UNIVERSITÄT BASEL

An algorithm for computing bisimulations in planning

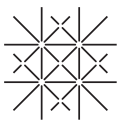
Bachelor Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science

Examiner: Malte Helmert
Supervisor: Martin Wehrle

Sascha Scherrer
s.scherrer@unibas.ch
09-058-454

31.07.2012



Acknowledgments

I would like to thank Prof. Dr. Malte Helmert for suggesting the topic for my bachelor thesis as well as for the opportunity to write this thesis in the area of artificial intelligence. I further would like to thank Dr. Martin Wehrle for his support and suggestions during the last three months.
Finally, I thank my friends, colleagues and my family for supporting and occasionally distracting me.

Abstract

Merge-and-shrink abstractions are a popular approach to generate abstraction heuristics for planning. The computation of merge-and-shrink abstractions relies on a *merging* and a *shrinking* strategy. A recently investigated shrinking strategy is based on using *bisimulations*. Bisimulations are guaranteed to produce perfect heuristics. In this thesis we investigate an efficient algorithm proposed by Dovier et al. [2004] for computing coarsest bisimulations. The algorithm however cannot directly be applied to planning and needs some adjustments. We show how this algorithm can be reduced to work with planning problems. In particular, we show how an edge labelled state space can be translated to a state labelled one and what other changes are necessary for the algorithm to be usable for planning problems. This includes a custom data structure to fulfil all requirements to meet the worst case complexity. Furthermore, the implementation will be evaluated on planning problems from the International Planning Competitions. We will see that the resulting algorithm can often not compete with the currently implemented algorithm in Fast Downward. We discuss the reasons why this is the case and propose possible solutions to resolve this issue.

Table of Contents

1 Introduction

Domain independent planning concerns the problem of searching and finding a sequence of actions from a start to a goal state. It can be used to find - optimal, if needed - solutions of complex problems. Planning can be done as heuristic search using a heuristic function to evaluate states and select the most promising one. The search is done by maintaining an open and a closed list. The former one contains all states that are reachable and have not yet been visited, the latter one contains all visited states. At each iterative step a state from the open list is chosen and then visited. Selecting the state in the open list with the lowest value returned by the heuristic function yields the state that is nearest to the goal (according to the heuristic function). Visiting a state renders all its successors reachable and if they aren't in either the open or the closed list they get added to the open list. Once a goal state is reached, the solution can be extracted from the closed list.

One category of heuristic functions are *abstraction heuristics*. Simplifying the search space by abstracting states yields a smaller search space which can be used to estimate the distance to a goal by measuring abstract goal distances. An example for such an abstraction heuristic is *merge-and-shrink*. Merge-and-shrink has originally been proposed by Dräger et al. [2006] in the context of model checking and Helmert et al. [2007] in the context of planning. Merge-and-shrink uses - as the name suggests - a merging and a shrinking strategy. It works on a set of atomic projections of the state space and iteratively selects two (by using the merging strategy) and then decides which needs to be shrunk and actually shrinks it (shrinking strategy). Nissim et al. [2011] suggested the use of bisimulation for computing the abstraction and also noted that such an abstraction heuristic is perfect.

A bisimulation is a binary relation between two state spaces under which they exhibit the same behaviour. Especially of interest is the coarsest bisimulation of a given state space, as it yields the smallest abstraction, and its computation. Dovier et al. [2004] introduced 'an efficient algorithm for computing bisimulation equivalence' in the context of model-checking using the partition refinement algorithm by Paige and Tarjan [1987].

The target of this thesis is to adapt and implement the algorithm by Dovier et al. [2004] for the use in planning. To be able to use it on a planning problem, we first need to translate the edge labelled state space into a state labelled state space by replacing all edges with states. The state space needs to be adjusted further so the algorithm correctly handles

the goal states. A custom data structure will be introduced to maintain all abstract states during the refinement.

2
Background

2.1 Planning

2.1.1 Notation

A planning task is a tuple (V, O, s_0, s_*) where V is a finite set of variables $v \in V$, each having its own finite domain D_v. A state over V is a function s with $s(v) \in D_v \; \forall v \in V$. A partial state over V is a state over a subset $V_s \subset V$ which means that not all variables are relevant. The initial state s_0 is a state while the goal s_* is a partial state. O is a final set of operators, each consisting of a pair of partial states representing the necessary preconditions and the effect of applying the operator. An operator is applicable if the current state fulfils the mapping of the preconditions. The successors of a state are all states that can be reached by applying an operator to the current state.

The labelled transition system of such a task is usually represented as a tuple (S, L, T, s_0, S_*), where S is a final set of states, L a final set of (transition) labels, T a set of labelled transition defined as $T \subseteq S \times L \times S$, $s_0 \in S$ is the start state and $S_* \subseteq S$ are the goal states.

A plan is the path from the start state to any of the goal states. A given plan is optimal if and only if its cost (or length if all transitions are unit cost) is equal to the cost of the shortest possible path from the start state to any goal state.

A heuristic is a function defined as $h : S \to \mathbb{N}_0 \cup \{\infty\}$ and is an estimator for the distance to a goal state. A given heuristic is admissible if the estimated distance is lower or equal to the actual distance for every state. A given heuristic is consistent if for every pair of connected states $(s_1, l, s_2) \in T$ the following holds: $h(s_1) \leq h(s_2) + 1$ (unit cost variant of the triangle inequality) and $h(s) = 0$ *for all* $s \in S_*$.

Planning can be done by heuristic search of the state space. Algorithm 1 shows pseudocode of such a search algorithm. Note that the definition of minimum on line 4 depends on the actual search algorithm (e.g. a greedy best first search uses the smallest heuristic value). The function *solution*() returns - as the name suggests - the solution.

An abstraction is a simplified and preferably smaller version of a transition graph gained by ignoring or concentrating some information. An abstraction is defined by its abstraction mapping function $\alpha : S \to S'$. The abstract transition graph $T' = (S', L', T', s'_0, S'_*)$ is calculated as followed: $L' = L$, $(\alpha(s), l, \alpha(s')) \in T' \; \forall (s, l, s') \in T$, $\alpha(s_0) = s'_0$ and $\alpha(s_*) \in$

$S'_* \forall s_* \in S_*$.

Algorithm 1 Heuristic search using a consistent heuristic function

```
1: open := {root}
2: closed := {}
3: while open not empty do
4:    n = pop min(open)
5:    if n not in closed then
6:       insert n into closed
7:       if n is goal then
8:          return solution(n)
9:       end if
10:      for all successor n' of n do
11:         if h(n') < ∞ then
12:            insert n' into open
13:         end if
14:      end for
15:   end if
16: end while
17: return no solution
```

A projection of a planning task is an abstraction which only uses a subset V' of the tasks variables. It maps states with the same values for all variables in the subset .

$$\alpha(s_1) = \alpha(s_2) \textit{ iff } s_1(v) = s_2(v) \; \forall v \in V' \tag{2.1}$$

An atomic projection is a projection which only uses a single variable v and is written as π_v.

2.1.2 Merge-and-shrink

Merge-and-shrink is an algorithm for calculating an abstraction using a *merging* strategy and a *shrinking* strategy. It starts with a set of all atomic projections of the given task and iteratively merges and shrinks (so their size after merging remains under a given size bound N) two abstractions at a time until only a single abstraction is left. The size of an abstraction is defined as the number of abstract states it contains. The merging strategy selects the abstractions to use in a given iteration while the shrinking strategy decides which of the two abstractions should be shrunk and how this should be done. The synchronized product $S^{\otimes} = (S_{\otimes}, L_{\otimes}, T_{\otimes}, s^0_{\otimes}, S^*_{\otimes})$ of two transition systems $S^1 = (S_1, L_1, T_1, s^0_1, S^*_1)$ and $S^2 = (S_2, L_2, T_2, s^0_2, S^*_2)$ is defined as followed: $S^{\otimes} = S^1 \otimes S^2$, $S_{\otimes} = S_1 \times S_2$, $s^0_{\otimes} = (s^0_1, s^0_2)$, $S^*_{\otimes} = S^*_1 \times S^*_2$, $T_{\otimes} = \{((s^s_1, s^s_2), l, (s^t_1, s^t_2)) \mid (s^s_1, l, s^t_1) \in T_1, (s^s_2, l, s^t_2) \in T_2\}$
The heuristic value is then calculated by measuring the the shortest distance of every state to a goal state. One possible way to shrink is to compute the coarsest bisimulation.

Algorithm 2 Merge-and-shrink

1: $abs := \{\pi_v | v \in V\}$
2: **while** $|abs| > 1$ **do**
3: *Select* $\mathcal{A}_1, \mathcal{A}_2 \in abs$
4: *Shrink* $\mathcal{A}_1 and/or \mathcal{A}_2$ *until* $size(\mathcal{A}_1) \cdot size(\mathcal{A}_2) \leq N$
5: $abs := (abs \setminus \{\mathcal{A}_1, \mathcal{A}_2\}) \cup \{\mathcal{A}_1 \otimes \mathcal{A}_2\}$
6: **end while**
7: **return** *only element of abs*

2.2 Bisimulation equivalence

2.2.1 Definition

A bisimulation is a binary relation $\sim$ over two transition systems S_1, S_2 defined as followed: For every pair of states $p \in S_1, q \in S_2$ in the bisimulation ($p \sim q$): if there is a $p' \in S_1$ with $p \to p'$ then there is a $q' \in S_2$ with $q \to q'$ and if there is a $q' \in S_2$ with $q \to q'$ then there is a $p' \in S_1$ with $p \to p'$ and $p' \sim q'$.

Two transition systems S_1 and S_2 are bisimilar if for all initial states in S_1 there exists a bisimilar initial state in S_2 and vice versa. A simple example with the mapping (dashed arrows) between the two systems is given in Figure 2.1. Note that while $n2$ maps to $n1'$, $n1'$ maps to $n1$.

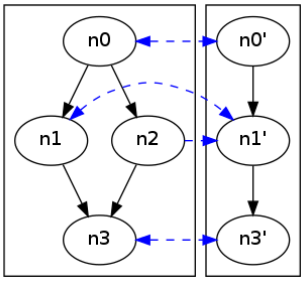


Figure 2.1: Two bisimilar graphs and their mapping

As stated by Nissim et al. [2011], there always exists a coarsest bisimulation (e.g. the right transition system in Figure 2.1) and this coarest bisimulation can be used to calculate a perfect heuristic.

3

Efficient algorithm for computing bisimulation equivalence

This chapter will introduce the algorithm used for computing the coarsest bisimulation and the idea behind it as well as what needs to be changed to be able to use it for planning.

3.1 The algorithm

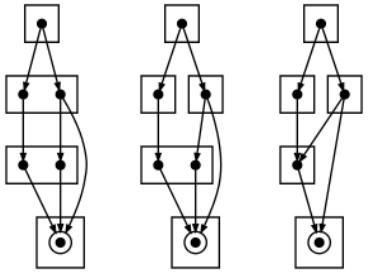
3.1.1 Notation

A partition of a set N is a set of pairwise disjoint subsets (called blocks) so that each element of N is in exactly one subset. The union of all blocks yields the set N and the intersection of any two blocks is always the empty set. Each block must hold at least one element of N.
A refinement P of a partition X is a partition with $\forall p \in P, \exists x \in X \ \textit{with}\ p \subseteq x$.
Given a binary relation E and its inverse relation E^{-1} on a set N is a subset of $N \times N$. A partition P of N is *stable* with respect to E if for every pair of blocks $B_1, B_2 \in P$ either $B_1 \subseteq E^{-1}(B_2)$ or $B_1 \cap E^{-1}(B_2) = \{\}$.
The subgraph $G' = \langle B, E \restriction B \rangle$ of $G = \langle N, E \rangle$ consists of the states of a subset $B \subset N$ of the states in N and the edges between them $E \restriction B := (B \times B) \cap E$.

3.1.2 Idea

Dovier, Piazza, and Policriti [2004] introduced an efficient algorithm for computing bisimulation equivalence which can be used to calculate the coarsest bisimulation of state-labelled graphs (a transition system with labelled states an no labels on edges). The algorithm uses a negative strategy by starting with the coarsest partition $P = \{N\}$ of a graph $G = \langle N, E \rangle$ and then splitting the blocks as long as P is not stable. The algorithm uses the idea of separating parts of the graph that aren't bisimilar and refining them separately. They define a rank and group all states with the same rank together. Figure 3.1 shows a simple example (the algorithm will be explained later on) with 4 rank induced blocks. Collapsing a block replaces all nodes in it with a new one inheriting all incoming and outgoing edges of the replaced nodes. In Figure 3.1a the block containing the goal state (on the bottom) gets

collapsed (has no effect). In Figure 3.1b the 2nd block from the top gets refined (separating connected from not connected states). In Figure 3.1c the 2nd block from the bottom gets collapsed.



(a) collapse (b) refine (c) collapse

Figure 3.1: Minimisation (well founded case)

3.1.3 Rank definitions

3.1.3.1 Well founded case

For an acyclic graph $G = \langle N, E \rangle$, the following recursive definition of rank can - according to Dovier, Piazza, and Policriti [2004] - be used to make the above stated separation:

$$rank(n) = \begin{cases} 0 & \textit{if } n \textit{ is a leaf} \\ 1 + max\{rank(m) : \langle n, m \rangle \in E\} & \textit{otherwise} \end{cases} \tag{3.1}$$

With this definition of rank, states that are bisimilar always get assigned the same rank. The opposite does not hold true.

3.1.3.2 General case

To be able to compute a rank for general graphs (i.e. cyclic graphs) the definition must be generalised. The following definitions are needed.
A strongly connected component of a given graph $G = \langle N, E \rangle$ is a set of states where each state can be reached from each state of this set. A graph $G^{scc} = \langle N^{scc}, E^{scc} \rangle$ built from the strongly connected components of an other graph is always acyclic.

$$\begin{aligned} N^{scc} &= \{c | c \textit{ is a strongly connected component of } G\} \\ E^{scc} &= \{\langle c_1, c_2 \rangle | c_1 \neq c_2 \textit{ and } (\exists n_1 \in c_1)(\exists n_2 \in c_2)(\langle n_1, n_2 \rangle \in E)\} \end{aligned} \tag{3.2}$$

The well founded set of a graph is a set of all states from which no acyclic state is reachable. The subgraph $G(n)$ of a graph $G = \langle N, E \rangle$ contains all states reachable from n. The well founded part of a graph $G = \langle N, E \rangle$ is defined as $WF(G) = \{n \in N | G(n) \textit{ acyclic}\}$.

$$rank(n) = \begin{cases} 0 & \textit{if } n \textit{ is a leaf} \\ -\infty & \textit{if } c(n) \textit{ is a leaf in } G^{scc} \textit{and } n \textit{ is not a leaf in } G \\ max(\{1 + rank(m) | \langle c(n), c(m) \rangle \in E^{scc}, m \in WF(G)\} \cup & \\ \quad \{rank(m) | \langle c(n), c(m) \rangle \in E^{scc}, m \notin WF(G)\}) & \textit{otherwise} \end{cases} \tag{3.3}$$

3.1.4 Paige Tarjan Partition Refinement

A given graph $G = \langle N, E \rangle$ can be interpreted as binary relation E on the set of states contained in G. The expression aEb for two states $a, b \in N$ is equivalent with $\langle a, b \rangle \in E$ meaning that there is an edge in G from a to b. Also $E^{-1}(Y) = \{x | \exists y \in Y \textit{ with } xEy\}$ are all predecessors of the states in Y. The partition refinement algorithm as proposed by Paige and Tarjan [1987] can be used to efficiently calculate the coarsest stable partition. It is the underlying algorithm for computing the coarsest bisimulation. The following pseudocode is a simple version of the efficient algorithm.
Splitting a block b with respect to some set s (as done on line 6 of Algorithm 3) yields two new blocks $b_1 = b \cap s$ and $b_2 = b \setminus (b \cap s)$. After splitting b with respect to s, these two sets fulfil the requirement(see Section 3.1.1) for the partition to be stable.

Algorithm 3 Compute coarsest stable partition of N

```
1: P = {N}
2: while P not stable do
3:    select b ∈ P
4:    compute s = E^{-1}(b)
5:    for all p ∈ P do
6:       split p with s
7:    end for
8: end while
```

In the actual implementation a more sophisticated version - also proposed by Paige and Tarjan [1987] - is used. Especially the selecting of a good splitter (one that actually splits some blocks) on line 3 is crucial for the algorithm. The more sophisticated version has a runtime complexity of $\mathcal{O}(|E| \cdot log(|N|))$.

3.1.5 Description

Below in Algorithm 4 the procedure to compute the coarsest bisimulation as proposed by Dovier et al. [2004] is shown. First it computes the rank of every state using the definition in Section 3.1.3.2. After the rank of every state is known, all states with rank $-\infty$ (which are dead ends in the transition graph) get collapsed and blocks at higher ranks get refined. This refinement step is done by splitting the blocks with both elements that are connected to the collapsed block and ones that are not. This refinement of blocks at higher ranks is also done after each collapse at these ranks. Next comes the main loop of the algorithm in which the higher ranks get processed. First, a new subgraph with all blocks at the current

rank (defined by the iteration) gets created and and refined with the Paige-Tarjan algorithm. After this refinement is done, the resulting blocks get collapsed and for every such block, all blocks at higher ranks get refined by splitting connected elements and not connected ones. The result after the main loop is a graph containing the coarsest bisimulation.
The algorithm has a runtime complexity of $\mathcal{O}(|E| \cdot log(|N|))$ as stated by Dovier, Piazza, and Policriti [2004].

Algorithm 4 Compute coarsest bisimulation

1: **for** $n \in N$ **do**
2: compute $rank(n)$;
3: **end for**
4: $p := max\{rank(n)|n \in N\}$;
5: **for** $i = -\inf, 0, ..., p$ **do**
6: $B_i := \{n \in N|rank(n) = i\}$
7: **end for**
8: $P := \{B_i|i = -\infty, 0, ..., p\}$;
9: $G := collapse(G, B_{-\infty})$;
10: **for** $n \in N \cap B_{-\infty}$ **do**
11: **for** $C \in P$ **and** $C \neq B_{-\infty}$ **do**
12: $P := (P \setminus \{C\}) \cup \{\{m \in C|\langle m, n\rangle \in E\}, \{m \in C|\langle m, n\rangle \notin E\}\}$
13: **end for**
14: **end for**
15: **for** $i = 0, ..., p$ **do**
16: $D_i := \{X \in P|X \subseteq B_i\}$;
17: $G_i := \langle B_i, E \restriction B_i\rangle$;
18: $D_i := Paige - Tarjan(G_i, D_i)$;
19: **for** $X \in D_i$ **do**
20: $G := collapse(G, X)$;
21: **end for**
22: **for** $n \in N \cap B_i$ **do**
23: **for** $C \in P$ **and** $C \subseteq B_{i+1} \cup ... \cup B_p$ **do**
24: $P := (P \setminus \{C\}) \cup \{\{m \in C|\langle m, n\rangle \in E\}, \{m \in C|\langle m, n\rangle \notin E\}\}$
25: **end for**
26: **end for**
27: **end for**

3.2 Translation to planning

The algorithm described can be adapted to be used for planning. As stated in Section 3.1, this algorithm works on a state-labelled graphs whereas we want one that works on edge labels as planning is modelled this way. The algorithm also doesn't work as expected on a graph which has only one strongly connected component (so each state can be reached from each state) because it doesn't handle goal states in any way. This problem will be discussed in Section 3.2.2.

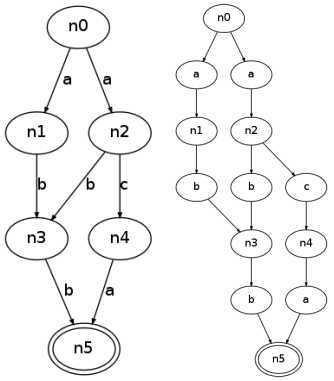
3.2.1 Edge labels to state labels

As suggested by Dovier et al. [2004], an edge-labelled graph can be converted in an equivalent state-labelled graph by simply replacing all edges with states.
This conversion to state labels has a drawback, it can increases the amount of states and edges in the graph significantly. By replacing the edges with states we double the amount of edges and increase the amount of states by the amount of edges.

$$|N'| = |N| + |E| \tag{3.4}$$

$$|E'| = |E| + |E| \tag{3.5}$$

So after the change the worst case complexity is $\mathcal{O}(|E| \cdot log(|E| + |N|))$. If the number of edges per state in a given problem is linked to the amount of states (i.e. with increasing number of states the number of edges per state increases) this transformation also increases the worst case complexity (in addition to the worse constant factor) of the algorithm. This increase in worst case complexity is problem specific and may vary strongly. An example of how this translation works and how the number of states and edges increases can be seen in Figure 3.2.



(a) edge labels (b) state labels

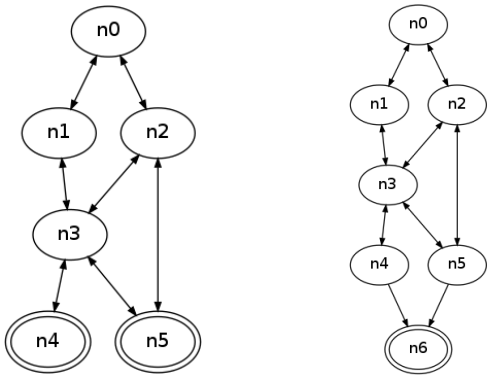
Figure 3.2: Transformation to state-labelled graph

In addition to the algorithm in Section 3.1 we need to make sure that states with different labels are never in the same equivalence group. To fulfil this requirement, we split all blocks after calculating the ranks into smaller blocks only containing one kind of state label each.

3.2.2 Avoiding one single equivalence group

As stated before, the algorithm behaves unexpectedly if all states are in one single strongly connected component and therefore have a rank of $-\infty$. A single strongly connected component can only happen if every state is reachable from every state (happens quite often in planning). This leads to the collapsing of all states into a single one. To solve this problem without changing the definition of rank or modifing the algorith proposed by Dovier et al. [2004] an additional state can be introduced representing a single goal state and connection all actual goal states to this new one. After this change, the algorithm can be used without

specially handling these states. This modification is displayed in Figure 3.3.



(a) multiple goal states

(b) single goal state

Figure 3.3: Transformation to state-labelled graph

Although the problem above can be solved by introducing a new state, it doesn't change the fact that the given definition of rank to split the task into smaller tasks yields only one rank for the whole graph for typical AI tasks. This is because actions can often be reversed by an opposite action (like making a step backwards). These action leading away from the goal states lead to the problem where all states are in one strongly connected component and therefore the same rank is assigned to all states. The algorithm looses an important optimization this way.

4 Implementation

All algorithms and code produced are written in C++.

4.1 Partition

To match up with the worst case complexity of $\mathcal{O}(|E| \cdot log(|N|))$ a special data structure with the following requirements is needed:

1. Get block of given element in $\mathcal{O}(1)$
2. Iterate over elements of a block B in $\mathcal{O}(|B|)$
3. Move element into different block in $\mathcal{O}(1)$

Given that the number of elements doesn't change during the refinement, an array can be used to store the elements, which allows access of individual elements in $\mathcal{O}(1)$. Storing a pointer to the block containing the element in each element fulfils requirement (1). To be able to iterate over all elements in a given block and also move the elements in $\mathcal{O}(1)$ a doubly linked list is a good choice. By storing the data needed for a doubly linked list (next and previous node of list) in the array introduced before requirements (2) and (3) are also met.

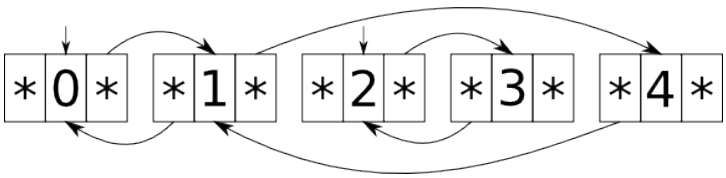


Figure 4.1: Example partition for 5 elements in 2 blocks
$P = \{\{0, 1, 4\}, \{2, 3\}\}$

The partition refinement algorithm by Paige&Tarjan needs a second (working) partition of the whole set for its calculation. In addition the blocks of the two partitions need to be linked if one is a subset of the other, therefore a list of pointers to these linked blocks is stored within each block.

4.2 Approximative bisimulation

As mentioned in Section 2.2.1 there always exists a coarsest bisimulation. If the upper bound for the amount of abstract states that the shrinking algorithm should return is lower than the number of abstract states in the coarsest bisimulation, an approximative bisimulation needs to be calculated. Because the algorithm proposed by Dovier et al. [2004] uses a negative strategy by starting with one abstract state and then splitting it until the coarsest bisimulation is reached, simply stopping before a split would increase the number of abstract states over the upper bound gives us an approximative bisimulation.

5 Evaluation

5.1 Fast Downward

The algorithm was implemented for the use within the Fast Downward planner by Helmert [2006]. Fast Downward contains many different algorithms to solve planning problems including a merge and shrink algorithm. The algorithm for computing the coarsest bisimulation is meant to be used as a shrinking strategy with the merge-and-shrink algorithm within Fast Downward.

5.2 Experimental setup

The newly, in conjunction with this thesis, implemented algorithm will be abbreviated as BD. Because Fast Downward already contains a merge-and-shrink heuristic using bisimulation (this algorithm will be abbreviated as FD from now on), all tests were compared with this implementation. The algorithm labelled BD2 is a preliminary version of the algorithm by Dovier et al. [2004] which works directly on edge labels and is still in development. It is included in the benchmarks to visualize the effect of translating edge labels to state labels. Tested were different domains, some known to be shrinkable using bisimulation and some that are not. All tests were done with and without an upper bound for the number of states. The test machine is equipped with 6GB of RAM, a Core 2 Duo with 2.66 Ghz and running linux (Kubuntu 12.04LTS).

5.3 Results

5.3.1 Gripper domain

The gripper domain is a scenario where there are two rooms and multiple balls which need to be transported from the first to the second room. There is also a robot who has multiple arms (usually two) and can pick up one ball (if it carries none) with each, move to a room and drop the ball(s). The problem size is increased by increasing the number of balls to transport to the second room. The gripper domain is a very good domain for calculating bisimulation as it has many bisimilar states.

5.3.2 Without upper bound

These tests were executed with no upper bound for the amount of abstract states and a threshold of 1, which means that the shrinking algorithm is executed for every abstraction no matter how big or small it is. The problem size increases with higher problem number. Time is measured in seconds needed to compute the abstraction and to search the solution. Memory (abbreviated Mem) is the peak memory usage. Figure 5.1 shows the measured time and memory usage and Figure 5.2 displays it in graphical form. The scale of the x-axis is a combination of the number of states and edges in the problem calculated as $x = |E| * log(|N|)$ for every data point. Figure 5.3

Problem	Time FD	Time BD	Time BD2	Mem FD	Mem BD	Mem BD2
1	0s	0s	0s	3160	3160	3160
2	0s	0.02s	0s	3160	3420	3288
3	0s	0.02s	0.02s	3288	3584	3292
4	0.02s	0.06s	0.04s	3400	3968	3416
5	0.04s	0.08s	0.08s	3580	4456	3656
6	0.08s	0.16s	0.12s	3732	5032	3980
7	0.12s	0.24s	0.2s	4024	5908	4284
8	0.18s	0.38s	0.3s	4304	6956	4688
9	0.24s	0.56s	0.44s	4644	8148	5172
10	0.34s	0.84s	0.62s	5204	9676	5792
11	0.48s	1.2s	0.86s	5796	11444	6568
12	0.66s	1.7s	1.16s	6408	13644	7488
13	0.86s	3s	1.62s	7440	15996	8556
14	1.12s	3.28s	1.96s	8352	18764	9740
15	1.42s	4.32s	2.5s	9496	21980	11184
16	1.76s	6.5s	3.28s	10792	25468	12812
17	2.4s	8.32s	4.12s	12564	29488	14608
18	2.76s	9.54s	5.12s	14484	33988	16652
19	3.36s	12.4s	7.82s	16748	38996	19028
20	4.1s	13.34s	9.94s	19012	44528	21692

Figure 5.1: Results gripper domain

5.3.3 With upper bound

A heuristic obtained by an approximative bisimulation is worse than a exact bisimulation and therefore it takes much longer to search for a solution to a problem. These tests were made with smaller problems and with an upper bound of 97% of the states an exact bisimulation would need. All values include searching for the solution (which takes much longer than computing the abstraction) as it is part of evaluating the abstraction. Figure 5.4 shows the measured values and the upper bound set.

5.4 Discussion

As seen in Figure 5.2 the memory usage of the new algorithm scales badly with the problem size. This comes from the fact that memory usage scales mainly with the number of states

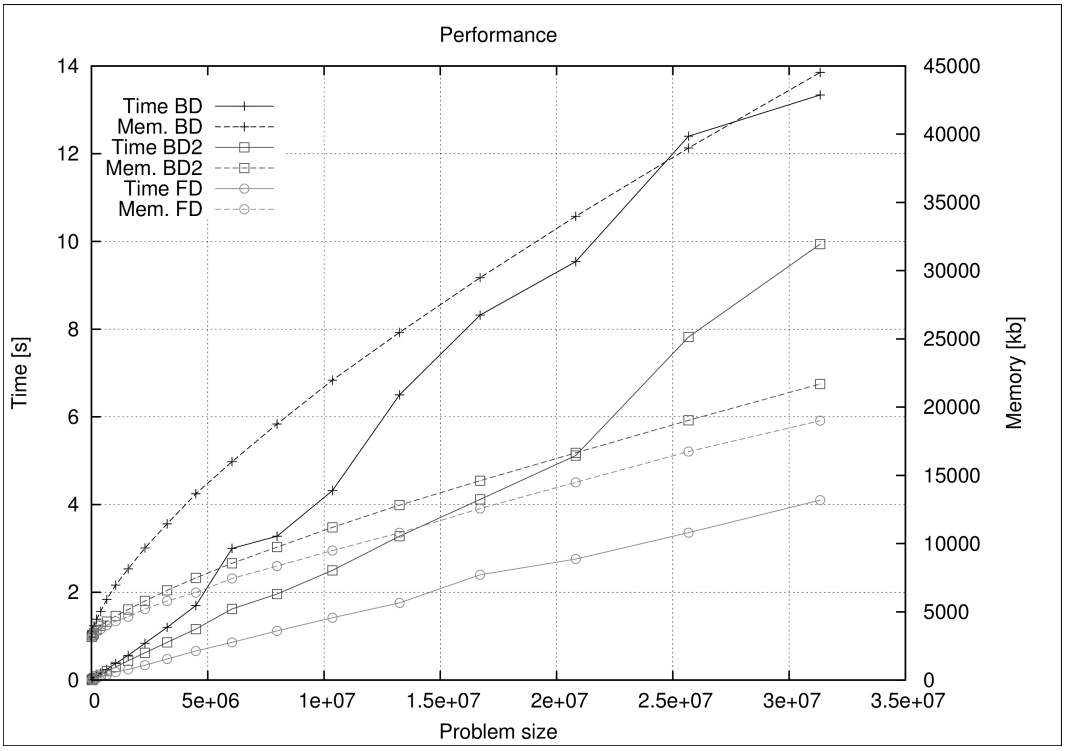


Figure 5.2: Results gripper domain

(higher constant factor than for edges). After translation from edge to state labels this new number of states scales with the old number of states and edges. This way the much higher number of edges gets multiplied with the higher constant factor.

Domain	Problem	FD	BD
airport	1	6.84	31.96
driverlog	6	4.82	26.00
blocks	6-0	1.50	4.22
elevator-opt11-strips	1	2.42	7.46
floortile-opt11-strips	1	2.04	7.56
miconic	9-0	4.10	11.60
mprime	1	10.54	27.00
mystery	1	0.54	34.78
No-mprime	1	10.40	not solvable
No-mystery	1	0.56	40.26
pegsol-opt11-strips	1	18.42	not solvable
Pipesworld-tankage	1	0.48	44.94
sokoban-opt11-strips	1	0.88	61.88
visitall-opt11-strips	4-full	1.02	1.76
woodworking-opt11-strips	1	4.64	22.76
zenotravel	8	8.22	35.68

Figure 5.3: Different domains

Problem	Max States	Limit	Time FD	Time BD	Mem FD	Mem BD
1	74	72	0	0	3.2	3.2
2	184	178	0	0	3.3	3.4
3	354	343	0.04	0.06	3.8	3.9
4	602	584	0.28	0.34	7.3	7.5
5	948	920	2.26	1.96	25.4	25.7
6	1394	1352	10.86	12.46	124.6	125
7	1962	1903	64.02	67.56	614	611.6

Figure 5.4: Gripper domain

6
Conclusion

As described before, multiple changes to the algorithm by Dovier et al. [2004] were necessary. In addition the implementation of the partition refinement algorithm by Paige and Tarjan [1987] was needed as well as a custom data structure to represent the partitions.
As the evaluation shows, the algorithm can in fact be adapted to correctly compute the coarsest bisimulation in the context of planning. Its performance in comparison to the already implemented algorithm within Fast Downward isn't as good as hoped for. One big problem is the much higher memory usage and the consequential slowdown that comes with it.
Further testing and developing of the edge label variant of the algorithm should be done as well as investigating the potential of this variant. As of now, this preliminary version seems to have more potential in planning than the direct attempt. Both variants of the algorithm need improvement for the case when there is an upper bound for the number of abstract states. Because of the way the partition refinement works it is not possible to directly stop refinement when the limit is reached. The current implementation stops refining before the actual limit is reached and therefore is not as exact as it could be.
The optimizations suggested by Dovier et al. [2004] and their usefulness as well as if they can be used with the edge label variant of the algorithm should be investigated. These suggested improvements tackle several problems including the case when there are only a few rank-induced blocks as it is usual for planning problems.

Bibliography

[Dovier et al., 2004] Agostino Dovier, Carla Piazza, and Alberto Policriti. An efficient algorithm for computing bisimulation equivalence. *Theor. Comput. Sci.*, 311(1-3):221–256, 2004.

[Dräger et al., 2006] Klaus Dräger, Bernd Finkbeiner, and Andreas Podelski. Directed model checking with distance-preserving abstractions. In Antti Valmari, editor, *SPIN*, volume 3925 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2006. ISBN 3-540-33102-6.

[Helmert, 2006] Malte Helmert. The fast downward planning system. *J. Artif. Intell. Res. (JAIR)*, 26:191–246, 2006.

[Helmert et al., 2007] Malte Helmert, Patrik Haslum, and Jörg Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In Mark S. Boddy, Maria Fox, and Sylvie Thiébaux, editors, *ICAPS*, pages 176–183. AAAI, 2007. ISBN 978-1-57735-344-7.

[Nissim et al., 2011] Raz Nissim, Jörg Hoffmann, and Malte Helmert. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Toby Walsh, editor, *IJCAI*, pages 1983–1990. IJCAI/AAAI, 2011. ISBN 978-1-57735-516-8.

[Paige and Tarjan, 1987] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, December 1987.

Declaration of Authorship

I hereby declare that this thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the bibliography and specified in the text.
This thesis is not substantially the same as any that I have submitted or will be submitting for a degree or diploma or other qualification at this or any other University.

Basel, 31.07.2012

Sascha Scherrer

Philosophisch-Naturwissenschaftliche Fakultät
der Universität Basel
Dekanat

Erklärung zur wissenschaftlichen Redlichkeit
(beinhaltet Erklärung zu Plagiat und Betrug)

(bitte ankreuzen)
☒ Bachelorarbeit
☐ Masterarbeit

Titel der Arbeit *(Druckschrift)*:

An algorithm for computing bisimulations in planning

Name, Vorname *(Druckschrift)*: Scherrer Sascha

Matrikelnummer: 09-058-454

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.

Diese Erklärung wird ergänzt durch eine separat abgeschlossene Vereinbarung bezüglich der Veröffentlichung oder öffentlichen Zugänglichkeit dieser Arbeit.

☐ ja ☒ nein

Ort, Datum: Basel, 31.07.2012

Unterschrift:

Dieses Blatt ist in die Bachelor-, resp. Masterarbeit einzufügen.