Università degli Studi di Padova

School of Engineering - Department of Information Engineering

Master Degree in Computer Engineering

MIP formulations for delete-free AI planning

Supervisor:
Prof. Salvagnin Domenico
University of Padova

Candidate:
Zanella Matteo
2122187

Academic Year 2024/2025

Abstract

We compare current state-of-the-art M.I.P. formulations to solve the delete-free relaxation in cost-optimal planning, integrating different preprocessing techniques proposed in different pubblications and devising primal heuristics to furnish a warm-start to the M.I.P. solver.
We, then, explored a novel way to model acyclicity by coumputing violated landmarks and adding them as constraints: computing all landmarks a priori is intractable, since the number of potential landmarks grows exponentially with respect to the problem size, making any brute-force approach computationally infeasible; instead we compute violated landmarks iteratively, looking for the minimal amount necessary to guarantee both feasibility and optimality of the final solution.
To conclude our work, we compared the current state-of-the-art with our formulation, showing that it can compete both in space and time efficiency.

Contents

Chapter 1

Introduction

AI Planning, a sub-area of Artificial Intelligence, has the objective of finding a sequence of actions, in a declaratively described system, to reach its goals while optimizing overall performance measures. Automated planners find the action to apply in each given state out of the set of possible actions for that state.
Fast Downward [Hel06] is a classical planning system based on heuristic search, which is leading the state-of-the-art in classical planning. One such heuristics that can be used to efficiently explore the search space, is the *delete-free relaxation* of the original problem, denoted as h^+.

1.1 Delete-free relaxation

The value of h^+ for a given planning problem is the optimal cost of the corresponding delete-relaxed planning problem, obtanied from the original problem by removing delete effects from all actions.
The value of h^+ is a lower bound of the optimal cost of the original problem, making it an admissible heuristic, and has been shown that having h^+ can significantly improve the efficiency of optimal planning [BH09]. Computing h^+, however, is NP-equivalent [Byl94] and is also hard to approximate [BH09].

Methods for computing h^+ can output optimal plans for a given delete-relaxed planning problem, equivalently important to find efficiently, since there exist delete-relaxed planning tasks that are of interest for the planning community. Moreover, optimal plans for classical planning problems, can be produced by iteratively solving and reformulating delete-relaxed tasks [Has12]. This can be done by repeatedly finding optimal plans for a delete-relaxed version of the original planning problem, and reducing the relaxation by reformulating the problem whenever a plan has been found for the delete-relaxed task, that is not a valid plan for the original problem.

We will first investigate on preprocessing techniques proposed by [IF15] and [HST12], aiming to simplify the problem by taking advantage of the less restrictive structure of the delete-relaxed task, to then explore the use of primal heuristics, to provide a warm-start

to our MIP solver, combining a greedy algorithm with concepts used to compute h^{add} [BG01].
Finally we will implement the models proposed by [IF15] and [FR22], which use different ways of modeling acyclicity in the model, respectively time labeling and vertex elimination, and compare them to our technique (of modeling acyclicity) of iteratively compiling violated disjunctive action landmarks [HST12], [BC11] and subtour elimination constraints (S.E.C.), adding those to the base model as cuts.

1.2 Preliminary definitions

There are multiple description languages for planning tasks: STRIPS, ADL, SAS$^+$ and PDDL; we will lean on the STRIPS formalism [FN71].

A *STRIPS planning task* is a 5-tuple $\Pi = (P, A, I, G, cost)$: P is a finite set of boolean variables (or *atomic propositions, atoms* for short); A is a finite set of *actions*: each action is a triple $(\text{pre}(a), \text{add}(a), \text{del}(a))$, where $\text{pre}(a), \text{add}(a), \text{del}(a) \subseteq P$ denote the set of *preconditions*, *positive effects* and *negative effects* of a; $I, G \subseteq P$ are respectively the *initial state* and *goal state*; $cost : A \to \mathbb{N}$ is a cost function that maps each action to a non-negative integer.

Executing an action a from a state $S \subseteq P$ transforms the state into $S' := exec_a(S) := S \setminus \text{del}(a) \cup \text{add}(a)$. Let $\pi = (a_0, ..., a_n)$ be a sequence of actions and $S \subseteq P$ a state: we denote with $S(\pi) = exec_{a_n}(...(exec_{a_0}(S)))$, the state achieved through π, starting from S. A *solution* to a planning task is a sequence of actions that transform the system from the initial state to a state $I(\pi) \subseteq G$. Moreover a feasible solution, or *plan*, is a solution $\pi = (a_0, ..., a_n)$ that satisfies $\forall i, pre(a_i) \subseteq I((a_0, ..., a_{i-1}))$, which means that each action's preconditions are achieved when using that action. Finally, the cost of an action sequence π is $cost(\pi) = \sum_{a \in \pi}(cost(a))$, and our goal is to find an *optimal plan* π^* with minimum cost (*optimal cost* c^*):

$$\pi^* = argmin_\pi\{cost(\pi),\ s.t.\ \pi \text{ is a plan for } \Pi\},\ c^* = cost(\pi^*)$$

We will limit ourselves to the *delete-relaxed task* $\Pi^+ = (P, A^+, I, G, cost)$, where A^+ is obtained by replacing, from each action, the negative effects with the empty set. Moreover, we will limit to the case in which $I = \emptyset$ for simplicity: in a delete-relaxed task, an atom that is true in the initial state will remain true throughout the whole plan, making it useless. This won't restrict the use case of this work to instances with empty initial states, since it's easy to remove the initial state from any instance without changing its optimal plan: $\Pi^+_I = (P \setminus I, A^+_I, \emptyset, G \setminus I, cost)$, where $A^+_I = \{(\text{pre}(a) \setminus I, \text{add}(a) \setminus I, \emptyset), \forall a \in A^+\}$.

Chapter 2

Current state of the art

To build a MIP formulation that solves the delete-relaxed task Π^+, we identify two sections:

1. Base IP model: defining variables for atoms, actions and *first achievers* [IF15], aswell as a basic set of constraints, aiming to enforce the basic functionalities of actions;

2. Modeling acyclicity: ensuring that the solution is in fact a plan, by adding constraints that make sure that a timeline of actions can be constructed from the solution obtained by the MIP solver.

The straightforward IP models, proposed by the literature, are often intractable and not useful in practice for computing h^+; this has lead the research community to develop preprocessing techniques to reduce the size and complexity of the IP models.

2.1 Base IP model

Different models have been proposed in past years; we will use the base model proposed in [FR22]:
$x_a \in \{0, 1\}$ indicates whether the action a is used in a solution π.

$$x_a = \begin{cases} 0 & \text{if } a \notin \pi \\ 1 & \text{if } a \in \pi \end{cases} \quad \forall a \in A^+$$

$x_p \in \{0, 1\}$ indicates whether an atom p is achieved in a solution π.

$$x_p = \begin{cases} 0 & \text{if } p \notin I(\pi) \\ 1 & \text{if } p \in I(\pi) \end{cases} \quad \forall p \in P$$

$x_{a,p} \in \{0, 1\}$ indicates whether a is the first achiever of p.

$$x_{a,p} = \begin{cases} 1 & \text{if } a \text{ is the first achiever of } p \\ 0 & \text{otherwise} \end{cases} \qquad \forall a \in A^+, \forall p \in \text{add}(a)$$

Having defined the variables that will be used in our formulation, the model is defined as follows:

$$\begin{cases} \min \sum_{a \in A^+} x_a \cdot cost(a) & \\ (C1) \quad \sum_{a \in A_p^+} x_{a,p} = x_p & \forall p \in P,\ A_p^+ = \{a \in A^+,\ s.t.\ p \in \text{add}(a)\} \\ (C2) \quad \sum_{a \in A_{p,q}^+} x_{a,q} \leq x_p & \forall p, q \in P,\ A_{p,q}^+ = \{a \in A^+,\ s.t.\ p \in \text{pre}(a), q \in \text{add}(a)\} \\ (C3) \quad x_{a,p} \leq x_a & \forall a \in A^+,\ p \in \text{add}(a) \\ (C4) \quad x_p = 1 & \forall p \in G \end{cases}$$

$(C1)$: An atom is achieved iff an action is its first achiever; moreover there can be only one first achiever per atom;

$(C2)$: An action can be a first achiever only if its preconditions are achieved;

$(C3)$: An action can be a first achiever only if it's used;

$(C4)$: Each atom in the goal state is achieved;

Note that, since there's no upper bound on action indicator variables, 0-cost actions might be flagged as used, even if they aren't, and still be a plan with optimal cost.
If we were just looking for the optimal cost c^*, this wouldn't bother us; however, since we are interested in the optimal plan π^*, we must acknowledge those actions by filtering the solution obtained by removing actions that aren't first achiever of any atom.

2.2 Modeling acyclicity

As shown both in [IF15] and [FR22], any solution π needs to be acyclic for it to be a plan, hence the need to add variables and constraints to force such acyclicity in the base model.
Multiple methods have been proposed: the time labeling method was first proposed in [IF15] while vertex elimination was proposed in [FR22]; the latter, however, proposed a revised time labeling method, proving that it was better than its predecessor, hence we will only mention the improved version.

2.2.1 Time labeling

For the time labeling method, we just need to create for each atom p, a timestamp $t_p \in \{0, ..., |A^+|\}$.
The constraints to add to the model will then be:

$$(C5) \quad t_p + 1 \leq t_q + |P|(1 - x_{a,q}) \qquad \forall a \in A^+, p \in \text{pre}(a), q \in \text{add}(a)$$

This constraint reads: if a is first achiever of q, then every $p \in \text{pre}(a)$ must have a timestamp that precedes than q.

The correctness of this model is implied from its construction, having tackled directly the problem of having to construct a timeline of actions out of the obtained solution with constraint $(C5)$.

2.2.2 Vertex elimination graph

To explain how to use a vertex elimination graph to model acyclicity, we first need to define a *causal relation graph* [RR22]. A causal relation graph of Π^+ is $G_{\Pi^+} = (P, E_{\Pi^+})$, where $E_{\Pi^+} = \{(p, q), s.t. \exists a \in A^+ \text{ with } p \in \text{pre}(a), q \in \text{add}(a)\}$. As shown in [FR22], let $\mathcal{O}$ be an elimination ordering for G_{Π^+}, and $G^*_{\Pi^+} = (P, E^*_{\Pi^+})$ be the vertex elimination graph of G_{Π^+} according to $\mathcal{O}$. Let Δ be the set of all triangles produced by elimination ordering $\mathcal{O}$ for graph G_{Π^+}: members of Δ are all ordered triples $(p, q, r)\, s.t.\, (p, r)$ is added to $E^*_{\Pi^+}$ by eliminating q.
We then create a variable for each edge $(p, q) \in E^*_{\Pi^+}$, $e_{p,q} \in \{0, 1\}\, \forall (p, q) \in E^*_{\Pi^+}$ and add the following constraints:

$$\begin{cases} (C6) & x_{a,q} \leq e_{p,q} & \forall a \in A^+, p \in \text{pre}(a), q \in \text{add}(a) \\ (C7) & e_{p,q} + e_{q,p} \leq 1 & \forall (p, q) \in E^*_{\Pi^+} \\ (C8) & e_{p,q} + e_{q,r} - 1 \leq e_{p,r} & \forall (p, q, r) \in \Delta \end{cases}$$

The correctness of the resulting model, composed of constraints $(C1)$ to $(C4)$ and $(C6)$ to $(C9)$, is proved in [FR22].

We will identify the two formulations respectively $\text{TL}(\Pi^+)$ and $\text{VE}(\Pi^+)$.

2.3 Preprocessing

2.3.1 Landmark-based model reduction

A *landmark* is an element which needs to be used in every feasible solution [HPS04]. We define *fact landmarks* and *action landmarks*, as in [GB12]: a fact landmark of a planning task Π is an atom that becomes true in some state of every plan, and similarly, an action landmark of a planning task Π is an action that is included in every plan.

Moreover, a fact or action landmark l if a landmark for an atom p if l is a landmark for the task $(P, A, I, \{p\}, cost)$, and similarly, l is a landmark for an action a if l is a landmark for the task $(P, A, I, \text{pre}(a), cost)$. For our delete-relaxed task Π^+, this leads to an obvious simplification of our model:

$$(P1) \quad x_p, x_a \equiv 1 \qquad \forall p \in P, a \in A^+ \text{ that is a fact/action landmark for (an atom in) } G$$

We can easily find a fact landmark p or an action landmark a by checking wether $(P, A^+ \setminus A_p^+, I, G, cost)$, with $A_p^+ = \{a \in A^+, \, s.t. \, p \in \text{add}(a)\}$, or $(P, A^+ \setminus \{a\}, I, G, cost)$ respectively, are infeasible tasks; however, since this is algorithm will be accompanied by other, more powerful, preprocessing techniques, we opted for the efficient extraction method, which however doesn't compute all landmarks, proposed in [IF15]:

Algorithm 1 Efficient landmark extracion algorithm

```
L[p] ← P for each p ∈ P        # L[p] stores candidates of fact landmarks for p ∈ P
S ← ∅
for a ∈ A+ do
    insert a into a FIFO queue Q if pre(a) ⊆ S
end for
while Q ≠ ∅ do
    retrieve an action a from Q
    for p ∈ add(a) do
        S ← S ∪ {p}
        X ← L[p] ∩ (add(a) ∪ ⋃_{p'∈pre(a)} L[p'])
        if L[p] ≠ X then
            L[p] ← X
            for a' ∈ {a ∈ A+, s.t. p ∈ pre(a)} do
                insert a' into Q if pre(a') ⊆ S and a' ∉ Q
            end for
        end if
    end for
end while
# Now L[p] contains the set of fact landmarks for p ∈ P
```

We conclude our landmark-based model reduction by extracting action landmarks as follows: if an atom p is a fact landmark for G, and if only one action a can achieve p, then a is an action landmark for G.

2.3.2 First achievers filtering

In [Has12], a first achiever is defined by achievability of a proposition; however, we can check wether an action *can* be a first achiever of an atom, and remove variables associated to those who can't [IF15]: an action a is a first achiever of a proposition p if $p \in \text{add}(a)$ and p is not a fact landmark for a. We can use this concept fo strenghten future preprocessing techniques: we define $\text{fadd}(a) = \{p \in \text{add}(a), \, s.t. \, p \text{ is not a fact landmark for } a\}$. We

can also use this new definition of first achievers to remove some variables from our formulation:

$$(P2) \quad x_{a,p} \equiv 0 \qquad \forall a \in A^+, p \in \text{add}(a),\ s.t.\ p \text{ is a fact landmark for } a$$

2.3.3 Relevance analysys

Relevance analysis is widely used to eliminate atoms and actions that are irrelevant to a task. In [IF15], a backchaining relevance analysis is proposed, where an action a is relevant if $\text{fadd}(a) \cap G \neq \emptyset$ or there exists a relevant action a' satisfying $\text{fadd}(a) \cap pre(a') \neq \emptyset$, and an atom p is relevant if $p \in G$ or there exists a relevant action a with $p \in \text{pre}(a)$. In [HST12], this relevance analysis is strenghtened by considering as relevant only actions that are possible first achievers of a relevant atom ($\text{fadd}(a) \cap \{p \in P,\ s.t.\ p \text{ is relevant}\} \neq \emptyset$). Actions and atoms that aren't relevant, can be removed from our formulation:

$$(P3) \quad x_a, x_p \equiv 0 \qquad \forall a \in A^+, p \in P,\ \text{that isn't relevant}$$

As noted in [IF15], a fact landmark might be found not relevant: in this case we just set $x_p \equiv 1$.

2.3.4 Dominated actions extraction

In a delete-relaxed task, if two actions have the same positive effects, it's clearly sufficient to use at most one of those two actions [IF15]: given a feasible delete-relaxed task Π^+, there exists an optimal plan that doesn't contain $b \in A^+$ if there exists an action $a \in A^+$ satisfying ($\text{fadd}(b) \subseteq fadd(a)$ and $\forall p \in pre(a),\ p$ is a fact landmark for b and $cost(b) \geq cost(a)$), and we call such b as a *dominated action*.

$$(P4) \quad x_a \equiv 0 \qquad \forall a \in A^+ \text{ that is a dominated action}$$

This constraint might prune feasible, even some optimal, solutions, but as it's shown in [IF15] at least one optimal solution remains.

2.3.5 Inverse actions extraction

For two actions $a, b \in A^+$, a is an *inverse action* of b if $\text{add}(a) \subseteq pre(b)$ and $add(b) \subseteq \text{pre}(a)$. Clearly, if a is an inverse action of b, then b must be an inverse action of a.
For a delete-relaxed task Π^+, if $a, b \in \pi$ and a is an inverse action of b, then there exist a feasible plan π' with at most one of a and b in it, and $cost(\pi') \leq cost(\pi)$ [IF15].
We can add this constraint as follows:

$$(P5) \quad x_a + x_b \leq 1 \qquad \forall a, b \in A^+,\ s.t.\ a \text{ is an inverse action of } b$$

2.3.6 Efficiency concerns

Section (2.3.4) is one of the most effective preprocessing technique among the ones we presented; however, a naive algorithm that would require $O(|A^+|^2)$ steps, might require too much time for some of the most demanding instances.

Algorithm 2 Naive dominated actions extraction

```
D ← ∅            # D is the set of dominated actions
for a ∈ A+ do
    for b ∈ A+ do
        if (fadd(b) ⊆ fadd(a) and ∀p ∈ pre(a), p is a fact landmark for b and cost(b) ≥ cost(a)) then
            D ← D ∪ {b}
        end if
    end for
end for
```

This concern extends also to section (2.3.5), therefore we felt the need to devise a data structure to speedup the filtering among actions. To explain how this data structure works, we will explain it using the example of the dominated actions extraction.
The problem lies in the inner loop: we, currently, are looking blindly among all actions for an action b that is dominated by a; we need a way to filter among all actions, a smaller set of actions that contains all, but not necessarily only, the actions dominated by a. Luckily we know that if $\text{fadd}(b) \not\subseteq \text{fadd}(a)$, b is not dominated by a, hence all those actions can be removed. We use a binary tree to store actions in a way that we can efficiently get all but those actions.
To add an action a to the tree T (T.insert$(a, \text{fadd}(a))$):

1. Start from the root
2. For each $p \in P$, if $p \notin \text{fadd}(a)$ we go to the left node, otherwise we go to the right node
3. Once we reached a leaf, we append a to a list in that leaf

Then, given an action a, if we need to find all actions b that satisfy $\text{fadd}(b) \subseteq \text{fadd}(a)$ (T.search(fadd(a))):

1. Start from the root
2. For each $p \in P$, if $p \notin \text{fadd(a)}$ we go to the left node, otherwise we expand both
3. At the end we'll have a list of leaves we reached: all actions stored in the lists of those leaves, are all the actions b that satisfy $\text{fadd}(b) \subseteq \text{fadd}(a)$

The proof of this last statement lies on the construction of the tree: consider a generic $p \in P$, if $p \in \text{fadd}(a)$ then fadd(b) might have or not have p, so we branch our search;

if $p \notin \text{fadd}(a)$, then we are interested only in actions b that satisfy $p \notin \text{fadd}(b)$, hence we can just go left without branching, since all actions that didn't have p among its first achievers, were inserted in the tree going left at this step.

Algorithm 3 Improved dominated actions extraction

```
D ← ∅                    # D is the set of dominated actions
T ← empty binary tree
for a ∈ A+ do
    T.insert(a, fadd(a))
end for
for a ∈ A+ do
    for b ∈ T.search(fadd(a)) do
        if (∀p ∈ pre(a), p is a fact landmark for b and cost(b) ≥ cost(a)) then
            D ← D ∪ {b}
        end if
    end for
end for
```

This technique, with some further optimizations, led to a huge performance increase in our dominated and inverse actions extraction: this is due to the fact that, usually, actions don't have many positive effects, hence we don't have many branching in our search; this means that we have an efficient search for possible dominated actions, and a smaller number of iterations in the inner cycle. This technique, however, is much less efficient for example when looking for all actions that can be applied from a specific state s (using $\text{pre}(a)$ for the insert part and s for the search part), since a state might contain many atoms, increasing exponentially the number of branchings needed to complete the search. This issue will be tackled in chapter 3, where we will compute a sub-optimal plan to provide as a warm-start to our MIP solver and we will need an efficient way of retrieving applicable actions.

We define additional sets to refer to the atoms/actions eliminated/fixed by this pre-processing:

$$P^e = \{p \in P,\ s.t.\ x_p \text{ has been set to } 0\},\ P^f = \{p \in P,\ s.t.\ x_p \text{ has been set to } 1\}$$
$$A^{+,e} = \{a \in A^+,\ s.t.\ x_a \text{ has been set to } 0\},\ A^{+,f} = \{a \in A^+,\ s.t.\ x_a \text{ has been set to } 1\}$$

By adding constraints $(P1)$-$(P5)$ to $\text{TL}(\Pi^+)$ and $(P1)$-$(P4)$ to $\text{VE}(\Pi^+)$, we obtain $\text{TL}^e(\Pi^+)$ and $\text{VE}^e(\Pi^+)$ respectively. We don't add $(P5)$ to $\text{VE}(\Pi^+)$ since $(P5)$ is included in $(C7)$, from our vertex elimination acyclicity modeling.

Chapter 3

Primal heuristics

A warm start in Mixed Integer Programming refers to providing an initial or partial solution to the solver before it begins its solving process. This can significantly improve performance in several ways:

1. Reduced computation time: A good initial solution allows the solver to establish upper bounds early, potentially pruning large portions of the search tree.

2. Better branch selection: With good incumbent solutions, branching decisions can be more effective as the solver has better bounds.

3. Improved cut generation: Warm starts can help generate more effective cutting planes early in the solution process.

In this chapter we explore efficient algorithms to compute good primal heuristic solutions, that can be used as warm start to speed up the search.

3.1 Greedy heuristic

When looking for a feasible solution to Π^+, starting from the initial state, we can simply iteratively look for actions that can be applied on the current state, adding them to the plan until we reach G: clearly, whenever we end up in a situation in which no action can be applied, and we didn't reach G yet, we prove the infeasibility of the task Π^+.

Algorithm 4 Greedy algorithm

$S \leftarrow I$ # current state
$p \leftarrow []$ # heuristic solution
while $G \not\subseteq S$ **do**
  candidates $\leftarrow \{a \in A^+, \; s.t. \; a \notin p, \text{ and } \text{pre}(a) \subseteq S\}$
  $a \leftarrow$ greedy choice(candidates)
  enqueue a to p
  $S \leftarrow S \cup \text{add}(a)$
end while

We can immediatelly improve this algorithm by noting that:

1. Actions that were fixed during our preprocessing should have the priority
2. Actions that were eliminated during our preprocessing can be ignored
3. Actions that only add already achieved atoms can be ignored

Algorithm 5 Improved greedy algorithm

$S \leftarrow I$ # current state
$p \leftarrow []$ # heuristic solution
while $G \not\subseteq S$ **do**
  if $\exists a \in A^{+,f}$, $s.t.$ pre$(a) \subseteq S$ **then**
    enqueue a to p
    $S \leftarrow S \cup \text{add}(a)$
  else
    candidates $\leftarrow \{a \in A^{+} \setminus A^{+,e}, s.t.\ a \notin p, \text{ and pre}(a) \subseteq S, \text{ and add}(a) \not\subseteq S\}$
    $a \leftarrow$ greedy choice(candidates)
    enqueue a to p
    $S \leftarrow S \cup \text{add}(a)$
  end if
end while

3.1.1 Filtering applicable actions

TODO ...

The last thing we're left to do is define a *greedy choice*: a decision that selects the action that is most appealing at the current moment, without considering the future consequences of that choice.

3.2 Naive greedy choice

The naive greedy choice will be to choose the action with lowest cost among the candidates:

$$\text{greedy choice}_c(\text{candidates}) = argmin_a\{cost(a),\ \forall a \in \text{candidates}\}$$

However there's another possible greedy choice, choosing the action that adds the highest number of new atoms:

$$\text{greedy choice}_a(\text{candidates}, S) = argmax_a\{|add(a) \setminus S|,\ \forall a \in \text{candidates}\}$$

We can combine those two into a single improved greedy choice:

$$\text{greedy choice}_{c\times a}(\text{candidates}, S) = argmin_a\{\frac{cost(a)}{|add(a) \setminus S|},\ \forall a \in \text{candidates}\}$$

3.3 Hadd estimator

TODO ...

The use of the simple $(c \times a)$ or hadd based greedy heuristics as warm-start to our formulations, will be indicated with $_g$ and $_h$ respectively, i.e. $\text{VE}^e(\Pi^+)$ will either become $\text{VE}^e_g(\Pi^+)$ or $\text{VE}^e_h(\Pi^+)$.

Chapter 4

Iterative approach to modeling acyclicity

TODO ...

4.1 Using landmarks to model acyclicity

TODO ...

4.1.1 Minimal landmarks

TODO ...

4.1.2 Complete landmarks

TODO ...

4.2 Using S.E.C. to model acyclicity

TODO ...

The formulation that use (both) landmarks to model acyclicity will be indicated with LM(Π^+), while the formulation that use (both) landmarks aswell as S.E.C. LMS(Π^+).

Chapter 5

Instances dataset

TODO ...

Chapter 6

Empirical results

	Time labeling	Vertex elimination	Landmarks / S.E.C.
Base Model:	$\mathrm{TL}(\Pi^+)$	$\mathrm{VE}(\Pi^+)$	$\mathrm{LM}(\Pi^+)$, $\mathrm{LMS}(\Pi^+)$
Base + preprocessing:	$\mathrm{TL}^e(\Pi^+)$	$\mathrm{VE}^e(\Pi^+)$	$\mathrm{LM}^e(\Pi^+)$, $\mathrm{LMS}^e(\Pi^+)$
Base + prep + greedy w.s.:	$\mathrm{TL}^e_g(\Pi^+)$	$\mathrm{VE}^e_g(\Pi^+)$	$\mathrm{LM}^e_g(\Pi^+)$, $\mathrm{LMS}^e_g(\Pi^+)$
Base + prep + hadd w.s.:	$\mathrm{TL}^e_h(\Pi^+)$	$\mathrm{VE}^e_h(\Pi^+)$	$\mathrm{LM}^e_h(\Pi^+)$, $\mathrm{LMS}^e_h(\Pi^+)$

Chapter 7

Future works

TODO ...

Chapter 8

Conclusions

TODO ...

Bibliography

[FN71] Richard E. Fikes and Nils J. Nilsson. "STRIPS: a new approach to the application of theorem proving to problem solving". In: Morgan Kaufmann Publishers Inc., 1971. DOI: 10.1016/0004-3702(71)90010-5.

[Byl94] Tom Bylander. "The computational complexity of propositional STRIPS planning". In: *Artificial Intelligence* 69 (1994). DOI: 10.1016/0004-3702(94)90081-7.

[BG01] Blai Bonet and Héctor Geffner. "Planning as heuristic search". In: *Artificial Intelligence* 129 (2001). DOI: 10.1016/S0004-3702(01)00108-4.

[HPS04] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. "Ordered landmarks in planning". In: *J. Artif. Int. Res.* 22 (2004). DOI: 10.1613/jair.1492.

[Hel06] M. Helmert. "The Fast Downward Planning System". In: *Journal of Artificial Intelligence Research* 26 (2006). DOI: 10.1613/jair.1705.

[BH09] Christoph Betz and Malte Helmert. "Planning with h+ in Theory and Practice". In: Springer Berlin Heidelberg, 2009. DOI: 10.1007/978-3-642-04617-9_2.

[BC11] Blai Bonet and Julio Castillo. "A complete algorithm for generating landmarks". In: AAAI Press, 2011. DOI: 10.1609/icaps.v21i1.13482.

[GB12] Avitan Gefen and Ronen Brafman. "Pruning Methods for Optimal Delete-Free Planning". In: *Proceedings of the International Conference on Automated Planning and Scheduling* 22 (2012). DOI: 10.1609/icaps.v22i1.13522.

[Has12] Patrik Haslum. "Incremental Lower Bounds for Additive Cost Planning Problems". In: *Proceedings of the International Conference on Automated Planning and Scheduling* 22 (2012). DOI: 10.1609/icaps.v22i1.13511.

[HST12] Patrik Haslum, John Slaney, and Sylvie Thiebaux. "Minimal Landmarks for Optimal Delete-Free Planning". In: *ICAPS 2012 - Proceedings of the 22nd International Conference on Automated Planning and Scheduling* 22 (2012). DOI: 10.1609/icaps.v22i1.13534.

[IF15] Tatsuya Imai and Alex Fukunaga. "On a Practical, Integer-Linear Programming Model for Delete-Free Tasks and its Use as a Heuristic for Cost-Optimal Planning". In: *Journal of Artificial Intelligence Research* 54 (2015). DOI: 10.1613/jair.4936.

[FR22] Masood Feyzbakhsh Rankooh and Jussi Rintanen. "Efficient Computation and Informative Estimation of h+ by Integer and Linear Programming". In: *Proceedings of the International Conference on Automated Planning and Scheduling* 32 (2022). DOI: 10.1609/icaps.v32i1.19787.

[RR22] Masood Feyzbakhsh Rankooh and Jussi Rintanen. "Efficient Encoding of Cost Optimal Delete-Free Planning as SAT". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36 (2022). DOI: 10.1609/aaai.v36i9.21228.