

To Merge or to Cost Partition?

Master's Thesis

Faculty of Science - The University of Basel
Department of Mathematics and Computer Science
Artificial Intelligence Research Group
<https://ai.dmi.unibas.ch/>

Examiner: Prof. Dr. Malte Helmert
Supervisor: Dr. Thomas Keller

Miruna-Alesia Muntean
miruna.muntean@unibas.ch
21-062-906

20/05/2023

Acknowledgments

First of all, I would like to thank Prof. Dr. Malte Helmert and Dr. Gabriele Röger for giving me the opportunity to study the topic of this thesis as part of the Artificial Intelligence research group. Additionally, I would like to thank Dr. Thomas Keller for guiding me throughout the process of this thesis with useful advice and valuable assistance. More, I want to thank my friends and family for being there for me at each step of this project with continuous support and understanding.

Abstract

This thesis aims to present a novel approach for improving the performance of classical planning algorithms by integrating cost partitioning with merge-and-shrink techniques. Cost partitioning is a well-known technique for admissibly adding multiple heuristic values. Merge-and-shrink, on the other hand, is a technique to generate well-informed abstractions. The "merge" part of the technique is based on creating an abstract representation of the original problem by replacing two transition systems with their synchronised product. In contrast, the "shrink" part refers to reducing the size of the factor. By combining these two approaches, we aim to leverage the strengths of both methods to achieve better scalability and efficiency in solving classical planning problems. Considering a range of benchmark domains and the Fast Downward planning system, the experimental results show that the proposed method achieves the goal of fusing merge and shrink with cost partitioning towards better outcomes in classical planning.

List of Figures

2.1	Atomic Projection of Variable a	6
2.2	Atomic Projection of Variable b	6
2.3	Merging Example	7
2.4	Pruning Example	8
2.5	Shrinking Example	8
2.6	Cost Partitioning for a with order (a, b)	10
2.7	Cost Partitioning for b with order (a, b)	10
2.8	Cost Partitioning for b with order (b, a)	11
2.9	Cost Partitioning for a with order (b, a)	11
3.1	Cost Partitioning Example in Figure 2.3	13
3.2	Merging of Examples in Figures 2.9 and 2.8	14
3.3	The Mechanics of The Algorithm	19
4.1	Baseline - Coverage Analysis	21
4.2	Baseline - Number of Merges Analysis	21
4.3	Merge and Prune - Coverage Analysis	22
4.4	Merge and Prune - Number of Merges Analysis	23
4.5	Merge and Prune - Expansions	24
4.6	Merge, Prune and Shrink - Coverage Analysis	25
4.7	Merge, Prune and Shrink - Number of Merges Analysis	25
4.8	Merge, Prune and Shrink - Expansions	26
4.9	Versus Cost Partitioning - Coverage	28
4.10	Versus Cost Partitioning - Number of Merges	29
4.11	Versus Cost Partitioning - Expansions	29
4.12	Versus Merge-and-Shrink - Coverage	30
4.13	Versus Merge-and-Shrink - Number of Merges	31
4.14	Versus Merge-and-Shrink - Expansions	31
4.15	Versus Sievers et al [11] - Coverage	32
5.1	Gantt Chart of Project Timetable	34
A.1	Benchmarks	39

List of Tables

4.1	Testing Larger Merging Thresholds	26
4.2	Testing Size in Quality Calculation	27
4.3	Testing Maximum Occurrences of Minimal Quality	28

Table of Contents

Acknowledgments	ii
Abstract	iii
List of Figures	iv
List of Tables	v
1 Introduction	1
1.1 Objectives and Thesis Outline	1
1.2 Aims and Motivation	2
2 Background Research	3
2.1 Key Concepts	3
2.2 Merge and Shrink	5
2.3 Cost Partitioning	9
3 Implementation and Development	12
3.1 Comparing Merge-and-Shrink with Cost Partitioning	12
3.2 Combining Merge-and-Shrink with Cost Partitioning	15
3.2.1 The Baseline Algorithm	16
3.2.2 Improvements to the Baseline	18
3.2.2.1 Pruning Unreachable and Irrelevant States	18
3.2.2.2 Shrinking	18
3.2.2.3 Flexible Quality and Merging Threshold	19
4 Experimental Analysis	20
4.1 The Baseline	20
4.2 Adding Pruning	22
4.3 Adding Shrinking	24
4.4 Quality Analysis	27
4.5 Comparing to Cost Partitioning	28
4.6 Comparing to Merge-and-Shrink	30
4.7 Comparing to Sievers et al Paper [11]	32

5 Conclusion	34
5.1 Reflection	34
5.1.1 Future Work and Further Improvements	35
5.2 Conclusion	36
Bibliography	37
Appendix A	39

1

Introduction

Under the umbrella of the vast realm of artificial intelligence, a planning problem can be expressed as one in which we have an initial starting state, which we wish to transform into a desired goal state by applying a sequence of actions.

'Planning is the art and practice of thinking before acting.'

- Patrick Haslum

On a more technical side, in the field of artificial intelligence, planning can be informally defined as the activity of coming up with a sequence of actions aiming to accomplish the target. Throughout the course of this study, classical planning will be used, which involves an environment that is fully observable, deterministic, static and discrete.

1.1 Objectives and Thesis Outline

The main objective of this thesis is to successfully combine merge-and-shrink heuristics with cost partitioning. This thesis paper aims to gain better knowledge regarding merge-and-shrink heuristics and cost partitioning techniques. While merge-and-shrink is used to replace two systems with their product [8], cost partitioning implies combining heuristics [2]. We eventually care about their influence on each other. This interaction of the two concepts has been previously studied by Sievers et al. [11], but so far, there is no combination of them per se. By the end, we should be a few steps closer to effectively making the right choice vis-a-vis merging, based on cost partitioned heuristics, to boost the performance and results. This is done by putting the decision whether to continue merging in the hands of a quality value, calculated based on the weight merging brings compared to simply cost partitioning. This process takes place at every iteration through the set of transition systems until there is no pair whose merge quality value exceeds the minimal allowed quality. At each iteration, the pair with the largest quality is eventually merged, and the process continues until no pair brings more information. Therefore, this thesis document introduces the key concepts that form the background research for the thesis, discussions regarding the matter of combining the two notions both theoretically and experimentally, as well as the baseline algorithm with additional improvements.

1.2 Aims and Motivation

As the previous section suggests, the primary motivation for this thesis lies in the basis of the previously studied concepts by Sievers et al. [11]. The paper is the first that aims to combine merge-and-shrink and cost partitioning. The authors propose a new method that extends the idea of cost partitioning to the concept of merge-and-shrink abstractions, investigating the results under both optimal and saturated cost partitioning. This is done using standard planning benchmarks, used in this thesis as well. The issue lies in the purpose of both the relevant literature and this thesis. Abstractions sometimes have helpful information for some parts of a planning task and totally ignore the remaining. Therefore, the operators' costs are distributed among the atomic projections, assuring that the sum of their heuristics remains admissible. For this, they are introducing label cost partitioning, transferring the notion of operator cost partitioning to merge-and-shrink abstractions. The approach suggests computing the cost partitioning heuristics over the transition systems, taking a single snapshot of each iteration of the merge-and-shrink algorithm. After, all the values at each iteration are experimentally evaluated. A drawback of this approach is that the two concepts are used one next to the other rather than actually influencing each other in the form of a combination. This thesis comes in here, attempting a different idea of integrating merge-and-shrink and cost partitioning. A more in-depth look at the paper as its comparison with this thesis will occur experimentally in Chapter 4.

The next chapter will present the background research, introducing and defining the key concepts needed for this thesis.

2

Background Research

The following chapter is aiming to give an insightful overview of the research that has been done for this project, including information on the domains of interest.

2.1 Key Concepts

Before diving deeper into the specifics of this paper's topic, there are a few concepts and ideas that revolve around the final goal. The subsequently discussed notions are going to be frequently used throughout the thesis.

Transition System

At the very base, we have the term of *transition system*, which is defined as:

Definition 1 (Transition System). *A transition system is a 6-tuple $\mathcal{T} = \langle S, L, c, T, s_0, S_* \rangle$:*

- S is a finite set of states,
- L is a finite set of (transition) labels,
- $c : L \rightarrow \mathbb{R}$ is a label cost function,
- $T \subseteq S \times L \times S$ is the transition relation,
- $s_0 \in S$ is the initial state,
- $S_* \subseteq S$ is the set of goal states.

We say that \mathcal{T} has the transition $\langle s, l, s' \rangle$ if $\langle s, l, s' \rangle \in T$. We also write this as $s \xrightarrow{l} s'$, or $s \rightarrow s'$ when not interested in l .

We can further define a **plan** π as a sequence of labels that can be followed to get from the initial state to a goal state. Moreover, the **cost of a plan** is the total value of the cost function c over the sequence of labels that took place from the initial state to the goal, namely the plan. Additionally, an **optimal plan** is a plan that has minimum cost among all possible plans achieving a goal state.

Planning Task

One of the most important concepts for this thesis is the *planning task*. In the previous chapter, this idea was briefly introduced as a problem in which we have some initial starting state, which we wish to transform into a desired goal state by applying a sequence of actions.

Definition 2 (Planning Task). *A planning task is a 4-tuple $\Pi = \langle V, I, O, \gamma \rangle$, where:*

- V is a finite set of state variables associated with a domain $\text{dom}(v)$ for all $v \in V$,
- I is a total assignment over V called the initial state,
- O is a finite set of operators over V with $o = \langle \text{pre}(o), \text{eff}(o), \text{cost}(o) \rangle$ for all $o \in O$, where $\text{pre}(o)$ and $\text{eff}(o)$ are partial assignments over V , and $\text{cost}(o) \in \mathbb{R}_0^+$ is the cost,
- γ is a partial assignment over V called the goal.

V must either consist only of propositional or only of finite-domain state variables. In our case, a finite-domain representation will be considered.

Transition System Induced by Planning Task

What we are going actually to work with are transition systems induced by planning tasks, as the transitions are defined by the application of specific operators to certain states, over a finite set of state variables.

Definition 3 (Transition System Induced by Planning Task). *A planning task $\Pi = \langle V, I, O, \gamma \rangle$ induces the transition system $\mathcal{T}(\Pi) = \langle S, L, c, T, s_0, S_* \rangle$, where:*

- S is the set of all states over V ,
- L is the set of operators O ,
- $c(o) = \text{cost}(o)$ for all operators $o \in O$,
- $T = \{ \langle s, o, s' \rangle \mid s \in S, \text{pre}(o) \subseteq s, s' = s[o] \}$,
- $s_0 = I$,
- $S_* = \{ s \in S \mid \gamma \subseteq s \}$.

We write $s[o]$ to denote the successor state that results from applying operator o in state s , with $s(v)$ being the value of v in state s :

$$s[o] = \begin{cases} d, & \text{if } v \rightarrow d \in \text{eff}(o) \\ s(v), & \text{otherwise} \end{cases} \quad (2.1)$$

Heuristics

In simple terms, heuristics refers to an instinctive approach to solving problems and making decisions that can produce results on their own or be used in more complex optimisation algorithms with the aim of improving efficiency.

Definition 4 (Heuristic). *Let s be a state in a transition system. A heuristic $h : S \rightarrow \mathbb{R}$ is a function $h(s)$ that maps a state to a real-valued number.*

The perfect heuristic defined as h^* maps each state s in the state space to the cost of an optimal plan. We say that a heuristic h is **admissible** if $h(s) \leq h^*(s)$ for all states s .

Due to our interest in the cost, we will use the notation $h : S \times cost \rightarrow \mathbb{R}$, $h(s, cost)$ for heuristic values. This denotes the value of the heuristic in a state s in a classical planning task with cost function $cost$.

Optimal Planning

As we are considering optimal classical planning tasks, the optimum of the plan is a requirement. Thus, we are solving the problem such that the plan offers a minimal cost.

Definition 5 (Optimal Planning). .

GIVEN a planning task Π

OUTPUT an optimal plan for Π , or unsolvable if no plan exists

Very popular for optimal planning is the **A* search algorithm**, being one of the best techniques for finding an optimal plan. It is a search algorithm that finds the shortest path between the initial and the goal state. The A* search algorithm acts at each step by considering a heuristic value, with all the heuristics sharing and maintaining a common property of being admissible [4]. As this thesis considers optimal planning, we will work fully with admissible heuristics. Moreover, both main concepts needed for this thesis, the merge and shrink method and cost partitioning, guarantee admissibility.

2.2 Merge and Shrink

Abstractions are one of the principal ways of deriving heuristics for planning tasks and transition systems. This notion can be defined as:

Definition 6 (Abstraction). *Let $\mathcal{T} = \langle S, L, c, T, s_0, S_* \rangle$ be a transition system. An abstraction of \mathcal{T} is a function $\alpha : S \rightarrow S^\alpha$ defined on the states of \mathcal{T} , where S^α is an arbitrary set. α induces the **abstract transition system** $\mathcal{T}^\alpha = \langle S^\alpha, L, c, T^\alpha, s_0^\alpha, S_*^\alpha \rangle$, where $\mathcal{T}^\alpha = \{ \langle \alpha(s), l, \alpha(s') \rangle \mid \text{for all } \langle s, l, s' \rangle \in \mathcal{T} \}$, $s_0^\alpha = \alpha(s_0)$ and $S_*^\alpha = \{ \alpha(s) \mid s \in S_* \}$.*

Definition 7 (Projection). *Let Π be a finite-domain representation planning task with variables V and states S . Let $P \subseteq V$, and let S_P be the set of states over P . The projection $\pi_P : S \rightarrow S_P$ is defined as $\pi_P(s) := s \upharpoonright_P$, where $s \upharpoonright_P (v) := s(v)$ for all $v \in P$.*

We call P the pattern of the projection π_P and denote the projection heuristic with h_P .

Atomic projections are a particular class of projections to a single state variable. They play a crucial role in merge-and-shrink abstractions, being the basis of the initial step. The process begins with computing the atomic projections of the transition systems.

An example can simply illustrate the aforementioned definitions and concepts. Let us consider the following planning task Π with:

- $V = \{a, b\}$, with $\text{dom}(a) = \{0, 1, 2\}$ and $\text{dom}(b) = \{0, 1\}$
- $I = \{a \rightarrow 0, b \rightarrow 0\}$
- $O = \{o_1, o_2, o_3\}$ with:
 - $o_1 = \langle \{a \rightarrow 0, b \rightarrow 0\}, \{a \rightarrow 1, b \rightarrow 1\} \rangle$
 - $o_2 = \langle \{b \rightarrow 1\}, \{b \rightarrow 0\} \rangle$
 - $o_3 = \langle \{a \rightarrow 0\}, \{a \rightarrow 2\} \rangle$
 - $o_4 = \langle \{a \rightarrow 1, b \rightarrow 0\}, \{a \rightarrow 2, b \rightarrow 1\} \rangle$
- $\gamma = \{a \rightarrow 2, b \rightarrow 1\}$

The atomic projections of a and b ($\pi_{\{a\}}$, respectively $\pi_{\{b\}}$) can be seen in Figures 2.1 and 2.2, marking the beginning step of the merge-and-shrink algorithm.

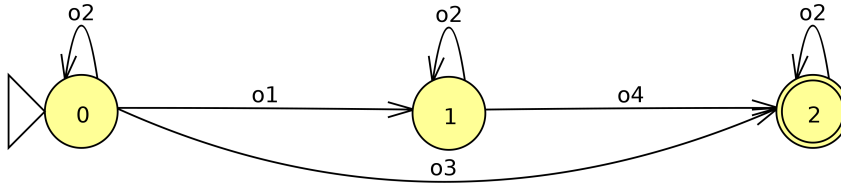


Figure 2.1: Atomic Projection of Variable a

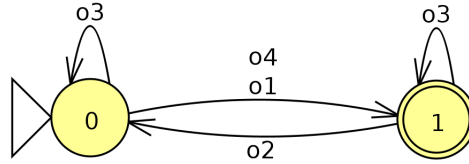


Figure 2.2: Atomic Projection of Variable b

Moving on, the synchronised product involves computing a product transition system of two abstractions. The result has the property of capturing all information from both systems.

Definition 8 (Synchronised Product of Transition Systems). For $i \in \{1, 2\}$, let $\mathcal{T}_i = \langle S^i, L, c, T^i, s_0^i, S_*^i \rangle$ be transition systems with the same labels and cost functions. The synchronised product of \mathcal{T}_1 and \mathcal{T}_2 , in symbols $\mathcal{T}_1 \otimes \mathcal{T}_2$, is the transition system $\mathcal{T}_\otimes = \langle S^\otimes, L, c, T^\otimes, s_0^\otimes, S_*^\otimes \rangle$, with:

- $S_\otimes = S_1 \times S_2$,
- $T_\otimes = \{ \langle \langle s_1, s_2 \rangle, o, \langle t_1, t_2 \rangle \rangle \mid s_1 \rightarrow t_1 \in T_1 \text{ and } s_2 \rightarrow t_2 \in T_2 \}$,
- $s_0^\otimes = \langle s_0^1, s_0^2 \rangle$,
- $S_*^\otimes = S_*^1 \times S_*^2$.

In the next part, we take each step of the merge-and-shrink algorithm:

Merge: Now, we calculate the synchronised product - the two projections are now merged into an abstraction. This case below is simply just an example of merging. We start with two transition systems and end up with a new system that is a product of them. We can introduce the process of merging as an action that "replaces two factors of the given factored transition system by their product system, leaving all other factors unchanged" [8]. When applying a sequence of linear merging transformations, each of the atomic factors contributes to exactly one factor in the resulting factored transition system.

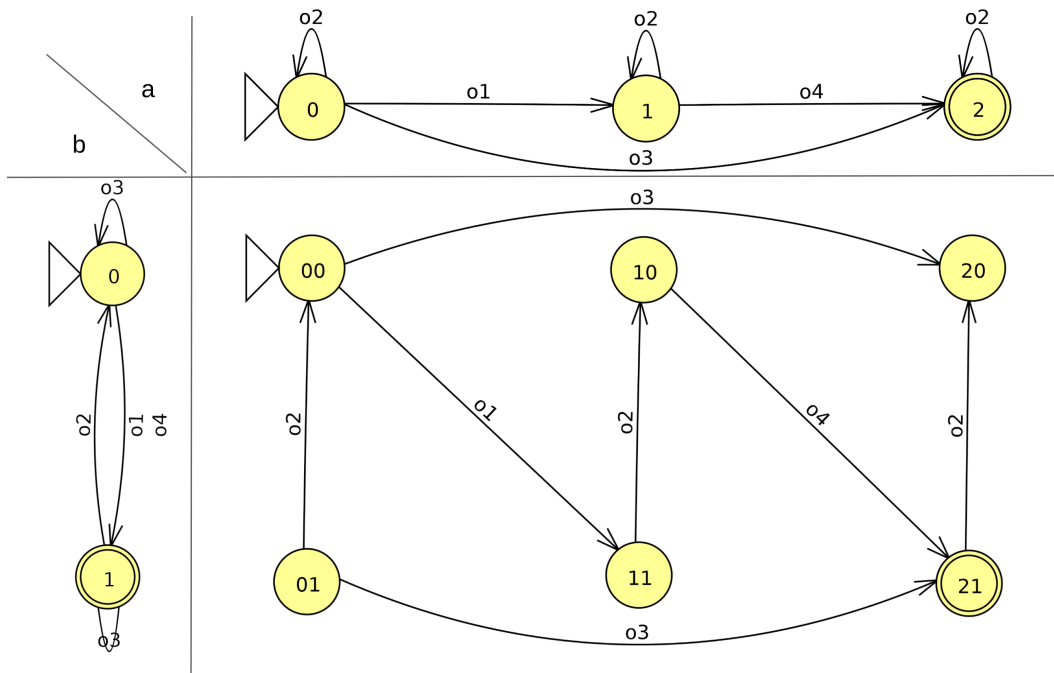


Figure 2.3: Merging Example

Prune: Subsequently, this is the process of removing unnecessary or redundant transitions to make the system more efficient. The goal of pruning is to reduce the number of transitions in the system while preserving its functionality [7]. More details regarding pruning are going to be discussed later in this thesis. The resulting product contains as states all pairs of the states from the atomic elements. In this presented case, we encounter unreachable (example state 01) and irrelevant (example state 20, which is a dead-end) states, which will be further handled and discussed by the end of the thesis, alongside the technical implementation. Below, the example illustrates the pruning of these irrelevant and unreachable states for the system in Figure 2.3. It removes two states, 01 and 20, and respectively the transitions related, under the reason that state 01 is unreachable and 20 is irrelevant as the goal state is not reachable from it.

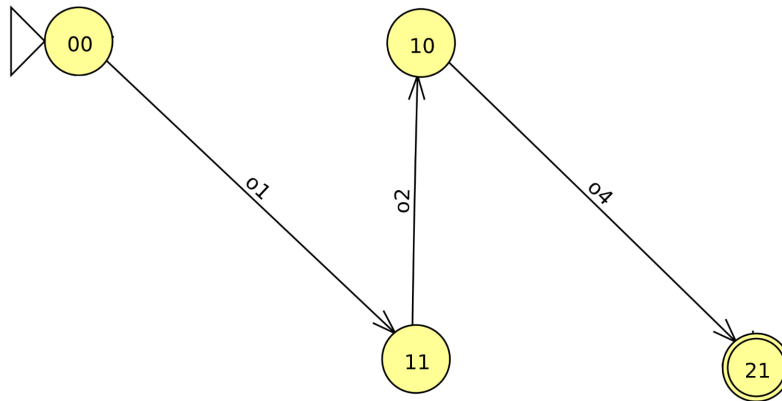


Figure 2.4: Pruning Example

Shrink: Next, we look further at the shrinking of the merged product above, illustrated in the following example. For this, we look at the merge example from Figure 2.3 and create abstract states by combining state 00 with state 20 and state 01 with state 21, as seen in states 0020 and 0121. Subsequently, shrinking refers to reducing the size of a single factor by abstraction [7]. So, we are replacing the factor with an abstraction of it. Making good shrinking decisions algorithmically is the job of the shrinking strategy.

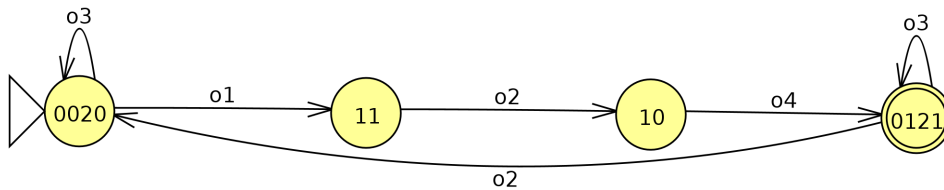


Figure 2.5: Shrinking Example

Label Reduction: Finally, the key idea involves recognising which transition labels can be combined into a single label while still retaining all relevant information. This approach offers various advantages, including the potential to greatly reduce the size of the transition system representation by consolidating parallel transitions with different labels into a single transition. This concept will not be a part of the implementation of this thesis, but its influence will be further discussed later as a future addition.

We can now understand the aim of merge-and-shrink, which is mainly to compute an abstraction for a large transition system, for instance, the one induced by a planning task. To do so, building the abstract transition system is needed. It is useful to note that the admissibility property of the heuristics is kept after doing any of the actions of merging, shrinking, pruning and label reducing.

2.3 Cost Partitioning

The idea of cost partitioning has its basis in admissibly combining admissible heuristics [2]. Classical planning problems often need the use of multiple heuristics to capture distinct areas of a problem. This is where cost partitioning comes in handy by splitting the cost between the heuristics with the aim of higher estimates.

Definition 9 (Cost Partitioning [11]). *Let Π be a planning task with operator costs \mathbf{cost} and let $\mathcal{H} = \langle h_1, \dots, h_n \rangle$ be admissible heuristics for Π . The cost functions $c = \langle cost_1, \dots, cost_n \rangle$ form a cost partition if for all $o \in O$, $\sum_{i=1}^n cost_i(o) \leq cost(o)$.*

The cost-partitioned heuristic is equal to $\sum_{i=1}^n h_i(s, cost_i)$ and it is admissible.

According to the paper by Sievers et al. [11], we can define optimal cost partitioning as:

Definition 10 (Optimal Cost Partitioning). *An optimal cost partitioning for a planning task Π with cost function \mathbf{cost} , a state s , and admissible heuristics $\mathcal{H} = \langle h_1, \dots, h_n \rangle$ is a cost partition $\mathcal{C}^* = \langle cost_1^*, \dots, cost_n^* \rangle$ where $\sum_{i=1}^n h_i(s, cost_i^*) \geq h_i(s, cost_i)$, for all $cost_i \in \mathcal{C}$, where $\mathcal{C} = \langle cost_1, \dots, cost_n \rangle$.*

Simply put, the optimal cost partitioning distributes the costs in an optimal way [2]. We get the largest possible heuristic value for the given state among all cost partitioning by computing the optimal cost partitioning for a state. Optimal partitioning dominates other types of cost partitioning.

However good the optimal cost partitioning is and even if sometimes it is possible to calculate it in polynomial time, it can get very expensive to calculate in practice. Thus, this thesis uses saturated cost partitioning as default, offering the best trade-off between computation time and heuristic guidance.

Definition 11 (Saturated Cost Function [6]). *A cost function \mathbf{scf} is saturated for a heuristic h , an original cost function \mathbf{cost} and a set S of states in the planning task, if*

- $scf(o) \leq cost(o)$ for all operators o ,
- $h(s, scf) = h(s, cost)$, for all states $s \in S$

Saturated cost partitioning is a greedy algorithm that can compute a sub-optimal cost partitioning quickly. In the case of saturated cost partitioning, the idea of order is at the centre. Hence, the quality of the resulting partitioning depends on the selected order. This study will consider all orders and choose the one that offers the best value in the end.

Definition 12 (Saturated Cost Partitioning). *A saturated cost partitioning of a planning task Π is a cost partitioning that uses saturated cost functions over an ordered sequence of heuristics.*

There is no news that cost partitioning is highly relevant to this study. We can better understand this concept by looking at an example that shows how cost partitioning can be used. Assume we have a planning task Π over $dom(a)$ and $dom(b)$ defined as seen below:

- $V = \{a, b\}$
- $I = \{a \rightarrow 0, b \rightarrow 0\}$
- $O = \{o_1, o_2\}$ with:
 - $o_1 = \langle \{b \rightarrow 0\}, \{b \rightarrow 1\} \rangle$
 - $o_2 = \langle \{a \rightarrow 0, b \rightarrow 1\}, \{a \rightarrow 1, b \rightarrow 0\} \rangle$
 - $o_3 = \langle \{a \rightarrow 1, b \rightarrow 1\}, \{a \rightarrow 2, b \rightarrow 0\} \rangle$
- $\gamma = \{a \rightarrow 2, b \rightarrow 1\}$

We calculate an optimal cost partitioning, $\langle cost_a^*, cost_b^* \rangle$ given the projections of a and b , π_a , respectively π_b , with heuristics $\mathcal{H} = \langle h_{\{a\}}, h_{\{b\}} \rangle$, for each of the projections:

- $cost_a^* = \{o_1 \rightarrow 0, o_2 \rightarrow 2, o_3 \rightarrow 2\}$
- $cost_b^* = \{o_1 \rightarrow 1, o_2 \rightarrow -1, o_3 \rightarrow -1\}$

For this study, we use saturated cost partitioning, meaning that we are considering both orders (a, b) and (b, a) before choosing the best one over $\mathcal{H} = \langle h_{\{a\}}, h_{\{b\}} \rangle$.

We start with the order (a, b) . Firstly, we take $h_{\{a\}}$ and calculate the cost partitioning:

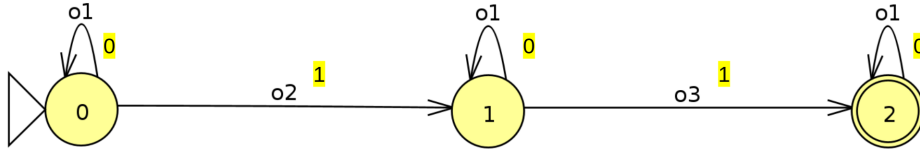


Figure 2.6: Cost Partitioning for a with order (a, b)

Next, we have $h_{\{b\}}$:

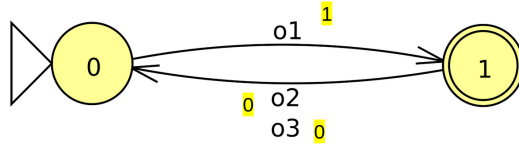


Figure 2.7: Cost Partitioning for b with order (a, b)

Therefore, for order (a, b) , we have:

- $cost_a = \{o_1 \rightarrow 0, o_2 \rightarrow 1, o_3 \rightarrow 1\}$
- $cost_b = \{o_1 \rightarrow 1, o_2 \rightarrow 0, o_3 \rightarrow 0\}$

Subsequently, we consider order (b, a) and start computing the cost partitioning for $h_{\{b\}}$:

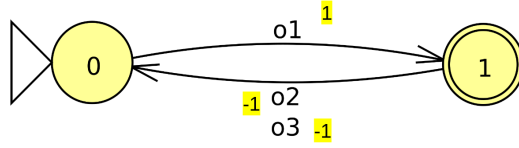


Figure 2.8: Cost Partitioning for b with order (b, a)

Now, we take $h_{\{a\}}$:

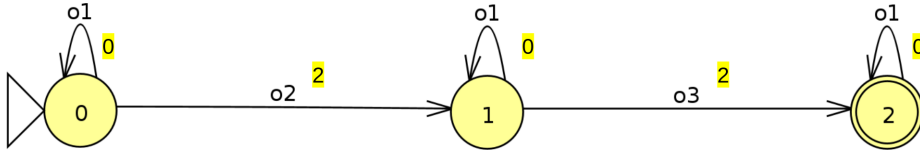


Figure 2.9: Cost Partitioning for a with order (b, a)

So, for order (b, a), we have:

- $cost_a = \{o_1 \rightarrow 0, o_2 \rightarrow 2, o_3 \rightarrow 2\}$
- $cost_b = \{o_1 \rightarrow 1, o_2 \rightarrow -1, o_3 \rightarrow -1\}$

In this example, the saturated cost partitioning for order (b, a) is the same as the optimal one. The initial unit cost of the operators has been split among the projections. We can understand that by simply looking at operator o_1 : for π_a it was assigned with 0, while for π_b with 1. Summing up $0 + 1 = 1$ gives us the initial cost, making it a correct partitioning. Secondly, the same applies for o_2 and o_3 , with π_a having it as 2 and π_b assigning it to -1. Again, $2 + (-1) = 1$ in both cases, so the process was done currently. The final value of the heuristic is given by the cost-partitioned shortest path from the initial state to the goal. It is important to note that, when using negative cost partitioning, we should ensure that no self-loop is assigned to a value lower than 0, as it would result in the shortest path to the goal being $-\infty$ [3].

The following chapter will put emphasis on the implementation and development of the algorithmic side of the thesis, with more attention to how merge-and-shrink and cost partitioning influence each other.

3

Implementation and Development

This chapter aims to give details concerning the development steps of the project and the implementation of the baseline algorithm, including improvements added for a deeper look at the combination between cost partitioning and the merge-and-shrink technique. The following sections will study the heuristic values and the sizes, emphasising when we want to decide to merge the atomic projections and when choosing cost partitioning is a better fit. It is known that merging gives us an equal or more informative result than cost partitioning. Still, sometimes the associated size increase is too big; therefore, cost partitioning becomes the best decision out of the two possibilities.

3.1 Comparing Merge-and-Shrink with Cost Partitioning

This section looks closely at cost partitioning and merge-and-shrink, comparing the two concepts based on explicit examples. For a more facile discussion, we take only the first step of the process and consider only two transition systems. At this stage, we will compare only the merging results with cost partitioning, not further considering shrinking, pruning or label reduction.

A Closer Look at Merging

We start by analysing the example in Figure 2.3 from the previous chapter, computing the cost partitioning for the atomic projections. For the sake of this example, an optimal cost partitioning is considered. As the following example suggests, sometimes cost partitioning cannot gather all the information. There can be cases where the cost partitioned elements are not informative enough, while the merged product is. However, it uses significantly more memory. Hence, even if merging the transition systems to get more information seems like the answer, it is not always the best choice, and it definitely does not guarantee the best choice all the time when size is also considered. The above reasoning lies at the basis of this thesis. The aim is to find a way of deciding between merging and not, ensuring achieving the best result is the ultimate goal. Therefore, as seen below, we take the projections from Figures 2.1 and 2.2 and cost partition.

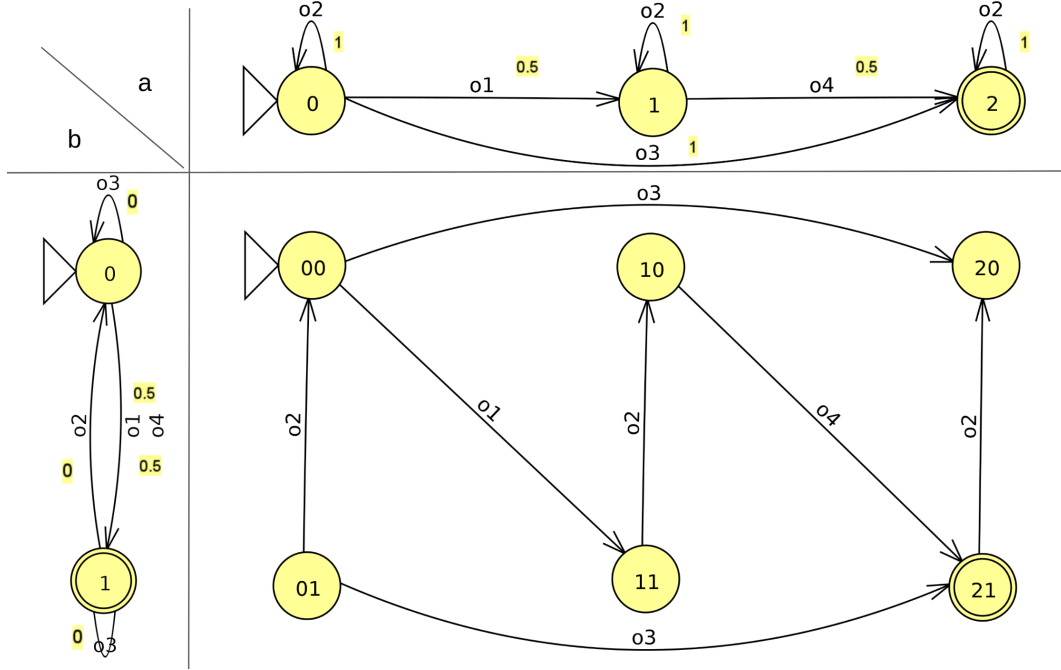


Figure 3.1: Cost Partitioning Example in Figure 2.3

Followingly, we move on to analysing the heuristic values in the initial state, with cost partitioning $\langle cost_a, cost_b \rangle$. Having these details, we can compute the heuristic values for each atomic projection and the resulting merged transition system. In this case we have $h^{\pi_a} + h^{\pi_b} \neq h^{\pi_{\{a,b\}}}$, however, it is known that $h^{\pi_a} \otimes h^{\pi_b} = h^{\pi_{\{a,b\}}}$:

- $h^{\pi_a}(s_0, cost_a) = 1$
- $h^{\pi_b}(s_0, cost_b) = 0.5$
- $h^{\pi_{\{a,b\}}}(s_0) = h^{\pi_a}(s_0) \otimes h^{\pi_b}(s_0) = 3$

We are switching our interest now to the total size:

- π_a is using total memory of 3
- π_b is using total memory of 2
- $\pi_a \otimes \pi_b$ is using total memory of 6

We are dealing with $h^{\pi_a} + h^{\pi_b} = 1 + 0.5 = 1.5$ and $h^{\pi_{\{a,b\}}} = h^{\pi_a} \otimes h^{\pi_b} = 3$, where we can clearly see that $1.5 < 3$. This implies the fact that the cost partitioning is not able to give us a result as good as the merging; hence computing the synchronised product would indeed offer more information. However, a closer look at memory usage tells us that, even if quality-wise merging seems like the better option, resource-wise, it gets costly. Keeping the non-merged projections brings a total memory of $3 + 2 = 5$, as we sum up the number of states in each. For the merging, we need to offer space for their product $3 \times 2 = 6$, which is larger in value than the sum.

A Closer Look at Cost Partitioning

Now, we focus on the atomic projections from the examples in Figures 2.9 and 2.8 and calculate their synchronised product. Analysing the example below, we notice that there can be cases where cost partitioning can offer as much information as the synchronised product. Now, we are dealing with the same heuristic value, but a smaller final memory usage for the cost partitioning compared to the merging. Therefore, considering both size and quality, the best decision would be to cost partition, showing an opposing point of view to the example from above. The example below illustrates the merging of the atomic projections from the examples in Figures 2.9 and 2.8, from the previous chapter:

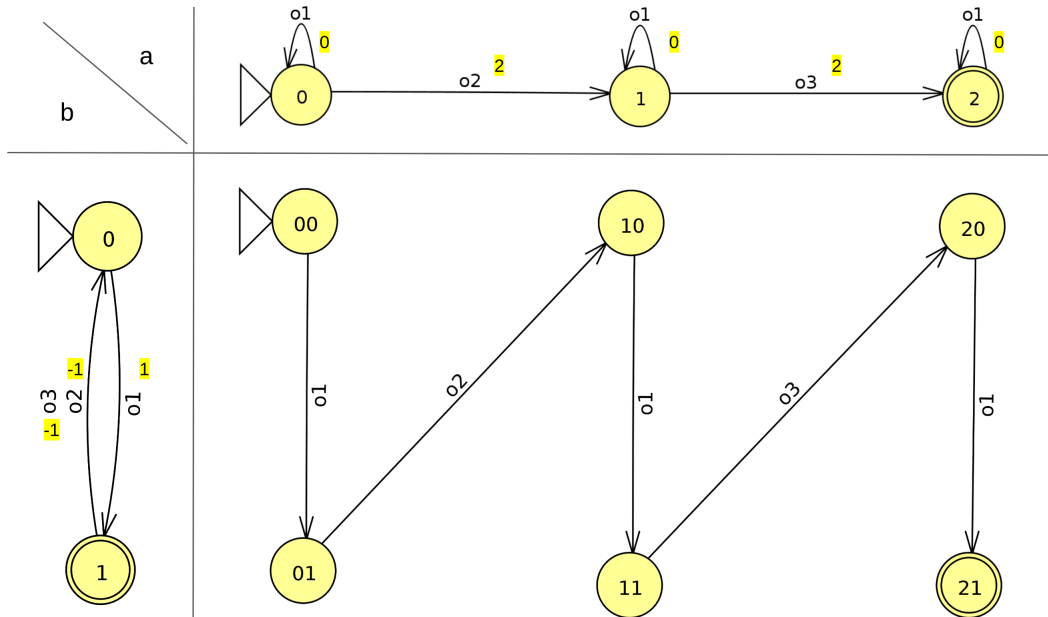


Figure 3.2: Merging of Examples in Figures 2.9 and 2.8

Taking a closer look, we can derive the heuristic values for the initial state, of both the atomic projections and the resulting synchronised product. Contrary to the previous example, in this case we notice that $h^{\pi_a} + h^{\pi_b} = h^{\pi_{\{a,b\}}} = h^{\pi_a} \otimes h^{\pi_b}$.

- $h^{\pi_a}(s_0, cost_a) = 4$
- $h^{\pi_b}(s_0, cost_b) = 1$
- $h^{\pi_{\{a,b\}}}(s_0) = h^{\pi_a}(s_0) + h^{\pi_b}(s_0) = 5$

Now, we look at the memory usage:

- π_a is using total memory of 3
- π_b is using total memory of 2
- $\pi_a \otimes \pi_b$ is using total memory of 6

The heuristic for the merged system has the same value as the sum of the cost-partitioned projections for one of the orders. This is telling us that the partitioning is getting everything from the synchronised product, being no information loss. So, merging will not offer us additional data as $h^{\pi_a} + h^{\pi_b} = 4 + 1 = 5 = h^{\pi_{\{a,b\}}}$. Therefore, as the cost partitioning can guarantee that all the information is kept in the individual atomic projections, doing such an expensive action as merging is not worth it in this case. Here, we have a 3×2 example, so the memory space of 6 for the synchronised product is larger than the summation of the memory taken by each atomic projection: $2 + 2 = 4$. In real life, we encounter much bigger situations, where doing the merging might take up significantly more memory, therefore, if calculating the product does not bring anything new to the table, the better choice might be not to calculate it at all. More, let us take one of the previously mentioned examples and assign a cost partitioning. Followingly, let us compute the new heuristic values and briefly compare variations of properties, such as quality and used resources.

In the rather small case from Figure 3.1, it might be worth merging even if this process implies greater resource usage. But, when dealing with very large examples, the memory needs might add up to colossal values that take ages to store, or even more that the machine can take. Therefore, even if the merging sometimes offers a better quality of results, it also comes with a severe memory increase. For instance, let us take an atomic projection with 200 states, and another with 300 - taking the cost-partitioned individual terms will require a total memory of 500, while the merged product will have a total of 60000. This trade-off between the perfect quality assured by the merge-and-shrink technique and the massive memory space it brings along with it is one of the goals that this thesis aims to study and discuss. Alongside, we will use cost partitioning as the main tool to help with this decision, predominantly in a quantitative manner that tells us whether merging is worth it or not, considering both the heuristic values and the memory space. The idea behind the merging algorithm while handling the heuristics is introduced by the pseudocode in the next section.

3.2 Combining Merge-and-Shrink with Cost Partitioning

The concepts and ideas mentioned above can be applied practically. For this, we came up with a baseline algorithm that attempts to make the best out of the combination between cost partitioning and the merge-and-shrink technique. The main idea of the algorithm is simply deciding whether to merge or to cost partition at any point - the aim is to merge if cost partitioning is not informative enough, and to cost partition if merging does not bring quality improvement. Let us take a look at the examples from Section 3.1. For Figure 3.2 and similar cases, it is very clear that we will cost partition - same value, less memory. But when it comes to cases like in Figure 3.1, merging brings to the table a better heuristic, more information, but also more memory. For this thesis, this case implies choosing to merge, as synchronised products that exceed a hard limit will be excluded from the calculation, as this section will later present.

3.2.1 The Baseline Algorithm

Therefore, on a more technical side, the approach of combining merge and shrink with cost partitioning considered for this thesis study relies on the following pseudocodes:

Algorithm 1: Pseudocode for Comparison Function

```

def compare_and_update( $T_i, T_j, \text{priority\_queue}$ ) as:
  compute  $T_i \otimes T_j$ ;
  compute  $h_{CP}^{T_i} + h_{CP}^{T_j}$ ;
   $quality \leftarrow \text{compare}(T_i \otimes T_j, h_{CP}^{T_i} + h_{CP}^{T_j})$ ;
   $\text{priority\_queue.add}(quality, \langle i, j \rangle)$ ;

```

Algorithm 2: Pseudocode for Baseline Algorithm

```

Input: A set of transition systems  $\mathcal{T}$ , a threshold
Output:  $\mathcal{T}$  updated based on decision to merge or cost partition
 $\text{priority\_queue} \leftarrow \text{Empty}$ ;
while active entries in  $\mathcal{T} \geq 2$  do
  if first iteration then
    foreach pair  $\langle T_i, T_j \rangle \in \mathcal{T}$  do
      |  $\text{compare\_and\_update}(T_i, T_j, \text{priority\_queue})$ ;
    end
  end
  else
    |  $T_{prev} \leftarrow \text{merge from last iteration}$ ;
    foreach  $T_i \in \mathcal{T}$  do
      |  $\text{compare\_and\_update}(T_{prev}, T_i, \text{priority\_queue})$ ;
    end
  end
   $\text{max\_quality} \leftarrow \text{priority\_queue.top().first}$ ;
   $\text{best\_pair} \leftarrow \text{priority\_queue.top().second}$ ;
  if  $\text{max\_quality} > \text{threshold}$  then
    | return  $\mathcal{T.add}(\text{merge}(\text{best\_par}))$ ;
  end
  else
    | break;
  end
end
return  $\mathcal{T}$ ;

```

The algorithm offers a greedy approach that aims to find an answer to the question that lies at the basis of this thesis: a decision on whether to merge or cost partition. It presents itself as a merge algorithm, taking transition systems as input and giving as output the updated version based on the number of merges executed. The algorithm itself starts by taking in a set of transition systems. It is worth mentioning that, for time and space matters, the merges for the pre-process are computed only once in the priority queue set up during the first iteration. After that, only the merges with the new transition system resulting from the previous iteration are executed, as the essential information is kept in a priority queue. The main loop checks if two or more active transition systems exist to ensure a possible merge.

During the set up, we are looping among all the existing and active transition systems to compute the quality measurement based on the merge and cost partitioning. The choice of whether to continue merging is guided by the quality value, calculated based on the value of merging compared to just using cost partitioning. This process takes place at every iteration through the set of transition systems until there is no pair whose merge quality value exceeds the minimal allowed quality. At each iteration, the pair with the largest quality is eventually merged, and the process continues until no pair brings more information. Each pair, alongside its quality, are stored in the data structure for later use. In the case of other iterations, the same is performed. Still, instead of looping through the whole factored transition system again, we are only computing and further storing the values for the newly added merged transition system in the last iteration, so in the end, everything is up to date, and we have all the information needed. When all the pairs have been traversed, the best pair is chosen based on the maximum quality value. If the quality exceeds the fixed threshold, the corresponding transition systems are merged, and the process begins again. In the opposite case, we are not merging, ending the loop with a cost partitioning. For two abstractions α and β with heuristics h^α and h^β , we can define the quality as:

$$quality = h^\alpha \otimes h^\beta - h^{\alpha+\beta}$$

In order to ensure a good quality of the results, several aspects have been considered in addition to the baseline. As previously mentioned, merging two transition systems can give us a very large result sometimes, timeouts or infinite loops might occur, or the merged pair might not bring as much as another possible pair would. For this reason, the following have been added to the baseline algorithm:

Space and Time Limits

While running experiments with the baseline algorithm, we ran out of time or space on several occasions. Also, in other cases, we were dealing with a prolonged run or too much memory usage. To fix this, space and time limitations have been introduced in addition.

- A countdown timer is now keeping everything under control, being set at the beginning of the main loop. If the maximum time allocated for the run passes, the loop will be ended with the decision not to merge anymore.
- An option to check if the synchronised product fits into memory was also implemented as an additional check for the estimated size of the merging. If the estimated size of merging a pair is greater than this allocated limit, the pair will not be considered.

Randomisation of Choice/Tie Breaking

There have been many times when more pairs had the most significant gain from the quality value, but the baseline algorithm would always choose the first in the queue. Sometimes, choosing another pair would result in a different course of the following steps in the algorithm. For this reason, randomisation was created when selecting the best pair. This tie-breaking only considers the pairs with the maximal quality value.

3.2.2 Improvements to the Baseline

With the goal of boosting the results of the baseline algorithm even more and aiming for a better performance and memory usage, improvements were considered. Within the limited time of this thesis, and considering the actual steps of the original merge-and-shrink technique, the following features were implemented:

3.2.2.1 Pruning Unreachable and Irrelevant States

Dealing with merged products can be very consuming regarding the memory and merging count. This is why opting to prune unreachable and irrelevant states could lead to a way better deal with out-of-memory errors. Moreover, as a merging size limit is present, this implementation could help consider merges that bring us much value but exceed the limit when such states are counted. Let us consider an example with an abstraction α of size $|\alpha| = 4$ and an abstraction β with size $|\beta| = 4$. The sum of the sizes for the atomic projections is 8. Assume the size of the merged system is $|\alpha \times \beta| = 16$, but the pruned synchronised product has a size of $|\alpha \times \beta|^{pruned} = 6$. Further, assume that the heuristic for merging is the same as for cost partitioning. In this case, without pruning, the pair would not be merged as the heuristic value brings nothing more to the table, but the size increases. When considering pruning, we have the same heuristic value but a smaller size for the merged product. Thus we would choose to merge the pair. Pruning is effectuated on the atomic projections when the priority queue is being set up, but also on each of the chosen synchronised products. The new dynamic of the program can be understood from the diagram in Figure 3.3, below. In the algorithm, pruning is used over the atomic projections at the start of the algorithm, as well as after merging the chosen pair.

3.2.2.2 Shrinking

On a high-importance scale, with the aim of a more complex and complete solution, shrinking was also integrated into this project. So far, we have been talking about merge-and-shrink but actually only used the merging side of this technique. Considering an example, let us take an abstraction α of size $|\alpha| = 6$ and an abstraction β with size $|\beta| = 3$. Moreover, consider the size of the atomic projections together as being 8. The merged size of the abstractions is $|\alpha \times \beta| = 18$, but let us assume that the size after shrinking is $|\alpha \times \beta|^{shrink} = 7$. Additionally, assuming that the heuristics are equal in both cases, originally, we would not choose to merge. However, the shrinking resulted in a smaller size for the product; therefore, merging and shrinking after is a better choice. Important to note is that cost partitioning itself cannot be better than merging, but if shrinking is involved as well, we can deal with the option when cost partitioning gives us more information than merge-and-shrink. To deal with this, special options to perform shrink and check shrink were added to the algorithm. The user can choose between shrinking and not shrinking, as well as decide whether shrinking should influence the quality calculation or not. Similarly to pruning, shrinking can be used initially, on atomic projections, to calculate the quality, but also at before merging the pair with the highest quality.

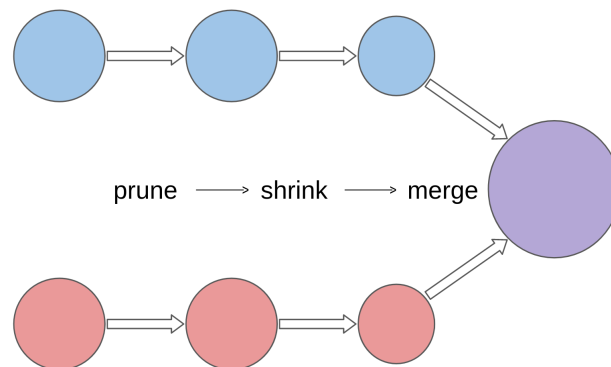


Figure 3.3: The Mechanics of The Algorithm

3.2.2.3 Flexible Quality and Merging Threshold

On a more experimental side, paving the path to the next chapter, we also took into consideration the flexibility of choosing the threshold for merging or a way of computing the quality. The latter can now be calculated using only the heuristic values (if we only care only about the actual and raw difference in information between merging and cost partitioning) but also using both the heuristics and the memory sizes (if controlling the size allocation in memory is of high importance as well), mode that the user can choose. We can use this in order to catch some edge cases, such as always merging if the size of the resulting synchronised product is equal to the size of the individual abstractions summed up.

We impose a quality threshold in the form of the minimal heuristic value of the synchronised product accepted for a merging to take place. With this, we make sure to merge only if it brings results good enough. For instance, with a threshold of 0, we suggest that the heuristic value of the synchronised product must be better than the cost partitioning of the atomic projection, while with -1 we allow a merge to take place even if it has the same heuristic. It can also be used to reproduce the main algorithms: merge-and-shrink and cost partitioning.

Moreover, we consider a parameter that dictates the maximum times allowed to overpass this quality threshold. Let us look at this example: we consider the threshold 0 and take a set of transition systems. Here in the first iteration, the maximum possible quality is 1. After merging, in the next iteration, we face the maximum quality of 0, which usually would terminate the process and not further merge. However, if we had continued to merge, the next best quality would be 3, which would even be the highest so far. This feature allows the calculation to keep going over the minimal accepted quality for as many times as we offer by the command line argument.

Moving on to the merging threshold, this parameter can put a higher limit on the size of the synchronised product, as well as a triggering point for shrinking. As the previous cases, it can also be chosen by the user for a more adaptable environment. More details on a practical level and experimental analysis of each parameter will be discussed and proved in the following chapter.

4

Experimental Analysis

This chapter aims to give an overview of the experimental side of the thesis, with insights regarding the tests that have been concluded for better development and implementation of the thesis. There are four main checkpoints of analysis present, referring to the baseline algorithm and each of the improvements. Further experiments took place concerning the parameters and arguments, as well as comparisons with the previous work done on this same topic carried out by Sievers et al [11]. The experiments have been conducted on the sciCORE infai server, under Linux, using Fast Downward and various benchmarks. The maximum loop is controlled by a constant timer of 900 seconds throughout the experimental phase.

4.1 The Baseline

On a vast spectrum of algorithms, the two main techniques considered for this thesis, merge-and-shrink and cost partitioning, are situated on opposite sides. In between, there are many algorithms with distinct aims and goals, among which the approach for this thesis can be found. Having the purpose of getting the best of the aforementioned techniques, we start with the baseline and therefore infuse only merging with cost partitioning at this step.

The experimental phase was run on a set of benchmarks, listed in Appendix A, that will be further referred to during this discussion. Additionally, the main subject of this analysis is parameter optimisation based on the quality threshold, merge threshold, coverage, number of merges and initial heuristic value. The quality threshold implies the minimum value allowed for the quality calculation - how much better does merging have to be in order to merge and not cost partition. Next, the merge threshold implies the maximum number of states of the transition systems further considered for merging. Next, the coverage suggests the number of problems solved. In this baseline variant of the algorithm, a simple approach that only uses the merging part of merge-and-shrink is considered. Lastly, as suggested by the name, the number of merges refers to how many merges have been executed using the given parameters, while the initial heuristic is the value of the heuristic initially calculated. The aim is getting larger values in all cases.

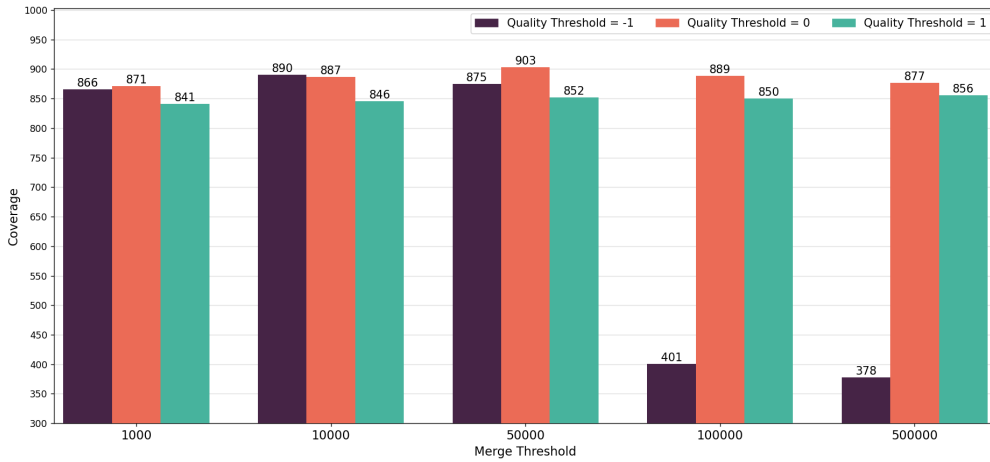


Figure 4.1: Baseline - Coverage Analysis

As illustrated in Figure 4.1, one could easily notice the outlier coverage values reached by using the quality threshold of -1 for merge thresholds exceeding 100,000. This phenomenon is most likely due to the very large size allowed for the transition systems, combined with a very permissive merging strategy. In turn, it can cause massive memory and time usage that can often exceed the hard limits imposed by this approach. Next, looking at 0 as the quality threshold, the values seem to reach their maximum coverage value at the 50,000 merge threshold, declining slightly afterwards. As for the quality threshold of 1, it seems to evolve similarly to the case of quality 0, with all the values being underachieving with regard to coverage. Looking at the average coverage for each merge threshold, it can be noticed that 50,000 offers the largest mean value of 876, followed closely by 10,000 with an average of 874. Consequently, the aforementioned factors signal that quality threshold 0 helps solve the most problems among the benchmarks. More, in the case of a merge threshold of 50,000, over 900 successfully solved problems are encountered, this result being the highest so far.

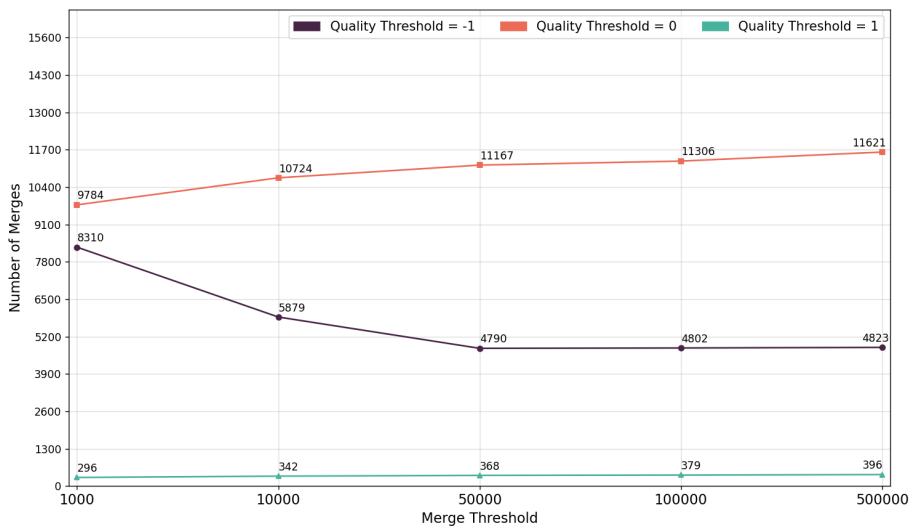


Figure 4.2: Baseline - Number of Merges Analysis

In Figure 4.2, the number of merges that took place is further analysed, keeping the attention on the merge and quality thresholds. There is a clear delimitation between the quality threshold values based on the number of merges. A large value of the quality threshold implies a more restrictive algorithm regarding the number of merges. This can be observed as well from the line chart, especially when looking at quality thresholds 0 and 1. With a quality of 0, the algorithm will merge in every case when the synchronised product offers a better value than the cost partitioning within limits imposed by the merge threshold, therefore having large and increasing values. For quality 1, extra weight is added to the maximum needed merging value to be for the pair to be considered; hence, there are very few merges. In theory, the quality threshold of -1 would be expected to perform more merges. However, due to the large number of states that occur with increased merges, the merge threshold prevents pairs from being considered, thus actually resulting in a lower amount of merged transition systems. This would also be the reason for the decrease in the number of merges as the merge threshold grows, contrary to our expectations. Otherwise, the values seem to follow an ascending trend with relatively small steps, as expected.

4.2 Adding Pruning

Merging brings value and information but can also bring massive memory usage. Sometimes memory is used to store irrelevant or unreachable states that bring nothing to the final result. As suggested by Figure 2.4, extra memory use can be eliminated by simply pruning. Taking a closer look at how the current algorithm works, only transition systems of size below a certain limit will be further processed and considered for merging. Hence, pruning brings us a step closer to the purpose of merging as much as possible without losing value or increasing the memory allocation significantly. Pruning can be controlled from the command line, as the user can choose whether to consider this option or not. Similarly to the previous section, we look at the quality and merge threshold. The experiment is split into 3 cases given by the quality threshold, to study the program for different merging thresholds.

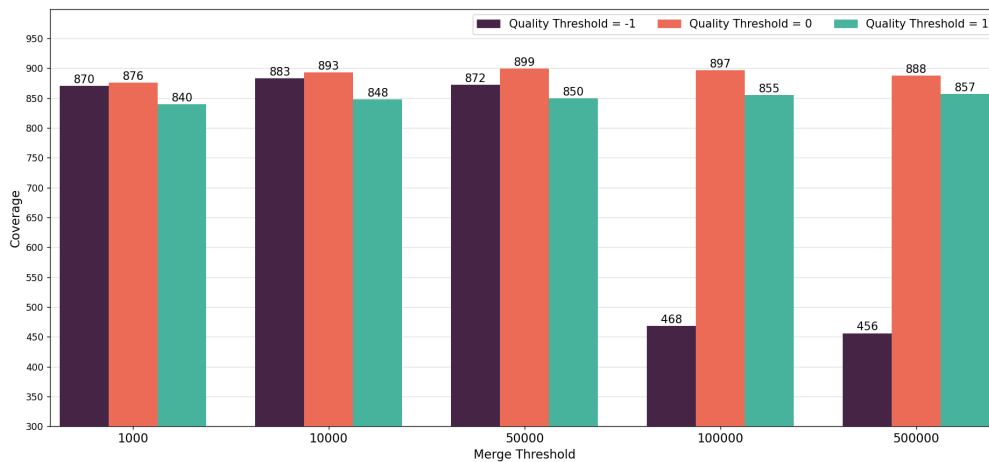


Figure 4.3: Merge and Prune - Coverage Analysis

As shown in Figure 4.3, in a similar manner as discussed in the previous section, there are 3 main study cases for different quality threshold values: -1, 0 and 1. It can be noticed that for value -1, the merge thresholds of 100,000 and 500,000 present significantly lower coverage. Similar to the baseline variant, we deal with a major increase in memory and runtime, behaviour explained by the permissive merging strategy and the allowed transition systems with many states. As for the default quality threshold of 0, the coverage reaches its peak at 50,000 maximum states and begins to degrade for larger amounts. This is not the case for a quality threshold of 1, as it can be observed a linear and slow increase. Overall, the most promising variant of the algorithm can be found between a merge threshold of 10,000 and 50,000, where the mean coverage of the three quality threshold cases is 874 for the first mentioned, while the second averages 873. Moreover, as the case of the merge threshold 50,000 with quality threshold 0 is the most valuable overall, this will be considered as the new main algorithm for further analysis and comparisons.

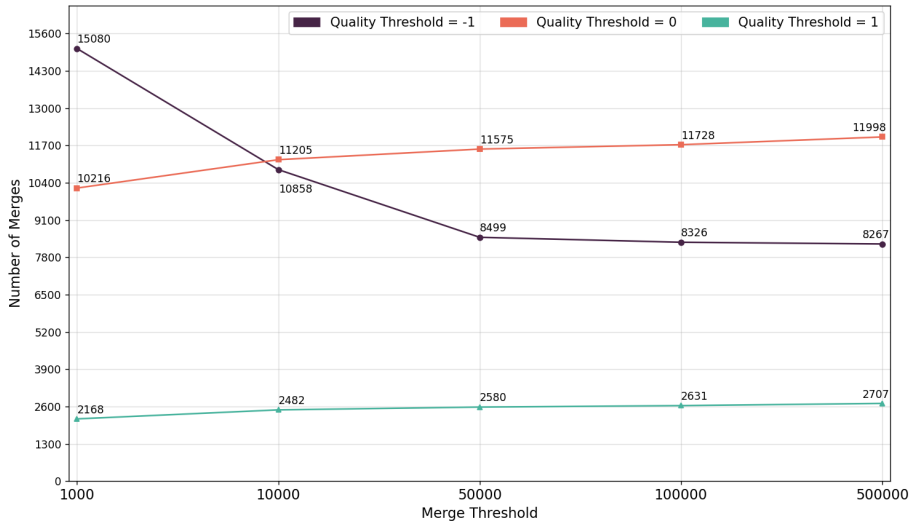


Figure 4.4: Merge and Prune - Number of Merges Analysis

Taking a closer look at the number of merges, Figure 4.4 shows pattern similarities to the variant of the algorithm that only considers merging discussed in the previous section. Generally, the number of merges is higher than previously. This is mainly due to the fact that pruning brings memory-saving options by removing irrelevant and unreachable states. As we are only storing and considering the informative states, there are even fewer cases where the merge threshold stops a pair of transition systems from being merged. All quality cases kept their trend evolution, with a slight change from quality threshold -1, that now starts all the way up at over 15,000 merges. The values for this particular case are yet still decreasing, signalling that not enough memory has been saved.

Even if coverage-wise the changes are not major, and sometimes even fewer problems are solved, by looking at the number of merges we can understand the value pruning brings into this approach. More merges are allowed to take place by saving enough memory that would

have been used by uninformative states. Not only the merged product is pruned, but also the atomic projections during the pre-process, making the starting point even more memory effective. By simply looking at quality threshold 1, we can notice a 700% increase in the number of merges.

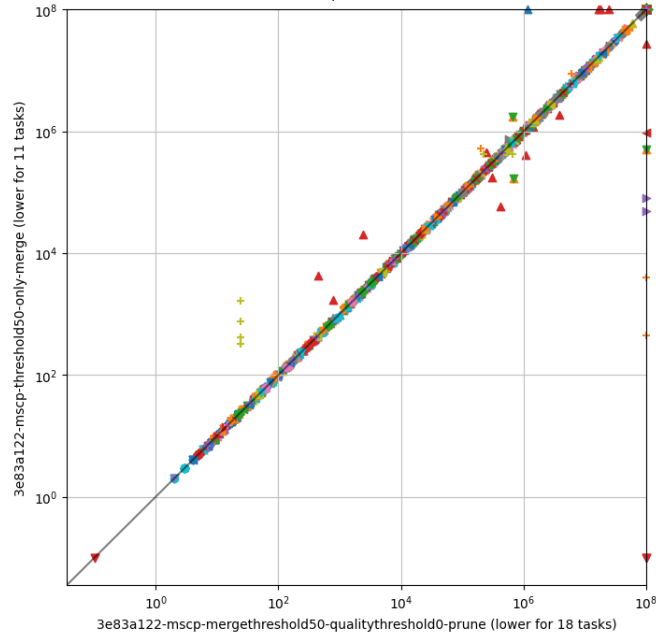


Figure 4.5: Merge and Prune - Expansions

Next, we switch our attention to the heuristic value, comparing the expanded states from before and after adding pruning as an option. We strive for a value of expansions as low as possible, implying that the initial heuristic calculates by the algorithm is valuable enough for the planning problem. As can be seen in the scatter plot above from Figure 4.5, there are different cases where each of the approaches performs better heuristic-wise than the other. There are lower expansions in 18 instances for the new implementation that also considers pruning, showing the potential of this feature. One example would be *openstack-strips*, a set of small to medium-sized problems. The big similarities in terms of initial heuristic value can be determined by the large number of items present on the diagonal line, with only a few cases of under 20 instances being expanded.

4.3 Adding Shrinking

Looking for a way to prune and subsequently merge even more, shrinking comes in as a helping hand. Therefore, by creating abstractions of the already existing transition systems, the memory is yet further decreased. However, contrary to pruning, the actual quality of the resulting synchronised product can be affected. Same as for pruning, the option of shrinking can be changed from the command line, choosing between performing shrinking, performing and calculating the quality using shrinking, or not shrinking at all.

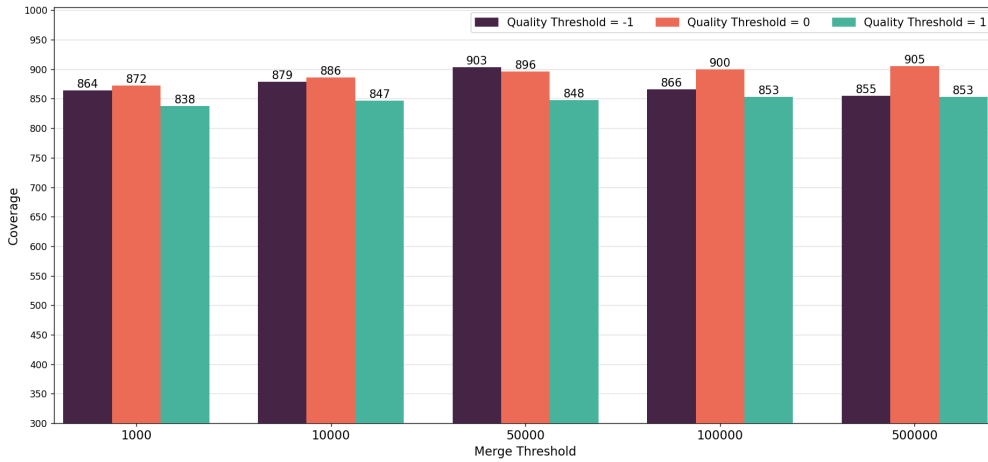


Figure 4.6: Merge, Prune and Shrink - Coverage Analysis

Figure 4.6 represents the coverage analysis after adding shrinking as a feature to the algorithm. As opposed to the previous cases, this version of the algorithm succeeded in getting a higher coverage even when allowing large transition systems with more states and a permissive merging strategy. Easily observable is the increasing trend for quality threshold 0, which is no longer having the peak at 50,000 but rather at the last tested value for the merge threshold of 500,000. This is not the case for quality -1, as the top value is seen as expected at 50,000. Taking a closer look at the average coverage per merge threshold value, even if the most valuable case is seen for quality 0 at 500,000 merge quality, we notice that the highest mean is once again present for the 50,000 merge threshold, with a value of 882. Overall, there has been an improvement in coverage of over 4%.

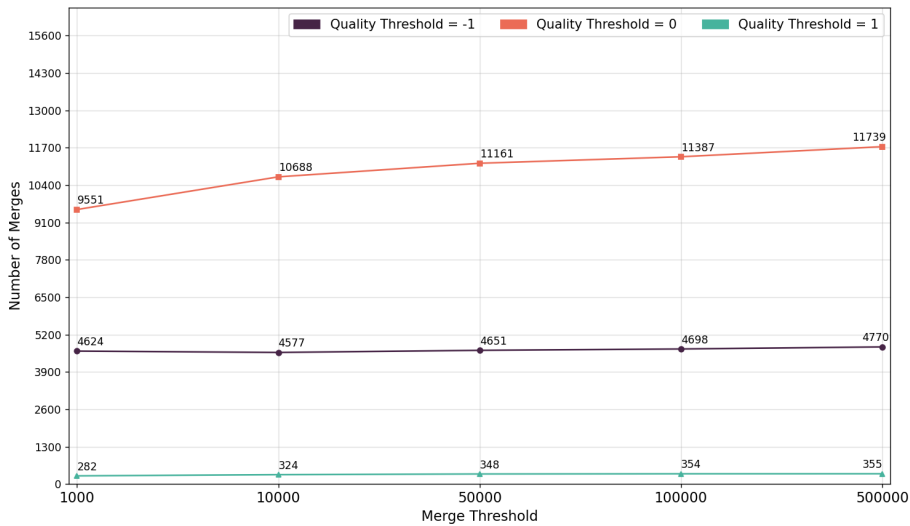


Figure 4.7: Merge, Prune and Shrink - Number of Merges Analysis

Next, Figure 4.7 illustrates the number of merges for all threshold cases, after the addition of shrinking. As we have seen when studying the coverage, we are dealing with pattern changes, especially for quality threshold -1. In the case of the number of merges, the values

are no longer decreasing but rather steadily growing. This, along with the coverage valued similarly to the other cases, implies better handling of the memory space. Therefore, it seems that the often-encountered timeouts and memory issues are now kept under control.

As mentioned, the coverage in the case of a quality threshold of 0 is increasing, having the peak at the 500,000 merge threshold, which is the last one analysed. Hence, for this study, and solely for the quality threshold valued at 0, additional merge thresholds have been further tested, as seen in Table 4.1:

Threshold	1k	10k	50k	100k	500k	700k	1M	2M	5M
Coverage	872	886	896	900	905	906	905	901	887
Merges	9551	10688	11161	11387	11739	11747	11753	11388	10964

Table 4.1: Testing Larger Merging Thresholds

We see that the peak is actually between 500,000 and 1,000,000 as the value of the merge threshold. However, with a closer look at memory usage, we can see a massive increase of over 63% as the value of the merge threshold grows.

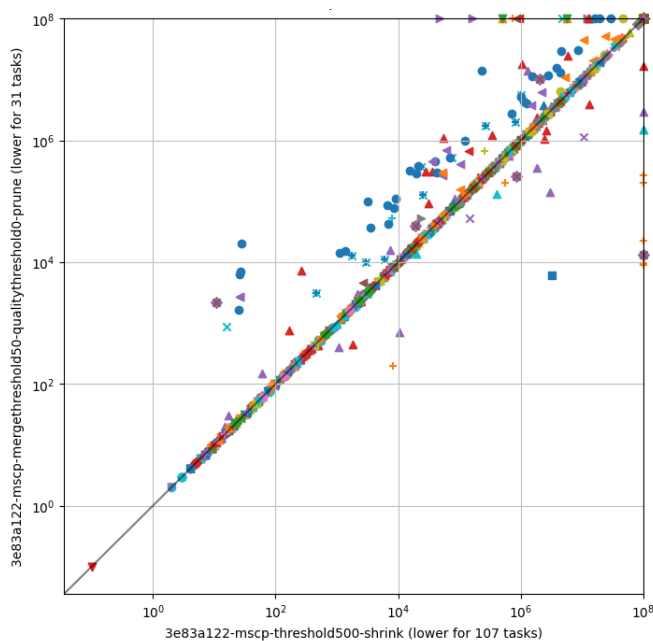


Figure 4.8: Merge, Prune and Shrink - Expansions

Similarly to the last section, Figure 4.8 plots the expansions, comparing the most promising configuration previously discussed with the highest achieving version after adding shrinking. We can see how the new approach brings us a larger amount of cases in which the number of expanded states is lower. There are still problems that can be better solved with one approach than the other. A good example would be *miconic*, represented by the dark blue circle, where it is clear that most instances are present on the upper side of the diagonal line. Here we are dealing with a domain of smaller problems with under 60 transition systems. The newest approach manages to offer a better initial heuristic value for all these problems,

illustrated by the lower number of expansions for all outliers. We can see that the approach including shrinking, in the bottom half, managed to score a better heuristic initially in 107 cases, while the older version only 31.

To summarise the parameter optimising section of the experimental phase, the variant of the algorithm employing both pruning and shrinking, with a quality threshold of 0 and a merge threshold of 500,000, outperforms all the other tested versions for this implementation.

4.4 Quality Analysis

As quality is the main idea behind the merging strategy proposed for this thesis, several testing methods have been used to determine the most value of this parameter. Therefore, two additional options have been added to the program, controlling the way quality is handled throughout the program. For the following results, we have used the aforementioned final version of the algorithm decided upon.

Considering Size in Quality: Sometimes, not only the final heuristic value is important, but also the space in memory used. Hence, an option to calculate the quality of merging by also weighing in the estimated size of a merge has been offered. This follows a mathematical formula that increases the quality of the heuristic as the synchronised product gets larger and its size gets smaller:

$$quality = \begin{cases} 999999, & \text{if } |\alpha \times \beta| \leq |\alpha| + |\beta| \text{ and } h^\alpha \otimes h^\beta \geq h^{\alpha+\beta} \\ 1, & \text{if } |\alpha \times \beta| \leq |\alpha| + |\beta| \text{ and } h^\alpha \otimes h^\beta = h^{\alpha+\beta} \\ h^\alpha \otimes h^\beta - h^{\alpha+\beta}, & \text{if } |\alpha \times \beta| \leq |\alpha| + |\beta| \text{ and } h^\alpha \otimes h^\beta < h^{\alpha+\beta} \\ \frac{h^\alpha \otimes h^\beta - h^{\alpha+\beta}}{|\alpha \times \beta| - (|\alpha| + |\beta|)}, & \text{otherwise} \end{cases} \quad (4.1)$$

where α and β are abstractions of size $|\alpha|$ and $|\beta|$, and the merged size of the abstractions is $|\alpha \times \beta|$, with heuristics h^α and h^β .

Allow Size	Yes	No
Coverage	900	905
Merges	58019	13909

Table 4.2: Testing Size in Quality Calculation

Table 4.2 shows not only a very similar coverage value in the case where size weighs in towards quality calculation but also a significant increase of almost 25% in the number of merges effectuated during the algorithm.

Allowing Occurences Minimal Quality: Furthermore, there are cases when the first few steps offer directly the minimum allowed quality, but the merging happening in later stages brings a very informative result. Due to time limitations, a proper look-ahead option to avoid this issue was not implemented, but a variable to allow the minimal quality to be overlooked a given amount of times was added to the algorithm. In this manner, if the

option is set to, for instance, 2 and the quality threshold is 0, the algorithm will continue to merge even if the merging gives a quality of 0 but allows this to happen only 2 times.

Occurrences of Min Quality	0	1	2	3
Coverage	905	889	886	881
Merges	13909	15144	16905	18593

Table 4.3: Testing Maximum Occurrences of Minimal Quality

As seen in Table 4.3, the number of merges grows linearly. The initial heuristic value also increases alongside the number of merges, with an increase of over 99%.

4.5 Comparing to Cost Partitioning

We can reproduce cost partitioning by setting the merge threshold to -1, the quality threshold to infinity, and all the other parameters to *false* so that we make sure that no merge, shrink or prune will take place during the process.

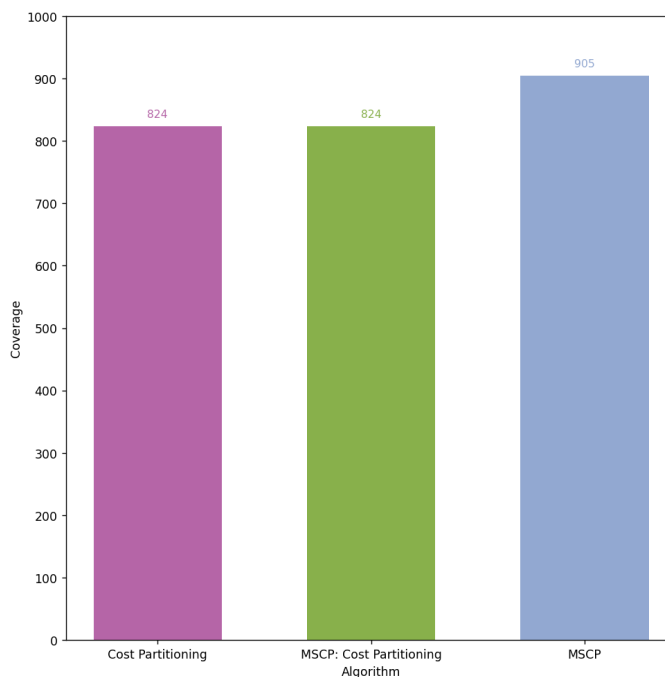


Figure 4.9: Versus Cost Partitioning - Coverage

In Figure 4.9, three algorithms are compared: cost partitioning, this algorithm's cost partitioning representation and the proposed algorithm for this thesis. Coverage-wise, while cost partitioning could only solve 824 problems, the highest-performing algorithm from the last section achieved a top value 905. Moreover, the algorithm was successful in reproducing precisely the coverage value 824 of cost partitioning using the parameter values mentioned above.

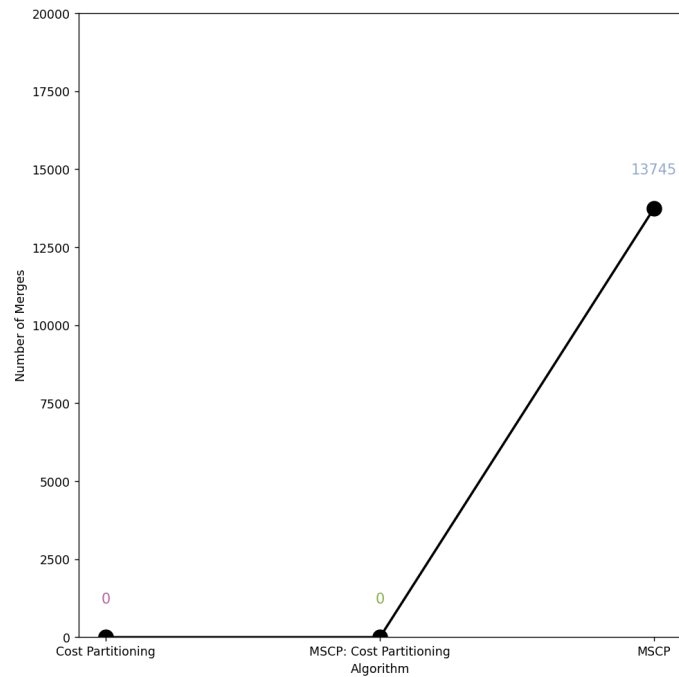


Figure 4.10: Versus Cost Partitioning - Number of Merges

As expected, we can see that the algorithm is able to reproduce pure cost partitioning with 0 number of merges. As cost partitioning does not perform any merges, the quality threshold of infinity prevents any merge from taking place.

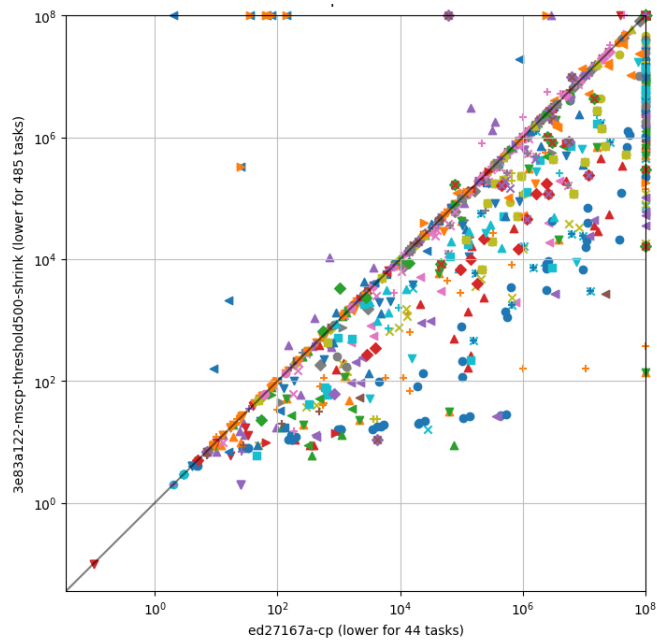


Figure 4.11: Versus Cost Partitioning - Expansions

Figure 4.11 aims to picture the initial heuristic value through the expanded states. The side of the plot over the diagonal represents the states expanded for the algorithm of this

thesis, while the bottom side is cost partitioning. Above the diagonal, there are fewer instances of the benchmarks, while the bottom half contains most of them. This implies that cost partitioning resulted in more expansion after the calculation of the initial heuristic in 485 cases, to our advantage. This represents an increase of over 1000% compared to cost partitioning, being lower for only 44 tasks.

4.6 Comparing to Merge-and-Shrink

Before diving deeper into the comparison results, it is important to mention that while the cost partitioning can be reproduced almost perfectly, it is not exactly the case for merge-and-shrink. The algorithm used a so-called merge strategy to compute the results, a parameter not used in the implementation suggested by this thesis. The reason behind this is that the proposed algorithm presents itself as a form of merging strategy, and its value consists, among others, of how the merging is being done. Moreover, the merge-and-shrink algorithm includes a fourth step of label reduction, a concept not yet implemented for this thesis. For the sake of these experiments and yielding towards a fair comparison, label reduction has not been used for the merge-and-shrink tests.

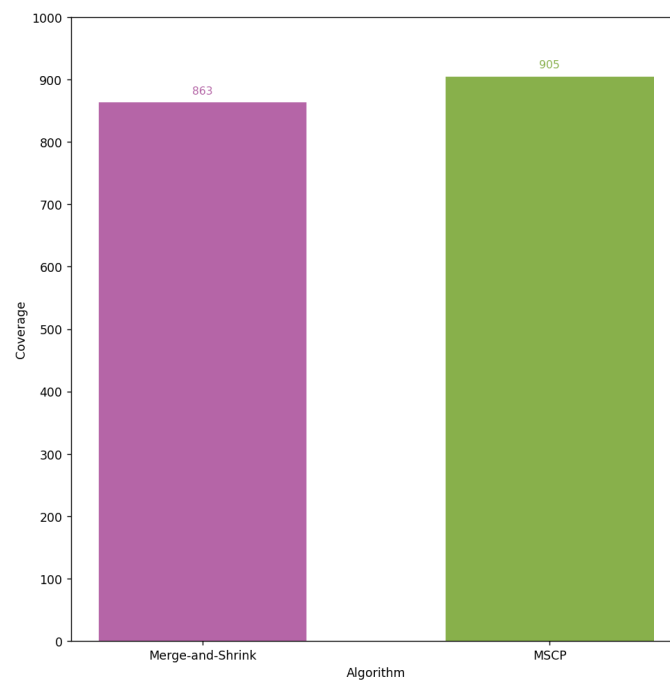


Figure 4.12: Versus Merge-and-Shrink - Coverage

As shown in Figure 4.12, the potential of this approach is suggested by the coverage. While merge-and-shrink solved a total of 863 problems, the algorithm of this thesis managed to score 905 as the top coverage. The difference of 42 unsolved cases, alongside the information that merge-and-shrink reaches 905 coverage only when adding label reduction, implies a very high chance of overall better results if label reduction is included in the algorithm.

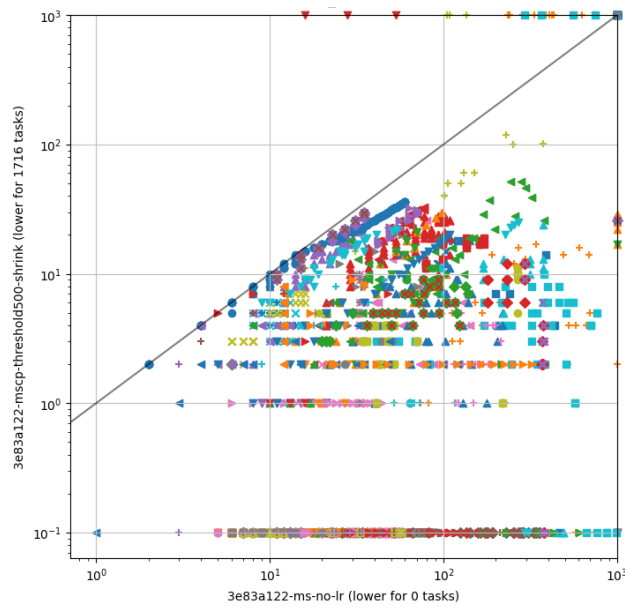


Figure 4.13: Versus Merge-and-Shrink - Number of Merges

Moving on to Figure 4.13 showing the number of merges, the expected behaviour can be observed. As the merge-and-shrink algorithm is used to merge all the transition systems until only one is left, but the algorithm suggested for this thesis only merges based on the quality of the resulting synchronised product, the number of merges is always higher for all the benchmarks. However, there are a few cases in which the number of merges is equal or of very close value, such as instances in *miconic*, *blocks* or *scanalyzer-opt11-strips*. Most cases indicate smaller problems with higher initial heuristic values. There are also cases such as *pipesworld-notankage*, where we merge considerably less, as from one point, most synchronised products of transition systems offer a quality of 0.

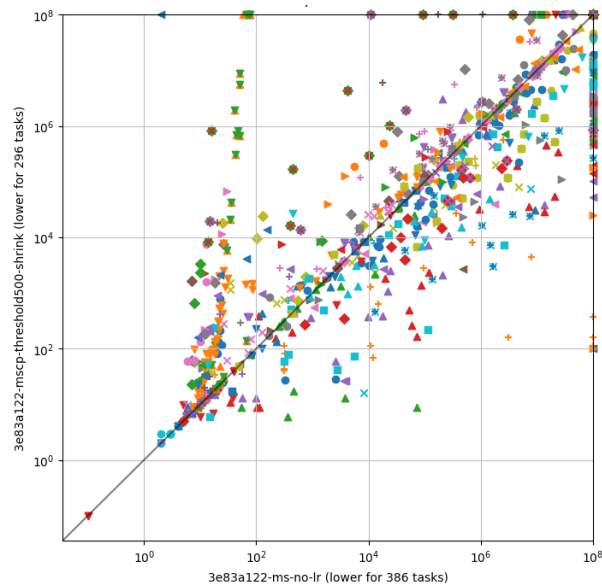


Figure 4.14: Versus Merge-and-Shrink - Expansions

Looking at expansions in Figure 4.14, we can see a rather balanced representation, comparing merge-and-shrink without label reduction with the main approach chosen for this thesis. The fewer values on the upper side suggest a more valuable initial heuristic for the algorithm from the scope of this thesis for those instances. This implies that there are problems that merge-and-shrink can solve better, but also many instances that the algorithm of this thesis can solve more efficiently. There are clearly cases where we perform better, showing once again the potential that this idea has.

4.7 Comparing to Sievers et al Paper [11]

This section takes a closer look at the previous work on the same subject, done by Sievers et al [11]. Similarly, as the original merge-and-shrink algorithm, their approach uses label reduction, a feature that was not included in this work due to time limitations. Moreover, slightly different benchmarks have been used. Therefore, there can be an offset in the values presented in this section.

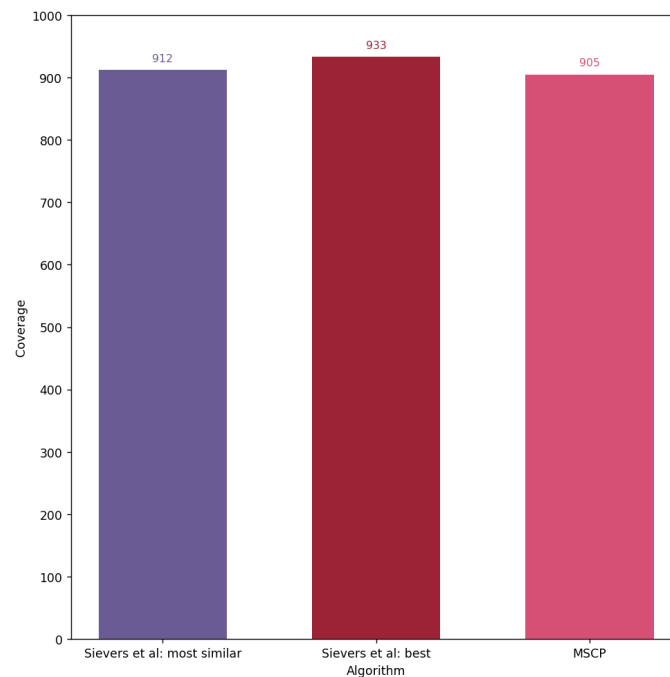


Figure 4.15: Versus Sievers et al [11] - Coverage

Figure 4.15 is another indicator towards the potential for further research of this approach. Having in mind that their algorithm also used label reduction, this can be one more hit at the improvement this feature can bring to our approach. The longer period of time associated with their work can be observed upon the high value of coverage and closer attention to the amount of testing for parameters. There are also similarities, such as the shrinking strategy used throughout the experiments, a non-greedy shrink bisimulation, as well as saturated cost partitioning. While the experiments behind our approach consider only random

orders, their approach also takes into consideration greedy and fixed orders. However, the highest results have been encountered using random orders. While our idea is based mainly on the quality value of merging two transition systems compared to the cost partitioning of the atomic projections, their approach relies on taking a single snapshot of each iteration in the merge-and-shrink algorithm, computing one cost partitioning heuristic over the input transition systems.

The next chapter will conclude the thesis with a brief reflection on the course of the study and an overview of possible future work that has great potential to boost the results and offer greater value to this implementation.

5

Conclusion

This last chapter offers a concluding step to this thesis, putting together all the notions and concepts studied throughout the six months to gain a deeper understanding of cost partitioning, merge-and-shrink heuristics, the way they eventually influence each other, and how we could use this connection to achieve better efficiency and value of results.

5.1 Reflection

Taking a retrospective look at the duration of this project, I am grateful to admit that I have accomplished all the aims and objectives set for the development. Even if not everything set in mind from scratch has been included in the final implantation of the project and there were slight deviations from the initial plan, I consider this a valuable learning opportunity that resulted in a worthwhile product. It is satisfying to see that my approach to integrating cost partitioning with merge-and-shrink techniques brought significant findings. I am delighted that this thesis has contributed to classical planning and has the potential to pave the way for further research in this area.

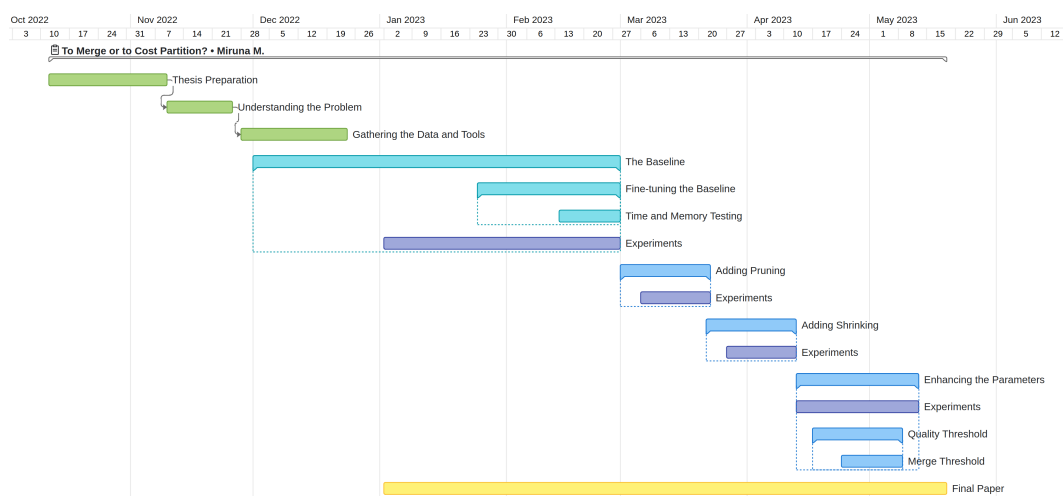


Figure 5.1: Gantt Chart of Project Timetable

In the above-displayed picture, my project's Gantt Chart contains all the relevant stages of my master's thesis experience, along with the duration associated with them. Starting the research earlier, during the thesis preparation, helped me get a better understanding of the project and the period that needs to be connected with each step of the project. My biggest regret is not being able to implement further improvements for this approach, mainly due to the very limited time, thus allocating more room for delivering my achievements in the best way that was intended from the beginning of the project.

5.1.1 Future Work and Further Improvements

As the time period for this project, as a master thesis, is limited, some ideas were left out of the final result that could have improved the results. Among the ideas I had regarding the enhancement of the implementation, I consider the following improvements very probable to sky-rocket the efficiency, effectiveness and value of my project. There have been attempts at addressing these implementations, but the limited time allows me to only theoretically present them and their purpose in extending the current version of the thesis.

Look-ahead Factor

One possible improvement is the option to know how much value a decision to merge can bring in future iterations. This would serve as a look-ahead factor that tells us if choosing to merge the transition systems would bring us a larger quality later in the process. Hence, let us take as an example 2 pairs to merge with the same quality of 4. Currently, the tie-breaking is made completely random, and one pair is chosen to proceed with. In the case of this new possible implementation, the algorithm computes the quality in the next iteration based on this iteration's calculations - if the first pair with current quality 4 assures the best quality next iteration of 3, but the other pair will guarantee the next quality to be 6, the second pair will be chosen. This addition could catch some edge cases not yet considered in this thesis, such as abstractions without goal variables, that we would like to merge as often as possible with the hope of better future results.

Non-Linear Merge Strategy

The next interesting extension of this thesis would be to weigh in non-linear merge strategies [1]. In this case, the main idea refers to being able to merge a transition system more times. Basically, in the current implementation, once a transition system is merged, it becomes inactive. This can cause a lower number of overall merges and, sometimes, even a lower heuristic value. With non-linearity, we can simulate parallelism, being very useful in very large problems with a very large amount of states [10]. This presents itself as a very flexible approach that could boost the overall coverage significantly. However, with this, all transition systems must remain alive during the process, potentially resulting in unwanted colossal memory usage, which we are trying to avoid.

Label Reduction

As part of the base merge-and-shrink algorithm, label reduction is a very probable further improvement that might boost this implementation. The main idea of label reduction refers to recognising which transition labels can be combined into one label with the property of still maintaining all relevant information [9]. Even if our approach assures better memory usage, we still face large numbers for the total size used. Therefore, label reduction presents the potential to lower the size of the transition system representation by fusing parallel transitions with distinct labels into one, simplifying the underlying problem.

Non-Random Saturated Cost Partitioning Order

We have talked about different improvements over the merging part of the algorithm, but cost partitioning can also come with a set of improvements. So far, our cost partitioning approach only considers random orders. One possibility that we have thought about is being able to specify a specific order. While we deal with a small enough number of heuristics, we can generate a good result most of the time by diversifying random orders. But, with the increase in the number of heuristics and, subsequently, the likelihood of orders, we need more concrete and fast options rather than waiting for the best one to perhaps happen [5].

5.2 Conclusion

To sum everything up, merge-and-shrink gives us a perfect product, while cost partitioning is not the case. But, using these techniques combined, we can decide whether doing such an expensive action as merging is the right choice. Merging brings in a significant memory increase, and it does not always guarantee a more informative result. This is where cost partitioning comes in very useful - we can calculate the heuristic of the synchronised product and compare it with the sum of the heuristics for the atomic projections. Now, if the value of the merged product is higher, it pays off to do the action despite the memory boost. However, if we face a value equal or smaller, there is no new information; therefore, merging would not bring anything new to the table. Thus, this thesis has presented a novel approach to integrate cost partitioning with merge-and-shrink heuristics, proposed for searching algorithms in solving classical planning problems by using a quality measurement to guide and control the number of merges. The experimental results, conducted on a range of benchmark domains on the Fast Downward planning system, have demonstrated the perks and drawbacks of this method by showing an improvement in coverage of approximately 10% compared to cost partitioning and of 5% compared to merge-and-shrink. By combining the strengths of both ways, we have successfully achieved our desired purpose of fusing merge and shrink with cost partitioning, with a 30% decrease in expanded states than merge-and-shrink and over 1000% lower expansions compared to cost partitioning. The suggested approach has the potential to be further studied and could serve as a basis for future research in classical planning. The thesis successfully reached its purpose, bringing a more detailed and experimental point of view to offer more certain answers to the question:

To merge or to cost partition?

Bibliography

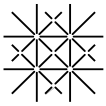
- [1] Gaojian Fan, Martin Müller, and Robert Holte. Non-linear merging strategies for merge-and-shrink based on variable interactions. In *Proceedings of the Seventh Annual Symposium on Combinatorial Search, SOCS 2014, Prague, Czech Republic, 15-17 August 2014*. AAAI Press, 2014.
- [2] Michael Katz and Carmel Domshlak. Optimal admissible composition of abstraction heuristics. *Artificial Intelligence*, 174(12):767–798, 2010.
- [3] Florian Pommerening, Malte Helmert, Gabriele Röger, and Jendrik Seipp. From non-negative to general operator cost partitioning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 3335–3341, 2015.
- [4] Luis Henrique Oliveira Rios and Luiz Chaimowicz. A survey and classification of a* based best-first heuristic search algorithms. In *Advances in Artificial Intelligence - SBIA 2010 - 20th Brazilian Symposium on Artificial Intelligence, São Bernardo do Campo, Brazil, October 23-28, 2010. Proceedings*, volume 6404 of *Lecture Notes in Computer Science*, pages 253–262. Springer, 2010.
- [5] Jendrik Seipp. Better orders for saturated cost partitioning in optimal classical planning. In *Proceedings of the Tenth International Symposium on Combinatorial Search, SOCS 2017, 16-17 June 2017, Pittsburgh, Pennsylvania, USA*, pages 149–154. AAAI Press, 2017.
- [6] Jendrik Seipp, Thomas Keller, and Malte Helmert. Saturated cost partitioning for optimal classical planning. *J. Artif. Intell. Res.*, 67:129–167, 2020.
- [7] Silvan Sievers. Merge-and-shrink heuristics for classical planning: Efficient implementation and partial abstractions. In *Proceedings of the Eleventh International Symposium on Combinatorial Search, SOCS 2018, Stockholm, Sweden - 14-15 July 2018*, page 99. AAAI Press, 2018.
- [8] Silvan Sievers and Malte Helmert. Merge-and-shrink: A compositional theory of transformations of factored transition systems. *J. Artif. Intell. Res.*, 71:781–883, 2021.
- [9] Silvan Sievers, Martin Wehrle, and Malte Helmert. Generalized label reduction for merge-and-shrink heuristics. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, pages 2358–2366. AAAI Press, 2014.

-
- [10] Silvan Sievers, Martin Wehrle, and Malte Helmert. An analysis of merge strategies for merge-and-shrink heuristics. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016*, pages 294–298. AAAI Press, 2016.
- [11] Silvan Sievers, Florian Pommerening, Thomas Keller, and Malte Helmert. Cost-partitioned merge-and-shrink heuristics for optimal classical planning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4152–4160, 2020.

A

x	agricola-opt18-strips
+	airport
●	barman-opt11-strips
■	barman-opt14-strips
▲	blocks
▼	childsack-opt14-strips
◀	data-network-opt18-strips
▶	depot
◆	driverlog
x	elevators-opt08-strips
+	elevators-opt11-strips
●	floortile-opt11-strips
■	floortile-opt14-strips
▲	freecell
▼	ged-opt14-strips
◀	grid
▶	gripper
◆	hiking-opt14-strips
x	logistics00
+	logistics98
●	miconic
■	movie
▲	mprime
▼	mystery
◀	nomystery-opt11-strips
▶	openstacks-opt08-strips
◆	openstacks-opt11-strips
x	openstacks-opt14-strips
+	openstacks-strips
●	organic-synthesis-opt18-strips
■	organic-synthesis-split-opt18-strips
▲	parcprinter-08-strips
▼	parcprinter-opt11-strips
◀	parking-opt11-strips
▶	parking-opt14-strips
◆	pathways
x	pegsol-08-strips
+	pegsol-opt11-strips
●	petri-net-alignment-opt18-strips
■	pipesworld-notankage
▲	pipesworld-tankage
▼	psr-small
◀	rovers
▶	satellite
◆	scanalyzer-08-strips
x	scanalyzer-opt11-strips
+	snake-opt18-strips
●	sokoban-opt08-strips
■	sokoban-opt11-strips
▲	spider-opt18-strips
▼	storage
◀	termes-opt18-strips
▶	tetris-opt14-strips
◆	tidybot-opt11-strips
x	tidybot-opt14-strips
+	tpp
●	transport-opt08-strips
■	transport-opt11-strips
▲	transport-opt14-strips
▼	trucks-strips
◀	visitall-opt11-strips
▶	visitall-opt14-strips
◆	woodworking-opt08-strips
x	woodworking-opt11-strips
+	zenotravel

Figure A.1: Benchmarks



Erklärung zur wissenschaftlichen Redlichkeit und Veröffentlichung der Arbeit (beinhaltet Erklärung zu Plagiat und Betrug)


Titel der Arbeit: To Merge or to Cost Partition?

Name Beurteiler*in: Dr Gabriele Roeger

Name Student*in: Miruna-Alesia Muntean

Matrikelnummer: 21-062-906

Mit meiner Unterschrift 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.


Ort, Datum: Basel, 20.05.2023 Student*in: 

Wird diese Arbeit veröffentlicht?

Nein

Ja. Mit meiner Unterschrift bestätige ich, dass ich mit einer Veröffentlichung der Arbeit (print/digital) in der Bibliothek, auf der Forschungsdatenbank der Universität Basel und/oder auf dem Dokumentenserver des Departements / des Fachbereichs einverstanden bin. Ebenso bin ich mit dem bibliographischen Nachweis im Katalog SLSP (Swiss Library Service Platform) einverstanden. (nicht Zutreffendes streichen)

Veröffentlichung ab: 20.05.2023

Ort, Datum: Basel, 20.05.2023 Student*in: 

Ort, Datum: _____ Beurteiler*in: _____

Diese Erklärung ist in die Bachelor-, resp. Masterarbeit einzufügen.