UNIVERSITÄT BASEL

Computation of h^{+} with Factored Planning

Master Thesis

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

Examiner:
Prof. Dr. Malte Helmert
Supervisors:
Gabriele Röger
Silvan Sievers

Gabriel Duss
g.duss@stud.unibas.ch
2008-054-942

31.07.2013

UNI
BASEL

Abstract

The main approach for classical planning is heuristic search. Many cost heuristics are based on the delete relaxation. The optimal heuristic of a delete free planning problem is called h^+. This thesis explores two new ways to compute h^+. Both approaches use factored planning, which decomposes the original planning problem to work on each subproblem separately. The algorithm reuses the subsolutions and combines them to a global solution.

The two algorithms are used to compute a cost heuristic for an A^* search. As both approaches compute the optimal heuristic for delete free planning tasks, the algorithms can also be used to find a solution for relaxed planning tasks.

Table of Contents

1 Introduction

Planning problems appear in many areas such as robotics, artificial intelligence of computer games or in logistics. For instance, a company manages its products in a warehouse where a robot collects ordered goods. The robot has to collect the items while considering important factors such as, which is the shortest path to the item, in which order should it collect the different items and how many items can it carry at the same time. The robot has to come up with a sequence of actions that is as cheap as possible to reach the goal. The task of finding a sequence that achieves this goal is called planning.

In order not to implement a new program for each planning problem, domain-independent planning approaches are used. Domain-independent planning systems can handle a huge variety of different problems if they are represented in a way that the general planning algorithm can read and understand. Within the planning system, the world is represented in *states*. A state describes the world with a set of facts that are true. To formalize a planning task, the planning system takes three different kinds of inputs. First, it has to know a description of the initial situation of the world (*initial state*). The second input is a description of how the world should look like at the end (*goal description*). Finally, the planning system has to know how it can change the world, which parts of the world can be changed and under which conditions. This is done by defining *operators*, which can add and delete facts from a world state. The goal of the planning system is to get from the initial state to a goal state. This is done by using operators, which change the current state of the world under certain preconditions. The planning system is looking for a sequence of operators that changes the world from the initial state to a goal state. Such a sequence is called *plan*. Each operator has a cost, which can be zero. The cost of the final plan is the sum of all used operators.

For example, there is a room A and a room B, a box and a robot. The robot and the box are both located in room A. The robot can lift and drop the box and it can move between the rooms. The goal is to move the box from room A to room B. In this case, three operators have to be defined, the lift, drop and move operator. The lift operator

has as a precondition that the robot is in the same room as the box. Otherwise, it is not possible to lift the box. The initial state specification encodes that the robot and the box are in room A and each state where the box is in room B is a goal state. The goal of the planning system is to find an sequence of operators (lift, drop, move) to get from the initial state to a goal state. One possible plan would be that the robot lifts the box, moves to room B and drops the box.
In practice, a lot of these planning problems are too complex to find a solution in reasonable time with an uninformed search. Because of these circumstances, heuristics are used to compute an estimation of the solution cost of the original task from a state. A possibility for a heuristic is to solve a relaxed version of the original task and to use the resulting plan cost as as an estimation for the solution cost of the original task. In planning tasks with simplified problems, certain factors that make the original problem more complicated are ignored. The *delete relaxation* is one of those simplifications. In a delete relaxation of a task all delete effects are ignored. That means, if the state of the world is described with binary properties, effects that turn a true fact to false are ignored. For instance, if a robot moves from room A to room B, the fact that the robot is not in room A anymore is ignored. Thus, both facts are true, that the robot is in room A and that it is in room B. Such tasks without any delete effects are called *delete free*. For instance, if in the example above the initial state is that robot and the box are in room A and the goal state is that the box is in room B and the robot in room A, then the solution of the delete relaxation would be that the robot picks up the box, moves to room B and drops the box. Unlike in the original problem, the robot does not have to move back to room A. The fact that the robot is in room A was already true at the beginning and the delete effect of that fact was ignored because of the delete relaxation. The cost of the optimal relaxed plan from a state s to a goal state, where all operator effects that set state variables to false are ignored is a heuristic called $h^+(s)$ [Hoffmann, 2001].

1.1 Goal and Contribution

The goal of this work is to explore a new implementation of the computation of the h^+ heuristic based on *factored planning* [Brafman and Domshlak, 2006]. The resulting algorithm can be used for both, to find the solution of a planning task that has no delete effects like the minimal seed set problem [Gefen and Brafman, 2011] and as a heuristic for planning tasks with delete effects.
We implemented two variants of the algorithm. Both variants are built on the *Local Iterative Deepening (LID) algorithm*, a factored planning algorithm presented by Brafman and Domshlak. The main change to the original LID algorithm is that the two new algorithms exploit that the task is delete free. Variant two uses a different encoding to improve the efficiency of the algorithm.
Both variants were implemented on top of the Fast Downward planning system [Helmert, 2006], where it is used as a cost heuristic for an A^* search. Finally, the efficiency of the

two algorithms will be evaluated.

1.2 Related Work

The paper *Factored Planning: How, When and When Not* [Brafman and Domshlak, 2006] presents the *Local Iterative Deepening (LID) algorithm* to solve planning problems with factored planning. Factored planning is a name for all algorithms that decompose the domain of the original problem into subdomains (*factors*) and then solve each subproblem and put it together into a valid global plan in a second step. The paper addresses the following two questions: How/What is the best way to decompose a problem? When should factored planning be expected to work better then standard planning? The tree width of the *causal graph* which describes the dependencies between the nodes/variables of the planning problem, is a key role to answer the question of *When.* A high tree width causes the variables to have many connections between them. Because of that, the complexity of putting all the subproblems together can exceed the complexity of solving the original problem. The answer of the question of *How* is very simple: factor = variable, where the variables are properties of the world. This means the problem is solved for each variable separately in a first step and in a second step, the plans for the individual variables are combined into a global solution.
The factored planning approach of Brafman and Domshlak [2006] is more efficient than the best previous method of Amir and Engelhardt [2003]. One reason for the efficiency was to use the causal graph as one of the key parameters that identifies the dependency between the variables.
Unfortunately, the presented LID algorithm generates too many subplans for each variable (factor), and because of that even small problems cannot be solved because the algorithm would use too much memory and time.
One approach to save those resources would be to use a delete relaxation of the problem, which restricts the size and number of possible subplans. But because computing h^+ is an NP-hard problem [Bylander, 1994], many methods exist to estimate h^+. Some of this estimations are admissible, which means that the heuristic never overestimates the true cost.
Admissible:

- h^{max} heuristic [Bonet and Geffner, 2001]
- *additive* h^{max} heuristic [Haslum *et al.*, 2005] that was later generalized [Katz and Domshlak, 2008]
- h^{LM-cut} heuristic [Helmert and Domshlak, 2009]

Inadmissible:

- h^{add} heuristic [Bonet and Geffner, 2001]
- h^{FF} heuristic [Hoffmann and Nebel, 2001]
- $h^{FF/add}$ heuristic [Keyder and Geffner, 2008]
- h^{sa} heuristic [Keyder and Geffner, 2008]
- h^{cs} cost-sharing heuristic [Mirkis and Domshlak, 2007]

– h^{lSt} local Steiner tree heuristic [Keyder and Geffner, 2009]

Pommerening and Helmert developed an approach for optimal planning for delete free tasks with incremental LM-cut [Pommerening and Helmert, 2012]. Just like the h^{LM-cut} heuristic, this approach uses landmarks as key parameters. Landmarks are points in the search space that have to be crossed by each possible valid solution path. The h^{LAMA} [Richter *et al.*, 2008] heuristic also uses landmarks and was the winner of the sequential satisficing tracks of the International Planning Competitions in 2008.

1.3 Structure

Chapter 2 gives an overview of the necessary concepts that are required for the theory of the thesis and delivers all important definitions. Section 2.2 presents the causal graph that describes the connections of the variables of the planning problem, which is a key concept for the LID algorithm. Section 2.4 and section 2.6 present the two key concepts for the resulting algorithms of the thesis: the h^+ heuristic and the LID algorithm.

Chapter 3 will present the theory of the h^+ implementation. It presents two variants of the developed Relaxed Local Iterative Deepening algorithm. The first variant (section 3.2.1) is based on the LID algorithm, the second variant (section 3.2.2) uses another encoding to improve the efficiency.

Chapter 4 presents an evaluation of the two implemented variants of the Relaxed Local Iterative Deepening algorithm. The chapter analyses the results.

Chapter 5 has two subsections. The first section 5.1 gives a conclusion of the thesis. The second section *Future Work* (section 5.2) presents possible improvements that could be made, like using other factored planning methods.

2 Background

The *Background* chapter provides all relevant formalisms, definitions and underlying information that are needed to develop the two versions of the *Relaxed Local Iterative Deepening algorithm* presented in the theory chapter of this thesis.

2.1 Planning Tasks

We consider STRIPS [Fikes and Nilsson, 1972] for the planning tasks. The STRIPS formalism is a very simple formalism that uses only binary variables. If the variables of the planning problem are binary, it is a simple task to do a delete relaxation by ignoring all delete effects.

Definition 2.1 (STRIPS Formalism)**.** *A STRIPS planning task* Π *is a four-tupel* $\Pi = \langle V, O, I, G \rangle$*, where*

- $V = \{v_1, ..., v_n\}$ *is a finite set of boolean variables.*
- $O = \{o_1, ..., o_m\}$ *is a finite set of operators, where each operator o consists of*

– *preconditions* $pre(o) \subseteq V$

– *add effects* $add(o) \subseteq V$

– *delete effects* $del(o) \subseteq V$

– *cost* $cost(o) \in \mathbb{R}_0^+$

Applying operator o in a state s leads to a new state s′ with $s' = (s \setminus del(o)) \cup add(o)$.

- $I \subseteq V$ *specifies the initial state.*
- $G \subseteq V$ *specifies a set of goal variables.*

We also introduce the SAS$^+$ formalism [Bäckström and Nebel, 1993], because some of the related work that is used for the thesis and introduced later uses SAS$^+$. Unlike the STRIPS formalism, the state variables of SAS$^+$ are multi-valued.

Definition 2.2 (SAS$^+$ Formalism). *A SAS$^+$ planning task* Π *is a four-tupel* $\Pi = \langle V, O, I, G \rangle$*, where*

- $V = \{v_1, ..., v_n\}$ *is a finite set of multi-valued variables, where each variable has a finite domain of possible values that can be assigned. Each assignment of all variables is a state.*
- $O = \{o_1, ..., o_m\}$ *is a finite set of operators, where each operator o consists of*

– *preconditions* $pre(o)$*, which is a partial variable assignment*

– *effects* $eff(o)$*, which is a partial variable assignment*

– *cost* $cost(o) \in \mathbb{R}_0^+$

- I *specifies the initial state.*
- G *is a partial variable assignment that describes the set of goal states.*

The goal of optimal planning application is to find a plan with minimal cost or to prove that the task has no valid solution.

Definition 2.3 (Plan). *A sequence of operators* $\langle o_1, ..., o_m \rangle$ *for a STRIPS planning task* $\Pi = \langle V, O, I, G \rangle$ *that achieves a goal assignment G from a state s is called s-plan. If state s is the initial state the resulting operator sequence that result to a goal assignment G is called I-plan, plan or solution of* Π.
The cost of a plan $\pi = \langle o_1, ..., o_m \rangle$ *is defined as* $cost(\pi) := \sum_{i=1}^{m} cost(o_i)$.
An optimal plan π^* *is a plan with minimal cost. This implies, that there is no plan with lower costs, for all other I-plans* π $cost(\pi^*) \leq cost(\pi)$.

To compute the h$^+$ heuristic a relaxed planning task $\Pi^+ = \langle V, O, I, G \rangle$ is used. The new task is a delete relaxation of the old, original task.
In this case, each action of Π^+ consists of $pre(O) \subseteq V$, $add(O) \subseteq V$ and $cost(O) \in \mathbb{R}_0^+$.

Definition 2.4 (Delete Free Task). *A given STRIPS planning task* $\Pi = \langle V, O, I, G \rangle$ *is delete free if an action never removes an element from the set of true facts. The delete relaxation of a STRIPS planning task* $\Pi = \langle V, O, I, G \rangle$ *is the delete free STRIPS planning task*

$$\Pi^+ := \langle V, \{o^+ \mid o \in O\}, I, G \rangle,$$

with $pre(o^+) = pre(o), add(o^+) = add(o), del(o^+) = \emptyset$.

2.2 Causal Graph

The causal graph [Bacchus and Yang, 1994; Knoblock, 1994; Brafman and Domshlak, 2003; Domshlak and Dinitz, 2001; Helmert, 2004; Williams and Nayak, 1997] represents the dependencies between the variables.

Definition 2.5 (Causal Graph). *Given a SAS$^+$ formalized planning problem* $\Pi = \langle V, O, I, G \rangle$*, the causal graph* CG_Π *describes the dependencies between the nodes* V.

Two nodes (v, w) are connected if $v \neq w$ and there exists an operator $\langle pre, eff \rangle \in O$ where eff(v) is defined as well as pre(w) or eff(w). This means if an operator has the variable v in the precondition set and the variable w in the set of effects or the other way around, the variables v and w have a directed connection in the causal graph. There is an undirected connection between the variables if both are in the set of effects of an operator.

It is easy to recognize that the construction of a causal graph of any given STRIPS planning problem comes with low costs. The construction is straightforward, each variable is represented by a node. The algorithm has only to iterate through all operators and to create the edges for each dependency. The structure of the causal graph contains valuable informations of the problem structure, which can be exploited to implement planning heuristics [Helmert, 2004]. One important information that can be extracted from the causal graph is that two variables are independent if they have no connection in the causal graph. For instance, if variable v and w have no connection, there is no need to take a look at variable v if an algorithm checks the preconditions of an operator that has an effect on variable w.
The SAS$^+$ representation gives an overview and understanding of the connection between the variables that is evident to humans. However, the STRIPS representation, which is used our implemented heuristic, obscures the important causal structure.

Example: We use the same example as shown in the introduction to create a causal graph but with two boxes instead of one. There are two rooms, room A and room B, two boxes z, box X and box Y, and a robot who can carry one box at a time and move between the two rooms. The actions that can be done are grabbing the box, dropping the box and the robot can move to the other room. We model this problem with the SAS$^+$ formalism, thus a variable/node is multi-valued. The variables x and y can take the positions (*at-roomA*, *at-roomB*, *at-robot*) of the boxes as a value. The variable hand-free f is false if the robot has a box. In this case, the robot cannot grab another box. The variable r represents the position of the robot (*at-roomA* or *at-roomB*). The example has three kinds of actions, which are move-u-v (u and v are rooms), grab-u-v (u is a room and v a box) and drop-u-v (u is a room and v a box) for $u \neq v$. The move-u-v actions have one precondition $-$ r = *at-u* and one postcondition $-$ r = *at-v*. The grab actions grab-u-v have three preconditions $-$ r = *at-u*, z = *at-u*, $f = false$ and one postcondition $-$ z = *at-robot*. Finally, the actions drop-u-v have one precondition z = *at-robot* and two postconditions z = *at-v*, $f = true$.
The corresponding causal graph is shown in Figure 2.1. Node r has a directed connection to nodes x and y because the value of r is a precondition for changing the values x and v in the two actions grap and drop. There is an undirected edge between f and y as well as from f to x. This is because the drop and grab actions change the value of f and x respectively y at the same time.

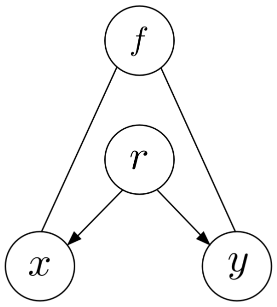

Figure 2.1: Causal Graph Example

2.3 Constraint Satisfaction Problem

A *constraint satisfaction problem (CSP)* is a problem where a value has to be assigned to each variable [Brailsford *et al.*, 1999]. Each variable has a finite domain of possible values that can be assigned. In addition, constraints restrict the values that can be assigned simultaneously. In other words, a problem is considered as solved when each variable has a value that fulfills all the constraints [Russell *et al.*, 2010].

Definition 2.6 (Constraint Satisfaction Problem)**.** *An instance of the constraint satisfaction problem is formalized as* $\Psi = \langle X, D, \mathcal{C}\rangle$*, where*

- $\mathcal{X} = \{X_1, ..., X_n\}$ *is a set of variables.*
- D *is a function that assigns to each variable* X *a nonempty set of possible values (the* domain *of* X*),* $D : \mathcal{X} \to \mathcal{P}^U$*, where* U *are all possible values for the variables.*
- $\mathcal{C} = \{C_1, ..., C_m\}$*. A constraint* C_i *is a pair* $((x_1, ..., x_k), R)$*, where* $(x_1, ..., x_k)$ *is a k-ary tuple of variables and* $R \subseteq D(x_1) \times D(x_2) \times ... \times D(x_k)$*. Thus each constraint restricts the possible values that the different variables can simultaneously take.*

The *constraint optimization problem (COP)* is an extension of the CSP (Definition: 2.6). The CSP is extended by a cost function that calculates the costs for each possible solution. In a COP each variable assignment has a cost. The total cost that the COP minimizes is the sum of all these costs.

2.4 Heuristics

In planning, a heuristic is an estimator for the cost of a solution. Heuristics are often used if the original problem is too complex to compute in a reasonable time. With a heuristic it is possible to calculate an estimation of the cost, which can be used for instance in an A^* search.
The optimal heuristic of a planning problem is called h^*.

Definition 2.7 (h* Heuristic)**.** *The h^* heuristic is the optimal heuristic that maps each state s to the cost of the cheapest path to the goal respectively to the cost of an optimal plan, or ∞ if there is no solution.*

In general, computing h^* is PSPACE-equivlent [Bylander, 1994]. The h^+ heuristic is as the h^* an optimal heuristic but only for delete free planning tasks and in general NP-equivalent [Bylander, 1994].

Definition 2.8 (h$^+$ Heuristic)**.** *The h^+ heuristic is the perfect heuristic for the delete relaxation Π^+ of a planning problem Π. It is a function mapping each state s to the length of the path with the lowest cost to a goal assignment respectively to the cost of an optimal plan, or ∞ if there is no solution.*

The heuristic h$^+$ is admissible. This means that the heuristic never overestimates the true cost.

Theorem 2.1 (h$^+$ is admissible)**.**
Proof sketch: *Each plan of Π is also a solution of Π^+ but not the other way around. Because Π^+ has less restrictions than Π, the cost of the solution has to be smaller or if there is no benefit of the relaxation, the cost will be the same. But the cost cannot be higher. So the following inequality must be satisfied: $h^+(s) \leq h^*(s)$.* □

Because of its complexity, h^+ has not been used for larger planning problems. Instead, approximations of h^+ have been used. But $h(s)/h^+(s)$ can be very large for inadmissible and $h^+(s)/h(s)$ for admissible heuristics. Therefore, the approximations of h^+ can be far away from the correct h^+ values.

2.5 Factored Planning

Factored planning is a term for planning algorithms that decompose the global problem to work on each subproblem separately. The solutions of each subproblem are reused and put together into a valid global plan. It is only possible to gain a benefit from factored planning if the component's values have little influence on other values (e.g. [Knoblock, 1990; Erol *et al.*, 1994]). If the components of the resulting subproblems have too many connections, the complexity of putting the subproblems together can exceed the solving of the non-decomposed problem. The subproblems that are created by factored planning can be overlapping [Laneky and Getoor, 1995], they share some variables and the goal to make them true. There are many different approaches to split the global problem into individual subproblems. The goal of all approaches is to find an algorithm that splits the global problem into subproblems with as little interaction as possible that comes with low compute costs. The best strategy depends on the planning problem itself. If the connections between the nodes of the problem are equally distributed or if it has only a few connections, it will probably not pay off to invest a lot of computation power to split the global problem into optimal subproblems. Because the gained computation time from solving the planning problem with the optimal decomposition in comparison to

a random decomposition is much smaller than the computation time that was invested to find the perfect partitioning. But if the planning problem has subproblems where the individual members of the group have many connections but there are only a few connections between the groups, it will probably pay off to invest the computation time into the analysis and decomposition of the structure of the problem.
One approach that does a good analysis of the problem structure is done by Kelareva et al. [2007]. They do factored planning by tree decompositions and present an SAT-based (introduced in [Kautz *et al.*, 1992]) implementation where they use an exernal SAT-Solver [Moskewicz *et al.*, 2001].
A simple factored planning approach is presented by Brafman and Domshlak [2006]. Their approach is to decompose the global problem in a way that each subproblem only has to solve one variable (factor = variable). With this method, the resulting algorithm creates all possible subplans for each variable, which will be combined into a global plan. We used that approach in this thesis. We will take a closer look at this approach in the next section.

2.6 Local Iterative Deepening Algorithm

The *Local Iterative Deepening (LID)* algorithm [Brafman and Domshlak, 2006] is an algorithm for factored planning. The LID algorithm decomposes the planning problem by creating a factor for each variable, thus the problem is solved for each subproblem separately. The LID algorithm computes all possible solutions for each subproblem. The focus of the algorithm is to put the different solutions of the different subproblems together, where it has to decide which solutions of the different variables are compatible. This is the reason why the algorithm needs all solutions of the subproblems, because it cannot know in advance which subsolutions of the different subproblems are combinable. If the LID algorithm would not compute all subsolutions, it could be that the existing subsolutions of a variable is not combinable with the subsolutions of the other variables. In this case, the algorithm cannot find a global plan, although it exists a plan.
Let $A_v \subseteq A$ be the set of actions that changes the variable $v \in V$. $SPlan(v)$ is a set of prescheduled action sequences that change the variable v to the goal value. Each subplan of $SPlan(v)$ is a set of pairs (a, t) with timepoint $t \in \mathbb{N}_0$ and action $a \in A$. At each timepoint t either one and only one action $a \in A$ or no action has to be performed. The plan and each subplan has the maximum length of $n \cdot d$, where d is a loop variable that will be explained later. Thus, timepoint t cannot be higher than $n{\cdot}d$. In this context, n is the number of variables. The planning problem can be solved with n timepoints if all postconditions are positive (the problem is delete free), because an order exists for the operators that makes all goal variables true, if the problem is solvable. In other words, no operator has to be used more than once. This fact is evident, because no variable that was true since the beginning or became true through an operator will change to false again. Thus, it is unnecessary to apply the same operator for a second time, because all variable that would be changed to true, are already true. But if the

planning problem is not delete free it can not be guaranteed that a variable that was set to true, will not be changed to false again. In this case, it is possible that an operator that was already used, has to be applied again. This is the reason why n timepoints are not sufficient for planning tasks that are not delete free. Therefore, the LID algorithm uses $n \cdot d$ timepoints, where d is a loop variable. This variable is initially one and will be incremented by one for each iteration where the problem cannot be solved.
The remaining question is how to construct a global plan by using these n sets of action sequences $SPlan(v)$. The problem will be solved by compiling it into a binary CSP (SeqCSP) over n variables X_1,..,X_n where:

1. The domain of X_i is exactly $SPlan(v_i)$, where $SPlan(v_i)$ is a given set of prescheduled action sequences.
2. The constraints of SeqCSP bijectively correspond to the edges of the causal graph CG_Π.

The constraints describes which subplans $SPlan(v_i) \times SPlan(v_j)$ where $i \neq j$ are combinable. If the two selected variables have no connections in the causal graph, the two variables are independent from one another. This means that no conflicts can appear and the subplans are combinable if they do not have different operators at the same time point.
If the causal graph has a directed connection from v_i to v_j an action that changes the value of v_j has a precondition which involves some value of v_i that has to be checked.
The pseudo code of the LID algorithm is shown in listing 2.1. The first loop incrementally increases the value of d by one, if the algorithm cannot find a global plan with a maximum length of $d \cdot n$. The inner loop iterates over all variables, because the algorithm has to find all solutions (that changes the value of a variable to the goal value) for all variables. After computing all solutions for all subproblems, the algorithm constructs a constraints satisfactions problem to restrict which subproblems are combinable. A CSP solver will try to solve the problem. If the solver has found a solution, it will return a subsolution number for each variable. With the help of this numbers, LID can combine the correct subsolution to a global plan and will return it. But if the CSP solver could not find a solution, d will be increased and the algorithm tries to find a solution with a new, higher maximum length of the possible plan.

Listing 2.1: Local Iterative Deepening Algorithm [Brafman and Domshlak, 2006]

```
procedure LID
$d := 0$
loop
    for i:=1...n do
        $Dom(X_i)$ := all subplans for $v_i$ of length up to $d$, over all schedules
                  of all operators that turn $v_i$ to the goal value, across $nd$
                  time points
    Construct $SeqCSP_\Pi(d)$ over $X_1$ ,... ,$X_n$ .
    if(solve-csp($SeqCSP_\Pi(d)$)) then
        Reconstruct a plan $\rho$ from the solution of $SeqCSP_\Pi(d)$.
        return $\rho$
    else
        $d := d + 1$
endloop
```

3 Theory

The *Theory* chapter presents two versions of the *Relaxed Local Iterative Deepening algorithm (RLID)* that were developed for this thesis. First, the two algorithms and their background will be presented. After the presentation of the RLID algorithms, the theory chapter presents how the cost optimality of the two algorithms was ensured.

3.1 A Simple Factored Planning Approach

The two variants of the RLID algorithm that were developed for this thesis, use the same simple factored planning approach as the LID algorithm by Brafman and Domshlak [2006]. The presented approach uses the most simple factored planning strategy possible. The global problem is factored into n subproblems, where n is the number of variables in the planning problem. This means, each subproblem has to change only one variable to the goal value. The resulting algorithm solves each individual subproblem and computes all possible subsolutions. The subproblems are each treated as an independent planning task. The algorithm searches for correct subplans and only has to consider the operators that affects the value of an variable that is part of the subproblem. All other operators are ignored.

3.2 Relaxed Local Iterative Deepening Algorithm

The RLID algorithm is an adaption of the LID algorithm to delete free planning tasks. Two versions of the RLID algorithm were implemented for this thesis. Version one is a slightly modified version of the LID algorithm. Version two of the algorithm includes more optimizations that can be used for delete free tasks. These optimizations change the algorithm completely. Therefore, version two of the RLID algorithm much further away from the original LID algorithm than the first version.

3.2.1 Variant One of the RLID Algorithm

The domain of a variable X_v is a set of all subplans for v over $|V|$ timepoints, where V is the set of all variables. In a delete free planning task $|V|$ timepoints are enough, because each operator sets at least one variable to true. If the planning task is solvable, there are enough timepoints that all goal variables can be set to true. Because of that, at most $|V|$ different operators are needed to find a plan. But in many cases not all time slots are needed. In contrast to a maximal amount of timepoints, there exists no general minimum of needed timepoints, because this value depends on the structure of the individual problem. For instance, if all variables $v \in G$ are also $v \in I$, the problem is already solved and no operator has to be used. An operator can also set multiple variables to true.
A subplan P_v for variable V is a set of pairs (a, t) where $t \in \{1, ..., |V|\}$ and $a \in A_v$. Each subplan P_v contains at most one pair for a certain t. If $v \in G$ (goal set) each subplan P_v has to include at least one action $a \in A_v$, otherwise a subplan can be empty. $SPlan(v)$ is the set of all possible subplans:

$$SPlans(v) = \{P_v \subseteq A_v \times \{1, ..., |V|\} \mid \text{ for } (a, t), (a', t) \in P_v, a = a'\}$$

A variable $v \in G$ has to be set to true for a valid solution. A variable $v \notin G$ does not need to set to true, it only has to be changed if it is a precondition of an operator that sets a variable $v \in G$ to true. In the other case, a value of an operator must not be changed if an extra operator has to be used, because more operators means more cost, if the cost of the operator is not zero.

$$Dom(X_v) = \begin{cases} SPlans(v) & \text{if } v \notin G \text{ or } v \in I \\ SPlans(v) \backslash \{\{\}\} & \text{otherwise} \end{cases}$$

An optimal plan only sets a variable $v \notin G$ to true, if the variable is a precondition for a needed operator or if an operator that sets a variable $v \in G$ to true changes the value of a variable $w \notin G$ simultaneously.

In the first step, the RLID algorithm creates $SPlans(v)$. Thus, there is a set of subplans for each variable and each subplan from this set solves the subtask for this variable. The next step is to combine the subplans of the different variables where $v \notin I$. The combination of the subplans creates a global plan that solves the subproblems of the different variables. The combination of the subplans can be done by a CSP solver after creating a set of constraints that describes which subplans of the different variables can be combined.

$R_{v,w}$ describes the constraints between the variables X_v and X_w. The constraints $R^a_{v,w}$ for each action $a \in A_w$ define the restrictions for the occurrence of action a within the

subplans of the variables v and w.
We can assume that $pre(a) \cap add(a) = \emptyset$ holds for all actions a. If that is not the case, we can remove all effects of the intersection from the actions, because it would not have an effect.
The constraints are a set of pairs of subplans (first element for a subplan P_v and second element for a subplan P_w) $R_{v,w} \subseteq SPlans(v) \times SPlans(w)$.
If $w \in add(a)$ and if $v \in pre(a)$ or $v \in add(a)$, we have to create the following restrictions:

- If $v \in pre(a)$: v has to be true before the usage of a for w. This means for each timepoint t with $(a, t) \in P_w$ there is a timepoint $t' < t$ and an action $a' \in A_v$ with $(a', t') \in P_v$ for all $(P_v, P_w) \in R^a_{v,w}$ (or v is already true in the beginning).

$$\begin{aligned} R^{a,pre}_{v,w} = \{(P_v, P_w)\ |P_v \in SPlans(v) \wedge P_w \in SPlan(w) \wedge \\ \forall\ t \text{ with } (a,t) \in P_w\ \exists\ t' < t, a' \in A_v, \\ \text{so that } (a', t') \in P_v \text{ or } v \in I\} \end{aligned}$$

- If $v \in add(a)$: For all $(P_v, P_w) \in R^a_{v,w}$, the action a is only allowed in P_w, if it occurs in P_v at the same timepoint: If $(a, t) \in P_w$ then $(a, t) \in P_v$.

$$\begin{aligned} R^{a,add}_{v,w} = \{(P_v, P_w)\ |P_v \in SPlans(v) \wedge P_w \in SPlan(w) \wedge \\ \forall\ t \text{ with } (a,t) \in P_w, (a,t) \in P_v\} \end{aligned}$$

  Otherwise, the final plan could have the same action at different timepoints. The second appearance has no effect but increasing the cost, because all delete effects are ignored.

Overall, $R^a_{v,w}$ is:

$$R^a_{v,w} = \begin{cases} R^{a,pre}_{v,w} & \text{if } a \in A_w \text{ and } v \in pre(a) \\ R^{a,add}_{v,w} & \text{if } a \in A_w \text{ and } v \in add(a) \\ SPlans(v) \times SPlans(w) & \text{otherwise} \end{cases}$$

The constraint between the variables v and w is the combination of the constraints of all actions $a \in A$:

$$R_{v,w} = \bigcap_{a \in A} R^a_{v,w}$$

The slightly modified version of the LID algorithm is shown in listing 3.1.

Listing 3.1: Relaxed Local Iterative Deepening (RLID) Algorithm: Variant 1

```
procedure LID
for i:=1...|V| do
        Dom(X_i) := all subplans for v_i, over all schedules of all operators
                that turn v_i to the goal value across |V| time points.
```

```
Construct $SeqCSP_{\Pi}(d)$ over $X_1, \dots, X_n$.
Reconstruct a plan $\rho$ from the solution of $SeqCSP_{\Pi}(d)$.
return $\rho$
```

The pipeline of the RLID algorithm is shown in Figure 3.1. It consist of six major steps and starts with the original planning task.

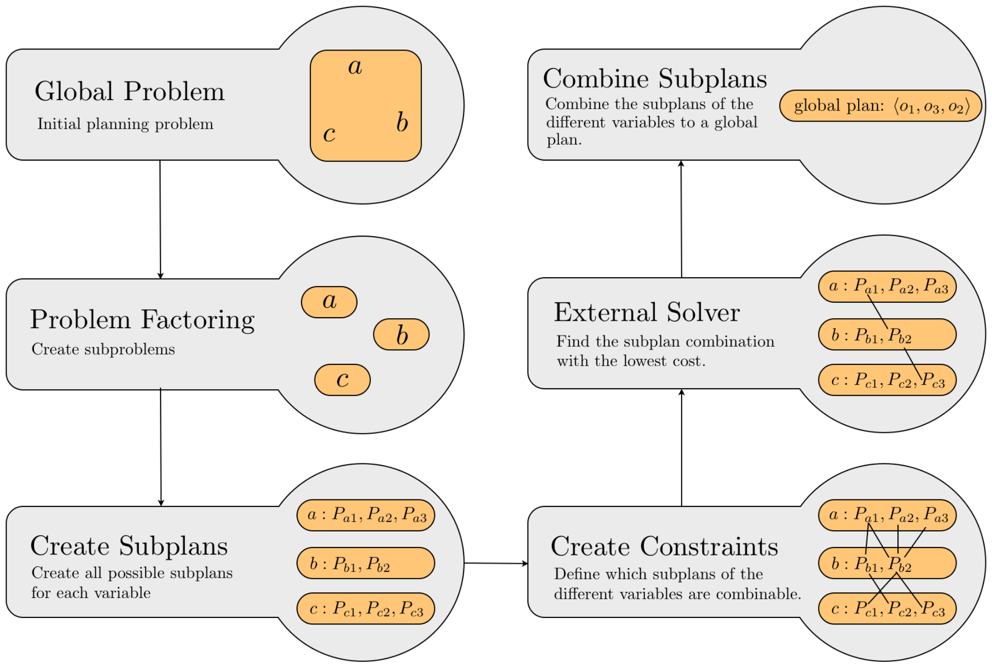
Global Problem
Initial planning problem
a
b
c
Problem Factoring
Create subproblems
a
b
c
Create Subplans
Create all possible subplans for each variable
$a: P_{a1}, P_{a2}, P_{a3}$
$b: P_{b1}, P_{b2}$
$c: P_{c1}, P_{c2}, P_{c3}$
Create Constraints
Define which subplans of the different variables are combinable.
$a: P_{a1}, P_{a2}, P_{a3}$
$b: P_{b1}, P_{b2}$
$c: P_{c1}, P_{c2}, P_{c3}$
External Solver
Find the subplan combination with the lowest cost.
$a: P_{a1}, P_{a2}, P_{a3}$
$b: P_{b1}, P_{b2}$
$c: P_{c1}, P_{c2}, P_{c3}$
Combine Subplans
Combine the subplans of the different variables to a global plan.
global plan: $\langle o_1, o_3, o_2 \rangle$

Figure 3.1: Pipeline of the Relaxed Local Iterative Deepening Algorithm

3.2.1.1 Cost Optimization

We add cost optimization to our algorithm, thus we have a COP instead of a CSP. This means, that the goal of the RLID algorithm is not just to find a sequence of operators that solves the relaxed problem, but a sequence of operators with minimal cost. To achieve this goal, we introduce a new variable $\textit{use-}a_i \in \{0, 1\}$. This variable is zero, if the operator that belongs to that variable is not included in the final plan. Otherwise, the variable is set to one.

If the operator a_i is used in a subplan of X_i, the value of $\textit{use-}a_i$ will be set to one if the subplan will be part of the final plan. Otherwise, the value will be zero. For all variables v and each action $a_i \in A_v$, we have the following constraint.

$$R_{v,a_i} = \textit{Dom}(X_v) \times \textit{Dom}(\textit{use-}a_i) \backslash \{((a,t),0) \mid (a,t) \in \textit{Dom}(X_v) \text{ with } a = a_i\}$$

To find a plan with optimal costs for the relaxed planning problem, the following equation has to be minimized.

$$\sum_{a \in A} \textit{use-}a \cdot \textit{cost}(a)$$

Minimizing the equation will result in choosing the operators that build the cheapest plan, because including any non needed operator can only increase the solution cost. Figure 3.2 illustrates the dependencies between the variables.

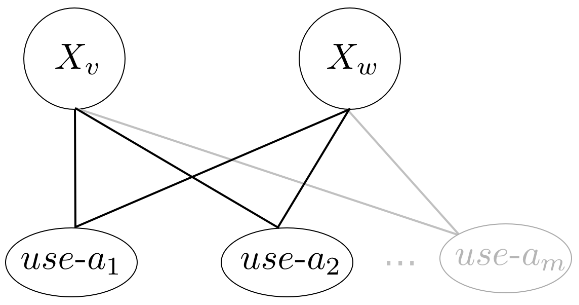

Figure 3.2: If a subplan of X_v uses a_1 the value of *use-*a_1 will be one if the subplan will be a part of the final plan. In all other cases the value of *use-*a_1 will be zero.

3.2.1.2 Disadvantage of the First Variant of the RLID algorithm

Variant one of the RLID algorithm has a huge disadvantage, it has to generate a lot of subplans for each variable. The number of subplans that are generated for one variable is a^p if a is the number actions that sets the variable to true and p is the maximal plan size, which is the number of variables in our case. A small example demonstrates how many subplans have to be generated even for small planning problems. If we have 85 variables, which is not very much for a planning problem with binary variables and each variable has eleven operators that can set the variable to true, then we have to generate $85 \cdot 11^{85} = 2.8 \cdot 10^{90}$ subplans. We have about 10^{89} particles in the universe, this means we would have to generate about 28 times more subplans than we have particles in the universe. Even for a very small planning problem like a planning task with 15 variables and five operators, that makes each variable true, we have to generate more than 457 billion subplans.
This is the reason why we implemented a second variant of the RLID algorithm that uses the characteristic of delete free planning tasks to drastically reduce the number of subplans that have to be generated.

3.2.2 Variant Two of the RLID Algorithm

Variant one of the RLID creates a huge amount of subplans. Because of that property, we run out of memory early, even for small planning problems. The second variant of the algorithm improves the memory usage and performance of the initial algorithm by taking advantage of properties of a delete free planning problem. We know that we have to make a variable true not more than once in an optimal plan. In this case, we have to create only $a \cdot p$ instead of a^p subplans (a = number of variables, p = maximal plan size). If we have the same example as above, with 85 variables and each variable has

eleven operators that can set the variable to true, variant two of the RLID algorithm has to generate $85 \cdot 11 \cdot 85 = 79475$ subplans instead of $2.8 \cdot 10^{90}$.
To save memory, the second variant only allows an action with a timepoint that changes a corresponding variable from false to true rather than a whole subplan. This variant has two special cases that need a special representation. First, if the variable does not have to be changed to true because the variable was already true in the initial state. For this special case, we introduce a dummy action a^T at the timepoint zero, with the cost of zero.
In the second special case, a variable does not have to be changed from false to true, because the variable does not have to be true. For this case, we introduce a dummy action a^F at the timepoint $|V| + 1$ that has a cost of zero, too.
The CSP domains are much simpler than in the first variant:

$$Dom(X_v) = \begin{cases} \{(a,t) \mid a \in A_v, t \in \{1, ..., |V|\}\} \cup \{(A^F, |V|+1)\} & \text{if } v \notin I \\ \{(a^T, 0)\} & \text{otherwise} \end{cases}$$

Variables $v \in I$ have not to changed to true, because v is already at the beginning true. For this reason, there is only one dummy operator $(a^T, 0)$ with a cost of zero. If we would allow other operators, the COP solver would find an optimal solution, nevertheless. The advantage of only allowing one dummy variable is that the domain of X_v is much smaller and thereby the algorithm will use less memory. In the other case where $v \notin I$, the COP solver has the possibility to choose the free subplan $(A^F, |V| + 1)$ for a variable v where $v \notin G$ and v is not a precondition of an operator that needs to be applied to set a variable $v \in G$ to true.

The restrictions for combining two subplans of the variables v and w are related to the restrictions shown in variant one. We compare each subplan of a variable v with each subplan of a variable w. In the second variant of the RLID algorithm, we have only two pairs (a, t) and (a', t') for each check between two subplans P_v and P_w. There are some conditions that have to be satisfied to combine two subplans if variable v is a precondition of a', action a has to set v to true. To do so, action a cannot be the dummy action $(a \neq a^F)$ and action a has to be executed before action a' $(t < t')$. Correspondingly the same has to be satisfied if w is a precondtion of action a: $a' \neq a^F$ and $t' < t$. If both actions (a and a') are dummy actions, the subplans are combinable. If a subplan has an action that sets both variables v and w to true, the only combinable subplan is a subplan that has the same action at the same timepoint. The last requirement is that if both subplans have an action at the same timepoint $(t = t')$ the actions have to be the same $(a = a')$.
We define the constraints $R_{v,w}$.

$$
\begin{aligned}
R_{v,w} = \{((a,t),(a',t')) \in Dom(X_v) \times Dom(X_w) \mid \\
&(\text{if } v \in pre(a') \text{ then } a \neq a^F \wedge t < t') \wedge \\
&(\text{if } w \in pre(a) \text{ then } a' \neq a^F \wedge t' < t) \wedge \\
&(add(a) \cap add(a') \cap \{v,w\} = \emptyset \text{ or} \\
&\quad (\text{if } t < t' \text{ then } w \notin add(a) \wedge \\
&\quad \text{if } t' < t \text{ then } v \notin add(a') \wedge \\
&\quad \text{if } t = t' \text{ then } a = a'))\}
\end{aligned}
$$

The constraints for the costs are the like in the first variant of the RLID algorithm.

The pseudo code of the second variant of the RLID algorithm is shown in listing 3.2. In difference to the LID and the RLID 1 algorithm, this algorithm has generate much less subplans in the loop over $|V|$. The second difference is the construction of the CSP.

Listing 3.2: Relaxed Local Iterative Deepening (RLID) Algorithm: Variant 2

```
procedure LID
for i:=1...|V| do
        $Dom(X_i)$ := all subplans that have only one operator $\in A_{v_i}$,
        across $n$ time points.
Construct $SeqCSP_\Pi(d)$ over $X_1$,...,$X_n$.
Reconstruct a plan $\rho$ from the solution of $SeqCSP_\Pi(d)$.
return $\rho$
```

4
Evaluation and Discussion

The *Evaluation and Discussion* chapter presents the experimental setup and makes an evaluation and comparison of the two versions of the implemented RLID algorithm.

4.1 Fast Downward

The Fast Downward planner is a planning system that is based on heuristic search [Helmert, 2006]. The planning problems for the Fast Downward planning system are encoded in PDDL2.2 [Edelkamp and Hoffmann, 2004]. The two variants of the the RLID algorithm are implemented as heuristics to approximate the the costs of a plan. These heuristics are used within the Fast Downward planner.

4.2 Experiments

The *Experiments* section presents the experimental setup and evaluates the two versions of the RLID algorithm. To evaluate the algorithms we make three comparisons.

- RLID One against RLID Two
- LM-cut against RLID Two
- Incremental LM-cut against RLID Two

In the first two comparisons, the algorithms are used as cost heuristic for an A^* search. In the third comparison, we compare the computation time to find an optimal plan for the delete relaxation of the planning tasks.

4.2.1 Setup

The two versions of the RLID algorithm uses the external, Java based COP solver Sugar [Tamura *et al.*, 2008] to find the optimal combinations of subplans to generate the final plan. All experiments were ran on a machine equipped with 2 × Nehalm Quad-core 2,6 Ghz cores and 24 gigabyte of memory.

4.2.2 Results and Discussion

This section presents the results of the two versions of the RLID algorithm. The section gives also explanations for the presented results.

4.2.2.1 Variant One of the RLID Algorithm

First, we made a comparison of the two implementation of the RLID algorithm. We set a time limit of thirty minutes and a memory limit of 4000 megabytes. As mentioned in the section 3.2.1.2 about the disadvantage of this variant of the algorithm, RLID 1 has to generate a huge amount of subplans. Because of that, a lot of computation power and memory are used, which is the reason that only small problems can be solved. As table 4.1 shows, RLID 1 could solve only a few small problems. The algorithm could solve five out of 150 problems of the Miconic domain. The second version of the RLID version could solve 21 problems of this domain and 43 of other domains. The RLID 1 algorithm could not solve any other problem from any other domain. In total, the algorithm could solve five out of 1116 problems. For each of the five problems, both algorithms used less than 0.1 seconds to solve. The reason why the first algorithm could not solve more problems was that is always exhausted the memory limit.

Coverage	**RLID One**	**RLID Two**	**Difference**
Miconic (150)	5	**21**	16

Table 4.1: Coverage of the first version of the RLID algroithm.

4.2.2.2 Variant Two of the RLID Algorithm

Variant two of the RLID algorithm solves much more planning tasks than the first variant. Thus, we evaluated variant two of the algorithm and compared it with the state-of-the-art heuristic LM-cut. We set the same limits as before (4000 megabytes ram, 30 minutes).

The coverage results of the experiment are presented in table 4.2. In total, the RLID 2 algorithm could solve 64 out of 1116 problems. LM-cut has a much better performance and found a solution in 596 problems. In the 30 problem domains where LM-cut had found at least one solution, the RLID 2 algorithm had only found a solution in 10 of it.

We evaluated the number of expansions that the planning system had to make with the two different heuristics. The h^+ heuristic is the perfect heuristic for delete relaxations of

Coverage	LM-Cut	RLID Two	Difference
airport (50)	**27**	1	-26.0
blocks (35)	**28**	0	-28.0
depot (22)	**7**	0	-7.0
driverlog (20)	**13**	1	-12.0
elevators-opt08-strips (30)	**22**	0	-22.0
freecell (80)	**15**	0	-15.0
grid (5)	**2**	0	-2.0
gripper (20)	**7**	1	-6.0
logistics00 (28)	**20**	4	-16.0
logistics98 (35)	**6**	0	-6.0
miconic (150)	**141**	21	-120.0
mprime (35)	**22**	0	-22.0
mystery (30)	**17**	0	-17.0
openstacks-opt08-strips (30)	**19**	0	-19.0
openstacks-strips (30)	**7**	0	-7.0
parcprinter-08-strips (30)	**18**	0	-18.0
pathways-noneg (30)	**5**	2	-3.0
pegsol-08-strips (30)	**28**	0	-28.0
pipesworld-notankage (50)	**17**	0	-17.0
pipesworld-tankage (50)	**11**	0	-11.0
psr-small (50)	**49**	23	-26.0
rovers (40)	**7**	4	-3.0
satellite (36)	**7**	2	-5.0
scanalyzer-08-strips (30)	**15**	0	-15.0
sokoban-opt08-strips (30)	**29**	0	-29.0
tpp (30)	**6**	5	-1.0
transport-opt08-strips (30)	**11**	0	-11.0
trucks-strips (30)	**10**	0	-10.0
woodworking-opt08-strips (30)	**17**	0	-17.0
zenotravel (20)	**13**	0	-13.0
Sum (1116)	**596**	64	-532.00

Table 4.2: Each table entry gives the sum of the coverage for that domain.

planning problems. Thus, the LM-cut heuristic cannot generate less expansions, because it is an approximation. In other words, h^+ is always at least as good as LM-cut. Table 4.3 sums up the number of expansions of the different problem domains. As the LM-cut heuristic computes accurate approximations, the number of expansions with the RLID algorithm are the same in seven out of ten domains. In total, the LM-cut heuristic generates 2336 expansions in comparison to the 2074 expansions of the RLID algorithm. Thus, the LM-cut algorithm leads to about 12 percent more expansions.

Expansions	**LM-Cut**	**RLID Two**	**Difference**
airport (1)	9	9	0
driverlog (1)	8	8	0
gripper (1)	107	**82**	-25
logistics00 (4)	174	174	0
miconic (21)	284	284	0
pathways-noneg (2)	20	20	0
psr-small (23)	1350	1350	0
rovers (4)	93	**60**	-33
satellite (2)	24	24	0
tpp (5)	267	**63**	-204
Sum (64)	2336	**2074**	-262

Table 4.3: Each table entry gives the sum of expansions for that domain.

In a next step, we evaluated the memory usage of the two algorithms. The table 4.4 shows that the RLID algorithm uses more memory than LM-cut. The bottleneck of the second variant of the RLID algorithm is the memory it needs to compute the heuristic. The RLID algorithm needs also more computation time. The huge search time differences are illustrated in table 4.5. With LM-cut, the planning system needed 6.4 seconds to solve the listed problems. With the RLID algorithm, it needed 2159.55 seconds overall, which is about 320 times longer.

Table 4.6 gives a summary of the comparisons between LM-cut and RLID 2. Overall, the RLID 2 algorithm used more time and memory to compute the optimal solution cost of the relaxed planning problem. Because of the resource limitation, the algorithm could solve only 64 task in comparison to LM-cut which solved 596 problems, which is more than nine times as many tasks.

In another experiment, we increased the memory limitation to 8000 megabytes and the time limitation to one hour per run. Table 4.7 presents the coverage result of this experiment.

Memory	LM-Cut	RLID Two	Difference
airport (1)	**5852**	20684	14832.0
driverlog (1)	**5716**	18524	12808
gripper (1)	**5716**	7560	1844
logistics00 (4)	**23120**	31420	8300
miconic (21)	**120036**	159128	39092
pathways-noneg (2)	**11432**	32384	20952
psr-small (23)	**131468**	187260	55792
rovers (4)	**22864**	23140	276
satellite (2)	**11432**	17512	6080
tpp (5)	**28708**	34960	6252
Sum (64)	**366344**	532572	166228

Table 4.4: Each table entry gives the sum of the memory usage (in bytes) for that domain.

Time	LM-Cut	RLID Two	Difference
airport (1)	**0.1**	22.67	22.57
driverlog (1)	**0.1**	72.78	12808
gripper (1)	**0.1**	40	39.9
logistics00 (4)	**0.4**	210.93	210.53
miconic (21)	**2.1**	430.13	428.03
pathways-noneg (2)	**0.2**	182	181.80
psr-small (23)	**2.3**	1051.37	1049.07
rovers (4)	**0.4**	19.37	18.97
satellite (2)	**0.2**	49.28	49.08
tpp (5)	**0.5**	81.22	80.72
Sum (64)	**6.4**	2159.55	2153.15

Table 4.5: Each table entry gives the sum of the search time in seconds for that domain.

Memory was the factor that limited the RLID algorithm to solve more problems. With 4000 megabytes more memory, the algorithm could solve 78 planning tasks instead of 64. In comparison, if the planning system uses the LM-cut heuristic, it can solve 608 tasks with 8000 megabyte of memory, instead of 596. Because h^+ is the perfect heuristic for delete free planning tasks, the algorithm leads to less expansions. The planning system

Summary	**LM-Cut**	**RLID Two**	**Difference**
Coverage Sum	**596**	64	-532.0
Expansions	2336	**2074**	-262
Memory (Bytes)	**366344**	532572	166228
Time (Seconds)	**6.4**	2159.55	2153.15

Table 4.6: Comparison between the LM-cut heuristic and the RLID Two heuristic with 30 minutes computation time and 4000 megabytes of memory.

has to do less expansions and can save up evaluations. Nevertheless, if the planning system uses the LM-cut heuristic, it has only to do about 12 percent more expansions for the evaluated planning tasks. This is a small amount of expansions, in comparison to the greater consumption of computation time and memory of the RLID 2 algorithm. The relatively small amount of expansions that can be saved with RLID 2 can be explained by the fact that LM-cut makes a very good approximation to the perfect h^+ heuristic. The estimated costs from the initial states (initial h value) of the 64 planning tasks which were solved by both heuristics were always the same but in one exception.

4.2.2.3 Relaxed Planning

The h^+ heuristic computes good approximations, it is even perfect on delete relaxations. But this advantage comes with the disadvantage that the computation of h^+ is very expensive. The RLID algorithm computes not only the cost of the optimal plan of the delete relaxation but also the plan itself. For this reason, we compared the computation time with a planner [Pommerening and Helmert, 2012] that also computes optimal plans of delete free planning tasks. The planner of Pommerening and Helmert uses incremental LM-cut.

The results of the experiment can be found in table 4.8. The incremental LM-cut algorithm outperformed the RLID 2 algorithm. For instance, RLID 2 needed almost 60 times more time to solve the relaxed version of the first problem of the trucks-strips domain.

Coverage	**LM-Cut**	**RLID Two**	**Difference**
airport (50)	**27**	1	-26.0
blocks (35)	**28**	3	-25.0
depot (22)	**7**	0	-7.0
driverlog (20)	**14**	2	-12.0
elevators-opt08-strips (30)	**22**	0	-22.0
freecell (80)	**15**	0	-15.0
grid (5)	**2**	0	-2.0
gripper (20)	**7**	1	-6.0
logistics00 (28)	**20**	6	-14.0
logistics98 (35)	**6**	0	-6.0
miconic (150)	**141**	25	-116.0
mprime (35)	**24**	0	-24.0
mystery (30)	**17**	0	-17.0
openstacks-opt08-strips (30)	**19**	0	-19.0
openstacks-strips (30)	**7**	0	-7.0
parcprinter-08-strips (30)	**19**	0	-19.0
pathways-noneg (30)	**5**	2	-3.0
pegsol-08-strips (30)	**28**	0	-28.0
pipesworld-notankage (50)	**18**	0	-18.0
pipesworld-tankage (50)	**12**	0	-12.0
psr-small (50)	**49**	25	-24.0
rovers (40)	**9**	4	-5.0
satellite (36)	**7**	2	-5.0
scanalyzer-08-strips (30)	**16**	0	-16.0
sokoban-opt08-strips (30)	**30**	0	-30.0
tpp (30)	**6**	5	-1.0
transport-opt08-strips (30)	**11**	0	-11.0
trucks-strips (30)	**10**	0	-10.0
woodworking-opt08-strips (30)	**19**	0	-19.0
zenotravel (20)	**13**	2	-11.0
Sum (1116)	**608**	78	-530.00

Table 4.7: Each table entry gives the sum of the coverage for that domain.

Time	**RLID Two**	**Incr. LM-Cut**	**Difference**
airport (1)	2.26	**0.1**	2.16
driverlog (3)	8.08	**0.3**	7.78
gripper (2)	1.91	**0.2**	1.71
logistics00 (12)	14.27	**1.2**	13.07
logistics98 (1)	1.58	**0.1**	1.48
miconic (29)	68.31	**2.9**	65.41
pathways-noneg (2)	2.25	**0.2**	2.05
psr-small (29)	19.85	**2.9**	16.95
rovers (6)	13.99	**0.6**	13.39
satellite (2)	0.75	**0.2**	0.55
tpp (5)	4.89	**0.5**	4.39
trucks-strips (1)	6.84	**0.12**	6.72
Sum (93)	144.98	**9.32**	135.66

Table 4.8: Each table entry gives the sum of time in seconds for that domain.

5 Conclusion and Future Work

The *Conclusion and Future Work* chapter gives a conclusion of the work done in this thesis. In a second part, the chapter presents possible future work and improvements of the implemented two variants of the RLID algorithm as well as additional work that was not realized in this thesis but could be used to improve the performance of factored planning based heuristics.

5.1 Conclusion

In this thesis we presented and evaluated two new approaches to compute the h^+ heuristic, which is the perfect heuristic for delete free planning tasks. The two resulting algorithms are called RLID 1 and RLID 2. We provided an approach to decompose the original planning task into subproblems. We made a subproblem for each variable of the original task. After the decomposition, we worked with each subproblem individually and presented an approach to compute all solutions of each subproblem. In a next step, we constructed a COP. The COP includes the information of which subplans of the subproblem v are combinable with which subplans of the suproblem w. To check if two subplans are combinable, the RLID algorithms have to ensure that the preconditions of the operators of a subplan are satisfied and that no conflicts appear. A conflict appears if the two subplans have a different operator at the same timepoint. The RLID algorithms have not to do these checks in each case. If the subplans $SPlans(v)$ are responsible to change v to true and the suplans $SPlans(w)$ set the variable w to true, the checks have only to be done, if these two variables have a connection in the causal graph. If they do not have a connection, the variables are independent from each other and each subplan of $SPlans(v)$ is combinable with each subplan of $SPlans(w)$. By solving the COP, we can find a combination of the different subplans, which gives us an optimal plan for the delete relaxation of the original planning task. The COP solver also returns the cost of the plan.

In the evaluation part of this thesis we showed that the RLID 1 algorithm needs a huge amount of memory. Because of that, the algorithm can only solve a few small problems with a memory limit of 4000 megabytes. Version two of the RLID algorithm is much more memory efficient. If the RLID 2 algorithm is used in a cost heuristic in an A^* search, it generates less expansions than the LM-cut heuristic, but it uses more memory and time to compute the cost heuristic. With the same memory and time limitation, the RLID 2 algorithm can solve only 64 planning tasks out of 1116, in comparison to 596 tasks that could be solved with the help of the LM-cut heuristic. Finally, we compared RLID 2 algorithm with a planner that uses incremental LM-cut to find a plan of delete free (or relaxed) planning tasks and compared the computation time. The RLID 2 was outperformed in each planning task. Some improvements that could be done, will be presented in the next section.

5.2 Future Work and Improvements

There are two main parts that can be improved. One possible improvement is to use other factored planning methods, like factored planning by tree decomposition. Another possible future work that could be done is to develop an algorithm that could decide which factored planning method is most suitable for a certain kind of planning problem.

5.2.1 Other Variants of Factored Planning

Factored planning is used for the presented algorithms in this thesis. A future work could be to use other decompositions of the domains. The two algorithms of the thesis define each variable as a factor and with this approach, the problem is solved for each variable and the resulting subplans are reused and put together into a valid, global plan. Another factored planning approach is to factor the domain with the help of decomposition trees (dtree) [Darwiche and Hopkins, 2001]. The goal of this idea is to create variable groups, in which the variables of one group have many connections but the groups should have few connections between the them. The work of Kelareva et al. [2007] could be used as foundation. They present the algorithm *dTreePlan* which is a factored planning algorithm that uses tree decomposition to partition the domain into subdomains with as little interaction as possible.

5.2.2 Select the Best Factored Planning Approach

As mentioned in the *Factored Planning* chapter (2.5), there is no best factored planning method. The best approach depends on the structure of the specific planning problem. A future work could be to implement a program that analyses the causal graph and computes the tree width of the planning problem to decide which factored planning method should be used. If a planning system uses factored planning, the program can decide which approach would be the best one for a specific problem. To give an algorithm such an ability, a lot of analysis with different planning tasks and different factored planning approaches has to be made. There are many possible factors that could influence such a decision, like the tree width, the number of connections among the nodes or the number of nodes. Already existing tools could be used to calculate the tree width, like LibTW [van Dijk *et al.*, 2006] or quickBB [Gogate and Dechter, 2004]. This information could then be used to choose between different factored planning methods or if the nodes have too many connections among them, the program could decide to use no factored planning method, because the complexity of putting all the subproblem together can exceed the complexity of solving the original problem.

Bibliography

[Amir and Engelhardt, 2003] Eyal Amir and Barbara Engelhardt. Factored planning. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, volume 18, pages 929–935, 2003.

[Bacchus and Yang, 1994] Fahiem Bacchus and Qiang Yang. Dawnward refinement and the efficiency of hierarchical problem solving. *Journal of Artificial Intelligence Research (JAIR 1994)*, 71(1):43–100, 1994.

[Bonet and Geffner, 2001] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1):5–33, 2001.

[Brafman and Domshlak, 2003] Ronen I Brafman and Carmel Domshlak. Structure and complexity in planning with unary operators. *Journal of Artificial Intelligence Research (JAIR 2003)*, 18:315–349, 2003.

[Brafman and Domshlak, 2006] Ronen I Brafman and Carmel Domshlak. Factored planning: How, when, and when not. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI 2006)*, volume 21, pages 809–814, 2006.

[Brailsford *et al.*, 1999] Sally C Brailsford, Chris N Potts, and Barbara M Smith. Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research*, 119(3):557–581, 1999.

[Bylander, 1994] Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1):165–204, 1994.

[Bäckström and Nebel, 1993] Christer Bäckström and Bernhard Nebel. Complexity results for SAS$^+$ planning. In *Proceedings of the 13th International Conference of Artificial Intelligence (IJCAI 1993)*, pages 1430–1435, 1993.

[Darwiche and Hopkins, 2001] Adnan Darwiche and Mark Hopkins. Using recursive decomposition to construct elimination orders, join trees, and dtrees. In *Proceedings of the Sixth European Conference for Symbolic and Quantitative Approaches to Reasoning under Uncertainty (EC-SQARU 2001)*, pages 180–191, 2001.

[Domshlak and Dinitz, 2001] Carmel Domshlak and Yefim Dinitz. Multi-agent off-line coordination: Structure and complexity. In *Pre-proceedings of the Sixth European Conference on Planning (ECP 2001)*, pages 277–288, 2001.

[Edelkamp and Hoffmann, 2004] Stefan Edelkamp and Jörg Hoffmann. PDDL2.2: The language for the classical part of the fourth international planning competition. *Fourth International Planning Competition (IPC 2004), at The International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 2004.

[Erol *et al.*, 1994] Kutluhan Erol, James Hendler, and Dana S Nau. HTN Planning: Complexity and Expressivity. In *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI 1994)*, volume 94, pages 1123–1128, 1994.

[Fikes and Nilsson, 1972] Richard E Fikes and Nils J Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208, 1972.

[Gefen and Brafman, 2011] Avitan Gefen and Ronen I Brafman. The minimal seed set problem. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS 2011)*, pages 319–322, 2011.

[Gogate and Dechter, 2004] Vibhav Gogate and Rina Dechter. A Complete Anytime Algorithm for Treewidth. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (UAI 2004)*, pages 201–208, 2004.

[Haslum *et al.*, 2005] Patrik Haslum, Blai Bonet, and Héctor Geffner. New admissible heuristics for domain-independent planning. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI 2005)*, pages 1163–1168, 2005.

[Helmert and Domshlak, 2009] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pages 162–169, 2009.

[Helmert, 2004] Malte Helmert. A planning heuristic based on causal graph analysis. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS 2004)*, volume 16, pages 161–170, 2004.

[Helmert, 2006] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26(1):191–246, 2006.

[Hoffmann and Nebel, 2001] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14, 2001.

[Hoffmann, 2001] Jörg Hoffmann. Local search topology in planning benchmarks: An empirical analysis. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, volume 17, pages 453–458. Lawrence Erlbaum Associates LTD, 2001.

[Katz and Domshlak, 2008] Michael Katz and Carmel Domshlak. Optimal additive composition of abstraction-based admissible heuristics. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS 2008)*, pages 174–181, 2008.

[Kautz *et al.*, 1992] Henry A Kautz, Bart Selman, et al. Planning as satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI 1992)*, volume 92, pages 359–363, 1992.

[Kelareva *et al.*, 2007] Elena Kelareva, Olivier Buffet, Jinbo Huang, and Sylvie Thiébaux. Factored planning using decomposition trees. In *Proceedings of the 20th International Joint Conference on Artificial Intelligenc (IJCAI 2007)*, volume 7, pages 1942–1947, 2007.

[Keyder and Geffner, 2008] Emil Keyder and Héctor Geffner. Heuristics for planning with action costs revisited. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, pages 588–592, 2008.

[Keyder and Geffner, 2009] Emil Keyder and Héctor Geffner. Trees of shortest paths vs. steiner trees: Understanding and improving delete relaxation heuristics. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1734–1739, 2009.

[Knoblock, 1990] Craig A Knoblock. Learning abstraction hierarchies for problem solving. In *Proceedings of the eighth National Conference on Artificial Intelligence (AAAI 1990)*, volume 2, pages 923–928, 1990.

[Knoblock, 1994] Craig A Knoblock. Automatically generating abstractions for planning. *Artificial intelligence*, 68(2):243–302, 1994.

[Laneky and Getoor, 1995] Amy L Laneky and Lise C Getoor. Scope and abstraction: Two criteria for localized planning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 1995)*, volume 2, pages 1612–1618, 1995.

[Mirkis and Domshlak, 2007] Vitaly Mirkis and Carmel Domshlak. Cost-sharing approximations for h^+. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007)*, pages 240–247, 2007.

[Moskewicz *et al.*, 2001] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Annual Design Automation Conference (DAC 2001)*, pages 530–535, 2001.

[Pommerening and Helmert, 2012] Florian Pommerening and Malte Helmert. Optimal planning for delete-free tasks with incremental LM-cut. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 2012.

[Richter *et al.*, 2008] Silvia Richter, Malte Helmert, and Matthias Westphal. Landmarks revisited. In *Proceedings of the 23th National Conference on Artificial Intelligence (AAAI 2008)*, pages 975–982, 2008.

[Russell *et al.*, 2010] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial Intelligence: A Modern Approach*. Pearson Education: Prentice Hall Upper Saddle River, Third edition, 2010.

[Tamura *et al.*, 2008] Naoyuki Tamura, Tomoya Tanjo, and Mutsunori Banbara. System description of a SAT-based CSP solver Sugar. *Proceedings of the Third International CSP Solver Competition*, pages 71–75, 2008.

[van Dijk *et al.*, 2006] Thomas van Dijk, Jan-Pieter van den Heuvel, and Wouter Slob. Computing Treewidth with LibTW, 2006.

[Williams and Nayak, 1997] Brian C Williams and P Pandurang Nayak. A reactive planner for a model-based executive. In *Proceedings of the 15th International Joint Conferences on Artificial Intelligence (IJCAI 1997)*, volume 97, pages 1178–1185, 1997.

UNIVERSITÄT BASEL

PHILOSOPHISCH-NATURWISSENSCHAFTLICHE FAKULTÄT

Declaration on Scientific Integrity
(including a Declaration on Plagiarism and Fraud)

~~Bachelor's~~ / Master's Thesis *(Please cross out what does not apply)*

Title of Thesis *(Please print in capital letters)*:
Computation of h+ with Factored Planning

First Name, Surname *(Please print in capital letters)*: Gabriel, Duss

Matriculation No.: 08-054-942

I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged.

I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

In addition to this declaration, I am submitting a separate agreement regarding the publication of or public access to this work.

☐ Yes [✓ No]

Place, Date: Basel, 31.07.2013

Signature: [illegible]

Please enclose a completed and signed copy of this declaration in your Bachelor's or Master's thesis .

UNI BASEL