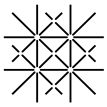

University
of Basel

Abstraction Heuristics for Rubik's Cube

Bachelor Thesis

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

Examiner: Prof. Dr. Malte Helmert
Supervisors: Patrick Ferber and Dr. Jendrik Seipp

Clemens Büchner
clemens.buechner@unibas.ch
15-059-603

June 2, 2018

Acknowledgments

Joyfully realizing what I have achieved until today, I am very grateful for the people who accompany the journey of my life and were also there for me while working on this thesis.
I am especially thankful for my girlfriend Stefanie and the time we spend together. Thank you for supporting me and always having an open ear for my interests and issues.
I would like to thank my parents Thomas and Beatrix who have always helped me find my way. Feeling their unconditional support is amazing. They provide me with anything I need and are there for me whenever needed. I am also very thankful for my siblings Benjamin and Damaris who make me feel home no matter where we are.
I owe my thanks to Prof. Dr. Malte Helmert for giving me the opportunity of writing my Bachelor thesis in the AI group at the University of Basel. A special thank you goes to my supervisors Patrick and Jendrik who have helped me understand the topics covered in this thesis. They have always offered me their help and were answering tons questions from my behalf even at times I did not expect to get a response anytime soon. Thank you for all your time to guide me through this thesis.
Very much thanks go to the people who have read my thesis and provided me with feedback. I especially want to thank Stefanie and Alexander who have spent many hours to help me improve my work.
Also many thanks to the sciCORE[1] high performance computation center and the team behind it for providing me with the possibility to work on their cluster.
I thank the people connected to the Jungschar Boa for accepting my limited capacities and availability. I am glad to be able to rely on my team members who have pitched in to relieve me from some of my responsibilities.
Last but not least I thank my coach Davor and our teammates at the BSC Chargers Baselland for understanding my absence at training sessions due to the workload this thesis entailed.

[1] https://scicore.unibas.ch

Abstract

We consider the problem of Rubik's Cube to evaluate modern abstraction heuristics. In order to find feasible abstractions of the enormous state space spanned by Rubik's Cube, we apply projection in the form of pattern databases, Cartesian abstraction by doing counterexample-guided abstraction refinement as well as merge-and-shrink strategies. While previous publications on Cartesian abstractions have not covered applicability for planning tasks with conditional effects, we introduce factorized effect tasks and show that Cartesian abstraction can be applied to them. In order to evaluate the performance of the chosen heuristics, we run experiments on different problem instances of Rubik's Cube. We compare them by the initial h-value found for all problems and analyze the number of expanded states up to the last f-layer. These criteria provide insights about the informativeness of the considered heuristics. Cartesian Abstraction yields perfect heuristic values for problem instances close to the goal, however it is outperformed by pattern databases for more complex instances. Even though merge-and-shrink is the most general abstraction among the considered, it does not show better performance than the others.

Table of Contents

1 Introduction

This thesis aims to give an overview of modern *abstraction heuristics* as they are used for solving *classical planning tasks*. In order to do so, we present three classes of abstractions: *Projections*, *Cartesian abstractions* and *merge-and-shrink abstractions*. They are applied to the problem of *Rubik's Cube* which is a puzzle where turning the layers of the cube are the applicable actions and the goal is to move all pieces of the same color onto the same face. Actions can be applied to Rubik's Cube so that over 43 quintillion different states can be reached. Searching the optimal plan in this enormous *state space* seems to be impossible for human beings and even with the aid of a computer it must be narrowed down to avoid running out of memory.

The presented abstractions use different strategies and therefore, we assume that their performance varies. Merge-and-shrink iteratively performs merge and shrink steps in order to create an abstraction that is the most general among the considered. We apply *counterexample-guided abstraction refinement* (CEGAR) in order to obtain Cartesian abstractions which are still more general than projections. The approach for getting projections is to use *pattern databases* (PDBs).

To confirm this assumption, we evaluate the performance of the considered abstraction heuristics using Fast Downward [5]. With merge-and-shrink as well as PDBs already implemented, we have left to adjust the concepts for CEGAR which has only been applied to planning tasks with no *conditional effects*. We develop an approach to make CEGAR also applicable for Rubik's Cube, which uses conditional effects, and introduce it as *factored effect tasks*. By running various experiments we compute initial h-values as well as numbers of expanded states until the last f-layer for all heuristics. We explore homogeneities and differences among the values for these and other attributes over all problem instances and heuristic configurations. The results of these experiments are presented and discussed in the last part of this thesis.

2

Rubik's Cube

Originally designed for students to understand three-dimensional problems, Prof. Ernõ Rubik invented Rubik's Cube, also known as „Magic Cube“, in the year 1974 [1]. While his prototype had a size of $4 \times 4 \times 4$, we will generally refer to the *classical Rubik's Cube* as a $3 \times 3 \times 3$ cube, since nowadays this is the most commonly used model. Meanwhile, a wide range of forms and variations have been introduced under the denomination of Rubik's Cube, but we restrict ourselves to the classical one which is a cube with six faces of equal magnitude which are perpendicular to one another, as it is depicted in Fig. 2.1.

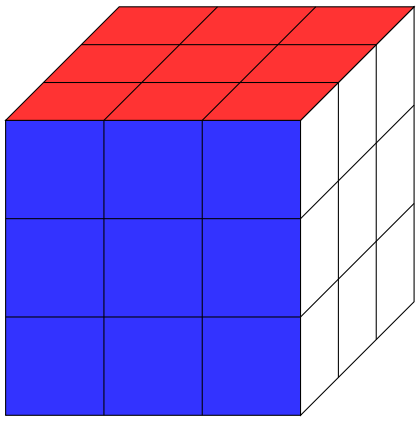

Figure 2.1: Solved classical Rubik's Cube of size 3.

2.1 Terminology

We introduce terms and expressions to simplify the wording for Rubik's Cube and its states. Certain terms introduced in this section correspond to the wording of a cube as a mathematical body. Please note that we only consider three-dimensional (Rubik's) cubes within this thesis.

Cubie Rubik's Cubes are „cut“ uniformly into multiple slices parallel to each face, resulting in a large amount of smaller cubes which we call *cubies*. On classical Rubik's Cubes we

find three different kinds of cubies. The term *category* when referring to all cubies of the same kind. The cubie in the inside of a cube can never appear on the surface as a result of any combination of actions as described in Section 2.2.2.

Face A set of cubies covering one whole side of Rubik's Cube is called *face*. In Fig. 2.2, the cubies marked grey are those of the face turned towards the viewer. To denote actions later on, we assign names to the faces. The face highlighted in gray in Fig. 2.2 is called the front face (F), in the opposite direction we have the back face (B). Furthermore, we have the left face (L), the right face (R), the upper face (U) and the downward face (D). We call the faces of a cubie *facelet* in order to avoid confusion with the term of faces on the cube as a whole.

Corner The *corners* of Rubik's Cube are those cubies positioned on three faces at once. There are eight corner cubies on a classical Rubik's Cube. There are three facelets on the surface on each corner cubie. Fig. 2.3 highlights the visible corner cubies in gray. The corner cubie that is located on the downward, left and back face is not visible at all.

Edge Adapting the concept from mathematics, edges are the cubies located between two corner cubies. In Fig. 2.4, edge cubies are visually highlighted in gray. Each of the edge cubies appears on two faces. There are exactly twelve edge cubies on a classical Rubik's Cube.

Center *Center* cubies are those which only have one facelet on the outside of Rubik's Cube. There is exactly one center-cubie on each face which means that we have a total of six center-cubies. In Fig. 2.5 they are highlighted in gray, while only half of them are visible from the given viewpoint.

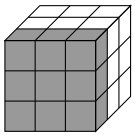

Figure 2.2: Front face.

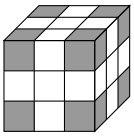

Figure 2.3: Corner cubies.

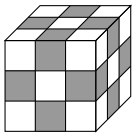

Figure 2.4: Edge cubies.

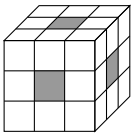

Figure 2.5: Center cubies.

Layer We define a *layer* to be a subset of cubies that form a plane parallel to a face of the cube. We denote them with the name of the face and an index subscript. The faces themselves are annotated with index 0. In the classical Rubik's Cube, a layer is either a face, or the cubies between two faces. For example we can refer to the front face F as the F_0-layer or also the B_2-layer.

Size The *size* denotes the number of layers in each dimension. Classical Rubik's Cubes have a size of 3, which means that each face has a total of $3^2 = 9$ cubies on it.

2.2 Problem Description

The cube described above is based on the puzzle invented by Prof. Ernõ Rubik. Each cubie is of different color on its facelets and when the puzzle is solved, each face has only facelets of one color on it but is colored different than all other faces. It is possible to independently turn every layer around the cubie in its center by 90 degrees. Turning any layer of the cube will shift all cubies on that layer and the colors on the faces scramble. Doing an arbitrary amount of such turns will most likely leave the cube in a state where each face of the cube has a mixture of facelet-colors on it. Then, the goal is to restore the state of the cube where each face has only facelets of one color on it, as it is shown in Fig. 2.1. Recently, it has been proven that any configuration of a classical Rubik's Cube can be solved with a maximum of 20 moves [14], which was already assumed at least 20 years ago [12].

2.2.1 Reducing the Problem Space

Given their size of 3, classical Rubik's Cubes consist of $3^3 = 27$ cubies. However, the one in the center is of no relevance since it is never visible from the outside. Furthermore, six of the remaining 26 cubies are fixed in their position: The cubies lying in the center of each face cannot be interchanged. Only turning a middle layer changes the location of such center cubies, which is equivalent to turning the layers parallel to it in the opposite direction. Also, the locations relative to all other center cubies do not change when applying such an action which is why we forbid turning middle layers. Although these cubies do rotate when their corresponding faces are rotated they form a fixed reference framework disallowing to turn the entire cube [12]. We do not have to consider them for solving the puzzle and therefore can narrow it down to a total of 20 cubies necessary for defining our problem space.
From these remaining 20 cubies, eight are corner cubies and twelve are edge cubies. They always stay within their corresponding category, so a corner cubie can never become an edge cubie and accordingly the other way around. Since it is only possible to turn layers by 90 degrees, corner cubies will always end up in a corner location and edge cubies accordingly on edge locations.

2.2.2 Operators

Given the six faces of a Rubik's Cube, we introduce a total of 18 operations applicable to the problem. This comes down to three operations per face. We present them on the front face but apply similarly to the other faces.
The first option is to turn the face by 90 degrees clockwise. We denote this operation with F, which is an abbreviation for the front face that is turned. The second possibility is to turn the face by 90 degrees counterclockwise. We denote this operation with F' (speaking „F prime"), where the prime implies the opposite direction of the turn. This is equivalent to three consecutive F operations. The last possible operation on the is denoted with F2 and describes the move that turns the face upside down, meaning we apply two consecutive F operations, which is also equivalent to doing two F' operations.
Applying the same semantics to the other faces B, L, R, U and D calls for an explanation of the terms clockwise and counterclockwise. These apply to the direction when looking

directly onto the considered face. For example, using the F operator moves the top left corner on the front face to the top right corner on the front face, whereas using the B operator will move the top left corner on the back face to the bottom left corner on the back face.

3
Classical Planning

Rubik's Cube as it is described in Chapter 2 is a *classical planning task*. The goal of classical planning is to find a path from one state to another in a *state space* by applying multiple actions [10]. Since such state spaces are mostly by far too large to search paths without any further information, we make use of heuristics that help us search towards promising directions.

3.1 Planning Tasks and State Spaces

A planning task is defined as an assignment of finding a path from an initial state to a goal state. The following definition in finite-domain representation is taken from Helmert and Röger [9].

Definition 1. *A* planning task *is a 4-tuple* $\Pi = \langle \mathcal{V}, I, \mathcal{O}, \gamma \rangle$ *where*

- $\mathcal{V}$ *is a finite set of* finite-domain state variables*,*
- I *is a state over* $\mathcal{V}$ *called the* initial state*,*
- $\mathcal{O}$ *is a finite set of* finite-domain operators *over* $\mathcal{V}$*, and*
- γ *is a formula over* $\mathcal{V}$ *called the* goal*.*

The following additions to this definition are adopted from Seipp and Helmert [15]:

- The mapping $V \mapsto v \in \mathit{dom}(V)$ of a variable $V \in \mathcal{V}$ is called an *atom*. We denote a formula over $\mathcal{V}$ as a *partial state*. Atoms are used to assign values in their respective domains to a subset of variables $\mathit{vars}(()\, s) \subseteq \mathcal{V}$ in partial state s. We write $s[V] \in \mathit{dom}(V)$ for the value which s assigns to the variable V. Partial states defined on all variables are called *states*, and $S(\Pi)$ is the set of all states of Π.
- Each operator $o \in \mathcal{O}$ has a *precondition* $\mathit{pre}(o)$, an *effect* $\mathit{eff}(o)$ and a non-negative *cost* $\mathit{costo} \in \mathbb{R}_0^+$. The precondition $\mathit{pre}(o)$ and effect $\mathit{eff}(o)$ are partial states. An operator $o \in \mathcal{O}$ is applicable in state s if $\mathit{pre}(o) \subseteq s$.

Each planning task induces a state space which we define according to Helmert [6].

Definition 2. *A* state space *is a 6-tuple* $\mathcal{S} = \langle S, A, cost, T, s_0, S_\star \rangle$ *with*

- S *a finite set of* states,
- A *a finite set of* actions,
- $cost : A \to \mathbb{R}_0^+$ *the* action costs,
- $T \subseteq S \times A \times S$ *the* transition relation *from one state to another through an action,*
- $s_0 \in S$ *the* initial state, *and*
- $S_\star \subseteq S$ *the set of concrete* goal states.

A plan *is a path from* s_0 *to* $s_\star \in S_\star$ *labeled by the actions of each transition taken to get to* $s_\star$. *The plan is called* optimal *if the sum of costs along the path is minimal.*

We further define effects according to Helmert and Röger [9].

Definition 3. Effects *over state variables* $\mathcal{V}$ *are inductively defined as follows:*

- *If* $V \in \mathcal{V}$ *is a state variable, then* $V \mapsto d \in dom(V)$ *is an effect* (atomic effect).
- *If* $e_1, \ldots, e_n$ *are effects, then* $(e_1 \wedge \cdots \wedge e_n)$ *is an effect* (conjunctive effect). *The special case with* $n = 0$ *is the* empty effect $\top$.
- *If* $\mathcal{X}$ *is a logical formula and* e *is an effect, then* $(\mathcal{X} \rhd e)$ *is an effect* (conditional effect).

3.2 Abstraction Heuristics

The solution strategies evaluated within this thesis are connected since they abstract the considered state space in order to simplify a planning task. Abstraction usually happens when a state space is modified into a coarser version by losing some distinctions [15]. Abstract state spaces have fewer states and lead to a more tractable analysis [8]. According to Helmert et al., every abstraction yields an admissible heuristic, since abstractions preserve paths in the underlying transition graph [7]. The following sections present the methods we apply in our evaluations.

3.2.1 Projections

Instead of solving a comparatively large problem, we can use projections to shrink the effort. By only focusing on parts of the underlying problem, projections disregard some of the available information on purpose. We use pattern databases and only take into account a subset of all state variables. The method can be seen as an adaption of human problem solving behavior for combinatorial puzzles or multi-player games where we often apply a „divide-and-conquer“ strategy to reach intermediate goals [3].
We use the term of *target patterns* similar to Culberson and Schaeffer: One target pattern describes a partial specification of the goal. Then the *pattern database* (PDB) is the set of

all patterns that can be obtained by permutations over the variables in a target pattern. We compute the cost of each pattern in the PDB, which corresponds to the minimal distance to its target pattern. These values are stored in a lookup table. This gives a lower bound to the complete planning task since we cannot find a solution without solving the sub-problem for the considered target pattern [12].
We can either look up the value found for one pattern or combine the values of multiple patterns in order to come up with a heuristic value for a state. For example, taking the maximum value over two or more patterns will always be at least as good as looking only at one of these values. Since the pattern that provides the maximum value is still a lower bound on satisfying its target pattern, this value is the minimum cost we must expect for solving the overall problem.

3.2.2 Cartesian Abstractions

The idea behind *Cartesian abstraction*, which is a rather new class of abstractions, is to start from a coarse and maybe inaccurate abstraction and iteratively improve it [15]. The refinements in each iteration happen only where flaws in the previous abstraction were detected. The method introduced by Seipp and Helmert is called counterexample-guided abstraction refinement (CEGAR). Its goal is to derive Cartesian abstraction heuristics for optimal classical planning. It is a variation of the work published by Clarke et al. [2] who have applied this technique for model checking.
The concept is more general than PDBs. In comparison, Cartesian abstraction does not choose patterns of variables present in a planning task but takes apart the corresponding domains of the variables. We define *Cartesian sets* and Cartesian abstraction according to Seipp and Helmert [15] in Definition 4.

Definition 4. *A set of states for a planning task with variables $\langle v_1, \ldots, v_n \rangle$ is called* Cartesian *if it is of the form $A_1 \times A_2 \times \cdots \times A_n$, where $A_i \subseteq dom(v_i)$ for all $1 \leq i \leq n$.*
An abstraction is called Cartesian *if all its abstract states are Cartesian sets.*
For an abstract state $a = A_1 \times \cdots \times A_n$, we define $dom(v_i, a) = A_i \subseteq dom(v_i)$ for all $1 \leq i \leq n$ as the set of values that variable v_i can have in abstract state a.

An abstraction of a state space is called Cartesian if all its abstract states are Cartesian sets [15]. Given a planning task with variables $\langle V_1, \ldots, V_n \rangle$, a set of states is Cartesian if it is of the form $A_1 \times A_2 \times \cdots \times A_n$ where $A_i \subseteq dom\,(V_i)$ for all $1 \leq i \leq n$. In other words, abstract states are collections of concrete states with certain similarities. A transition between two abstract states is present if a transition between two concrete states, one from either abstract state, exists.
Another difference to pattern databases is the variance in granularity of the different abstract states. In Cartesian abstraction, abstract states can either consist of a single concrete state or of a large number of concrete states [15].
Instead of searching for one abstraction of the state space, Seipp and Helmert suggest using additive abstractions that can also be used to estimate a heuristic value when combined together. In order to combine Cartesian abstractions admissibly, they further introduce saturated cost partitioning. Since this part diverges from the aims of this thesis, we will

not go any further into detail. However, their work also provides an overview over other abstraction methods present in our evaluations.

3.2.3 Merge-and-Shrink Abstractions

The idea behind merge-and-shrink abstractions originated in model checking for automata networks [8]. The idea was carried further to be used as a heuristic function and was later applied by Helmert et al. to planning as a new class of abstracting planning tasks.
Building merge-and-shrink abstractions is an iterative procedure and consists of the merge step and the shrink step. Starting from atomic projections, abstractions are merged together and therefore grow during the merge step, whereas in the shrink step, others are shrunk and apply additional abstractions.
Merge-and-shrink is the most general among the considered classes of abstractions. In other words, instances of the less general abstractions can also be the result when applying merge-and-shrink. So, merge-and-shrink can derive the same abstractions as Cartesian abstraction and projections, but can also find abstractions that are considered to be neither Cartesian nor projections. We therefore expect merge-and-shrink to perform better in the evaluations.

4

Related Work

When talking about Rubik's Cube and abstraction heuristics it is essential to mention the publication of „Finding Optimal Solutions to Rubik's Cube Using Pattern Databases“ published by Korf [12] in the year 1997.
He begins by introducing Rubik's Cube as the problem on which he applies his experiments. Also working on the classical Rubik's Cube of $3 \times 3 \times 3$ cubies, he only uses the corner cubies and edge cubies (20 cubies in total) to represent a state in the state space and there are 18 applicable operators. This corresponds exactly to our model of the state space, which is described in Chapter 5.
Discussing the memory used for the lookup tables, he shows that it is possible to use memory in a very compact and therefore efficient manner, even though the state space of Rubik's Cube is enormous. He uses three pattern databases for which the the manhattan distance is computed for all cubies in the target pattern. One pattern covers all corner cubies and two patterns each cover one half of the edge cubies. The difference that he observes with taking the maximum over these three patterns compared to only the one of the corner cubies is stated to be a significant increase in performance. All problem instances he used for his experiments could be solved optimally.
In the last part, the presented research focuses on analyzing the performance of the method. He compares his expectations with the computations of his implementation and often finds discrepancies because of duplicates in the generated search tree. He suggest storing larger heuristic tables which was not possible on his hardware given the latency of disks as the storage component would not have been bearable.

5
Modeling Rubik's Cube

Even though Rubik's Cube exists in a range of various forms and sizes, we decided to restrict our evaluations to use only the classical $3 \times 3 \times 3$ cube as described in Chapter 2. This chapter introduces the chosen formalization for the problem.

5.1 Choice of State Space

When choosing a state space representation of a problem, we always want to keep it as small as possible in order to find solutions faster. The smaller the state space is, the less states have to be considered within the search. For Rubik's Cube, we have examined two different approaches.

5.1.1 Model using Facelets

The state of Rubik's Cube can be described by its facelets. There are 24 facelets on corner cubies and 24 facelets on edge cubies. Also, there are 24 locations where a corner-facelet can possibly be placed, as well as 24 locations for edge-facelets. This results in a total amount of $24^{24} \cdot 24^{24} = 24^{48}$ possible permutations of all facelets.

For each operator we find the following amount of effects: Having 12 facelets on the corner cubies of one face and 24 facelets on corner cubies in total, one operator will yield $12 \cdot 24 = 288$ effects. Additionally, with 8 facelets on the edge cubies of one face and totally 24 facelets on edge cubies, we have another $8 \cdot 24 = 192$ effects . This sums up to a total of 480 effects per operator.

Because we ignore the fact that all facelets on the same cubie are stuck together and cannot be moved around the cube independently, this model allows a lot of unreachable states. The resulting numbers of states and effects are thus suboptimal.

5.1.2 Model using Cubies

We can improve this representation by focusing on cubies instead of facelets. Given 8 corner cubies with 3 possible rotations and 12 edge cubies with 2 possible rotations and only 8 locations for corner cubies and 12 locations for edge cubies, we get a reduction

of the state space. Thus, the number of permutations within our state space reduces to $(8 \cdot 3)^8 \cdot (12 \cdot 2)^{12} = 24^8 \cdot 24^{12} = 24^{20}$ states.
The number of effects per operator is also reduced: For any action, we have 4 corner cubies on the according face, each with three rotations. Therefore, there are $4 \cdot 3 \cdot 8 = 96$ effects for corner cubies. Accordingly, we have 4 edge cubies on a face with 2 rotations each, which yields $4 \cdot 2 \cdot 12 = 96$ effects as well. Summing up these yields a total of 192 effects per operator.
We could go even further and remove one more cubie per category from the model. This is because its position and rotation can be determined by the positions and rotations of all other cubies of the same category [12]. However, this is not done within the scope of this thesis in order to keep the generation of problem files a little simpler and more intuitive.

5.2 Coordinate System

We establish an ordering of cubies on Rubik's Cube by enumerating them starting at the top left corner of the front face with a value of 0. Each category is considered for itself, since cubies cannot move between locations of different categories. From top to bottom, left to right, front to back, we assign the next higher integer number to each cubie of the given category. Fig. 5.1 illustrates the values of the visible cubies where the categories are distinguished by the color of the labels: Corner cubies are displayed in red whereas edge cubies are marked in blue.

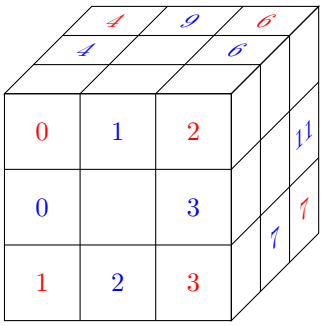

Figure 5.1: The coordinate system used for our model of a classical Rubik's Cube.

5.3 Locations and Rotations

Each cubie considered for classical Rubik's Cubes has two values which define its state unambiguously. Firstly, we need to specify where on Rubik's Cube each cubie is located in a state. For example, a corner cubie can be located on either of the eight corner cubie locations. Secondly, it can lie in one of three different rotation directions. Edge cubies, however, can be located on either of the twelve edge cubie locations and their rotation can be either of two directions.
We take the coordinate as the value for denoting its location on the cube. Even though we have the same values for coordinates of corner cubies as for edge cubies, this is still

unambiguous since we know whether a cubie is a corner cubie or an edge cubie.
For describing the rotation value, we imagine a three-dimensional object as shown in Fig. 5.2. We describe its current rotation by a triple $\langle x, y, z\rangle$ where x corresponds to the color of the plane perpendicular to the x-axis, namely the y-z-plane. Accordingly y is the color of the x-z-plane and z the color of the x-y-plane. The example shown in Fig. 5.2 is denoted as $\langle r, g, b\rangle$, where the starting letter of the colors are taken as the values in the triple.
As in the Rubik's Cube-problem, we only allow rotations by 90 degrees around either of the x-, y- or z-axis. Applying a rotation around the x-axis switches the values of y and z in the triple and in our example results in the triple $\langle r, b, g\rangle$ which corresponds to Fig. 5.3. Similarly, turning around the y-axis will switch the values of x and z and turning around the z-axis switches x with y accordingly. The resulting rotation triple is not dependent on the direction of the turn and turning by 180 degrees does not have any effects.

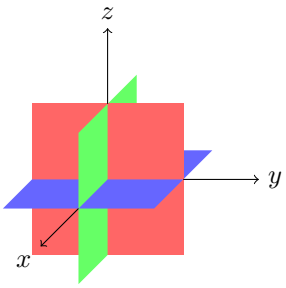

Figure 5.2: Rotation imagination aid.

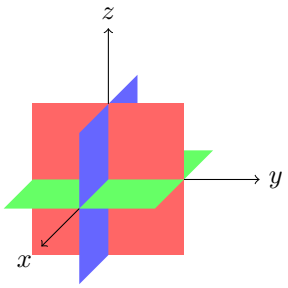

Figure 5.3: Rotation after a turn of 90 degrees around the x-axis.

For edge cubies we use the same notation, but denote one plane's color as # (speaking „blank"), which corresponds to the direction where no facelet is visible from the outside of the cube. Then, rotations on the rotation triple can be performed similar to the case of corner cubies.
We have stated before that we only need three rotation values to manifest a cubie's state together with its location. The reason for this is that every corner cubie has three visible facelets which cannot be interchanged and each facelet can be turned towards either of three directions in one position. However, there are six different rotation triples that can be achieved by applying rotations. In order to reduce the representation as a rotation triple to only three values (or rather two for edge cubies) we apply the following: From a rotation triple $\langle x, y, z\rangle$ we only need information about the value in one direction and then together with the position of the cubie can conclude the other two. Therefore, the first non-blank element of the triple is taken to be the rotation value, which has three possibilities in the case of corner cubies and two for edge cubies.

5.4 Formalization

We formalize planning tasks considered within this thesis in the commonly used SAS$^+$ formalism. As we have established in Section 5.1, we have a rather compact description of the state space when using cubies.

5.4.1 Separating Locations from Rotations

Our first version of the model uses 40 variables, which makes two per cubie. One of these is concerned with the location, the other denotes the rotation. The domain of the location variable for corner cubies therefore is a value from 0 through 7, whereas for edge cubies it is a value from 0 through 11. The domain of the rotation variable for corner cubies is a value from 0 through 2, whereas for edge cubies it is either 0 or 1.

If one corner cubie is at any location, no other corner cubie can be located there as well. This knowledge can be modeled with eight mutexes for corner cubies and twelve for edge cubies. There are even more restrictions that could be captured with mutexes in the SAS^+ formalism. For example, it is not possible to mutate a single cubie, meaning that we cannot apply a series of actions that returns to its starting state and only changes the rotation value of one cubie. Even interchanging two cubies is not possible, so literature talks about *3-cycles* that can be performed on Rubik's Cube [11]. However, we did not model these in our description of the state space.

Operators for Rubik's Cube do not have preconditions, so they are all applicable in every state. However, the effects of each operator only happen under certain circumstances. We call these the effect conditions and all effects of operators have at least one such condition. For example, a cubie will only be affected by the F operator if it is located on the front face before F is applied. Consequently, it will also be on the front face after applying F since all cubies not on F keep their location. Effects for location variables, on one hand, have exactly this effect condition. On the other hand, effects for the rotation variable of the same cubie need information about its location as well as its rotation value in order to tell whether or not the effect triggers. Example 1 illustrates the explanations above.

Example 1. *Let c_i be the variable for the cubie that is located at coordinate i in the solved Rubik's Cube. Let r_i be the rotation of the same cubie. The atom $c_i \mapsto j$ maps c_i to coordinate j. Accordingly, $r_i \mapsto j$ maps r_i to one of the possible rotation values. Then applying operator U to turn the upper face by 90 degrees has the following effects $a \rhd b$ where a is the effect condition and b is the effect fact:*

$$
\begin{aligned}
c_0 \mapsto 0 \quad &\rhd \quad c_0 \mapsto 4 \\
c_0 \mapsto 0 \wedge r_0 \mapsto 0 \quad &\rhd \quad r_0 \mapsto 1 \\
c_0 \mapsto 0 \wedge r_0 \mapsto 1 \quad &\rhd \quad r_0 \mapsto 2 \\
c_0 \mapsto 0 \wedge r_0 \mapsto 2 \quad &\rhd \quad r_0 \mapsto 0 \\
c_0 \mapsto 2 \quad &\rhd \quad c_2 \mapsto 0 \\
&\dots \\
c_7 \mapsto 6 \wedge r_7 \mapsto 2 \quad &\rhd \quad r_7 \mapsto 0 \\
c_8 \mapsto 1 \quad &\rhd \quad c_9 \mapsto 4 \\
&\dots \\
c_{19} \mapsto 9 \wedge r_{19} \mapsto 1 \quad &\rhd \quad r_{19} \mapsto 1
\end{aligned}
$$

5.4.2 Combining Locations and Rotations

The model described above does not meet all requirements needed for purposes explained in Chapter 6. We want our model to have effects with only one effect condition and furthermore, the condition can only depend on the variable that is changed by the effect fact. We can keep our cube model as it is, but we have to apply some minor changes to its formalization. The SAS$^+$-representation we established combines the rotation variable with the location variable. This leads to a total number of only 20 variables, which means one variable per cubie. The domain of each variable corresponds to the cross-product of the possible locations and rotations of the according cubie. To calculate the value for any cubie, we multiply the location value by the number of possible rotations and add the rotation value. For example, a corner cubie at coordinate 5 with rotation 2 is assigned the value $5 \cdot 3 + 2 = 17$. The other way around, if v is the value of a cubie in our adjusted state space and r is the number of possible rotations for the cubie, we get the location of a cubie by calculating $\left\lfloor \frac{v}{r} \right\rfloor$ and its rotation value by $v \mod r$. Example 2 shows the same effects as Example 1 but for the newly introduced model.

Example 2. *Let c_i be the variable for the cubie that is located at coordinate i in the solved Rubik's Cube. Let r_i be the number of possible rotations for the same cubie. The atom $c_i \mapsto j$ maps c_i to coordinate $\left\lfloor \frac{j}{r_i} \right\rfloor$ and the rotation value $j \mod r_i$. Then applying operator U to turn the upper face by 90 degrees has the following effects $a \rhd b$ where a is the effect condition and b is the effect fact:*

$$
\begin{aligned}
c_0 \mapsto 0 \quad &\rhd \quad c_0 \mapsto 13 \\
c_0 \mapsto 1 \quad &\rhd \quad c_0 \mapsto 14 \\
c_0 \mapsto 2 \quad &\rhd \quad c_0 \mapsto 12 \\
c_0 \mapsto 6 \quad &\rhd \quad c_0 \mapsto 5 \\
&\dots \\
c_7 \mapsto 20 \quad &\rhd \quad c_7 \mapsto 6 \\
c_8 \mapsto 2 \quad &\rhd \quad c_8 \mapsto 8 \\
&\dots \\
c_{19} \mapsto 19 \quad &\rhd \quad c_{19} \mapsto 13
\end{aligned}
$$

6

Factored Effect Tasks for Cartesian Abstraction Refinement

The theory presented by Seipp and Helmert [15] computes refinements of the abstractions in each iteration by applying regression. However, the regression of a Cartesian state over general operators with conditional effects is not Cartesian. Since Rubik's Cube has only operators with conditional effects, we provide the theory for allowing conditional effects by introducing *factored effect tasks*.

The first step is to understand why the theory proposed would not work for conditional effects. The crucial point is to see that with conditional effects, the value of a variable in the state space is dependent on the state before applying the operator with conditional effects. However, this is not the case when using operators without conditional effects: When applied, they will effect each variable occurring in their effects no matter which value they had before. In their theory, Seipp and Helmert [15] used the function $post(o)$ to denote the partial state over all variables occurring either in the precondition $pre\,(o)$ or the effect $eff\,(o)$ of an operator o.

For the problem of Rubik's Cube considered in this thesis, only one special case of conditional effects occurs: For all (conditional) effects triggered by an operator o it is never necessary to check for other variables than the one that is changed by the effect. We therefore introduce *factored effect tasks* using *factored effect operators*.

Definition 5. *Operator o is a* factored effect operator *if $eff\,(o)$ has the following form:*

$$X \mapsto x_1 \rhd X \mapsto x_2 \wedge X \mapsto x_3 \rhd X \mapsto x_4 \wedge \cdots \wedge Z \mapsto z_1 \rhd Z \mapsto z_2$$

We write $effects\,(o)$ for the set of effects $\langle X \mapsto x_1, X \mapsto x_2 \rangle$ where $x_1, x_2 \in dom(X)$, $x_1 \neq x_2$ and $X \mapsto x_1$ is the effect condition *and $X \mapsto x_2$ is the* effect fact*. A factored effect operator cannot have two effects with equal effect conditions, i.e., $x_1 \neq x_3$ for all pairs of effects $\langle X \mapsto x_1, X \mapsto x_2 \rangle, \langle X \mapsto x_3, X \mapsto x_4 \rangle \in effects\,(o)$.*

We say $X \in vars\,(o)$ if either $X \in vars\,(pre\,(o))$ or there is an effect $\langle X \mapsto x_1, X \mapsto x_2 \rangle \in effects\,(o)$.

Let further $effects\,(o)\,[X] \subseteq effects\,(o)$ be the set of fact pairs $\langle X \mapsto x_1, X \mapsto x_2 \rangle$ that are concerned with variable $X \in \mathcal{V}$. The set $effects\,(o)\,[X]$ can be the empty set.

Definition 6. *A* factored effect task *is a planning task* $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ *where all* $o \in \mathcal{O}$ *are factored effect operators.*

Let us clarify the meaning of Definitions 5 and 6 with the following example:

Example 3. *Let* $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ *be the factored effect task of counting from zero to three with*

- $\mathcal{V} = \{c\}$ *where* c *is the counter variable with* $dom(c) = \{0, 1, 2, 3\}$,
- $\mathcal{O} = \{count\}$ *with* $effects\,(count) = \{\langle c = 0, c = 1\rangle, \langle c = 1, c = 2\rangle, \langle c = 2, c = 3\rangle\}$ *and* $pre\,(count) = \emptyset$,
- $s_0 = \{c \mapsto 0\}$ *and* $s_\star = \{c \mapsto 3\}$

In Example 3 it is impossible to tell the value of c after applying *count*, when we do not have any information about the value of c before applying *count*. Therefore, it is impossible to take on the concept of a function $post(o)$. However, we can find another way to achieve the same goal.
Considering a factored effect task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$, let a be an abstraction of Π, let $o \in \mathcal{O}$ be a factored effect operator and let $X \in \mathcal{V}$ be a state variable. Then the function $resulting\_fact\,(X \mapsto x_1, o)$ computes the value of X after applying o if X has the value x_1 before applying o.

$$resulting\_fact\,(X \mapsto x_1, o) = \begin{cases} X \mapsto x_2 & \text{if } \langle X \mapsto x_1, X \mapsto x_2 \rangle \in effects\,(o) \\ X \mapsto x_1 & \text{otherwise} \end{cases} \tag{6.1}$$

We define the function $possible\,(a, o, X)$ to obtain a set of values that X can possibly have after applying o in any concrete state $s \in a$.

$$possible\,(a, o, X) = \bigcup_{x \in dom(X,a)} \{resulting\_fact\,(x, o)\} \tag{6.2}$$

We go on by working those revisions into the pseudo-code provided within the work of Seipp and Helmert [15]. Algorithm 1 checks whether a transition exists between two abstract states via a given operator. This corresponds to Algorithm 4 in the submission of Seipp and Helmert [15].

Algorithm 1 Transition check. Returns true iff factorized effect operator o induces at least one transition between abstract states a and b.

```
1: function CheckTransition(a, o, b)
2:     for all v ∈ V do
3:         if v ∈ vars(pre(o)) and pre(o)[v] ∉ dom(v, a) then
4:             return false
5:         if v ∈ vars(o) and possible(a, o, v) ∩ dom(v, b) = ∅ then
6:             return false
7:         if v ∉ vars(o) and dom(v, a) ∩ dom(v, b) = ∅ then
8:             return false
9:     return true
```

We also need to update the regression described by Seipp and Helmert [15] in property **P4**. In order to do that, we first need to define effect conditions.

Definition 7. *Let $\ell \mapsto dom(v)$ be an atomic effect. The* effect condition *$effcond\,(\ell, e)$ under which ℓ triggers given the effect e is a propositional formula defined as follows:*

- $effcond\,(\ell, \ell) = \top$
- $effcond\,(\ell, \ell') = \bot$ *for atomic effects* $\ell' \neq \ell$
- $effcond\,(\ell, (e_1 \wedge \cdots \wedge e_n)) = effcond\,(\ell, e_1) \vee \cdots \vee effcond\,(\ell, e_n)$
- $effcond\,(\ell, (\mathcal{X} \rhd e)) = \mathcal{X} \wedge effcond\,(\ell, e)$

The regression computes which value a variable X can have before an operator o has been applied, given the value assigned to X at the moment. Before applying o, X can be mapped to either one of the following options:

- The value of the effect condition of an effect with effect fact $X \mapsto x_1$, or
- x_1 itself, if there is no effect condition that triggers an effect $X \mapsto x_2$ where $x_1 \neq x_2$.

Definition 8. *Let $X \mapsto x_1$ be an atomic effect and let o be an operator in a (general) planning task. Then the regression of X through o is defined as follows:*

$$\begin{aligned} regr\,(X \mapsto x_1, eff\,(o)) \quad = \quad & pre\,(o)\,[X] \wedge (effcond\,(X \mapsto x_1, eff\,(o)) \vee \\ & (X \mapsto x_1 \wedge \neg effcond\,(X \neq x_1, eff\,(o)))) \end{aligned}$$

Considering the special case of factored effect tasks, we can specialize the regression for the case of factored effect operators. Let o be such a factored effect operator, leading Equation 6.3 to show the regression of an atomic effect $X \mapsto x_2$ as described in Definition 8.

$$regr\,(X \mapsto x_2, o) \quad = \quad pre\,(o)\,[X] \wedge \left(\bigvee_{\substack{\langle X \mapsto x_1, X \mapsto x_2 \rangle \in \\ effects(o)}} X \mapsto x_1 \vee \left(X \mapsto x_2 \wedge \neg \bigvee_{\substack{\langle X \mapsto x_3, X \mapsto x_4 \rangle \in \\ effects(o) : x_2 \neq x_4}} X \mapsto x_3 \right) \right) \tag{6.3}$$

We can rewrite Equation 6.3 by reasoning that either $X \mapsto x_2$ occurs as the effect condition of a factored effect or it does not. This information is contained in the term $\neg \bigvee_{\langle X \mapsto x_3, X \mapsto x_4 \rangle \in effects(o) : x_2 \neq x_4} X \mapsto x_3$: For all factored effect pairs which do not have $X \mapsto x_2$ as their effect fact, we check whether their effect condition is false. If so, we add x_2 to the possibilities of previous values for X in order to end up with $X \mapsto x_2$ after applying o. In other words, if $X \mapsto x_2$ occurs as the effect condition for any effect, this effect triggers for the case where $X \mapsto x_2$ held before applying o and therefore cannot hold anymore afterwards. This contradicts our knowledge since we are regressing over the fact $X \mapsto x_2$ and therefore x_2 is not a valid value for X before applying o. This leads to the case distinction shown in Equation 6.4.

$$regr\,(X \mapsto x_2, o) \quad = \quad pre\,(o)\,[X] \wedge \left(\bigvee_{\substack{\langle X \mapsto x_1, X \mapsto x_2 \rangle \in \\ effects(o)}} X \mapsto x_1 \vee \begin{cases} X \mapsto x_2 & \text{if } X \mapsto x_2 \text{ does not occur as an effect condition in } o \\ \bot & \text{otherwise} \end{cases} \right) \tag{6.4}$$

This definition is illustrated by Example 4.

Example 4. *Let o be an operator of a planning task with no precondition ($pre\,(o) = \top$) and the following effect: $eff\,(o) = X \mapsto x_1 \rhd X \mapsto x_2 \wedge Z \mapsto z_1 \rhd Z \mapsto z_2 \wedge Z \mapsto z_2 \rhd Z \mapsto z_1$. Then*

$$\begin{aligned} regr\,(Z \mapsto z_1, o) &= \top \wedge regr\,(Z \mapsto z_1, eff\,(o)) \\ &= (Z \mapsto z_2 \vee (Z \mapsto z_1 \wedge \neg(Z \mapsto z_1))) \\ &= Z \mapsto z_2 \end{aligned}$$

Since we want to compute the regression not only for one atomic effect but for a whole abstract state, we need to add the following to our definition:
For a set $\mathcal{X} \subseteq dom(X)$ of values for a variable X we define the regression as

$$regr\,(\mathcal{X}, o) = \bigcup_{x \in \mathcal{X}} regr\,(X \mapsto x, o) \tag{6.5}$$

and for an abstract state a as

$$regr\,(a, o) = A_1 \times \cdots \times A_n \tag{6.6}$$

where $A_i = regr\,(dom(v_i, a), o)$. These changes suffice to make CEGAR applicable to factored effect tasks.

7 Evaluation

In this chapter we present our evaluations of the abstraction heuristics introduced in Section 3.2. Before doing so, we describe the setting of our experiments.

7.1 Experiment Setup

In order to evaluate abstraction heuristics for Rubik's Cube we first needed to generate a set of problem files. We did so by writing a Python script which takes input parameters like the size, a parameter to choose which operators should be allowed, and the number of turns for scrambling Rubik's Cube. When allowing all 18 operators, this last parameter gives an upper bound on the number of turns needed to solve the problem instance that is generated, since we could simply reverse the order of the scrambling moves and switch the turn direction to obtain a plan to solve the task. In some instances of the scrambling process, moves cancel each other out, thus the actual optimal plan could be shorter than the number of turns passed to the script.
Using the script described above, which we called 10 times for each value from 1 through 20 for the turns parameter, we then generated a set of 200 problem instances for the Rubik's Cube. In the process, we made sure to avoid duplicates by storing hashes of the initial states and redoing the generation step for detected duplicates in the hash map.
We used Downward Lab [16] in order to facilitate to set up the evaluation of our problems. Furthermore, we ran our experiments on the sciCORE high-performance computing infrastructure. The following paragraphs provide more information about the heuristics used for our final experiments. The search was done with the commonly used and well known A* search algorithm [4].

Blind Search Heuristic As a base line we used the blind search heuristic. We expected it to only work on the problems that have short solutions, but we use the number of states expanded in comparison to the other heuristics. The heuristic is part of Fast Downward [5].

Maximum over Manual Patterns We divided the 20 variables into five patterns of equal size. Following Korf [12], each pattern consists either only of corner cubies or only of edge

cubies. Since we were using an implementation of pattern databases that is not optimized for Rubik's Cube, we were not able to make our patterns as large as the ones Korf used. While he kept all corner cubies in one pattern and divided the edge cubies only into two patterns, we had two patterns for corner cubies and three for edge cubies. This makes four cubies per pattern, which was the limit that could be handled by the implementation that was already provided in Fast Downward [5]. Our patterns are the following:

- corner cubies of the F_0-layer,
- corner cubies of the F_2-layer,
- edge cubies of the F_0-layer,
- edge cubies of the F_1-layer, and
- edge cubies of the F_2-layer.

After looking up the abstract goal distance for each pattern independently, we took the maximum over these values for each pattern to get a lower bound on the number of moves necessary to get to the goal.

Maximum over Systematic Patterns The strategy applied here is the same as for the fixed patterns: We take the maximum value over the values computed for a set of smaller patterns. The difference, however, lies in the choice of these patterns. While before, we were selecting patterns manually, we now use all *interesting* patterns up to a given size [13]. This method too was already implemented in Fast Downward [5]. The only parameter needed by the algorithm is the maximum size of each pattern. We ran a separate experiment that tested different values from 2 up to 6 in order to find this value. The most promising one was found to be 3 which was then adopted into our final experiment.

Single Pattern of Corner Cubies or Edge Cubies We also wanted to find out how informative a single pattern of four variables is. Assuming to have symmetries in the patterns of corner cubies and also in the patterns of edge cubies, we include the pattern for the corner cubies of layer F_0 and the pattern for the edge cubies of the same layer.

CEGAR As already discussed in Chapter 6, the implementation provided by Seipp and Helmert [15] was not applicable for tasks that have operators with conditional effects. However, their implementation was taken as a basis for our evaluations. Therefore, we had to change the code in the same places as the theory. We then did some experiments on varying parameters for the search with CEGAR. Concretely, we tested which configuration of limiting the number of transitions or the time yields the best result. In the end, we chose to limit time by a maximum of 900 seconds and allowing infinitely many transitions for the final experiments.

Merge-and-Shrink For the merge-and-shrink heuristic it was possible to take the already implemented version which can also handle conditional effects. It uses the currently recommended merge-and-shrink configuration that employs the DFP-SCC merge strategy, bisimulation and at most 50'000 states.

7.2 Results

For the results presented and discussed in this section we refer to the tables gathered in Appendix A, which show a portion of the reports generated with Downward Lab [16]. They contain the initial h-values computed or rather the number of expanded states until the last f-layer for all of our 200 problem instances and all heuristics evaluated.

7.2.1 Coverage and Errors

Table 7.1 provides an overview of the number of solved problems with each configuration and also the reason of failure regarding the unsolved problems. The coverage denotes the amount of problems that could be solved by either configuration. The sum over coverage, out-of-memory and timeout sums up to 200 for each configuration. The last entry denoted as total time is the calculated geometric mean of time used to find a solution. Problems where no solutions were found are not considered in this value.

Summary	blind	man	syst	corner	edge	cegar	m&s
coverage	68	128	122	108	105	111	95
out-of-memory	132	72	0	92	95	89	105
timeout	0	0	78	0	0	0	0
total time	0.32	16.98	141.36	2.64	2.21	0.61	19.94

Table 7.1: Values summed up over all 200 instances and the geometric mean for the time needed to find a solution.

Within the limits of our working environment, all heuristic search algorithms find more solutions than blind search. The highest coverage is found by applying projection using pattern databases with our manual patterns, followed by systematic patterns. The configuration using systematic pattern generation needs significantly more time for finding paths. It is also the only configuration that ever runs out of time before memory. In fact, it never runs out of memory. Since an initial h-value is found with systematic PDBs for all problem instances, we can rule out that the process of generating the patterns takes too long. We deduce that this heuristic is slow to evaluate whereas all other heuristics are fast to evaluate. The systematic pattern size of three finds 1'350 *interesting* patterns for Rubik's Cube, which are actually all patterns of size three or smaller. Thus, we assume that the difference in time needed for evaluating is dominated by the number of patterns. Still, it performs better than only using one single pattern to estimate the goal distance.
Only using approximately double the time of blind search, our CEGAR implementation is in third place considering coverage. Meanwhile, the memory overhead of merge-and-shrink seems to be rather high, given that it only finds solutions for less than 50% of the problems, while our manual patterns cover almost 65% of the problems. We find nine instances with

a solution cost 12, which is the highest solution cost among all instances. This leads to the conclusion that we are not able to solve problems of Rubik's Cube with an optimal plan cost of 13 or more using any of our configurations. This means that with our configurations we can only solve 0.0001% of all 43'252'003'274'489'856'000 reachable states in the state space. This is according to the list[2] that emerged as a byproduct of the proof that all configurations of classical Rubik's Cubes can be solved in 20 steps [14].
The plan costs found in our experiments are identical throughout either configuration for each problem. They are never higher than the number of turns to initialize the problem.

7.2.2 Initial h-value

In order to compare the configurations by their informativeness, we use the h-values found for the initial states of our problem instances. The higher the value is, the more informed we assume the heuristic to be. In Table A.1 we see that CEGAR is the undisputed leader in this category. It shows the highest value over all configurations for every problem instance. Furthermore, it agrees with the costs for the found plans up to 9 initial turns. From then on, the initial h-value varies somewhere in the range of 8 and 9, but interestingly never goes up to 10 or higher.
The second best informed heuristic among the evaluated is one of the projections. Both manual patterns as well as systematic patterns perform somewhere on the same level where they mostly compute the same value and only rarely vary by more than 1. It is not clear which one is more informative since none of them is always better or worse than the other. They never compute an initial h-value of more than 7. The projection heuristics that only consider one pattern do even worse, where again it is not clear whether one of them outperforms the other. For example, for problem p_2 of 4 turns we can see that they can vary a lot, even though both are concerned with the cubies belonging to the front face.
Surprisingly, merge-and-shrink performed rather poorly. Even though being the most general class of abstractions among the considered, it has the lowest coverage and does not convince with the found initial h-values. At least for Rubik's Cube, it seems that our implementation does not fulfill the expectations of higher potential over other classes of abstractions. While the initial h-value is perfect for problem instances with 1 through 3 turns, the initial h-value only varies between 3 and 4 for all other instances. Therefore, assuming that most states in the state space will have a value of 3 or 4, we suppose that not much change in the h-value can be expected from one state to another. We can only assume that this behavior originates where the abstraction is built, even though there is no evidence of running out of time or anything alike.

7.2.3 Expansions Until Last f-layer

In this section, we set the heuristics in relation to one another. In order to do so, we compare the strategies in pairs, where the pairs are neighbors in the order of generality. We use the number of expansions until the last f-layer and we display the values in scatter plots. This

[2] available online at http://cube20.org (accessed: May 29, 2018)

number denotes how many expansions are necessary to get to the point where the next expansion could reach a goal state.
Fig. 7.1 shows that our implementation of CEGAR outperforms the merge-and-shrink heuristic. Most of the data points are situated on the bottom line of the scale, which means that CEGAR barely has to do any expansions before reaching the goal. Only very few values approach the diagonal, which indicates the threshold on whether a data point speaks for one or the other implementation.

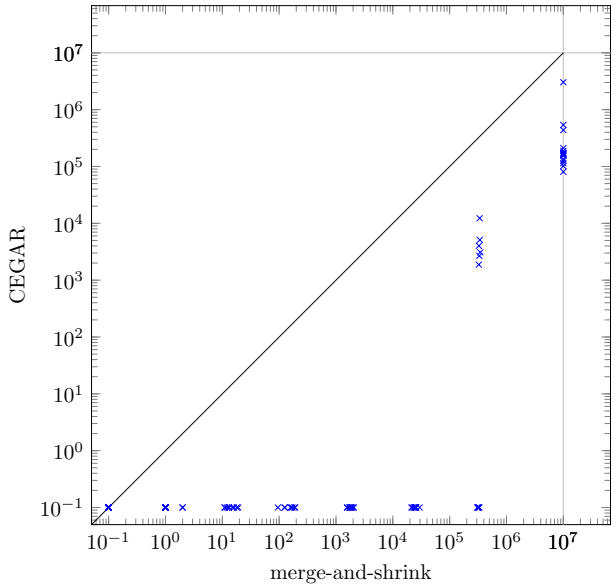

Figure 7.1: Expansions until the last f-layer for merge-and-shrink vs. CEGAR on a loglog scale.

Next up in the ordering are the projections. We compare CEGAR to the fixed patterns in Fig. 7.2. While at first CEGAR again has a set of values very close to zero, at some point its numbers of expansions until the last f-layer increase rapidly and the data points fall below the boundary set by the diagonal. The behavior is similar for either manual patterns as well as for systematic pattern generation. We interpret that for problem instances that are further away from the goal, the heuristics using pattern databases both outperform CEGAR. Also, when comparing the singleton PDB's for the corner cubies, or rather edge cubies, on the front face, we recognize high correlation. In Fig. 7.3 we see that neither can get ahead of its counterpart.
When using CEGAR, we find a lot of problem instances for which we do not have to do any expansions until the last f-layer, which denotes the heuristic to be perfect for these instances. However, as soon as we get to more complex instances, the number goes up and CEGAR is outperformed by PDBs, which have also shown higher coverage.

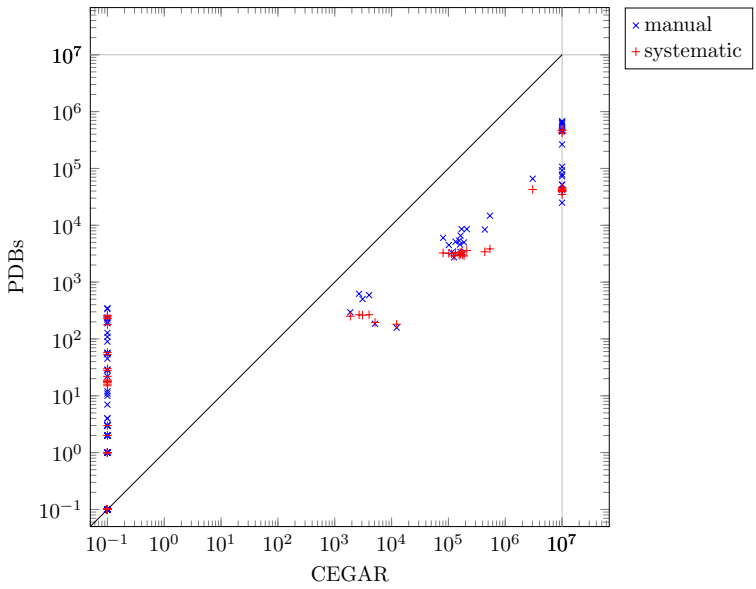

Figure 7.2: Expansions until the last f-layer for CEGAR vs. PDB heuristics on a loglog scale.

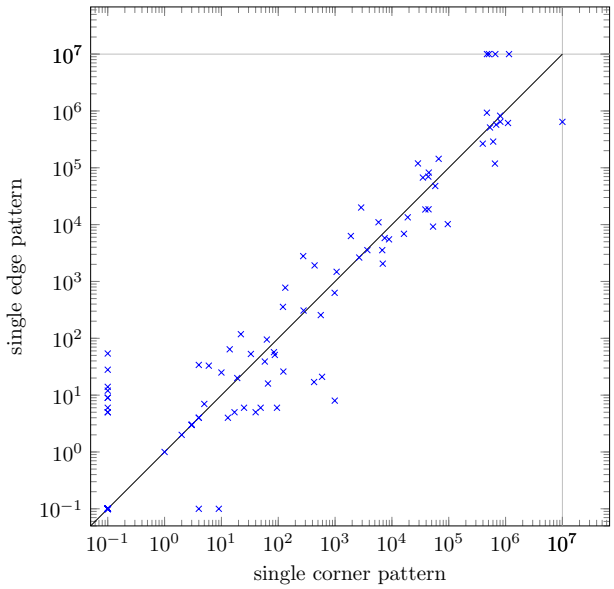

Figure 7.3: Expansions until the last f-layer for one pattern of corner cubies vs. one pattern for edge cubies on a loglog scale.

8 Conclusion

This chapter provides a summary of the work we have done within this thesis and also looks ahead of what else could be done in the same area.

8.1 Results

We have evaluated a range of different abstraction heuristics on Rubik's Cube. In order to do so, we formalized the state space with a focus on keeping it compact. The model chosen consists of a variable for each cubie and represents both its location as well as its rotation. The heuristics considered for our studies have in common that they abstract the state space spanned by the planning task of solving Rubik's Cube. We used three different classes of abstractions to do so: projection, Cartesian abstraction and merge-and-shrink abstraction. For evaluating these heuristics, we used Fast Downward [5] where we need to update the implementation for CEGAR since it does not support conditional effects in order to apply regression. Changes were only needed for the rewiring process where we have to distinguish whether it is possible given one abstract state to reach another via an operator. We introduced the function *possible* that computes all values a variable can have after applying an operator in an abstract state.
Within our evaluations we then found that CEGAR yields a perfect heuristic for a majority of the solved problem instances. We can see this by the number of expanded states until the last f-layer, which is zero for these instances. However, PDBs perform better on more challenging instances. The merge-and-shrink configuration, in turn, has shown a performance worse than expected: it finds the least solutions among all evaluated heuristics, has the lowest initial h-values and needs a lot of expansions until the last f-layer.

8.2 Future Work

The results presented in this thesis provide a baseline on how abstraction heuristics perform in Fast Downward [5]. The work could be continued in several ways.
For example, there is another class of abstraction heuristics that we have completely left out in our analysis. They are called *domain abstractions* and are placed between projections

and Cartesian abstraction in terms of generality. It could be interesting to see how they perform compared to our evaluated configurations.
Furthermore, we are aware of some efficiency issues in our implementations of CEGAR with conditional effects. Fixing these and rerunning our experiments could lead to different findings and provide further knowledge. We have already seen that it behaves perfectly for problem instances that are close to the goal. We assume that with more efficient code, we could reach problems further away from the goal without doing expansions until the last f-layer.

Bibliography

[1] (2018). Rubik's, the home of Rubik's Cube. https://eu.rubiks.com. Accessed: May 13, 2018.

[2] Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H. (2000). Counterexample-guided abstraction refinement. In *International Conference on Computer Aided Verification*, pages 154–169. Springer.

[3] Culberson, J. C. and Schaeffer, J. (1998). Pattern databases. *Computational Intelligence*, 14(3):318–334.

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

[5] Helmert, M. (2006). The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246.

[6] Helmert, M. (2018). Foundations of artificial intelligence. See online at http://informatik.unibas.ch/fs2018/lecture-foundations-of-artificial-intelligence/, Accessed: May 23, 2018.

[7] Helmert, M., Haslum, P., Hoffmann, J., et al. (2007). Flexible abstraction heuristics for optimal sequential planning. In *International Conference on Applied and Practical Sciences*, pages 176–183.

[8] Helmert, M., Haslum, P., Hoffmann, J., and Nissim, R. (2014). Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM (JACM)*, 61(3):16.

[9] Helmert, M. and Röger, G. (2017). Planning and optimization. See online at http://informatik.unibas.ch/hs2017/lecture-planning-and-optimization/, Accessed: May 27, 2018.

[10] Hoffmann, J. (2011). Everything you always wanted to know about planning. In *Annual Conference on Artificial Intelligence*, pages 1–13. Springer.

[11] Joyner, D. (2008). *Adventures in group theory: Rubik's Cube, Merlin's machine, and other mathematical toys*. Springer.

[12] Korf, R. E. (1997). Finding optimal solutions to Rubik's Cube using pattern databases. In *AAAI Conference on Artificial Intelligence*, pages 700–705.

[13] Pommerening, F., Röger, G., and Helmert, M. (2013). Getting the most out of pattern databases for classical planning. In *International Conference on Automated Planning and Scheduling*, pages 2357–2364.

[14] Rokicki, T., Kociemba, H., Davidson, M., and Dethridge, J. (2014). The diameter of the Rubik's Cube group is twenty. *SIAM Review*, 56(4):645–670.

[15] Seipp, J. and Helmert, M. (2018). Counterexample-guided Cartesian abstraction refinement for classical planning. *Journal of Artificial Intelligence Research*.

[16] Seipp, J., Pommerening, F., Sievers, S., and Helmert, M. (2017). Downward Lab. https://doi.org/10.5281/zenodo.790461.

A

Results of Final Experiments

The tables following on the next few pages display the most important results of the experiments done in this thesis. In order to keep table headers short, we introduce the following shortcuts for the experiments outlined in Section 7.1.

blind	blind search
man	maximum over manual patterns
syst	maximum over systematic patterns
corner	single pattern of corner cubies
edge	single pattern of edge cubies
cegar	single Cartesian abstraction heuristic
m&s	merge-and-shrink heuristic

Table A.1 shows the initial h-values for all problem instances. Table A.2 does the same for the number of expanded states before the last f-layer to find the goal.

turns	problem	blind	man	syst	corner	edge	cegar	m&s
1	p_0	1	1	1	1	1	1	1
1	p_1	1	1	1	1	1	1	1
1	p_2	1	1	1	1	1	1	1
1	p_3	1	1	1	1	1	1	1
1	p_4	1	1	1	1	1	1	1
1	p_5	1	1	1	1	1	1	1
1	p_6	1	1	1	1	1	1	1
1	p_7	1	1	1	1	1	1	1
1	p_8	1	1	1	1	1	1	1
1	p_9	1	1	1	0	0	1	1
2	p_0	1	2	2	2	2	2	2
2	p_1	1	2	2	2	2	2	2
2	p_2	1	2	2	2	2	2	2
2	p_3	1	2	2	2	2	2	2
2	p_4	1	2	2	2	2	2	2
2	p_5	1	2	2	1	1	2	2
2	p_6	1	2	2	2	2	2	2
2	p_7	1	2	2	2	2	2	2
2	p_8	1	2	2	2	2	2	2
2	p_9	1	2	2	2	2	2	2
3	p_0	1	3	3	3	2	3	3
3	p_1	1	3	3	3	2	3	3
3	p_2	1	2	3	2	2	3	3
3	p_3	1	3	3	3	3	3	3
3	p_4	1	2	3	2	2	3	3
3	p_5	1	3	3	3	3	3	3
3	p_6	1	2	3	2	2	3	3
3	p_7	1	3	3	3	3	3	3
3	p_8	1	3	3	3	3	3	3
3	p_9	1	3	3	2	2	3	3
4	p_0	1	4	4	3	4	4	3
4	p_1	1	4	4	4	2	4	3
4	p_2	1	4	4	4	1	4	3
4	p_3	1	4	4	3	2	4	3
4	p_4	1	3	4	3	3	4	3
4	p_5	1	4	4	3	3	4	3
4	p_6	1	4	4	3	3	4	3
4	p_7	1	4	4	4	2	4	3
4	p_8	1	3	4	3	3	4	3
4	p_9	1	4	4	3	4	4	3
5	p_0	1	5	5	5	4	5	4
5	p_1	1	4	5	3	3	5	3
5	p_2	1	5	5	2	4	5	3
5	p_3	1	5	5	5	5	5	4
5	p_4	1	5	5	5	4	5	3
5	p_5	1	5	5	4	4	5	3
5	p_6	1	4	5	3	4	5	3
5	p_7	1	5	5	4	3	5	3
5	p_8	1	3	5	3	3	5	3
5	p_9	1	5	5	2	3	5	3

Table A.1: Initial h-values computed for our problems, part 1.

turns	problem	blind	man	syst	corner	edge	cegar	m&s
6	p_0	1	6	6	4	5	6	4
6	p_1	1	5	6	5	5	6	3
6	p_2	1	5	5	4	5	6	3
6	p_3	1	4	4	2	3	4	3
6	p_4	1	5	5	4	5	6	4
6	p_5	1	6	6	5	5	6	4
6	p_6	1	5	5	5	5	6	4
6	p_7	1	5	6	5	5	6	4
6	p_8	1	5	6	4	4	6	4
6	p_9	1	4	4	4	4	4	4
7	p_0	1	6	6	5	5	7	4
7	p_1	1	7	6	5	4	7	4
7	p_2	1	4	4	4	3	4	3
7	p_3	1	6	6	6	5	7	4
7	p_4	1	5	6	5	5	7	4
7	p_5	1	4	5	4	4	5	4
7	p_6	1	5	5	5	4	7	4
7	p_7	1	6	6	6	5	7	4
7	p_8	1	6	6	4	6	7	3
7	p_9	1	6	6	6	5	7	3
8	p_0	1	6	6	6	6	8	4
8	p_1	1	6	6	6	5	8	4
8	p_2	1	6	5	6	5	8	3
8	p_3	1	5	6	3	4	8	3
8	p_4	1	5	5	5	4	5	3
8	p_5	1	6	6	5	5	8	4
8	p_6	1	5	6	5	5	8	4
8	p_7	1	7	6	5	5	8	3
8	p_8	1	6	5	4	5	8	4
8	p_9	1	5	6	4	4	8	3
9	p_0	1	5	6	5	5	8	3
9	p_1	1	6	6	4	6	7	3
9	p_2	1	6	6	5	5	9	4
9	p_3	1	7	6	5	6	8	4
9	p_4	1	6	6	5	5	9	4
9	p_5	1	6	6	6	5	9	4
9	p_6	1	6	6	6	5	8	3
9	p_7	1	6	6	6	6	8	3
9	p_8	1	6	7	6	6	9	4
9	p_9	1	6	6	5	4	8	3
10	p_0	1	4	6	4	3	7	3
10	p_1	1	6	6	5	5	8	3
10	p_2	1	6	6	6	5	9	3
10	p_3	1	6	6	6	4	8	4
10	p_4	1	6	6	5	6	8	4
10	p_5	1	6	6	5	5	8	3
10	p_6	1	6	6	6	5	9	4
10	p_7	1	6	6	5	6	8	3
10	p_8	1	6	6	6	5	9	3
10	p_9	1	6	6	6	6	8	3

Table A.1: Initial h-values computed for our problems, part 2.

turns	problem	blind	man	syst	corner	edge	cegar	m&s
11	p_0	1	5	5	4	5	8	4
11	p_1	1	7	6	5	6	8	4
11	p_2	1	6	6	3	4	8	3
11	p_3	1	5	6	4	4	8	3
11	p_4	1	6	6	4	5	8	3
11	p_5	1	6	6	6	6	9	4
11	p_6	1	5	6	5	5	8	3
11	p_7	1	6	6	6	5	9	4
11	p_8	1	6	6	5	5	9	4
11	p_9	1	7	6	4	5	8	4
12	p_0	1	7	6	5	7	8	4
12	p_1	1	6	6	6	5	8	3
12	p_2	1	5	6	5	5	8	3
12	p_3	1	6	6	4	6	8	3
12	p_4	1	5	6	5	3	8	3
12	p_5	1	7	6	4	7	8	4
12	p_6	1	6	6	3	6	8	3
12	p_7	1	6	6	4	6	8	3
12	p_8	1	6	6	5	6	9	4
12	p_9	1	6	6	5	6	8	3
13	p_0	1	6	6	5	6	8	4
13	p_1	1	7	6	6	5	8	4
13	p_2	1	6	6	6	3	8	3
13	p_3	1	6	7	6	5	9	4
13	p_4	1	6	6	6	6	8	3
13	p_5	1	7	6	6	6	8	3
13	p_6	1	6	7	6	5	9	4
13	p_7	1	6	6	3	6	8	3
13	p_8	1	6	6	3	6	8	3
13	p_9	1	7	7	5	6	8	4
14	p_0	1	5	6	5	5	8	3
14	p_1	1	6	6	5	6	8	3
14	p_2	1	6	7	5	5	8	4
14	p_3	1	7	6	6	5	9	4
14	p_4	1	7	6	6	5	9	4
14	p_5	1	7	6	6	7	9	4
14	p_6	1	6	6	5	6	8	3
14	p_7	1	6	6	5	6	8	4
14	p_8	1	7	6	5	5	8	3
14	p_9	1	6	6	4	6	8	4
15	p_0	1	6	6	5	6	8	4
15	p_1	1	7	7	6	6	8	4
15	p_2	1	5	6	5	5	8	3
15	p_3	1	6	7	6	6	8	4
15	p_4	1	6	6	4	5	8	3
15	p_5	1	7	7	5	7	8	4
15	p_6	1	7	6	4	7	8	4
15	p_7	1	6	6	6	4	9	4
15	p_8	1	7	6	4	6	8	3
15	p_9	1	5	6	5	4	8	4

Table A.1: Initial h-values computed for our problems, part 3.

turns	problem	blind	man	syst	corner	edge	cegar	m&s
16	p_0	1	6	7	5	6	8	4
16	p_1	1	6	6	5	6	8	4
16	p_2	1	6	6	5	5	8	4
16	p_3	1	7	6	6	7	8	4
16	p_4	1	6	6	6	5	9	4
16	p_5	1	7	7	6	5	9	4
16	p_6	1	6	6	5	6	8	4
16	p_7	1	7	6	5	4	8	4
16	p_8	1	6	6	6	6	9	3
16	p_9	1	6	6	5	6	8	3
17	p_0	1	7	7	5	7	8	3
17	p_1	1	7	6	6	6	9	4
17	p_2	1	6	6	4	5	8	3
17	p_3	1	7	6	5	5	8	4
17	p_4	1	6	6	4	5	8	4
17	p_5	1	6	6	5	6	8	4
17	p_6	1	6	6	6	6	9	4
17	p_7	1	6	6	6	6	9	3
17	p_8	1	6	6	6	5	9	4
17	p_9	1	7	6	4	7	8	4
18	p_0	1	6	6	6	3	9	4
18	p_1	1	6	6	6	6	9	4
18	p_2	1	7	6	5	7	8	4
18	p_3	1	5	6	4	4	8	3
18	p_4	1	7	7	6	7	8	4
18	p_5	1	6	6	6	5	9	3
18	p_6	1	7	7	6	6	9	3
18	p_7	1	6	7	6	6	9	4
18	p_8	1	5	6	5	5	8	3
18	p_9	1	7	6	5	7	8	3
19	p_0	1	6	6	5	5	8	3
19	p_1	1	6	7	5	6	8	4
19	p_2	1	7	7	7	6	9	4
19	p_3	1	7	6	6	5	9	4
19	p_4	1	6	6	6	6	9	3
19	p_5	1	7	6	7	7	9	4
19	p_6	1	6	6	4	6	8	3
19	p_7	1	6	6	5	6	8	3
19	p_8	1	6	6	5	6	8	3
19	p_9	1	7	6	6	3	8	4
20	p_0	1	6	6	5	6	8	4
20	p_1	1	6	6	5	5	8	3
20	p_2	1	6	7	6	5	9	3
20	p_3	1	6	7	5	6	8	4
20	p_4	1	7	6	5	6	8	4
20	p_5	1	6	6	6	6	9	3
20	p_6	1	6	6	5	6	9	4
20	p_7	1	6	6	6	5	9	4
20	p_8	1	7	7	5	7	8	3
20	p_9	1	6	6	5	5	8	3

Table A.1: Initial h-values computed for our problems, part 4.

turns	problem	blind	man	syst	corner	edge	cegar	m&s
1	p_0	0	0	0	0	0	0	0
1	p_1	0	0	0	0	0	0	0
1	p_2	0	0	0	0	0	0	0
1	p_3	0	0	0	0	0	0	0
1	p_4	0	0	0	0	0	0	0
1	p_5	0	0	0	0	0	0	0
1	p_6	0	0	0	0	0	0	0
1	p_7	0	0	0	0	0	0	0
1	p_8	0	0	0	0	0	0	0
1	p_9	0	0	0	1	1	0	0
2	p_0	1	0	0	0	0	0	0
2	p_1	1	0	0	0	0	0	0
2	p_2	1	0	0	0	0	0	0
2	p_3	1	0	0	0	0	0	0
2	p_4	1	0	0	0	0	0	0
2	p_5	1	0	0	2	2	0	0
2	p_6	1	0	0	0	0	0	0
2	p_7	1	0	0	0	0	0	0
2	p_8	1	0	0	0	0	0	0
2	p_9	1	0	0	0	0	0	0
3	p_0	19	0	0	0	5	0	0
3	p_1	19	0	0	0	5	0	0
3	p_2	19	2	0	3	3	0	0
3	p_3	19	0	0	0	0	0	0
3	p_4	19	1	0	3	3	0	0
3	p_5	19	0	0	0	0	0	0
3	p_6	19	2	0	3	3	0	0
3	p_7	19	0	0	0	0	0	0
3	p_8	19	0	0	0	0	0	0
3	p_9	19	0	0	3	3	0	0
4	p_0	262	0	0	4	0	0	1
4	p_1	262	0	0	0	28	0	1
4	p_2	262	0	0	0	54	0	1
4	p_3	262	0	0	4	34	0	1
4	p_4	262	4	0	17	5	0	1
4	p_5	262	0	0	5	7	0	1
4	p_6	262	0	0	4	4	0	1
4	p_7	262	0	0	0	14	0	1
4	p_8	262	1	0	4	4	0	1
4	p_9	262	0	0	9	0	0	1
5	p_0	3502	0	0	0	9	0	2
5	p_1	3502	1	0	33	53	0	13
5	p_2	3502	0	0	95	6	0	18
5	p_3	3502	0	0	0	0	0	2
5	p_4	3502	0	0	0	12	0	11
5	p_5	3502	0	0	49	6	0	16
5	p_6	3502	1	0	40	5	0	13
5	p_7	3502	0	0	14	64	0	12
5	p_8	3502	7	0	87	51	0	15
5	p_9	3502	0	0	84	57	0	19

Table A.2: Number of expanded states until last f-layer, part 1.

turns	problem	blind	man	syst	corner	edge	cegar	m&s
6	p_0	46741	0	0	427	17	0	171
6	p_1	46741	1	0	19	20	0	125
6	p_2	46741	2	1	66	16	0	159
6	p_3	262	0	0	13	4	0	1
6	p_4	46741	4	1	123	26	0	162
6	p_5	46741	0	0	58	39	0	186
6	p_6	46741	3	1	10	25	0	125
6	p_7	46741	1	0	6	33	0	95
6	p_8	46741	2	0	63	95	0	192
6	p_9	262	0	0	0	0	0	0
7	p_0	-	3	1	279	308	0	1858
7	p_1	-	0	1	982	631	0	1710
7	p_2	262	0	0	0	6	0	1
7	p_3	621649	1	2	121	357	0	1573
7	p_4	621649	11	2	560	256	0	1793
7	p_5	3502	2	0	25	6	0	11
7	p_6	-	45	15	275	2791	0	1770
7	p_7	621649	1	1	22	118	0	1564
7	p_8	-	2	1	590	21	0	1900
7	p_9	-	3	1	133	777	0	1817
8	p_0	-	12	18	1064	1479	0	21253
8	p_1	-	22	16	2660	2635	0	23694
8	p_2	-	54	53	1895	6300	0	25708
8	p_3	-	109	18	16291	6891	0	29599
8	p_4	3502	0	0	0	9	0	13
8	p_5	-	28	19	6759	3552	0	25148
8	p_6	-	57	17	3700	3557	0	25607
8	p_7	-	10	18	6903	2056	0	25094
8	p_8	-	128	28	7458	5802	0	24857
8	p_9	-	90	22	5810	11003	0	25358
9	p_0	-	624	266	57897	47820	2679	334127
9	p_1	621649	3	1	987	8	0	2018
9	p_2	-	336	235	44534	82312	0	331620
9	p_3	-	158	181	96079	10198	12281	337013
9	p_4	-	199	227	28718	119293	0	306260
9	p_5	-	238	251	35060	67285	0	312170
9	p_6	-	295	251	43734	69930	1877	325420
9	p_7	-	503	262	44115	18614	3083	346482
9	p_8	-	189	178	18987	13438	0	317322
9	p_9	-	590	267	66453	143833	4018	327089
10	p_0	621649	29	3	436	1916	0	2057
10	p_1	-	4581	3279	801478	823975	160173	-
10	p_2	-	2730	2888	396664	265510	125787	-
10	p_3	-	57	30	2879	19996	0	22338
10	p_4	-	3141	3028	649535	118553	175686	-
10	p_5	-	6576	3113	660254	-	163346	-
10	p_6	-	3336	3175	469899	-	113988	-
10	p_7	-	5144	3337	532140	511765	133556	-
10	p_8	-	6021	3255	469114	931033	80724	-
10	p_9	-	3406	3280	605098	288976	163086	-

Table A.2: Number of expanded states until last f-layer, part 2.

turns	problem	blind	man	syst	corner	edge	cegar	m&s
11	p_0	-	220	58	8896	5505	0	24331
11	p_1	-	185	195	52936	9229	5110	339528
11	p_2	-	52061	40634	-	-	-	-
11	p_3	-	107135	46206	-	-	-	-
11	p_4	-	78290	43905	-	-	-	-
11	p_5	-	348	260	38743	18464	0	328611
11	p_6	-	8568	3417	-	642863	170888	-
11	p_7	-	51344	39776	-	-	-	-
11	p_8	-	4495	3200	509300	-	101737	-
11	p_9	-	24969	34911	-	-	-	-
12	p_0	-	264838	422388	-	-	-	-
12	p_1	-	5381	2953	-	-	154561	-
12	p_2	-	72654	41225	-	-	-	-
12	p_3	-	8425	3422	1103345	614099	436403	-
12	p_4	-	92674	42822	-	-	-	-
12	p_5	-	648582	-	-	-	-	-
12	p_6	-	504767	470133	-	-	-	-
12	p_7	-	465130	472772	-	-	-	-
12	p_8	-	461938	-	-	-	-	-
12	p_9	-	593546	-	-	-	-	-
13	p_0	-	65972	42606	-	-	3037235	-
13	p_1	-	-	-	-	-	-	-
13	p_2	-	-	-	-	-	-	-
13	p_3	-	42149	39036	-	-	-	-
13	p_4	-	-	-	-	-	-	-
13	p_5	-	-	-	-	-	-	-
13	p_6	-	-	-	-	-	-	-
13	p_7	-	-	-	-	-	-	-
13	p_8	-	678664	-	-	-	-	-
13	p_9	-	-	-	-	-	-	-
14	p_0	-	5001	2890	683755	573371	188261	-
14	p_1	-	632243	-	-	-	-	-
14	p_2	-	-	-	-	-	-	-
14	p_3	-	-	-	-	-	-	-
14	p_4	-	-	-	-	-	-	-
14	p_5	-	-	-	-	-	-	-
14	p_6	-	446262	-	-	-	-	-
14	p_7	-	-	-	-	-	-	-
14	p_8	-	-	-	-	-	-	-
14	p_9	-	8574	3596	808694	646654	210704	-
15	p_0	-	-	-	-	-	-	-
15	p_1	-	-	-	-	-	-	-
15	p_2	-	-	-	-	-	-	-
15	p_3	-	-	-	-	-	-	-
15	p_4	-	-	-	-	-	-	-
15	p_5	-	-	-	-	-	-	-
15	p_6	-	-	-	-	-	-	-
15	p_7	-	-	-	-	-	-	-
15	p_8	-	-	-	-	-	-	-
15	p_9	-	-	-	-	-	-	-

Table A.2: Number of expanded states until last f-layer, part 3.

turns	problem	blind	man	syst	corner	edge	cegar	m&s
16	p_0	-	-	-	-	-	-	-
16	p_1	-	-	-	-	-	-	-
16	p_2	-	-	-	-	-	-	-
16	p_3	-	-	-	-	-	-	-
16	p_4	-	-	-	-	-	-	-
16	p_5	-	-	-	-	-	-	-
16	p_6	-	-	-	-	-	-	-
16	p_7	-	-	-	-	-	-	-
16	p_8	-	-	-	-	-	-	-
16	p_9	-	-	-	-	-	-	-
17	p_0	-	-	-	-	-	-	-
17	p_1	-	-	-	-	-	-	-
17	p_2	-	-	-	-	-	-	-
17	p_3	-	-	-	-	-	-	-
17	p_4	-	-	-	-	-	-	-
17	p_5	-	-	-	-	-	-	-
17	p_6	-	-	-	-	-	-	-
17	p_7	-	-	-	-	-	-	-
17	p_8	-	-	-	-	-	-	-
17	p_9	-	-	-	-	-	-	-
18	p_0	-	-	-	-	-	-	-
18	p_1	-	-	-	-	-	-	-
18	p_2	-	-	-	-	-	-	-
18	p_3	-	14721	3839	1149696	-	537668	-
18	p_4	-	-	-	-	-	-	-
18	p_5	-	-	-	-	-	-	-
18	p_6	-	-	-	-	-	-	-
18	p_7	-	-	-	-	-	-	-
18	p_8	-	-	-	-	-	-	-
18	p_9	-	-	-	-	-	-	-
19	p_0	-	-	-	-	-	-	-
19	p_1	-	-	-	-	-	-	-
19	p_2	-	-	-	-	-	-	-
19	p_3	-	-	-	-	-	-	-
19	p_4	-	-	-	-	-	-	-
19	p_5	-	-	-	-	-	-	-
19	p_6	-	-	-	-	-	-	-
19	p_7	-	-	-	-	-	-	-
19	p_8	-	-	-	-	-	-	-
19	p_9	-	-	-	-	-	-	-
20	p_0	-	-	-	-	-	-	-
20	p_1	-	-	-	-	-	-	-
20	p_2	-	-	-	-	-	-	-
20	p_3	-	-	-	-	-	-	-
20	p_4	-	-	-	-	-	-	-
20	p_5	-	-	-	-	-	-	-
20	p_6	-	-	-	-	-	-	-
20	p_7	-	-	-	-	-	-	-
20	p_8	-	-	-	-	-	-	-
20	p_9	-	-	-	-	-	-	-

Table A.2: Number of expanded states until last f-layer, part 4.

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

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

Author — Autor
Clemens Büchner

Matriculation number — Matrikelnummer
15-059-603

Title of work — Titel der Arbeit
Abstraction Heuristics for Rubik's Cube

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, June 2, 2018

Signature — Unterschrift