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
Graduation date 12/09/2025

Abstract

We compare current state-of-the-art MIP formulations for solving the delete-free relaxation in cost-optimal planning, integrating various preprocessing techniques from different publications and developing primal heuristics to provide warm-start solutions to the MIP solver. We then explore a novel approach to model acyclicity by computing violated landmarks and subtour elimination constraints (S.E.C.), adding them as constraints: computing all landmarks a priori is intractable, since the number of potential landmarks grows exponentially with the problem size, just like S.E.C., making any brute-force approach computationally infeasible; instead we employ a constraint generation approach where new constraints are identified dynamically: upon encountering an infeasible solution, we detect violated cuts within that solution and incorporate them as additional constraints, progressively constructing a constraint set required for both feasibility and optimality. Our experimental evaluation demonstrates that this landmark-based formulation achieves competitive performance with existing methods in both space and time efficiency.

Contents

Chapter 1

Introduction

AI Planning, a sub-area of Artificial Intelligence, aims to find a sequence of actions for a declaratively described system to reach its goals while optimizing overall performance measures.

The A^* algorithm is a fundamental graph traversal and pathfinding algorithm, that plays a central role in optimal classical planning. A^* systematically explores the search space by maintaining a priority queue of states ordered by $f(s) = g(s) + h(s)$, where $g(s)$ represents the cost of the path from the initial state to state s, and $h(s)$ is a heuristic estimate of the cost from s to the goal. The algorithm's optimality guarantee depends on the heuristic being admissible (never overestimating the true cost to goal) and consistent (satisfying the triangle inequality). In planning, states correspond to world configurations, and actions represent transitions between states. The quality of the heuristic function significantly impacts A^*'s performance: while an admissible heuristic ensures optimality, a more informed (higher-valued) admissible heuristic typically leads to fewer state expansions and faster goal discovery. Various heuristics have been developed for planning, ranging from simple domain-independent heuristics like h^{max} and h^{add} to more sophisticated approaches like the delete-relaxation heuristic h^+, landmark-based heuristics, and pattern database heuristics.

Methods for computing h^+ can output optimal plans for a given delete-relaxed planning problem, which is also important to compute efficiently, since there exist delete-relaxed planning tasks that are of interest for the planning community [GB11]. Moreover, optimal plans for classical planning problems can be found by iteratively solving and reformulating delete-relaxed tasks [Has12]. This is achieved by iteratively solving the delete-relaxed version of the planning problem and if the resulting plan is not valid for the original problem, the relaxation is reduced by reformulating the problem, and the process is repeated.

1.1 Mixed Integer Programming

Mixed Integer Programming (MIP) is a mathematical optimization technique that extends linear programming to handle variables that must take integer values, making it particularly suitable for modeling discrete decision problems like planning. A MIP formulation consists of an objective function to minimize or maximize, subject to a set of linear constraints, where some variables are restricted to integer values (typically binary 0-1 variables). In the context of planning, MIP models typically use binary variables to represent action selection decisions, with constraints encoding precondition-effect relationships and goal achievement requirements. The strength of MIP lies in its ability to naturally express logical relationships and combinatorial constraints that arise in planning problems. Modern MIP solvers like CPLEX, Gurobi, and SCIP employ sophisticated techniques including branch-and-bound, cutting planes, and presolving to efficiently explore the solution space. These solvers can leverage both the linear programming relaxation (which provides lower bounds) and various heuristics to find high-quality integer solutions. The branch-and-bound framework systematically partitions the solution space by branching on fractional variables, while cutting planes add valid inequalities to strengthen the relaxation. Additionally, MIP solvers benefit from warm-start capabilities, where good initial solutions can significantly accelerate the search process, and from the ability to incrementally add constraints (cuts) during the solving process, making them well-suited for iterative refinement approaches in planning.

1.2 Delete-free relaxation

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

We will first investigate 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. Then we will 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^{max} and h^{add} [BG01].

Finally, we will implement the models proposed by [IF15; FR22], which use different ways of modeling acyclicity, namely time labeling and vertex elimination. We will compare them to our technique of modeling acyclicity, which involves iteratively compiling violated disjunctive action landmarks [HST12; BC11] and subtour elimination constraints (S.E.C.), adding them to the base model as cuts.

1.3 Preliminary definitions

A *STRIPS* [FN71] *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*, *add effects* and *delete effects* of a; $I, G \subseteq P$ are respectively the *initial state* and *goal state*; $cost : A \to \mathbb{R}_+$ is a cost function that maps each action to a non-negative real.

Executing an action a from a state $S \subseteq P$ transforms the state into $S' \coloneqq exec_a(S) \coloneqq S \setminus \text{del}(a) \cup \text{add}(a)$. 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 $I(\pi) \supseteq G$. Moreover a feasible solution, or *plan*, is a solution $\pi = (a_0, ..., a_n)$ that satisfies $\forall i, pre(a_i) \subseteq I((a_0, ..., a_{i-1}))$, meaning that the preconditions of each action are met before it is executed. 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 delete effects with the empty set.

Moreover, we will limit ourselves 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, rendering it irrelevant. This will not restrict the use case of this work to instances with empty initial states, since it is 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], as well 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.

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 action achieving p in π.

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

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

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

$(C1)$: An atom is achieved iff an action is its first achiever; moreover there can be at most 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 is no upper bound on action indicator variables, 0-cost actions might be flagged as used, even if they are not, and still be a plan with optimal cost.
If we were just looking at 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

The causal graph associated with a problem Π^+ needs to be acyclic for the solutions provided by the base model to be plans [FR22], hence we need to add variables and constraints to force such acyclicity.
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 dominated 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 $t_p \in \{0, ..., |P|\}$.
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 the first achiever of q, then every $p \in \text{pre}(a)$ must have a timestamp that precedes that of 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 the *causal relation graph* [RR22]. A causal relation graph of Π^+ is a graph $G_{\Pi^+} = (P, E_{\Pi^+})$, 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, i.e. eliminating a vertex with minimal total number of incoming and outgoing edges in the remaining graph.
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:

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

We implemented the elimination ordering $\mathcal{O}$ using a minimum degree heuristic, choosing to eliminate vertices with the smallest degree (counting both in-going and out-going edges) first.

The correctness of the resulting model, composed of constraints $(C1)$ to $(C4)$ and $(C6)$ to $(C8)$, is proved in [FR22].

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

2.3 Preprocessing

Such straightforward IP models are often too slow in practice for computing h^+; this has led the research community to develop preprocessing techniques to reduce the size and complexity of the IP models.

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 is 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 extraction 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 to strengthen 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:

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

2.3.3 Relevance analysis

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 $\text{fadd}(a) \cap \text{pre}(a') \neq \emptyset$, 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 add 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 ($\text{fadd}(b) \subseteq \text{fadd}(a)$ and $\forall p \in \text{pre}(a)$, 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 is shown in [IF15] at least one optimal solution remains.

Efficiency concerns

The dominated actions extraction is one of the most effective preprocessing techniques among the ones we presented; however, a naive algorithm that would require $O(|A^+|^2)$ steps, would be too time-consuming for some of the most demanding instances. We felt the need to devise a data structure to speedup the filtering among actions.

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

Algorithm 2 Naive dominated actions extraction

$D \leftarrow \emptyset$ # D is the set of dominated actions
for $a \in A^+$ **do**
  for $b \in A^+$ **do**
    if (fadd(b) $\subseteq$ fadd(a) and $\forall p \in$ pre(a), p is a fact landmark for b and $cost(b) \geq cost(a)$) **then**
      $D \leftarrow D \cup \{b\}$
    end if
  end for
end for

by a. Luckily we know that if fadd(b) $\not\subseteq$ 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, fadd(a))):

1. Start from the root
2. For each $p \in P$, if $p \notin$ 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 fadd(b) $\subseteq$ fadd(a) (T.search(fadd(a))):

1. Start from the root
2. For each $p \in P$, if $p \notin$ 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 fadd(b) $\subseteq$ fadd(a)

The proof of this last statement lies in the construction of the tree: consider a generic $p \in P$, if $p \in$ fadd(a) then fadd(b) might have or not have p, so we branch our search; if $p \notin$ fadd(a), then we are interested only in actions b that satisfy $p \notin$ 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.

This technique, with some other minor optimizations, led to a huge performance increase in our dominated actions extraction: this is due to the fact that, usually, actions don't have many add 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 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.

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 issue will be tackled in chapter 3, where we will compute a primal heuristic 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 preprocessing:

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

By adding constraints $(P1)$-$(P4)$ to $\mathrm{TL}(\Pi^+)$ and $\mathrm{VE}(\Pi^+)$, we obtain $\mathrm{TL}^e(\Pi^+)$ and $\mathrm{VE}^e(\Pi^+)$ respectively.

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 particular via reduced cost fixing.

In this chapter we explore efficient algorithms to compute good primal heuristic solutions that can be used as a 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 in 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 have not yet reached G, we prove the infeasibility of the task Π^+.

Algorithm 4 Greedy algorithm

$S \leftarrow \emptyset$
$plan \leftarrow []$
while $G \not\subseteq S$ **do**
    candidates $\leftarrow \{a \in A^+,\ s.t.\ a \notin plan,\ \text{and pre}(a) \subseteq S\}$
    $a \leftarrow$ greedy_choice(candidates)
    enqueue a to $plan$
    $S \leftarrow S \cup \text{add}(a)$
end while

We can immediately improve this algorithm by noting that:

1. Actions that were fixed during our preprocessing should have the priority ($A^{+,f}$, defined in Chapter 2)

2. Actions that were eliminated during our preprocessing can be ignored ($A^{+,e}$, defined in Chapter 2)

3. Actions that only add already achieved atoms can be ignored

Algorithm 5 Revised greedy algorithm

$S \leftarrow \emptyset$
$plan \leftarrow []$
while $G \not\subseteq S$ **do**
  if $\exists a \in A^{+,f} \setminus plan$, *s.t.* $\text{pre}(a) \subseteq S$ **then**
    enqueue a to $plan$
  else
    candidates $\leftarrow \{a \in A^{+} \setminus A^{+,e}, \; s.t. \; a \notin plan, \; \text{pre}(a) \subseteq S, \; \text{and add}(a) \not\subseteq S\}$
    $a \leftarrow$ greedy_choice(candidates)
    enqueue a to $plan$
  end if
  $S \leftarrow S \cup \text{add}(a)$
end while

3.1.1 Filtering applicable actions

We proposed in section 2.3.5 a way to efficiently filter among actions, only those that meet a specific criterion: we already anticipated that this technique is not suitable for this specific problem, since we would have a large number of branches in our search algorithm of the binary tree. Looking for candidate actions among all actions at each step is inefficient; instead we can iteratively compute candidate actions based on newly achieved atoms:

1. Start by computing, for each atom, the set of actions that have that atom among its preconditions: $A^{+}_{pre,p} = \{a \in A^{+} \setminus A^{+,e}, \; s.t. \; p \in \text{pre}(a)\} \quad \forall p \in P$

2. Before the greedy algorithm, initialize the candidates list as the set of actions with no preconditions; since the initial state is empty, those are the initial candidates: candidates $= \{a \in A^{+} \setminus A^{+,e}, \; s.t. \; \text{pre}(a) = \emptyset\}$

3. After making the greedy choice, update the candidates by expanding that list with actions that have newly achieved atoms among their preconditions: by using $A^{+}_{pre,p}$ for each newly added atom, we are looking only among possible new candidates, thus skipping most of the unreachable actions.

Algorithm 6 Efficient greedy algorithm

$S \leftarrow \emptyset$
$plan \leftarrow []$
candidates $\leftarrow \{a \in A^+ \setminus A^{+,e},\ s.t.\ \text{pre}(a) = \emptyset\}$
$A^+_{pre,p} \leftarrow \{a \in A^+ \setminus A^{+,e},\ s.t.\ p \in \text{pre}(a)\}\ \forall p \in P$
while $G \not\subseteq S$ **do**
  if $\exists a \in A^{+,f} \cap$ candidates **then**
    enqueue a to $plan$
  else
    $a \leftarrow$ greedy_choice(candidates)
    enqueue a to $plan$
  end if
  for $p \in \text{add}(a) \setminus S$ **do**
    for $a' \in A^+_{pre,p}$ **do**
      if $\text{pre}(a') \subseteq (S \cup \text{add}(a))$ and $a' \notin$ candidates **then**
        candidates $\leftarrow$ candidates $\cup \{a'\}$
      end if
    end for
  end for
  $S \leftarrow S \cup \text{add}(a)$
  for $a' \in$ candidates **do**
    if $\text{add}(a) \subseteq S$ **then**
      remove a' from candidates
    end if
  end for
end while

3.2 Naive greedy choice

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

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

However, there is another possible greedy choice, choosing the action that adds the largest number of new atoms:

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

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

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

Note that $|\text{add}(a) \setminus S| \neq 0$ since we remove all actions whose add effects are already in S.

3.3 h^{max} and h^{add} Estimators

Computing h^+ is NP-equivalent, hence techniques to estimate it have been proposed: h^{max} and h^{add} [BG01] aim to estimate the cost of achieving the goal atoms and then set their values to a combination of these costs. Suppose we are in the state S and we aim to reach G:

$$h_S^{max}(p) = \begin{cases} 0 & \text{if } p \in S \\ \min_{a \in A^+_{add,p}} \{cost(a) + \max_{q \in pre(a)} \{h_S^{max}(q)\}\} & \text{otherwise} \end{cases}$$

$$h_S^{add}(p) = \begin{cases} 0 & \text{if } p \in S \\ \min_{a \in A^+_{add,p}} \{cost(a) + \sum_{q \in pre(a)} \{h_S^{add}(q)\}\} & \text{otherwise} \end{cases}$$

with $A^+_{add,p} = \{a \in A^+ \setminus A^{+,e},\ s.t.\ p \in \text{add}(a)\}$, and

$$h_S^{max} = \max_{p \in G} \{h_S^{max}(p)\}, \qquad h_S^{add} = \sum_{p \in G} h_S^{add}(p)$$

h_I^{max} is considering the "cost" of the "most expensive" atom in the goal state, while h_I^{add} is isolating each atom in the goal state, and adding up all their "costs": we integrated an iterative computation into our greedy algorithm:

$$\text{greedy\_choice}_{h^{max}}(\text{candidates}, S) = argmin_a \{h_{S'}^{max},\ S' = S \cup \text{add}(a),\ \forall a \in \text{candidates}\}$$
$$\text{greedy\_choice}_{h^{add}}(\text{candidates}, S) = argmin_a \{h_{S'}^{add},\ S' = S \cup \text{add}(a),\ \forall a \in \text{candidates}\}$$

Computing from scratch $h_{S'}^{max}$ and $h_{S'}^{add}$ for each $a \in$ candidates and for each iteration of the greedy algorithm would require too much time, so we integrated an iterative computation of h^{max} and h^{add} into our greedy algorithm, which reuses the values from the previous iteration of h^{max} and h^{add} and updates only the necessary ones, using a trail to keep track of the changes that need to be reverted for future iterations.

Chapter 4

Iterative approach to modeling acyclicity

Another proposed technique for computing h^+ is to compute a *minimum-cost hitting set* for a complete set of disjunctive action landmarks generated on the fly [BC11; HST12]. We can integrate the work on MIP formulations and hitting sets for complete sets of disjunctive action landmarks by solving our base model without the acyclicity constraints, then using the MIP solver's callbacks for the candidate solutions to add iteratively violated landmarks as cuts. This is a promising approach because we are not just trying to solve the minimum cost hitting set, but we already have other constraints in our base model that can better guide the search process.

Moreover, it has been shown [HST12] that usually, the number of violated landmarks that need to be added to obtain the optimal plan is relatively small; a much smaller number than the number of constraints that would be needed to model acyclicity.

We provide the base model to our MIP solver, without the constraints aiming to model acyclicity; this means that the solutions it provides are not plans, and we can try to fix them from two perspectives:

1. The solution is infeasible because it violates at least one landmark. It uses infeasible "shortcuts" to reduce costs, which causes it to avoid the landmark.

2. Removing cycles: if a solution is infeasible, this means that we can find a cycle in the justification graph built from it; if no cycles can be found in such a graph, then the solution is also a plan; hence, by using Subtour Elimination Constraints (S.E.C.) on those cycles, we are pruning all infeasible solutions.

4.1 Using landmarks to model acyclicity

In contrast to action landmarks defined in section 2.3.1, henceforth, when we talk about a *landmark*, we are referring to a *disjunctive action landmark*, a set of actions where at least one action in the set must be included in any valid plan for Π^+ [BC11;

HST12]. Each time we find a new violated landmark $L \subseteq A^+$, we add a cut to the model:

$$\sum_{a \in L} x_a \geq 1$$

In [BC11; HST12], the authors solve a planning task by compiling it as a minimum cost hitting set problem, where they try to find the minimal subset of actions that does not violate any landmark. We can integrate their work with ours by complementing our base model with a set of hitting set constraints obtained from landmarks that are violated by our infeasible solutions.

As shown in both publications, solving the delete-free relaxation as a minimum cost hitting set is sufficient to obtain a plan π for Π^+; hence, by integrating their work into our base model, we will implicitly model the acyclicity in our model, guaranteeing its correctness.

We first define $R(\pi)$ as the set of facts actually reachable by only using actions in π: note that if π is not a plan $R(\pi) \neq I(\pi)$, because $R(\pi)$ is the set of facts that are actually reachable by executing only the actions in π, respecting the logical dependencies and ordering constraints, while $I(\pi)$ is the set of facts that would be true if we could execute all actions in π ignoring the preconditions constraints.

4.1.1 Cut section landmarks

If $G \subseteq R(\pi)$ then π (without loss of generality) is a plan. Otherwise consider the cut $(R(\pi), R(\pi)^C)$ of P; we generate a precondition choice function as follows:

$$\text{pcf}(a) = \begin{cases} \text{any } p \in \text{pre}(a) & \text{if pre}(a) \subseteq R(\pi) \\ \text{any } p \in \text{pre}(a) \setminus R(\pi) & \text{if pre}(a) \not\subseteq R(\pi) \end{cases}$$

Our claim is that π doesn't hit the cut-set, i.e. that there is no operator $a \in \pi$ that labels an edge in the causal graph, going from $R(\pi)$ into $R(\pi)^C$. Let $a \in \pi$:

1. If $\text{pre}(a) \subseteq R(\pi)$ then $\text{add}(a) \subseteq R(\pi)$ since it would be applicable, and no edge labeled by a crosses the cut

2. If $\text{pre}(a) \not\subseteq R(\pi)$ then $\text{pcf}(a) \notin R(\pi)$ and no edge labeled by a crosses the cut because such edges originate in $R(\pi)^C$

The violated landmark is then computed as the set of actions that label an edge in $(R(\pi), R(\pi)^C)$. This means these actions connect facts reachable by the current solution to facts not yet reachable, indicating a necessary path to the goal.

The complete explanation and proof of this method of extracting violated landmarks can be found in [BC11].

4.1.2 Complement landmarks

For any solution π such that $G \not\subseteq R(\pi)$, then $\pi^C := A \setminus \pi$ is a landmark for Π^+ (clearly, if with π we can not reach the goal, we must use one of the unused actions, hence the complement of π is a landmark). Moreover, if π is an *inclusion-maximal* such infeasible solution, π^C is an *inclusion-minimal landmark*. This leads to the following algorithm:

Algorithm 7 Find complementary landmarks

if $G \subseteq R(\pi)$ **then**
  π is a plan
else
  while $\exists a \in \pi^C$ s.t. $G \not\subseteq (R(\pi \cup a))$ **do**
    Add a to π
  end while
  π^C is an inclusion-minimal landmark for Π^+
end if

Detailed implementation steps of this approach are given in [HST12].

4.2 Using S.E.C. to model acyclicity

Another direct approach to modeling acyclicity that doesn't require adding variables to the model is to use Subtour Elimination Constraints: every time a cycle is found in the causal relation graph associated to our solution we can simply prevent that cycle c from appearing by adding a Subtour Elimination Constraint:

$$\sum_{x^{\pi}_{a,p} \in c} x_{a,p} \leq |c| - 1$$

Detecting cycles in a solution is straightforward: first we construct the causal graph associated to our solution π: $G_\pi := (P, E_\pi = \{(p,q),\ s.t. \exists a \in A^+ \text{ with } p \in \text{pre}(a), q \in \text{add}(a) \text{ and } x^{\pi}_{a,q} = 1\})$; we then use a depth-first approach to detect cycles in such a graph: once a DFS iteration sees a node twice, a cycle has been detected.

The formulation that uses (both) landmarks to model acyclicity will be indicated with LM(Π^+), the formulation that uses S.E.C. will be indicated with SEC(Π^+), and the formulation that uses (both) landmarks as well as S.E.C., LMS(Π^+).

Chapter 5

Integration with LMCUT

LMCUT (Landmark-Cut) [HD09] represents one of the most significant breakthroughs in admissible heuristics for optimal planning: it works by computing an admissible heuristic by identifying and exploiting disjunctive action landmarks in a systematic way that provides tight lower bounds for optimal planning problems.

5.1 Theoretical Foundation of LMCUT

The core principle of LMCUT lies in its iterative extraction of landmark cuts from the planning problem. In each iteration, LMCUT identifies a single landmark cut L and computes its cost contribution as $\min_{a \in L} \text{cost}(a)$. The problem is then simplified by reducing the cost of all actions in L by its cost contribution, and the process repeats until no further landmarks can be extracted. The final heuristic value is the sum of all cost contributions across iterations.

5.1.1 Landmark Cut Identification Process

LMCUT operates by constructing a relaxed planning graph that ignores delete effects and analyzes the flow of facts through this graph. The algorithm identifies landmark cuts through the following systematic approach:

Graph Construction: LMCUT constructs the justification graph during the computation of $h^{\max}$. The justification graph has facts F as vertices, and directed edges (arcs) from preconditions to their corresponding add effects. Specifically, when $h^{\max}$ is computed for each fact, a precondition choice function (pcf) maps each action to one of its preconditions with maximal $h^{\max}$-value. The justification graph then contains an arc from the precondition chosen by the pcf to each add effect of that action, with the arc labeled by the action.

Cut Identification: A landmark cut L is identified as a minimal set of actions such that every path from the initial state to any goal fact in the relaxed planning graph must pass through at least one action in L. This ensures that L represents a necessary bottleneck in achieving the goals.

Admissibility Preservation: The key insight ensuring admissibility is that any valid plan must execute at least one action from each identified landmark cut. Therefore, the minimum cost among actions in each cut provides a safe lower bound contribution to the total plan cost.

5.1.2 Iterative Cost Reduction

After identifying a landmark cut L with cost contribution $c = \min_{a \in L} \text{cost}(a)$, LMCUT modifies the problem by reducing the cost of every action $a \in L$ by c. This cost reduction reflects the fact that this contribution has already been accounted for in the heuristic value. The modified problem is then used for the next iteration, ensuring that:

1. *No double counting occurs*: Each unit of cost is counted exactly once across all iterations

2. *Monotonicity is preserved*: The heuristic value never decreases as more landmarks are discovered

3. *Termination is guaranteed*: The process terminates when no landmark cuts with positive cost contribution can be found

5.2 Connection to Our Work

Our integration of LMCUT into the MIP-based framework leverages this landmark generation capability by converting the discovered landmarks directly into constraints for our base model.

This integration provides several theoretical and practical advantages:

- *Enhanced preprocessing*: LMCUT landmarks serve as high-quality initial constraints that capture essential structural properties of the problem before the iterative constraint generation begins.

- *Improved root relaxation*: By incorporating LMCUT landmarks a priori, we strengthen the linear programming relaxation at the root node, potentially reducing the initial optimality gap and accelerating convergence.

- *Complementary landmark discovery*: While our iterative approach discovers landmarks dynamically from infeasible solutions, LMCUT identifies landmarks through structural analysis of the problem, providing complementary coverage of the constraint space.

5.3 Exploiting Tie-Breaking for Enhanced Landmark Generation

A critical aspect of LMCUT's algorithm is the selection of a precondition choice function (pcf) when multiple candidate preconditions exist for landmark extraction. As highlighted in [LF20], different tie-breaking strategies in the pcf selection can lead to the discovery of distinct sets of landmarks, each capturing different structural bottlenecks in the problem.

We exploit this flexibility by executing LMCUT multiple times with different tie-breaking approaches, accumulating the discovered landmarks to create a more comprehensive set of initial constraints. The tie-breaking strategies we implemented include:

- *Arbitrary Choice (ARB)*: Select the first optimal precondition according to the natural ordering of atoms. This provides a consistent baseline approach.
- *Inverse Choice (INV)*: Select the last optimal precondition according to the natural ordering. This strategy often discovers landmarks complementary to those found by the arbitrary choice method.
- *Random Choice (RND)*: Randomly select among all optimal preconditions with uniform probability. By running this strategy multiple times with different random seeds, we can explore various regions of the landmark space.
- *Value Decrease Minimization (VDM)*: As proposed in [LF20], this strategy prefers a precondition whose $h^{\max}$-value has decreased the least since the first iteration. This aims to reduce the depth of the backward exploration by selecting preconditions that should not have many zero-cost actions in the justification graph between the initial state and the selected precondition.

5.4 Algorithm Integration

The integration process follows these steps:

1. Execute LMCUT multiple times with different tie-breaking strategies on the preprocessed problem instance.
2. Collect all unique disjunctive action landmarks discovered across all executions.
3. Convert each aggregated landmark into a hitting set constraint for the base MIP model and add these constraints to the base model before initiating the iterative constraint generation process.

This multi-strategy approach significantly enriches the initial constraint set, potentially reducing the number of callback iterations required during the MIP solving process while maintaining the correctness guarantees of our formulation.

Chapter 6

Empirical results

Model	Description	Acyclicity Method
	State-of-the-art Models	
$\mathrm{TL}^{e}(\Pi^{+})$	Time labeling with integrated preprocessing: landmark-based model reduction (P1), first achievers filtering (P2), relevance analysis (P3), and dominated actions extraction (P4)	Time labeling
$\mathrm{VE}^{e}(\Pi^{+})$	Vertex elimination with integrated preprocessing: landmark-based model reduction (P1), first achievers filtering (P2), relevance analysis (P3), and dominated actions extraction (P4)	Vertex elimination
	Warm-Start Enhanced Models	
$\mathrm{TL}_{h}^{e}(\Pi^{+})$	Time labeling with preprocessing and h_{add}-based greedy heuristic providing warm-start solutions to the MIP solver	Time labeling
$\mathrm{VE}_{h}^{e}(\Pi^{+})$	Vertex elimination with preprocessing and h_{add}-based greedy heuristic providing warm-start solutions to the MIP solver	Vertex elimination
	Iteratively Generated Models	
$\mathrm{SEC}_{h}^{e}(\Pi^{+})$	Base model with preprocessing and warm-start, using iterative Subtour Elimination Constraints (S.E.C.) to model acyclicity through cycle detection and elimination	Iterative S.E.C.
$\mathrm{LM}_{h}^{e}(\Pi^{+})$	Base model with preprocessing and warm-start, using iterative disjunctive action landmark generation (cut section and complement landmarks) to model acyclicity	Iterative landmarks
$\mathrm{LMS}_{h}^{e}(\Pi^{+})$	Base model with preprocessing and warm-start, combining both iterative landmark generation and S.E.C. separation for comprehensive acyclicity modeling	Iterative landmarks + S.E.C.

Table 6.1: Summary of MIP formulations and variations evaluated in this work

We implemented our models using IBM ILOG CPLEX Optimization Studio version 22.1. All experiments were conducted on a cluster of Linux machines running Rocky Linux 8.10, each equipped with Intel(R) Xeon(R) CPU E5-2623 v3 processors featuring 4 cores at 3.00 GHz and 16GB of RAM. We used a timeout of 900 seconds (15 minutes) per problem instance and a memory limit of 4 GB for the CPLEX solver.

The benchmark set consists of 2799 problem instances extracted from 49 different domains across both satisficing and optimal tracks of various International Planning Competitions (IPC). These instances span a wide range of problem complexities and structural characteristics, ensuring a comprehensive evaluation of our proposed methods.

For reproducibility and fair comparison, all experiments used deterministic settings in CPLEX. Each instance was solved independently with all model variants to ensure consistent experimental conditions.

The summary of models explored in this work is listed in Table 6.1. Our evaluation methodology focuses on two primary metrics: the average number of nodes explored by CPLEX (computed using shifted geometric mean with a shift of 1) and the average (again a shifted geometric mean) total time required to read, preprocess, and solve each instance. The experimental results are categorized by instance difficulty, where difficulty is determined by the shortest solution time achieved across all compared models for each instance.

6.1 Effectiveness of a warm start

		Time (s)			
Category	Count	greedy_choice$_c$	gc$_{c\times a}$	gc$_{h^{max}}$	gc$_{h^{add}}$
$[0,1)s$	2432	0.11	**0.10**	0.20	0.14
$[1,10)s$	266	3.10	**3.00**	6.80	4.70
$[10,100)s$	81	23.03	**22.94**	63.02	27.75
$[100,900)s$	4	**253.54**	256.67	320.08	358.96

Table 6.2: Time required to compute each heuristic.

As shown in Table 6.2, the heuristic functions provide a warm start for the MIP solver quite efficiently: for 87% of the instances, the greedy algorithm terminates in under 1s.

Figure 6.1 illustrates how close the proposed solutions are to h^+. The x axis shows the ratio between h^+ and heuristic solution's cost, while the y axys shows the fraction of instances achieving at most that ratio.

Although the two naive greedy choices are the fastest, they do not yield accurate approximations of h^+. The h^{max}-based greedy algorithm performs the worst among the explored options, as h^{max} tends to underestimates the true cost of achieve an atom; on the other hand, the h^{add} function is not admissible and often overestimates costs, penalizing

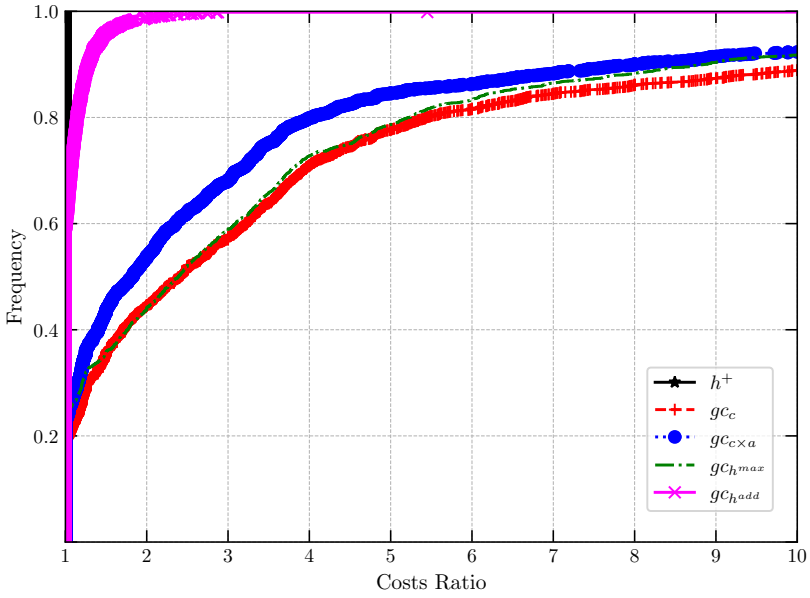

Figure 6.1: Ratio between h^+ and heuristic solutions cost.

more paths that could turn out expensive. Thanks to this aspect, the h^{add}-based greedy heuristic manages to achieve a 2-approximation ratio to h^+ in 95% of the instances.

Given that the h^{add}-based greedy algorithm is only slightly slower than the fastest heuristic, its cost ratio advantage is the deciding factor for determining the warm start to use in out experiments.

We will indicate the use of the h^{add}-based greedy algorithm as warm start with the symbol $_h$ ($\text{VE}^e(\Pi^+)$ with the h^{add}-based warm start becomes $\text{VE}^e_h(\Pi^+)$).

		Nodes			Time (s)		
Category	Count	$\text{TL}^e(\Pi^+)$	$\text{TL}^e_h(\Pi^+)$	Ratio	$\text{TL}^e(\Pi^+)$	$\text{TL}^e_h(\Pi^+)$	Ratio
$[0,1)s$	1746	1.00	**0.60**	0.60	0.21	**0.15**	0.73
$[1,10)s$	303	39.86	**20.98**	0.53	5.03	**3.20**	0.64
$[10,100)s$	163	197.41	**163.40**	0.83	41.40	**27.02**	0.65
$[100,900)s$	63	15566.20	**14195.58**	0.91	336.07	**307.58**	0.96

Table 6.3: Warm start effectiveness for algorithm $\text{TL}^e(\Pi^+)$.

		Nodes			Time (s)		
Category	Count	$\text{VE}^e(\Pi^+)$	$\text{VE}^e_h(\Pi^+)$	Ratio	$\text{VE}^e(\Pi^+)$	$\text{VE}^e_h(\Pi^+)$	Ratio
$[0,1)s$	1797	0.27	**0.19**	0.72	0.15	**0.15**	0.99
$[1,10)s$	365	4.96	**3.31**	0.67	3.98	**3.26**	0.82
$[10,100)s$	215	10.70	**8.41**	0.79	33.09	**28.99**	0.88
$[100,900)s$	67	191.57	**155.02**	0.81	357.59	**350.27**	0.98

Table 6.4: warm start effectiveness for algorithm $\text{VE}^e(\Pi^+)$.

With the warm start, we reduce the number of nodes expanded by CPLEX by 30-40%

with both models, and we reduced the total time required to compute h^+ by 30-35% for the $\text{TL}^e(\Pi^+)$ model and by 10-15% for some of the instances of the $\text{VE}^e(\Pi^+)$ model (Table 6.3, Table 6.4). The total number of instances solved and the time required are also displayed in Figure 6.2 and Figure 6.3.

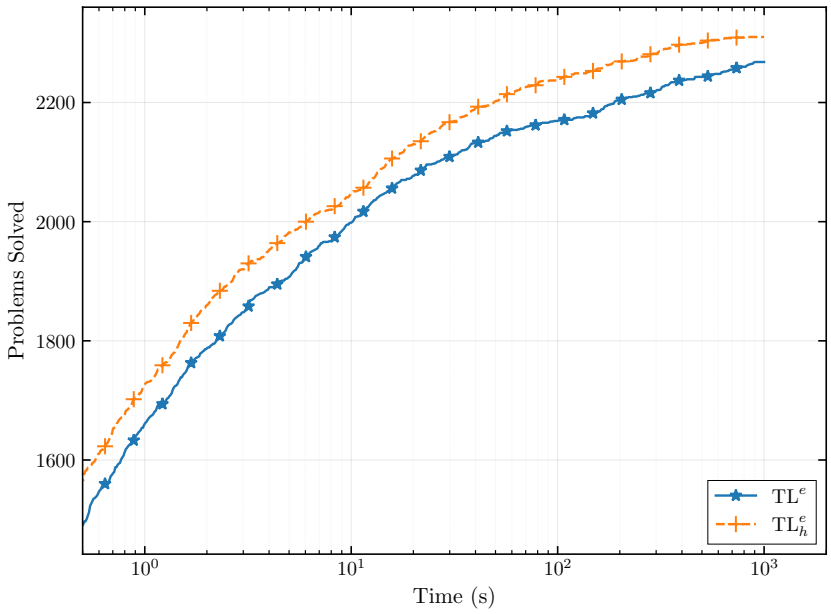

Figure 6.2: Instances solved by $\text{TL}^e(\Pi^+)$, $\text{TL}^e_h(\Pi^+)$.

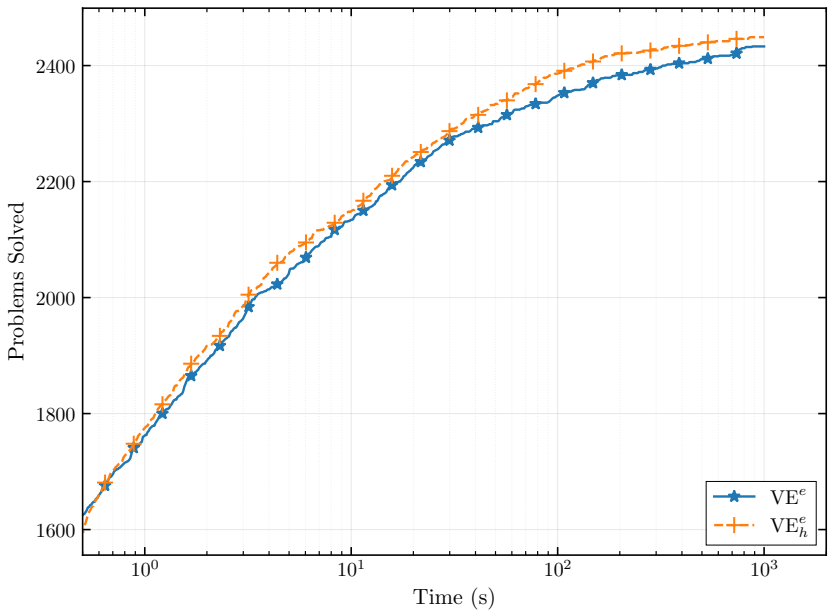

Figure 6.3: Instances solved by $\text{VE}^e(\Pi^+)$, $\text{VE}^e_h(\Pi^+)$.

6.2 Iteratively separating cuts

		Nodes			Time (s)		
Category	Count	$\text{VE}^e_h(\Pi^+)$	$\text{SEC}^e_h(\Pi^+)$	Ratio	$\text{VE}^e_h(\Pi^+)$	$\text{SEC}^e_h(\Pi^+)$	Ratio
$[0,1)s$	1864	**0.19**	2.27	12.25	**0.20**	0.36	1.82
$[1,10)s$	365	**4.51**	220.57	48.92	**5.20**	18.81	3.62
$[10,100)s$	196	**9.08**	328.36	36.18	**48.69**	83.35	1.71
$[100,900)s$	58	**93.45**	3500.87	37.46	**377.25**	518.62	1.38

Table 6.5: Nodes and time comparison: $\text{VE}^e_h(\Pi^+)$ - $\text{SEC}^e_h(\Pi^+)$.

		Nodes			Time (s)		
Category	Count	$\text{VE}^e_h(\Pi^+)$	$\text{LM}^e_h(\Pi^+)$	Ratio	$\text{VE}^e_h(\Pi^+)$	$\text{LM}^e_h(\Pi^+)$	Ratio
$[0,1)s$	1913	**0.33**	0.87	2.61	0.24	**0.23**	0.95
$[1,10)s$	369	**3.92**	24.89	6.35	8.19	**7.00**	0.86
$[10,100)s$	181	**8.37**	42.78	5.11	75.24	**44.99**	0.60
$[100,900)s$	61	**75.38**	767.53	10.18	492.70	**406.79**	0.83

Table 6.6: Nodes and time comparison: $\text{VE}^e_h(\Pi^+)$ - $\text{LM}^e_h(\Pi^+)$.

		Nodes			Time (s)		
Category	Count	$\text{VE}^e_h(\Pi^+)$	$\text{LMS}^e_h(\Pi^+)$	Ratio	$\text{VE}^e_h(\Pi^+)$	$\text{LMS}^e_h(\Pi^+)$	Ratio
$[0,1)s$	1905	**0.32**	0.84	2.60	0.24	**0.20**	0.85
$[1,10)s$	361	**3.91**	28.45	7.28	**7.35**	7.63	1.04
$[10,100)s$	199	**8.82**	61.37	6.96	78.71	**42.91**	0.55
$[100,900)s$	59	**62.96**	832.39	13.22	513.77	**363.81**	0.71

Table 6.7: Nodes and time comparison: $\text{VE}^e_h(\Pi^+)$ - $\text{LMS}^e_h(\Pi^+)$.

Using S.E.C. to separate cuts is not enough to justify the absence of the acyclicity constraints (Table 6.5), however, iteratively generating landmarks helps us improve the time needed to compute h^+ (Table 6.6, Table 6.7). Moreover, the higher number of nodes is expected since we have an incomplete model that is integrated with additional constraints as the nodes are expanded.

Looking at Figure 6.4 we can see that both $\text{LM}^e_h(\Pi^+)$ and $\text{LMS}^e_h(\Pi^+)$ are outperforming the state-of-the-art on the easy instances: this is a direct consequence of the fact that "easier" instances might not need all the acyclicity constraints to compute h^+, but just a smaller set of constraints.

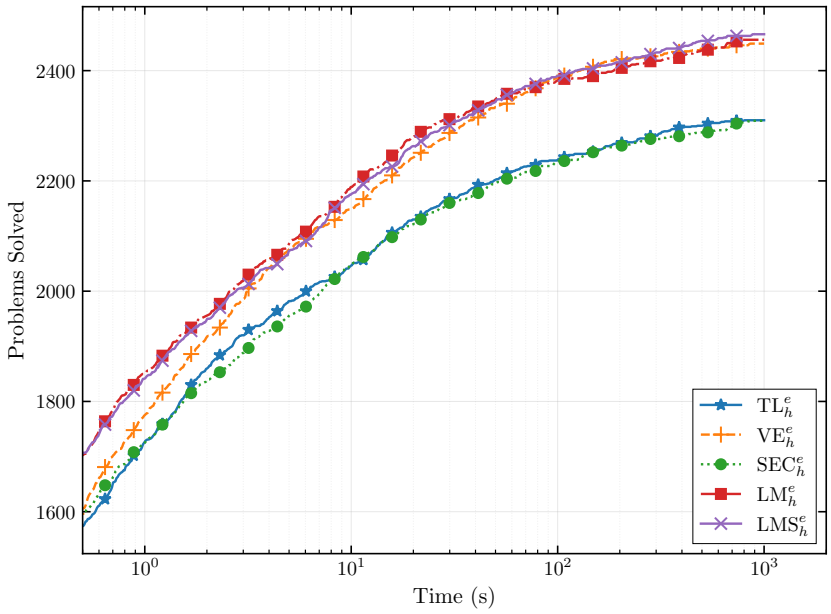

Figure 6.4: Comparison of all the models.

It is also important to note the existence of inherently acyclic instances, as shown in [IF15], in which the base model is enough to compute h^+ without any additional constraints; this speeds up the computation significantly since there is no need for the callbacks or the construction of the "complete" model with the acyclicity constraints.

The proposed models are also more suitable for bigger instances, in which the size of the problem might become an issue: with an iterative generation of the missing constraints, we start with a small model and add only the necessary constraints, which have been proven to be usually small in number, especially with respect to the number of constraints used to model acyclicity of the complete models $\mathrm{TL}_h^e(\Pi^+)$ and $\mathrm{VE}_h^e(\Pi^+)$, as shown in Tables 6.8, 6.9 and 6.10.

Category	Count	Acyclicity constraints for $\mathrm{VE}_h^e(\Pi^+)$	Cuts added in $\mathrm{SEC}_h^e(\Pi^+)$
$[0, 1)s$	1864	431.42	1.46
$[1, 10)s$	365	32006.50	20.53
$[10, 100)s$	196	259238.82	59.42
$[100, 900)s$	58	264010.68	615.06

Table 6.8: Acyclicity constraints comparison: $\mathrm{VE}_h^e(\Pi^+)$ - $\mathrm{SEC}_h^e(\Pi^+)$.

Category	Count	Acyclicity constraints for $\mathrm{VE}_h^e(\Pi^+)$	Cuts added in $\mathrm{LM}_h^e(\Pi^+)$
$[0, 1)s$	1913	431.42	1.68
$[1, 10)s$	369	32006.50	16.90
$[10, 100)s$	181	259238.82	34.56
$[100, 900)s$	61	264010.68	368.56

Table 6.9: Acyclicity constraints comparison: $\mathrm{VE}_h^e(\Pi^+)$ - $\mathrm{LM}_h^e(\Pi^+)$.

Category	Count	Acyclicity constraints for $\text{VE}_h^e(\Pi^+)$	Cuts added in $\text{LMS}_h^e(\Pi^+)$
$[0,1)s$	1905	431.42	1.90
$[1,10)s$	361	32006.50	18.52
$[10,100)s$	199	259238.82	54.74
$[100,900)s$	59	264010.68	456.24

Table 6.10: Acyclicity constraints comparison: $\text{VE}_h^e(\Pi^+)$ - $\text{LMS}_h^e(\Pi^+)$.

6.3 Integration with LMCUT

We first measure the impact of the integration of the basic LMCUT: we will denote the run that uses a single execution of LMCUT, with the ARB tie-breaking approach, with LMCUT_1.

		Time (s)		
Category	Count	$\text{LMS}_h^e(\Pi^+)$	$\text{LMS}_h^e(\Pi^+) + \text{LMCUT}_1$	Ratio
$[0,1)s$	1935	0.27	**0.13**	0.47
$[1,10)s$	324	5.93	**3.72**	0.63
$[10,100)s$	198	40.82	**30.15**	0.74
$[100,900)s$	67	400.59	**365.89**	0.91

Table 6.11: Time comparison: $\text{LMS}_h^e(\Pi^+)$ - $\text{LMS}_h^e(\Pi^+) + \text{LMCUT}_1$.

		Nodes		
Category	Count	$\text{LMS}_h^e(\Pi^+)$	$\text{LMS}_h^e(\Pi^+) + \text{LMCUT}_1$	Ratio
$[0,1)s$	1935	0.95	**0.31**	0.33
$[1,10)s$	324	15.65	**5.63**	0.36
$[10,100)s$	198	67.32	**31.26**	0.46
$[100,900)s$	67	1925.40	**1799.26**	0.93

Table 6.12: Nodes comparison: $\text{LMS}_h^e(\Pi^+)$ - $\text{LMS}_h^e(\Pi^+) + \text{LMCUT}_1$.

As expected, the addition of the landmarks generated by the LMCUT algorithm, drastically reduce the number of nodes explored by CPLEX (around 60% less nodes); this also leads to a decrease in the computation time, between 30% and 50% (Table 6.11, Table 6.12).

We then measured the impact of various combined tie-breaking approaches; after some tuning we found the ideal execution: repeat LMCUT 3 times, with respectively ARB, INV and VDM as tie-breaking approaches in the choice of the pcf; this approach will be denoted with LMCUT_2.

		Time (s)		
Category	Count	$\text{LMS}^e_h(\Pi^+) + \text{LMCUT}_1$	$\text{LMS}^e_h(\Pi^+) + \text{LMCUT}_2$	Ratio
$[0, 1)s$	1957	0.14	**0.12**	0.83
$[1, 10)s$	330	4.52	**3.80**	0.84
$[10, 100)s$	190	33.78	**31.34**	0.93
$[100, 900)s$	72	476.62	**335.86**	0.71

Table 6.13: Time comparison: $\text{LMS}^e_h(\Pi^+) + \text{LMCUT}_1$ - $\text{LMS}^e_h(\Pi^+) + \text{LMCUT}_2$.

		Nodes		
Category	Count	$\text{LMS}^e_h(\Pi^+) + \text{LMCUT}_1$	$\text{LMS}^e_h(\Pi^+) + \text{LMCUT}_2$	Ratio
$[0, 1)s$	1957	0.38	**0.24**	0.63
$[1, 10)s$	330	6.88	**4.91**	0.71
$[10, 100)s$	190	32.66	**23.37**	0.72
$[100, 900)s$	72	1930.29	**1395.37**	0.72

Table 6.14: Nodes comparison: $\text{LMS}^e_h(\Pi^+) + \text{LMCUT}_1$ - $\text{LMS}^e_h(\Pi^+) + \text{LMCUT}_2$.

With the multiple LMCUT execution gathering more landmarks to add as initial cuts, we can further reduce both the computation time and the number of nodes explored by CPLEX (Table 6.13, Table 6.14).

6.4 Overall improvements

Measuring the cumulative impact of our proposed techniques, compared to the previous state-of-the-art approach of modeling acyclicity with the Vertex Elimination Constraints, we achieve a speedup of 2× (Table 6.15) on the computation of h^+ as well as a reduction on the size of the models needed to comupute it as the base model is way smaller (Table 6.10); however, our technique explores more nodes in more complex instances (Table 6.7), even though using LMCUT to generate initial cuts reduces this gap (Table 6.16).

		Time (s)		
Category	Count	$\text{VE}^e(\Pi^+)$	$\text{LMS}^e_h(\Pi^+) + \text{LMCUT}_2$	Ratio
$[0, 1)s$	1972	0.32	**0.15**	0.47
$[1, 10)s$	343	14.63	**4.67**	0.32
$[10, 100)s$	157	89.56	**43.35**	0.48
$[100, 900)s$	45	635.07	**349.81**	0.55

Table 6.15: Time comparison: $\text{VE}^e(\Pi^+)$ - $\text{LMS}^e_h(\Pi^+) + \text{LMCUT}_2$.

		Nodes		
Category	Count	$\text{VE}^e(\Pi^+)$	$\text{LMS}^e_h(\Pi^+) + \text{LMCUT}_2$	Ratio
$[0, 1)s$	1972	0.66	**0.40**	0.61
$[1, 10)s$	343	**4.91**	5.19	1.06
$[10, 100)s$	157	**19.14**	60.62	3.17
$[100, 900)s$	45	**183.78**	1064.97	5.80

Table 6.16: Nodes comparison: $\text{VE}^e(\Pi^+)$ - $\text{LMS}^e_h(\Pi^+) + \text{LMCUT}_2$.

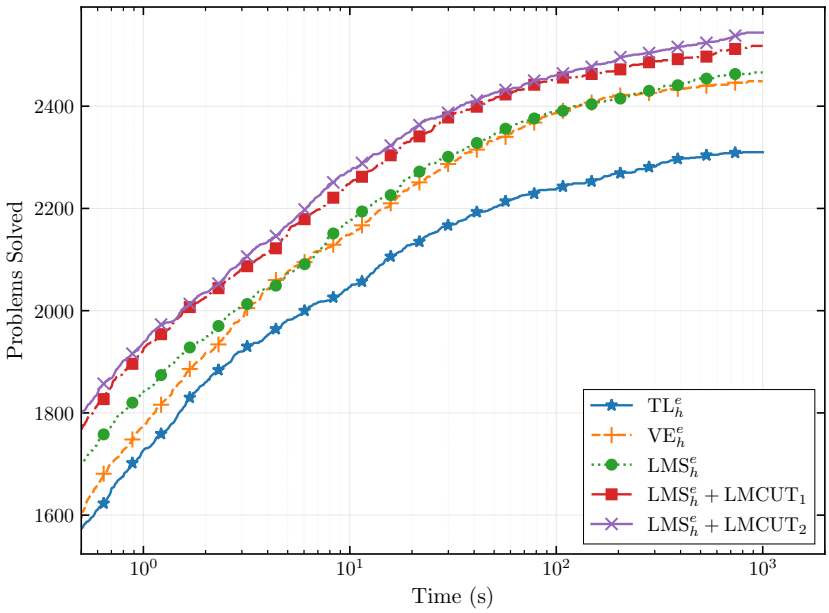

Figure 6.5: Comparison of the models + integration of LMCUT.

Chapter 7

Future works

This work is far from being complete since there are a few aspects that have room for improvement:

1. Separating cuts on fractional solutions: we noticed that in some instance classes, computing h^+ with $\text{VE}^e_h(\Pi^+)$ is extremely easy, while $\text{LMS}^e_h(\Pi^+)$ requires a relatively large amount of cuts before being able to compute h^+. By finding a way to separate cuts out of a fractional solution, we might be able to close the optimality gap earlier in the search, drastically reducing the number of nodes that CPLEX has to explore

2. LP-relaxation for quick estimates of h^+: in other publications, such as [IF15; FR22], in addition to computing the exact value of h^+, the authors solve the LP relaxation of the $\text{TL}^e(\Pi^+)$ and $\text{VE}^e(\Pi^+)$ models, which is faster to obtain an estimate of h^+: this blends with the previous open point, with which we would be able to separate landmarks from the solutions of the LP relaxation, obtaining a tighter approximation of h^+.

3. Expanding this work to classical planning: MIP solvers have already been proposed as black-box tools to solve classical planning tasks, with little success. However, this new approach might be an inspiration for new ways of exploring the search space: this might also be very helpful for satisficing planners, where we are looking for a good solution instead of the optimal one.

Chapter 8

Conclusions

This thesis addressed the challenge of efficiently computing h^+, the optimal cost of the delete-relaxed planning problem, a crucial lower bound for optimal planning heuristics. We explored and integrated various preprocessing techniques from existing literature to simplify the problem instances and reduce the complexity of the underlying MIP models. Furthermore, we developed and evaluated primal heuristics to provide effective warm-start solutions to the MIP solver, demonstrating their significant impact on reducing computation time and nodes explored.

A core contribution of this work is the proposal of a novel iterative approach to model acyclicity within the h^+ computation. Instead of relying on a priori complete acyclicity constraints, which can lead to very large and intractable models, our method dynamically generates and adds violated disjunctive action landmarks and S.E.C. as cuts. This constraint generation approach ensures that only the necessary constraints are added, leading to more compact and manageable models.

Our empirical evaluation, conducted using the IBM ILOG CPLEX Optimization Studio on a diverse set of IPC domains, confirmed the effectiveness of the proposed methods. The warm-start heuristics consistently reduced the number of nodes explored and the overall solution time for both Time Labeling and Vertex Elimination based models. More importantly, the iterative landmark and S.E.C. generation approach demonstrated competitive performance, particularly for "easy" instances where they often outperformed state-of-the-art complete models. For larger instances, this iterative approach proved to be more suitable due to the significantly smaller number of constraints required compared to the complete acyclicity models. This highlights the practical advantage of our approach in tackling complex planning problems.

Integrating LMCUT with our work, proved to be a key factor in reaching a $2\times$ speedup on the state-of-the-art and closing the gap on the number of nodes expanded by our proposed models.

While this work provides a strong foundation, several promising avenues for future research exist. These include further leveraging LM-CUT heuristics for preprocessing, exploring methods for separating cuts from fractional solutions to close optimality gaps

earlier, investigating LP-relaxations for quick h^+ estimates, and potentially expanding this iterative constraint generation approach to classical planning tasks.

8.1 Acknowledgments

I would like to thank those who contributed to the completion of this thesis, starting with my supervisor, Prof. Salvagnin Domenico, for his guidance and insightful feedback throughout this research, and Modolo Riccardo for his valuable assistance with the implementation of the LMCUT integration.

I am also grateful to my fellow colleagues for their stimulating discussions and for creating a collaborative research environment that enriched this work.

Finally, I extend my deepest gratitude to my family and friends for their constant support and encouragement throughout my academic journey.

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.

[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.

[HD09] Malte Helmert and Carmel Domshlak. "Landmarks, critical paths and abstractions: what's the difference anyway?" In: ICAPS'09. 2009. DOI: 10.1609/icaps.v19i1.13370.

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

[GB11] Avitan Gefen and Ronen Brafman. "The Minimal Seed Set Problem". In: *Proceedings of the International Conference on Automated Planning and Scheduling* 21 (2011). DOI: 10.1609/icaps.v21i1.13485.

[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.

[LF20] Pascal Lauer and Maximiliam Fickert. “Beating LM-cut with LM-cut: Quick Cutting and Practical Tie Breaking for the Precondition Choice Function”. In: *ICAPS 2020 Workshop on Heuristics and Search for Domain-independent Planning*. 2020. URL: https://openreview.net/forum?id=anKt6unvU-.

[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.