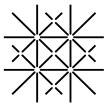

University of Basel

Mutex Based Potential Heuristics

Bachelor thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Artificial Intelligence
https://ai.dmi.unibas.ch

Examiner: Prof. Dr. Malte Helmert
Supervisor: Dr. Salomé Eriksson

Salome Müller
salo.mueller@unibas.ch
2017-063-058

06. 11. 2020

Acknowledgments

First, I want to thank Prof. Dr. Malte Helmert for the opportunity of writing my bachelor thesis in the AI group at the University of Basel. I am very grateful for the support of Dr. Salomé Eriksson, who patiently answered all my questions and guided me through this work. I would like to acknowledge Dr. Florian Pommerening for helping me understand the LP constructor of Fast Downward.
Further, I recieved a great deal of support and assistance from Lucas Galery Käser, as well as from my flatmates. Thank you all for keeping me nourished.
Last, I want to thank Ada Lovelace, the first programmer there was, for being an inspiration to women in computer science.
Calculations were performed at sciCORE (http://scicore.unibas.ch/) scientific computing center at University of Basel.

Abstract

This thesis discusses the thesis template using some examples of the Turing Machine.

Table of Contents

1
Introduction

My work is brilliant, as can be shown with this very nice example.

2
Background

The goal of this chapter is to define and explain the terminology used in this thesis. For visualization, we use the 8-Tiles problem as an example. This is a classical planning problem, in which 8 tiles are arranged in a 3x3-Grid. One spot remains empty, the goal is to bring the tiles in a specific order by sliding them around.

2.1 Planning Tasks

We define $\mathcal{V}$ as the finite set of **variables**, each of the variables $V \in \mathcal{V}$ has a finite set of **domains** $\mathrm{dom}(V)$. For 8-Tiles, the variables could be defined as the 9 fields in the grid $(v_1, \ldots, v_9)$, and their domains hold the values of all tiles and the blank space (1 to 8 and 0 for the blank tile). A **fact** $f = \langle V, v \rangle$ consists of a variable $V \in \mathcal{V}$ and one of its values $v \in \mathrm{dom}(V)$. The fact for tile number 5 being in the first position would be $\langle v_1, 5 \rangle$. $\mathcal{F}_V$ is the set of all possible facts of variable $V \in \mathcal{V}$ while $\mathcal{F}$ is the set of all facts of this problem. A **partial state** p of size t contains t facts of t different variables, i.e., it is the variable assignment over the variables $\mathrm{vars}(p) \subseteq \mathcal{V}$ with $|\mathrm{vars}(p)| = t$. $p[V]$ is the value assigned to V in p. In other words, $p = \{\langle V, p[v] \rangle | V \in \mathrm{vars}(p)\}$. A **state** s is not partial, if all variables are assigned, i.e., $\mathrm{vars}(s) = \mathcal{V}$. It **extends** the partial state $p \subseteq s$, if $s[v] = p[v]$ for all $v \in \mathrm{vars}(p)$. The partial state $p = \{v_1 \mapsto 0, v_2 \mapsto 1\}$ represents all states where the first grid in the field of the 8-Tiles puzzle is the blank space while tile number one lies in the second field.

I is the **initial state**, in 8-Tiles this is some specific random order of the tiles. G is a partial state representing the **goal**. s is a **goal state**, if it is an extension of G. In 8-Tiles it is one specific order of the tiles e.g. sorted by number: $s = \{v_1 \mapsto 1, v_2 \mapsto 2, v_3 \mapsto 3, v_4 \mapsto 4, v_5 \mapsto 5, v_6 \mapsto 6, v_7 \mapsto 7, v_8 \mapsto 8, v_9 \mapsto 0\}$.

$\mathcal{O}$ is a finite set of **operators**. Each $o \in \mathcal{O}$ has a precondition $\mathrm{pre}(o)$ and an effect $\mathrm{eff}(o)$ which are both partial states over $\mathcal{V}$, and a cost $\mathrm{c}(o) \in \mathbb{R}_0^+$.

The operator o is **applicable** in state s iff $\mathrm{pre}(o) \subseteq s$. We call the **resulting state** $o[\![s]\!]$. $o[\![s]\!][v] = \mathrm{eff}(o)[v]$ holds for all $v \in \mathrm{eff}(o)$ in resulting state $o[\![s]\!]$ while the other variables do not change, i.e., $o[\![s]\!][v] = s[v]$ for all $v \notin \mathrm{eff}(o)$. In 8-Tiles, the operators encode the movement of one tile to the blank space. The precondition assures that the tile is next to

the blank space, the effect swaps the values of the corresponding two variables, while all other tiles remain at the same position.
In order to reach the goal multiple operators need to be applied in a specific order. A sequence of operators $\pi = \langle o_1, \ldots, o_n \rangle$ is called a path, $\pi[\![s]\!] = s_n$. π is a **s-plan**, if π is applicable in s and $\pi[\![s]\!]$ is a an extension of G and therefore a goal state. If it has minimal cost among all s-plans it is called **optimal**.
The set $\mathcal{R}$ is defined as the set of all **reachable** states. A state s is reachable, if a plan π is applicable in I such that $\pi[\![I]\!] = s$. An operator o is reachable, if it is applicable in a reachable state. A state s is a **dead-end state** if it does not extend the goal state, and no s-plan exists.
Fišer et al. use the finite domain representation, where Π is specified by a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, I, G \rangle$ to represent problems as a **planning task** [4]. They can then be solved with heuristic search.

2.2 Heuristics

A **heuristic** $h : \mathcal{R} \to \mathbb{R} \cup \{\infty\}$ estimates the cost of the optimal plan for a state s. The problem of 8-Tiles has uinform cost, as sliding a tile always costs the same, i.e. 1, and there are no other operators. Therefore, the cost of a s-plan of any state which is not a dead-end equals the amount tiles, which need to be slid in the plan. The **optimal heuristic** $h^*(s)$ maps each state s to its actual optimal cost, or to ∞ if it is a dead-end state. We aim to approach this heuristic.
This thesis uses heuristics in the forward heuristic search where unreachable states are never expanded. Therefore they are defined over $\mathcal{R}$ instead of over all states and the above defined rules hold for reachable states only.
A heuristic is **admissible**, if it never overestimates the optimal heuristic, i.e., $h(s) \leq h^*(s)$. It is **goal-aware** iff $h(s) \leq 0$ for all reachable goal states, i.e., it recognizes a goal sate as such. Further, it is **consistent** iff $h(s) \leq h(o[\![s]\!]) + c(o)$. This rule assures, that the heuristic of the successor of a state is not a a lot lower than the heuristic of the state itself.
A heuristic which is goal aware and consistent is also admissible.
One class of heuristics are potential heuristics which assign a potential to each possible fact of the planning task.

Definition 1. *Let* Π *denote a planning task with facts* $\mathcal{F}$*. A* ***potential function*** *is a function* $\mathtt{P} : \mathcal{F} \mapsto \mathbb{R}$*. A* ***potential heuristic*** *for* $\mathtt{P}$ *maps each state* $s \in \mathcal{R}$ *to the sum of potentials of facts in* s*, i.e.,* $h^{\mathtt{P}}(s) = \sum_{f \in s} P(f)$*.*

Potential heuristics are goal-aware, consistent and admissible [1]. The potentials themselves are obtained through optimization which will be further analyzed in Chapter 3.
One further approach is **ensemble heuristics**. Instead of only one heuristic, this approach uses multiple heuristics and chooses the highest value as heuristic value for each state.

2.3 Mutexes and Disambiguations

Mutex means, that two or more things mutually exclude each other.

Definition 2. *Let* Π *denote a planning task with facts* $\mathcal{F}$*. A set of facts* $\mathcal{M} \subseteq \mathcal{F}$ *is a* ***mutex*** *if* $\mathcal{M} \not\subseteq s$ *for every reachable state* $s \in \mathcal{R}$

Facts are a mutex if they never appear together in any reachable state. If a partial state p in 8-Tiles holds $p[v_3] = 1$, then tile one may not be in any other spot of the grid, i.e., the fact $\langle v_3, 1\rangle$ is mutex with all other facts $\langle v, 1\rangle$ with $v \in \mathcal{V} \setminus \{v_3\}$.

Definition 3. *Let* Π *denote a planning task with variables* $\mathcal{V}$ *and facts* $\mathcal{F}$*. A set of sets of facts* $\mathcal{M} \subseteq 2^{\mathcal{F}}$ *is called a* ***mutex-set*** *if the following hold: (a) every* $M \in \mathcal{M}$ *is a mutex; and (b) for every* $M \in \mathcal{M}$ *and every* $f \in \mathcal{F}$ *it holds that* $M \cup \{f\} \in \mathcal{M}$*; and (c) for every variable* $V \in \mathcal{V}$ *and every pair of facts* $f, f' \in \mathcal{F}_V$*,* $f \neq f'$*, it holds that* $\{f, f'\} \in \mathcal{M}$*.*

We can say that $s \in \mathcal{M}$ if s contains a subset of facts which are a mutex.
Mutexes can be used to derive disambiguations.

Definition 4. *Let* Π *denote a planning task with facts* $\mathcal{F}$ *and variables* $\mathcal{V}$*, let* $V \in \mathcal{V}$ *denote a variable, and let* p *denote a partial state. A set of facts* $F \subseteq \mathcal{F}_V$ *is called a* ***disambiguation*** *of* V *for* p *if for every reachable state* $s \in \mathcal{R}$ *such that* $p \subseteq s$ *it holds that* $F \cap s \neq \emptyset$ *(i.e.,* $\langle V, s[V]\rangle \in F$*).*

The disambiguation of a variable V for a partial state p is the set of facts $F \in \mathcal{F}_V$ which occur in all reachable extended states of p. This means, that each fact of V which is not in F is a mutex with p. If F contains exactly one fact then p can be safely extended with that fact, as it is the only non-dead-end extension of the state. If F is the empty set every extended state of p is a dead-end. This knowledge can be used to prune operators o for which $p \subseteq \text{pre}(o)$ and unreachable states $s \subseteq p$. If the goal state G is one of this states, the problem is unsolvable.
If a partial state s of the 8-Tiles problem holds $p[v_3] = 1$ and $p[v_2] = 1$, then it is a dead-end, as these facts are a mutex. If $p = \{v_1 \mapsto 1, v_2 \mapsto 2, v_3 \mapsto 3, v_4 \mapsto 4, v_5 \mapsto 5, v_6 \mapsto 6, v_7 \mapsto 7, v_8 \mapsto 8\}$ then p is not a dead-end and v_9 can safely be assigned with 0, as it is the only fact in $\mathcal{F}_{v_9}$ which does not form a mutex with any of the already assigned facts.
The set $\mathcal{M}_p = \{f | f \in \mathcal{F}, p \cup \{f\} \in \mathcal{M}\}$ is the set of facts which are mutex with p. All facts of a variable $f \in \mathcal{F}_V$ not contained in $\mathcal{M}_p$ build the disambiguation F of V for p. In 3 we will use this to improve potential heuristics by narrowing down possible extensions of partial states.

3

Strengthening Potential Heuristics

Fišer et al. propose a method to improve potential heuristics with mutexes and disambiguations. This chapter contains the changes which are required to do so, regarding the transformation of a planning task into TNF and the adaption of the optimization functions. It shows how the equations which were later implemented (Section Implementation) are derived.

3.1 Potential Heuristics

When Pommerening et al. first introduced potential heuristics, they showed that two inequalities are sufficient to proof admissibility.

Theorem 5. *Let* $\Pi = \langle \mathcal{V}, \mathcal{O}, I, G \rangle$ *denote a planning task,* $\mathtt{P}$ *a potential function, and for every operator* $o \in \mathcal{O}$*, let* $\text{pre}^*(o) = \{\langle V, \text{pre}(o)[V]\rangle | V \in \text{vars}(\text{pre}(o)) \cap \text{vars}(\text{eff}(o))\}$ *and* $\text{vars}^*(o) = \text{vars}(\text{eff}(o)) \setminus \text{vars}(\text{pre}(o))$*. If*

$$\sum_{f \in G} \mathtt{P}(f) + \sum_{V \in \mathcal{V} \setminus \text{vars}(G)} \max_{f \in \mathcal{F}_V} \mathtt{P}(f) \leq 0 \tag{3.1}$$

and for every operator $o \in \mathcal{O}$ *it holds that*

$$sum_{f \in \text{pre}^*(o)} \mathtt{P}(f) + \sum_{V \in \text{vars}^*(o)} \max_{f \in \mathcal{F}_V} \mathtt{P}(f) - \sum_{f \in \text{eff}(o)} \mathtt{P}(f) \leq c(o) \tag{3.2}$$

then the potential heuristic for $\mathtt{P}$ *is admissible.*

Eq. (3.1) of the Theorem 5 assures goal-awareness of the potential heuristic. As all variables are assigned in the goal state, the potential of one fact per variable has to be summed up. For the variables $v \in \text{vars}(G)$ we can simply use the potentials of their respective facts. Meanwhile we assume the worst case for the other variables, by using the maximal potential over their facts, as we do not know what fact they are assigned.
Eq. (3.2) assures consistency. Recall the general heuristics consistency equation $h(s) \leq h(o[\![s]\!]) + \text{c}(o)$. It can be rewritten as $h(s) - h(o[\![s]\!]) \leq \text{c}(o)$. As the facts which do not occur in the effect are the same in both s and $o[\![s]\!]$ we can leave them aside. For s we know what facts of the variables of the preconditions are assigned and sum the potentials of the

facts which are in the effect as well. For the variables which are in the effect but not in the precondition we proceed similarly to (3.1), as we do not know their values. The potentials of the facts in the effect can be used without modification for $o[\![s]\!]$.
The advantage of these equations is that they are not state-dependent, even though they do not tell us explicitly what the potentials should be. However, they can be used as the constraints for a linear program, the solution of which is a potential function that forms an admissible potential heuristic. More about this in Section 3.2.

3.1.1 Generalization with Mutexes

Mutexes can be used to reduce the domain of variables, which are not yet assigned in a partial state p. This property is very helpful, as it minimizes the amount of facts which are candidates for the not assigned variables in Equations (3.1) and (3.2) of Theorem 5.
make this algorithm look a little nicer...

Algorithm 1 Multi-fact fixpoint disambiguation.

Input: A planning task Π with variables $\mathcal{V}$ and facts F, a partial state p, and a mutex-set $\mathcal{M}$.
Output: A set of disambiguations $\mathcal{D}_p$ of all variables $\mathcal{V}$ for p.

```
1:  D_v ← F_V for every V ∈ V
2:  A ← M_p
3:  change ← True
4:  while change do
5:      change ← False
6:      for all V ∈ V do
7:          if D_V \ A ≠ D_V then
8:              D_V ← D_V \ A
9:              A ← A ∪ ⋂_{f∈D_V} M_{p∪{f}}
10:             change ← True
11:         end if
12:     end for
13: end while
14: D_p ← {D_V | V ∈ V}
```

At the beginning, the set D_V contains all possible values for the variable $V \in \mathcal{V}$, while A contains all facts which are a mutex with any fact in p. In each iteration of the while-loop, all $f = \langle v, V \rangle$ which are in A and in D_V are removed from the corresponding D_V. On line 9, A is extended with all facts that form a mutex with all facts remaining in D_V, i.e., which are a mutex with $p \cup \{f\}$ for all $f \in D_V$.
In conclusion, after applying mutli-fact fixpoint disambiguation p can be extended with any fact in $\mathcal{D}_p$ without reaching a dead-end state. If any $D_V \in \mathcal{D}_p$ is empty, then p is already a dead-end itself.
This algorithm is used for several applications during the remainder of this chapter. In this section, it can be used to generalize Theorem 5 by the following theorem.

Theorem 6. *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, I, G \rangle$ denote a planning task with facts $\mathcal{F}$, and let $\mathtt{P}$ denote a potential function, and*

(i) for every variable $V \in \mathcal{V}$, let $G_V \subseteq \mathcal{F}_V$ denote a disambiguation of V for G s.t. $|G_V| \geq 1$, and

(ii) for every operator $o \in \mathcal{O}$ and every variable $V \in \text{vars}(\text{eff}(o))$, let $E^o_V \subseteq \mathcal{F}_V$ denote a disambiguation of V for $\text{pre}(o)$ s.t. $|E^o_V| \geq 1$.

If

$$\sum_{V \in \mathcal{V}} \max_{f \in G_V} \mathtt{P}(f) \leq 0 \tag{3.3}$$

and for every operator $o \in \mathcal{O}$ it holds that

$$\sum_{V \in \text{vars}(\text{eff}(o))} \max_{f \in E^o_V} \mathtt{P}(f) - \sum_{f \in \text{eff}(o)} \mathtt{P}(f) \leq \text{c}(o) \tag{3.4}$$

then the potential heuristic P *is admissible.*

Fišer et al. prove the theorem by showing that Equations (3.3) and (3.4) are generalizations of Equations (3.1) and (3.2), respectively.
The disambiguation G_V equals $D_V \in \mathcal{D}_G$ with $V \in \mathcal{V}$, which is generated by applying Algorithm 1 on the goal state. If it is empty for any of the variables, then the problem is unsolvable, as the goal contains a mutex and is therefore not a reachable state. E^o_V is equal to $D_V \in \mathcal{D}_{\text{pre}(o)}$. o is not applicable in any (partial) state if E^o_V is empty for any $V \in \text{vars}(\text{eff}(o))$.
To show the reader why this property is useful in practice, we first introduce the Transition Normal Form.

3.2 Transition Normal Form

Planning tasks can be in Transition Normal Form (TNF, Pommerening and Helmert). A planning task in TNF has a fully defined goal ($\text{vars}(G) = \mathcal{V}$) and all variables of the effect are also in the precondition for each operator $o \in \mathcal{O}$ ($\text{vars}(\text{pre}(o)) = \text{vars}(\text{eff}(o))$). These properties are essential to form the Linear Program, which we will do in the next section. Any planing task $\Pi = \langle \mathcal{V}, \mathcal{O}, I, G \rangle$ can be transformed into TNF with the following rules cited from Fišer et al.:

- Add a fresh value U to the domain of every variable.
- For every variable $V \in \mathcal{V}$ and every fact $f \in \mathcal{F}_V$, $f \neq \langle V, U \rangle$, add a new *forgetting* operator o_f with $\text{pre}(o_f) = \{f\}$ and $\text{eff}(o_f) = \{\langle V, U \rangle\}$ and the cost $\text{c}(o_f) = 0$.
- For every operator $o \in \mathcal{O}$ and every variable $V \in \mathcal{V}$:
  - If $V \in \text{vars}(\text{pre}(o))$ and $V \notin \text{vars}(\text{eff}(o))$, then add $\langle V, \text{pre}(o)[V] \rangle$ to $\text{eff}(o)$.
  - If $V \in \text{vars}(\text{eff}(o))$ and $V \notin \text{vars}(\text{pre}(o))$, then add $\langle V, U \rangle$ to $\text{pre}(o)$.
- For every $V \in \mathcal{V} \setminus \text{vars}(G)$ add $\langle V, U \rangle$ to G.

Each Variable $V \in \mathcal{V}$ gets a new value U in its domain, which can be seen as a sort of placeholder. The fact $\langle V, U\rangle$ can be assigned with cost 0, as the forgetting operator o_f which assigns it has no cost, regardless of the current state and especially the current assignment of V. The next point is to assure that for each operator the variables which are in the precondition are also in the effect.
If V is present in the preconditions of an operator $o \in \mathcal{O}$ but not in the effect, then we can simply add the variable and the value it is already assigned to the effect. This is a formal change, but does not change the effect of the operator at all, as it would not have changed this fact anyway.
The case of an operator $o \in \mathcal{O}$, where V is in the effect but not in the precondition, is a little more complicated. Here, the precondition is changed such that it contains also the fact $\langle V, U\rangle$. If o was applicable in s before, then, after transforming the plan into TNF, the corresponding o_f needs to be applied beforehand in order to forget the value of V. This change of the variable is insignificant, as the value then gets changed by applying the operator anyways.
Last, all variables which were not included in the partial state G need to be added into it. If they are assigned the fresh value U, then the goal state can be reached from every state which expanded it before. Without creating more cost, the values of all unimportant variables are changed to the fresh value. The compilation into TNF can produce a plan twice the size of the original task in worst case [8].

3.2.1 Generalization with Mutexes

Similar to Section 3.1.1, these rules can be generalized with disambiguations. Therefore, to replace U we introduce the fresh values U_{G_V} and $U_{E^o_V}$. Instead of adding forgetting operators from every fact in every $V \in \mathcal{V}$, we use the disambiguation sets G_V and E^o_V.

- Add fresh value U_{G_V} to the domain of every $V \in \mathcal{V}$.
- For every variable $V \in \mathcal{V}$ and every fact $f \in G_V$, $f \neq \langle V, U_{G_V}\rangle$, add new *forgetting* operators o_{f_G} with $\text{pre}(o_{f_G}) = \{f\}$ and $\text{eff}(o_{f_G}) = \{\langle V, U_{G_V}\rangle\}$ and the cost $\text{c}(o_{f_G}) = 0$.
- For every $V \in \mathcal{V} \setminus \text{vars}(G)$ add $\langle V, U_{G_V}\rangle$ to G.
- For every operator $o \in \mathcal{O}$ add fresh value $U_{E^o_V}$ to the domain of every $V \in \mathcal{V}$:
  - If $V \in \text{vars}(\text{pre}(o))$ and $V \notin \text{vars}(\text{eff}(o))$, then add $\langle V, \text{pre}(o)[V]\rangle$ to $\text{eff}(o)$.
  - If $V \in \text{vars}(\text{eff}(o))$ and $V \notin \text{vars}(\text{pre}(o))$, then add $\langle V, U_{E^o_V}\rangle$ to $\text{pre}(o)$.
- For every variable $V \in \mathcal{V}$, every operator$o \in \mathcal{O}$ and every fact $f \in E^o_V$, $f \neq \langle V, U_{E^o_V}\rangle$, add new forgetting operators $o_{f,o}$ with $\text{pre}(o_{f,o}) = \{f\}$ and $\text{eff}(o_{f,o}) = \{\langle V, U_{E^o_V}\rangle\}$ and the cost $\text{c}(o_{f,o}) = 0$.

For the goal state, forgetting operators are only created for the facts in F_V which are not a mutex with any $f \in G$ for every $V \notin \text{vars}(G)$. Similarly, facts in F_V which are a mutex with any $f \in \text{pre}(o)$ are not taken into account for all $o \in \mathcal{O}$ and every $V \in \text{vars}(\text{eff}(o))$.

This creates at most $|\mathcal{O}| * |\mathcal{V}|$ $U_{E_V^o}$ and $|\mathcal{V}|$ U_{G_V} and the amount of forgetting operators is in the worst case the same, multiplied with the sum of the cardinalities of the domains of all $V \in \mathcal{V}$. In practice, Fišer et al. show that the transformation with disambiguations is never bigger than without disambiguations.

3.3 Linear Program

The formulas and rules which were defined in the previous two sections can now be used to form a Linear Program (LP).
An LP consists of LP-variables which are constrained by multiple inequalities (constraints) and which are part of an optimization function. An LP-solver then assigns each LP-variable a value, such that all constraints are satisfied and the optimization function is optimized. We will look at different optimization functions in the next section (3.4).

Definition 7. *Let f be a solution to the following LP:*
Maximize opt *subject to* $\sum_{V \in \mathcal{V}} \mathtt{P}_{\langle V, s[V] \rangle} \leq 0$ *and* $\sum_{V \in \text{vars}(\text{eff}(o))} (\mathtt{P}_{\langle V, \text{pre}(o)[V] \rangle} - \mathtt{P}_{\langle V, \text{eff}(o)[V] \rangle}) \leq \text{c}(o)$ *for all $o \in \mathcal{O}$, where the objective function* opt *can be chosen arbitrarily.*
Then the function $\text{pot}_{\text{opt}}(\langle V, v \rangle) = f(\mathtt{P}_{\langle V,v \rangle})$ *is the* potential function optimized for opt *and* $h^{\mathtt{P}}$ *is the* potential heuristic optimized for opt*.*

In order to find a potential heuristic, the LP-variables are the potentials of the facts, $P(f)$. Further we define the constraints as Equation (3.3) for the goal state and Equations (3.4) for every operator. Since these two formulas assure admissibility, any solution of the LP builds an admissible $h^{\mathtt{P}}$. We introduce the the LP-variables $X_f = \mathtt{P}(f)$ for every $f \in \mathcal{F}$ and M_{G_V} and $M_{E_V^o}$ corresponding to U_{G_V} and $U_{E_V^o}$, respectively, with the constraint that $X_f \leq M_{G_V}$ for every $f \in G_V$ and $X_f \leq M_{E_V^o}$ for every $f \in E_V^o$. This gives the constraints

$$\sum_{f \in G} X_f + \sum_{V \in \mathcal{V} \setminus \text{vars}(G)} M_{G_V} \leq 0 \tag{3.5}$$

and

$$\sum_{f \in \text{pre}^*(o)} X_f + \sum_{V \in \text{vars}^*(o)} M_{E_V^o} - \sum_{f \in \text{eff}(o)} X_f \leq c(o) \tag{3.6}$$

which are coherent to the formulas in Definition 7 (Pommerening et al.). M_{G_V} and $M_{E_V^o}$ correspond to U_{G_V} and $U_{E_V^o}$, respectively, since these are the facts which are assigned if a variable is not defined in the goal state or in a precondition of an operator.
The solution of the LP might differ vastly depending on the used optimization function. We will look at multiple different possibilities for optimization functions.

3.4 Optimization Functions

An optimization function opt is a linear combination of the LP-variables. In our case, the goal is to have best possible heuristic value for as many states as possible. Using different optimization functions optimizes different aspects of a heuristic. The perfect heuristic would be achieved, if we optimized the potentials for each single state, but this is computationally too expensive.

In the first proposal for potential heuristics from Pommerening et al., the optimization for the initial state was used,

$$\text{opt}_I = \sum_{f \in I} \mathtt{P}(f). \tag{3.7}$$

It optimizes the heuristic value for the initial state. The drawback of is that facts which do not appear in the initial state are not taken into account.
Alternatively, we could optimize the potentials for all reachable states, with the all-states-potentials optimization function:

$$\text{opt}_{\mathcal{R}} = \frac{1}{|\mathcal{R}|} \sum_{s \in \mathcal{R}} \sum_{f \in s} \mathtt{P}(f). \tag{3.8}$$

It calculates the weighted sum of all facts, i.e., it multiplies the potential of a fact with the amount of reachable states containing this fact and normalizes it with the total amount of reachable states. The potentials generated with this optimization function would result in the heuristic with the maximal average heuristic value over all reachable states. Unfortunately, calculating this is, again, computationally expensive or even infeasible, if the planning task and therefore the size of $\mathcal{R}$ are big, as the set of reachable states is not known. To avoid this, we could sample some states $S \subseteq \mathcal{R}$, and calculate Equation (3.8) over these states, instead of $\mathcal{R}$:

$$\text{opt}_{\hat{\mathcal{S}}} = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \sum_{f \in s} \mathrm{P}(f). \tag{3.9}$$

The optimization function could also assume uniform distribution and give all facts the same weight. Instead of going over all facts of all states, we would sum over the domains of all variables:

$$\text{opt}_{\mathcal{S}} = \sum_{\langle V, v \rangle \in \mathcal{F}} \frac{1}{|\text{dom}(V)|} \mathtt{P}(\langle V, v \rangle). \tag{3.10}$$

These two approaches for the all-states-potential optimization function can be strengthened with mutexes and disambiguations, which we will describe in the next sections.

3.4.1 Strengthening All State Potentials

To estimate the amount of reachable states containing $f = \langle V, v \rangle$, we will calculate the upper bound of these states, and try to lower it for each $f \in \mathcal{F}$. The product of the domains of all variables except V, since V is already assigned, is the total amount of states, reachable and non-reachable, which contain f. With Algorithm 1 from Section 3.1.1 we can remove all facts from all domains which are mutex with f. These facts could never be assigned in any reachable state containing f. Therefore, the product over all domains in the resulting disambiguation set is again an upper bound of appearances of f.
This holds not only for a single fact, but can be applied to estimate the appearances of any partial state p. As the product of all domains in the disambiguation set is taken, the value is zero, if p is a mutex itself.
If we define $\mathcal{M}$ as the superset of all mutexes and $\mathcal{M}_p$ as the set of all facts which form a mutex with patial state p, then an upper bound for the amount of reachable states of size k containing f is

$$\mathcal{C}_f^k(\mathcal{M}) = \sum_{p \in \mathcal{P}_k^{\{f\}}} \prod_{V \in \mathcal{V}} |F_V \setminus \mathcal{M}_p|. \tag{3.11}$$

The Equation has the constraint to only extend states to size k, as it would get computationally too expensive to compute all reachable states for which f holds.
We will now use Equation (3.11) to calculate the weights for the potential of each $f \in \mathcal{F}$ and form the following optimization function:

$$\text{opt}_{\mathcal{M}}^k = \sum_{f=\langle V,v \rangle \in \mathcal{F}} \frac{\mathcal{C}_f^k(\mathcal{M})}{\sum_{f' \in \mathcal{F}_V} \mathcal{C}_{f'}^k(\mathcal{M})} \mathtt{P}(f). \tag{3.12}$$

The result is a modification of $\text{opt}_{\hat{S}}$ (Eq. (3.9)) that des not assume uniformly distributed facts. Each fact is weighted according to its estimated appearance in all reachable states of size k in relation to the other facts of the corresponding variable. The sum of the weights for all facts of one variable is 1. We can use this not only as a single heuristic, but also in ensemble heuristics as we will show in the next section.

3.4.2 Strengthening Conditioned Ensemble Potentials

If we divide $\mathcal{R}$ into multiple subsets $S_i \subset \mathcal{R}$, with $i = 1, \ldots, n$ and $S_1 \cup \cdots \cup S_n = \mathcal{R}$, the solution of the LP with optimization function $\text{opt}_{\hat{S}_i}$ gives better heuristic values for the states in S_i than $\text{opt}_{\mathcal{R}}$. If we calculate the potentials for the optimization of all S_i, the resulting heuristics can be used as ensemble heuristics. This gives a result which lies somewhere between the all-states-potential heuristic and calculating the potentials for each individual state.
We will choose S_i as the states extending one particular partial state t, and use the potentials generated with an adjusted $\text{opt}_{\mathcal{M}}^k$ as one of multiple ensemble heuristics. For this we adapt Equation (3.11), such that it counts how many reachable states of size $|t| + k$ extend t:

$$\mathcal{K}_f^k(\mathcal{M}, t) = \sum_{p \in \mathcal{P}_{|t|+k}^{t \cup \{f\}}} \prod_{V \in \mathcal{V}} |\mathcal{F} \setminus \mathcal{M}_p|. \tag{3.13}$$

The corresponding optimization function uses the weights calculated with $\mathcal{K}_f^k(\mathcal{M}, t)$,

$$\text{opt}_{\mathcal{M}}^{t,k} = \sum_{f=\langle V,v \rangle \in \mathcal{F}} \frac{\mathcal{K}_f^k(\mathcal{M}, t)}{\sum_{f' \in \mathcal{F}_V} \mathcal{K}_f^k(\mathcal{M}, t)} \mathtt{P}(f). \tag{3.14}$$

For this thesis, the partial states t were sampled uniformly at random, as did Fišer et al.. Future research could be to investigate other ways to choose them.

3.4.3 Adding Constraint on Initial State

Both $\text{opt}_{\mathcal{M}}^k$ and $\text{opt}_{\mathcal{M}}^{t,k}$ optimize the potentials in regard of the entire reachable state space. However, the importance of states may vary strongly, depending on where we start and what path is taken from there towards the goal. For example, a planning task with multiple goal states may have smaller heuristic values, even though the constraints enforce goal-awareness.

This is why we now look at a third approach which gives the initial state more weight and adds the constraint

$$\sum_{f \in I} \mathtt{P}(f) = h^{\mathtt{P}}_I(I) \tag{3.15}$$

to the Linear Program, where $h^{\mathtt{P}}_I$ denotes the potential heuristic optimized for the initial state (Eq. (3.7)). By calculating the potentials for this heuristic first and then taking it into account for the actual potentials, we force the solver to find potentials which guarantee high heuristic values for the initial state. This approach can be combined with all previously discussed optimization functions.

3.4.4 Adding Constraints on Random States

Taking the idea of the additional constraint a step further, we decided to add constraints on random states. Similar to the constraint on the initial state, the constraints on random states gives more importance to these states.
The states are generated with random walks of the size [half of the heuristic value for the initial state], the states in the middle of the way (heuristically speaking) are given more importance.

$$\sum_{f \in s} \mathtt{P}(f) = h^{\mathtt{P}}(s) \tag{3.16}$$

proof-read

3.5 Implementation

Our implementation of mutex based potential heuristics is embedded in the planning system Fast Downward by Helmert and written in C++.
We first implemented Algorithm 1, which uses the Fast Downward hm-heuristic with $m = 2$ to build a mutex table. The computation of the hm-heuristic in Fast Downward is very slow. Therefore, we implemented a new generator for the mutex table, which is very similar to the hm-heuristic, but optimized for finding mutexes and therefore a lot faster.
Fast Downward has a potential optimizer, which initializes and constructs an LP-solver. We implemented a new LP-constructor which builds the constraints according to Equations (3.3) and (3.4). For both the non-mutex and the mutex constructor, we added the option to use the additional constraints on the initial state and random states.
Next, we implemented the optimization function $\text{opt}^k_{\mathcal{M}}$. All partial states of cardinality one ($p_f = \{f\}$ for all $f \in \mathcal{F}$) are generated, and then recursively all partial states of size k extending p_f are created, using the disambiguation set $\mathcal{D}_{p_f}$. Even for small tasks the amount of extended states can grow very fast. For memory efficiency partial states are implemented as maps, containing the assigned facts only. Using these states, we calculate the weight of each fact with $\mathcal{C}^k_f(\mathcal{M})$. The weights are stored as vectors and passed to the potential optimizer, which uses them to generate the optimization function.
The optimization function $\text{opt}^{t,k}_{\mathcal{M}}$ was implemented implicitly. As all former mentioned methods can handle partial states of cardinalities ≥ 1, only one new method was needed. It

generates n mutex based potential heuristics, each of which uses a random state of size t. With each of these we perform the same procedure as above with p_f.
Running Fast Downward, the mutex based potential heuristic can be used with the command `--search "astar(mutex_based_potential())"`, where astar can be replaced by any other search algorithm of Fast Downward. The size to which p_f should be extended can be set with `k`, the default value is `k=2`.
The command `--search "astar(mutex_based_ensemble_potential())"` uses the mutex based ensemble heuristics. The variables can be set manually and their respective default values are `t=1`, `k=2` and `n=50`. `k` must be greater than `t`, but smaller than the size of a fully extended state. **mention: This is not the same k as in the evaluation section?** `n` can be chosen arbitrarily, however the evaluation in Chapter 4 shows that the results are best with `n=10`.
Further, all potential heuristics can be solved using LP-constraints built with mutexes through the option `mutex=1`. The default value is `mutex=0` for all potential heuristics, except for the mutex based ones. The constraint for the initial state is added through `init-const=1`, the default value for this is 0 for all potential heuristics. With `rand-con-num` the amount of additional constraints on random states can be set, with the default value 0 none are added.
We will evaluate our implementation in the next chapter.

4

Experimantal Evaluation

We tested our implementation of the strengthened potential heuristics on 1827 STRIPS problems with the official domains from the International Planning Competition (IPCs). Further, we used Downward Lab [9] to set up the tests and ran them on the sciCORE high-performance computing infrastructure. We set the time limit for each problem to 30 minutes and the memory limit to 7.6 GB. The different heuristics were all used in an $A^\star$ search (Hart et al.).

We test whether the LP constraints built with disambiguations improves the performance and compare the different optimization functions against each other. Further, we compare different ensemble heuristics and their respective amount of heuristics used, as well as the additional constraints on the initial state and on random states. We refer to the compared variants of heuristics as follows:

`lmc`: The LM-Cut heuristic, introduced by Helmert and Domshlak

`init`: The initial state potential heuristic (3.7)

`all`: The all states potential heuristic (3.8)

`max`: The maximization over `init` and `all`

`div`: The diversification heuristic **reference?**

$\mathbf{S}^{\mathtt{n}}_{\mathtt{i}}$: The sample based potential heuristic (3.9), with `n` being the umber of heuristics, `i` he number of samples per heuristic.

$\mathbf{M}_{\mathtt{k}}$ The strengthened all states potential heuristic with $\text{opt}^k_{\mathcal{M}}$ (3.12)

$\mathbf{K}^{\mathtt{n}}_{\mathtt{k}}$ The strengthened ensemble potential heuristic with $\text{opt}^{t,k}_{\mathcal{M}}$ (3.14) with $|t| = 1$

$\mathbf{L}^{\mathtt{n}}_{\mathtt{k}}$ The strengthened ensemble potential heuristic with $\text{opt}^{t,k}_{\mathcal{M}}$ (3.14) with $|t| = 2$

In addition, **`N`** refers to the non-strengthened LP, while **`D`** uses the LP strengthened with disambiguations. When the additional constraint on the initial state is used, **`I`** is appended to the name of the heuristic and **`R`** if the constraint on random states is used.

The attributes we use to compare different configurations are:

Coverage: The amount of problems solved with the respective configuration.
Expansions: The geometric mean of expansions needed to solve the problem.
Total Time: The geometric mean of the total time to solve the problems.
Search Time: Total time minus the time needed to build the mutex table.
Out of Memory: The amount of problems which failed due to a lack of memory.
Out of Time: The amount of problems which failed due to a lack of time.

The attributes expansions, total time and search time are the geometric mean over all problems which were solved by all configurations. The search time contains the time needed to build the Linear Program plus the time for the search. It always differs from total time, as total time also takes the time for translation and other preprocessing into account. In all tables, the best value per attribute is written in bold. For coverage, this is the highest value, for the other attributes it is the lowest.

4.1 Results

The following table shows results for the heuristics which were already provided in the Fast Downward planning system.

	lmc	**all-N**	**init-N**	**max-N**	**div**	**S_1^{100}-N**	**S_{1000}^{1}-N**
Coverage	958	929	891	948			961
Expansions	1856	11411	24145	9139			21266
Total Time	1.03	0.30	0.57	0.34			0.69
Search Time	0.95	0.24	0.47	0.26			0.46
Out of Memory	0	870	911	854			843
Out of Time	852	11	8	8			6

Table 4.1: Test results for the already provided heuristics.

We see that `lmc`, S_{1000}^{1}-N and `max-N` have the highest coverage over all problems. Further, `lmc` has the lowest number of expansions and therefore the lowest out of memory errors, while `max` is the fastest of the three configurations, therefore having only few out of time errors.
In the following subsections, we compare these results to the results we gathered with the features we implemented.

4.1.1 Mutex Based Linear Program

First, we compare the results from above with the ones obtained with the LP built with mutexes and disambiguations. The following table shows multiple configurations and their respective results.
The coverage is lower for these configurations. However, the number of expansions is better for all configurations, which decreases the out of memory errors.
The out of time errors, on the other hand, are higher. This is no surprise, as the total time is higher for all configurations as well. The search time is roughly the same, which indicates that the building of the mutex table is the most time consuming difference. Hence, it is the

	all-D	init-D	max-D	div	S_1^{100}-D	S_{1000}^{1}-D
Coverage	879	881	932	837	853	**952**
Expansions	12941	19259	8511	**5831**	6095	15655
Total Time	**0.65**	0.86	0.80	4.15	3.18	1.42
Search Time	0.28	0.40	0.25	**0.22**	0.34	0.37
Out of Memory	824	824	770	560	**273**	729
Out of Time	75	**73**	76	381	652	97

Table 4.2: Test results for mutex based Linear Program.

reason for the high amount of out of time errors, and therefore the relatively low coverage. We already greatly enhanced the performance of this part However, finding an even better way for building the mutex table would lead to a higher coverage. We optimized the (very slow) Fast Downward hm-heuristic for $m = 2$ and ignored heuristic values, as we are only interested in the binary reachability. Before the optimization, the mutex table was built for 1450 problems, in 117 seconds on average. Now, 1776 mutex tables are built in 34 seconds on average. Thus, over 300 additional mutex tables can be built in less than 30 minutes, some of them in less than 20 seconds.

Comparing the configurations amongst each other, we see that the coverages of S_{1000}^{1}-D and max-D are still good, as they only decrease by 10 problems. So does the coverage of init. These are also the heuristics, for which the expansions and search time sunk, as well as the out of memory errors. The other configurations were vastly worse with disambiguations. We can conclude this, too, from the higher expansions, search times, and little less out of memory errors.

comparison for div and s-1-100 missing

4.1.2 Mutex based optimization functions

Second, we look at the results for the mutex based potential heuristics. The mentioned configurations for ensemble heuristics (K and L) are chosen for display, as they have the highest coverage among the ones we tested (Table 4.4).

	M_1-D	M_2-D	K_1^{10}-D	K_2^{50}-D	L_1^{10}-D	L_2^{10}-D	k=3
Coverage	900	859	907	888	922	840	
Expansions	9111	9045	6892	8603	6394	6278	
Total Time	0.65	1.07	0.74	3.95	0.74	2.13	
Search Time	0.21	0.21	0.65	3.76	0.65	2.03	
Out of Memory	802	726	720	495	681	589	
Out of Time	77	203	154	398	178	360	

Table 4.3: Test results for mutex based potential heuristics.

For the three different categories, M, K and L, we see that the configuration with smaller k is better. This is due to the higher time and memory consumption needed for bigger extensions, which can be concluded from the increasing sum of errors. Coincidentally, there is a shift in where more error occur. For bigger k, time becomes a problem before memory

does. Configurations with k=2, and how fast it increases. **For k=3...**
The expansions however decrease, which indicates that the approach is very good, as the solution is found with less effort, once the search has started.
The comparison of these results with table 4.1 shows, that the coverage is not higher. However, less out of memory errors occurred, while the out of time errors raised. This is, very similar to the results for the mutex based LP (sec. 4.1.1), due to the building of the mutex table.
Currently, the implementation for mutex based potential heuristics is able to work with any $k \in \mathbb{N}$ smaller than the size of a state. An optimization for $k = 1$ and $k = 2$ could, similar to the optimization of building the mutex table, enhance the coverage as well. **Especially, as the results for k=3 are getting worse. The configuration with k=3 we chose, is the best among the ones we tested. For even higher k, the coverage would drop vastly, as the memory and time limits are not high enough to extend multiple partial states to this size.**
We also tested the configurations M_1-N and M_2-N, which have a higher coverage. With the non-mutex based LP, the expansions and therefore the amount of out of memory errors are higher. The mutex based LP variant on the other hand throws more out of time errors, as it takes five seconds longer to be build.
Plot vor expansion, search time. more comparisons of how search time, expansions and errors differ. But where and how?

4.1.3 Mutex Based Ensemble Heuristics

The next table shows the coverage for different sample based ensemble heuristics, for different amounts of used heuristics.

n	5	10	50	100	250	500
S_1	**889**	888	867	852	815	773
K_1	904	**907**	896	861	790	747
K_2	835	834	**888**	865	669	615
L_1	911	**922**	884	856	771	734
L_2	**840**	**840**	796	756	670	615

Table 4.4: Comparison over different n for different ensemble heuristics.

The best results are achieved with smaller n. The more heuristics are used, the more out of time errors occur. This is both because more heuristics need to be build and because they all need to be considered for each state which is evaluated. However, the expansions get lower the bigger the n. The best coverage has L_1^{10} with 922. **more comparisons of how search time, expansions and errors differ.**

4.1.4 Additional Constraint on the Initial State

The best coverage was gained with the additional constraints **on the initial state**.
all: higher coverage, higher expansions, more time, less out of time/memory.

	all-N-I	div-D-I	S_1^{100}-N-I	M_1-N-I	M_1-D-I
Coverage	965	844	963	946	950
Expansions	19468	11332	20391	20909	14086
Total Time	0.65	7.9	0.67	1.09	1.31
Search Time	0.42	0.33	0.45	0.44	0.33
Out of Memory	837	598	843	796	729
Out of Time	8	352	4	68	115

Table 4.5: Test results for the additional constraint on the initial state.

M-N: higher coverage, higher expansions, more time, slightly less out of time/memory.
M-D: higher coverage, higher expansions, more time, slightly less out of time/memory.

4.1.5 Additional Constraint on Random States

	all-N-I	div-N-I	S_1^{100}-N-I	M_1-N-I	M_1-D-I
Coverage					
Expansions					
Total Time					
Search Time					
Out of Memory					
Out of Time					

Table 4.6: Test results for the additional constraints on random states.

Here are two different tables. **new subsection for second table? we chose n=5 since our pretests (better word?) showed that this gives the best results**

	all-N-I	div-N-I	S_1^{100}-N-I	M_1-N-I	M_1-D-I
Coverage					
Expansions					
Total Time					
Search Time					
Out of Memory					
Out of Time					

Table 4.7: Test results for the combination of the additional constraints on the initial state and random states.

4.2 Comparison to Fišer et al.

compare to fiser et al., with results of tests from cppdl

5
Conclusion

In conclusion, Disambiguations are okay, while the additional constraints are good.

Bibliography

[1] Daniel Fišer, Rostislav Horčík, and Antonín Komenda. Strengthening potential heuristics with mutexes and disambiguations. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 124–133, 2020.

[2] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[3] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[4] Malte Helmert. Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence*, 173(5-6):503–535, 2009.

[5] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: what's the difference anyway? In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2010.

[6] Florian Pommerening and Malte Helmert. A normal form for classical planning tasks. 2015.

[7] Florian Pommerening, Malte Helmert, Gabriele Röger, and Jendrik Seipp. From non-negative to general operator cost partitioning. 2015.

[8] Jendrik Seipp, Florian Pommerening, and Malte Helmert. New optimization functions for potential heuristics. 2015.

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

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor
Salome Müller

Matriculation number — Matrikelnummer
2017-063-058

Title of work — Titel der Arbeit
Mutex Based Potential Heuristics

Type of work — Typ der Arbeit
Bachelor thesis

Declaration — Erklärung
I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, 06. 11. 2020

Signature — Unterschrift