>
<td>0.42</td>
<td>3.38</td>
</tr>
<tr>
<td>Pathways:P3</td>
<td>1.38</td>
<td>13.18</td>
<td>5.16</td>
<td>5.46</td>
<td>5.48</td>
<td>5.4</td>
<td>6.68</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>4.78</strong></td>
<td><strong>16.92</strong></td>
<td><strong>5.7</strong></td>
<td><strong>5.98</strong></td>
<td><strong>6.04</strong></td>
<td><strong>6.06</strong></td>
<td><strong>10.7</strong></td>
</tr>
</tbody>
</table>

### Table A.5: Times of Results Integrated Wisp=1

<table>
<thead>
<tr>
<th>Domain</th>
<th>$h_{\text{max}}$</th>
<th>$h_{\text{add}}$</th>
<th>$h_{\text{cg}}$</th>
<th>$h_{\text{cea}}$</th>
<th>$h_{\text{ff}}$</th>
<th>$h_{\text{lm-cut}}$</th>
<th>$h_{\text{MS}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sokoban:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
</tr>
<tr>
<td>Sokoban:P4</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.08</td>
</tr>
<tr>
<td>Sokoban:P5</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>0</td>
<td>0.02</td>
<td>0.1</td>
</tr>
<tr>
<td>Sokoban:P6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.04</td>
</tr>
<tr>
<td>Mystery:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.32</td>
</tr>
<tr>
<td>Mystery:P2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>0</td>
<td>0.02</td>
<td>1.88</td>
</tr>
<tr>
<td>Mystery:P3</td>
<td>3.12</td>
<td>0.08</td>
<td>0.06</td>
<td>0.1</td>
<td>0.08</td>
<td>1</td>
<td>32.32</td>
</tr>
<tr>
<td>Parcprinter:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Parcprinter:P2</td>
<td>0.02</td>
<td>0.02</td>
<td>0</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>Parcprinter:P3</td>
<td>0.02</td>
<td>0.02</td>
<td>0</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>Pathways:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pathways:P2</td>
<td>0.18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>0.02</td>
<td>3.5</td>
</tr>
<tr>
<td>Pathways:P3</td>
<td>2.5</td>
<td>0.1</td>
<td>0.14</td>
<td>0.1</td>
<td>0</td>
<td>0.34</td>
<td>6</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>5.82</strong></td>
<td><strong>0.18</strong></td>
<td><strong>0.22</strong></td>
<td><strong>0.24</strong></td>
<td><strong>0.12</strong></td>
<td><strong>1.42</strong></td>
<td><strong>44.34</strong></td>
</tr>
</tbody>
</table>

* Abort after a memory exception: not enough memory  
† Operator initiated termination after an run over 24h
## Appendix A. Times of Results

### Table A.6: Times of Results Integrated Wisp=10

<table>
<thead>
<tr>
<th>Domain</th>
<th>$h_{\text{max}}$</th>
<th>$h_{\text{add}}$</th>
<th>$h_{\text{cg}}$</th>
<th>$h_{\text{cea}}$</th>
<th>$h_{\text{ff}}$</th>
<th>$h_{\text{lm-cut}}$</th>
<th>$h_{\text{MS}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sokoban:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sokoban:P3</td>
<td>0</td>
<td>0.02</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>Sokoban:P4</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>Sokoban:P5</td>
<td><strong>0.02</strong></td>
<td><strong>0.02</strong></td>
<td>0.04</td>
<td><strong>0.02</strong></td>
<td>0.02</td>
<td>0.02</td>
<td>0.04</td>
</tr>
<tr>
<td>Sokoban:P6</td>
<td>0</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>0.02</td>
<td>0.04</td>
</tr>
<tr>
<td>Mystery:P1</td>
<td>0.22</td>
<td>0.14</td>
<td>0.08</td>
<td>0.2</td>
<td>0.16</td>
<td>0.36</td>
<td>0.34</td>
</tr>
<tr>
<td>Mystery:P2</td>
<td>47.78</td>
<td>3.48</td>
<td><strong>2.62</strong></td>
<td>4.62</td>
<td>7.24</td>
<td>24.12</td>
<td>4.98</td>
</tr>
<tr>
<td>Mystery:P3</td>
<td>Ø*</td>
<td>1191.12</td>
<td>351.78</td>
<td>1868.84</td>
<td>1706.76</td>
<td>Ø*</td>
<td>Ø*</td>
</tr>
<tr>
<td>Parcprinter:P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Parcprinter:P2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Parcprinter:P3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pathways:P1</td>
<td>0.02</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.06</td>
<td>0.06</td>
</tr>
<tr>
<td>Pathways:P2</td>
<td><strong>0.1</strong></td>
<td>0.12</td>
<td>0.12</td>
<td><strong>0.1</strong></td>
<td><strong>0.1</strong></td>
<td>0.12</td>
<td>3.48</td>
</tr>
<tr>
<td>Pathways:P3</td>
<td>0.04</td>
<td>5.1</td>
<td>5.1</td>
<td>5.1</td>
<td>5.18</td>
<td>45.4</td>
<td>5.7</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>54.2</strong></td>
<td><strong>9.14</strong></td>
<td><strong>8</strong></td>
<td><strong>10.14</strong></td>
<td><strong>52.96</strong></td>
<td><strong>30.4</strong></td>
<td><strong>18.36</strong></td>
</tr>
</tbody>
</table>

* Abort after a memory exception: not enough memory
Appendix B

Bar Charts of Methods

The following charts describe the count of found deadlines by each domain problem per applied heuristic.

Figure B.1: Deadlines by domains for a normal search with Fast Downward
Appendix B. Balance Sheet

Figure B.2: Deadlines by domains for a Breadth First search

Figure B.3: Deadlines by domains for a heuristic complete search
Figure B.4: Deadlines by domains for a uniform cost search with heuristic tendency
Figure B.5: Deadlines by domains for a modified heuristic search
Figure B.6: Deadlines by domains for wisp = 1
Figure B.7: Deadlines by domains for wisp = 10
Appendix C

Charts of Direct Comparison

The following charts describe a direct comparison of a choose from the methods. They show the found deadlines of each problem by each heuristic. The number of deadlines are presented as a static sequence. The axes are partial logarithmic plotted for a better view.
Figure C.1: Comparison of normal run and modified heuristics method 1
[Domain problems Sokoban 1 - 6 and Mystery 1 - 2]
Figure C.2: Comparison of normal run and modified heuristics method 2
[Domain problems Mystery 3, Parcprinter 1 - 3 and Pathways 1 - 3]
Figure C.3: Comparison of normal run and wisp=10 integrated method 1
[Domain problems Sokoban 1 - 6 and Mystery 1 - 2]
Figure C.4: Comparison of normal run and wisp=10 integrated method 2
[Domain problems Mystery 3, Parcprinter 1 - 3 and Pathways 1 - 3]
Erklärung

Ich erkläre hiermit gemäß §17. Abs 2 APO, dass ich die vorstehende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Datum: 

Unterschrift: 

79