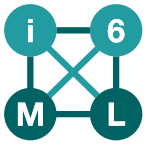

Chair of
Computer Science 6
(Machine Learning
and Reasoning)

RWTH AACHEN UNIVERSITY

The present work was submitted to the Chair of Machine Learning and Reasoning.
Diese Arbeit wurde vorgelegt am Lehrstuhl für Maschinelles Lernen und Inferenz.

Tail Reasoning in Classical Planning

Tail Reasoning in Classical Planning

Bachelor Thesis
Bachelorarbeit

Presented by / Vorgelegt von

Cedric Becks
378329

Supervised by / Betreut von Jonas Goesgens, M.Sc.

1st Examiner / 1. Prüfer Prof. Hector Geffner, Ph.D.

2nd Examiner / 2. Prüfer Prof. Dr. rer. nat. Christopher Morris

Aachen, February 20, 2025

Abstract

This thesis investigates forward search in classical planning and aims to improve it by incorporating information about the end of a solution. Traditional forward search struggles with implicit goal orderings. This work explores two approaches using the Fast Forward (HFF) heuristic: modifying HFFs delete and precondition lists to include goal-ordering information and a hybrid approach combining satisfiability (SAT) solving for a partial regressive solution with a subsequent HFF forward pass. The first approach, modifying HFF, proved largely ineffective. The hybrid approach leverages SAT solvers for complex subproblems and HFF for more straightforward sections, specifically targeting tail-heavy problems where complexity resides in the final solution steps. A new domain, a variation of Sokoban, is introduced to illustrate these challenges. While the hybrid approach showed some promise in specific scenarios, particularly those with looser goal orderings towards the end of the plan, it ultimately did not outperform HFF. Despite the overall mixed results, the thesis contributes a new benchmark domain, analysis of several planning domains, and insights into the challenges and potential of hybrid planning strategies.

Contents

1 Introduction

Classical planning, specifically STRIPS-planning (Fikes and Nilsson, 1971), involves finding a finite sequence of grounded actions (called operators) that, when applied to a given initial state, transitions to a state that satisfies a set of given goals. It usually includes more restrictions, such as the state space being fully observable and the agents' actions as the only way to change the state space. The key challenge for finding that sequence is determining efficient, scalable algorithms to guide the search process.

Classical planning can model various problems, ranging from robotics and logistics - like transportation and supply chain management - over (non-adversarial) games to biosynthesis.

One of the significant approaches to classical planning is state-based planning (as opposed to plan state-based planning) (Kambhampati and Srivastava, 1996), where the planner searches the (implicit) graph of all states for any path between the initial state and any goal state. Planners may initiate a forward search starting from the initial state or a regression (backward) search from the set of goal states. Both approaches have their advantages and problems. In this thesis, we explore how to improve a forward search guided by a heuristic by incorporating methods of the regression search, which we termed *tail reasoning*. Forward search struggles with things such as implicit goal ordering and partial solutions. We explore how much using a goal topology discovered through regression search can help forward search and look at various ways to use structural differences between forward and regression search. All approaches presented here are domain-independent but work better on some domains - especially those with a strict goal ordering, like particular blocksworld examples - than others - like general sokoban problems.

1.1 Problem Statement and Research Questions

Planning problems often require solving subgoals in a specific order, which, at least in STRIPS planning, must be inferred by the planner. Heuristic-based planners like Fast Forward Heuristic Search (HFF) (Hoffmann and Nebel, 2011) use greedy strategies that prioritize immediate goal satisfaction but struggle with inferring goal topologies. Such strategies often lead to inefficient search patterns, excessive backtracking, and, in the worst case, infeasible solving times. Addressing this limitation requires incorporating methods to handle implicit goal topology more effectively.

How can information about the end of a solution be gathered and utilized to improve forward search in classical planning?

To explore this question, we investigate two distinct approaches around HFF:

1. Modifying HFF by replacing its delete and precondition lists to incorporate goal-ordering information.
2. Using satisfiability to partially solve the problem regressively, followed by applying HFF for a forward pass.

Relaxed Heuristic Search. The first approach considered was modifying the well-known HFF to incorporate knowledge about the problem's end state. The idea was to analyze the reversed problem and integrate its properties into forward-solving strategies. It was motivated by studying the properties of the reversed (negative) planning problem and, specifically, the difference of operators compared to the original problem. We noticed that while delete and precondition effects essentially swapped places, the add effects were the same for both the problem and its negation. Because precondition and delete effects tend to be very similar, we explored using the negated operators as input for the relaxed forward search.

However, this approach mainly proved ineffective, as the modified heuristic performed worse or, at best, equally to the original across nearly all test domains. We only observed improvements in one case out of our ten benchmark domains, and even that was dubious.

Bidirectional Search. Following this, we turned to SAT solvers and bidirectional search techniques. The choice was motivated by the observation that different solvers deal more efficiently with either short solutions of high complexity or long solutions of low complexity. Complexity here refers to strictness regarding the order in which a plan must achieve its goals. To address this, we explored a hybrid approach by splitting the solution between different solvers, where one solver attempts to reach a valid intermediate state from which a different solver can take over to complete the solution.

SAT solvers excel at solving complex logical problems but suffer from scalability issues. As problem sizes increase, SAT-based approaches become computationally infeasible. As stated before, relaxed forward search has the opposite problem, where it can handle almost arbitrarily large problems but quickly falls into states from which it struggles to continue. Our solution was to use SAT solvers to handle complex subproblems while leveraging HFF to navigate more straightforward sections efficiently.

To use two different search paradigms in such a way requires an algorithm capable of distinguishing between simple and complex subproblems. The development of such an algorithm is out of scope for this thesis, so we focus specifically on tail-heavy problems, where we find most of the complexity in the final steps of the solution. The idea is to use an SAT solver to identify

a set of intermediate goals regressively, which are no longer strictly ordered, so that they can be reached efficiently with a forward heuristic search.

Tunnel Domain. To illustrate the challenges of tail-heavy problems, we introduce the *tunnel domain*, a variation of the Sokoban domain. In this domain, an agent must move blocks into a narrow tunnel, filling it up. The winning strategy places blocks at the final space of the tunnel first, then the second-to-last space, until the tunnel is complete. In this regard, it is somewhat similar to blocksworld (Gupta and Nau, 1992).

1.2 Research Direction and Contributions

This research explores several key directions:

- Evaluating the performance of the hybrid SAT-heuristic approach on the Tunnel domain and traditional benchmarks.
- Analyzing whether solutions generated by this approach are closer to optimal than pure heuristic search.
- Investigating potential vulnerabilities, such as adversarial modifications to problem domains that counteract the benefits of tail reasoning.
- Studying the impact of state invariants, crafted by hand, for any of the benchmark domains.
- Identifying bottlenecks in the approach, determining whether heuristic or SAT components contribute more to computational overhead.

By addressing the challenges of implicit goal ordering, this research contributes to developing more efficient and intelligent classical planning methods. The insights gained during this thesis may serve as a foundation for future work in hybrid planning strategies and heuristic optimization.

1.3 Thesis Structure

We organized this thesis as follows:

- Chapter 2 provides technical background information on STRIPS planning, heuristics, and prior work related to tail reasoning.
- Chapter 3 discusses previous work related to forward and regression search
- Chapter 4 details our methodology, including the SAT-based backward reasoning approach and heuristic modifications.
- Chapter 5 introduces and analyzes the Tunnel domain, as well as the other benchmark domains we used. All domains have been provided with additional state constraints.

- Chapter 6 presents experimental results, comparing our approaches across the benchmark domains.
- Chapter 7 discusses insights gained, limitations of our approach, potential areas for future research, and a summary of findings and their implications for classical planning.

1.4 Tool Assistance

This thesis used Grammarly and Gemini for proofreading, translation, and summarization purposes. No code, citations, logic, factual information, or other substantive content was obtained through any use of generative AI. All text fragments generated were additionally scrutinized, checked for consistency, and edited.

2 Background

This section will present a short introduction to Stanford Research Institute Problem Solver (STRIPS) planning and search systems.

2.1 STRIPS and Classical Planning

STRIPS has been the first successful attempt to formalize the construction and solving of deterministic planning. It defines the basic building blocks classical planning would build upon - a model of well-formed formulas (wffs) and operators. The wffs in the model describes either facts or rules while operators update said model. The task is to select a sequence of such operators that the final model satisfies an arbitrary second model, the goals, or prove that such a sequence does not exist.

We use $Part_k(S)$ to describe the set of all possible k-partitions of an arbitrary set S, $\overline{x}$ for either a negated fact or set of negated facts, and $\langle \cdot, \cdot, \cdot \rangle$ for a tuple of multiple variables.

Definition 1 (STRIPS Problem) *A STRIPS problem is a tuple $\langle \mathcal{F}, \mathcal{O}, I, G \rangle$, with $\mathcal{F}$ being a finite set of facts and $\mathcal{O}$ being a set of operators. The initial state $I \subseteq \mathcal{F}$ is a set of facts, and so are the goals $G \subseteq \mathcal{F}$. We write operators as $o = \langle pre(o), add(o), del(o) \rangle \subseteq \mathcal{F}^3$, For all operators o, it holds that*

$$add(o) \cap (del(o) \cup pre(o)) = \varnothing$$

Not all implementations of STRIPS planning support negative preconditions (i.e., a negated fact as part of their preconditions). This creates no reduction in expressiveness (Massey, 1998), because we can create an extended problem $\langle \mathcal{F}', \mathcal{O}', I', G \rangle$ where

$$\begin{aligned} \mathcal{F}' &:= \mathcal{F} \cup \overline{\mathcal{F}} \\ \mathcal{O}' &:= \left\{ \left\langle pre(o), add(o) \cup \overline{del(o)}, del(o) \cup \overline{add(o)} \mid o \in \mathcal{O} \right\rangle \right\} \\ I' &:= I \cup \overline{\mathcal{F}/I} \end{aligned}$$

Essentially, we keep track of which facts are not part of any state by creating a fact that represents them in their absence.

Without loss of generality, we assume that negative preconditions, states, and goals are supported unless otherwise stated.

An operator op can be applied to a state if it contains its preconditions. The state is then updated to contain all add-facts and removes all del-facts. Formally, we have:

Definition 2 (STRIPS Solution) *The solution to a STRIPS Problem $P = \langle \mathcal{F}, \mathcal{O}, I, G \rangle$, with $\mathcal{F}$ is any tuple $\pi = (o_1, ..., o_n) \in \mathcal{O}^n, n \in \mathbb{N}$ that with*

$$o_i(s) := (s/del(o_i)) \cup add(o_i)$$
$$s_0 = I$$
$$s_i = o_i(s_{i-1})$$

both of the following propositions are satisfied.

$$pre(o_i) \subseteq S_{i-1}, 1 \leq i \leq n$$
$$G \subseteq s_n$$

To denote that π is a solution to P, we write $\pi \vDash P$.

A STRIPS-Problem may have any number of solutions and final states, including none. We call it solvable and π a plan if it has at least one solution. A plan is optimal if no shorter plan exists to reach a solution.

Schemata for groups of STRIPS-Problems are called domains.

Definition 3 (STRIPS Domain) *A STRIPS Domain is a tuple $D = (\mathcal{P}, \mathcal{A})$, where $\mathcal{P}$ is a set of fact schemata (called predicates) and $\mathcal{A}$ is a set of operator schemata (called actions). Each predicate has the form $name(v_1, ..., v_n), n \in \mathbb{N}_0$, where each v_i is a (typed) variable. Each action $a \in \mathcal{A}$ has the form $\langle pre(a), add(a), del(a), v \rangle$, where v is a set of (typed) variables and $pre(a), add(a), del(a) \subseteq \mathcal{P}$.*

By replacing the variables of a predicate with some constants, we create a fact. This process is called grounding. For a predicate $p(v)$, some constants c, and a mapping $f : v \mapsto c$, we write $p(f)$ to ground the predicate.

By grounding the preconditions and effects of action $a = \langle pre(a), add(a), del(a), v \rangle$ with a mapping $f : v \mapsto c$ for some constants c, we obtain an operator. We similarly write $a(f)$ to ground an action.

To obtain a problem from a domain, we define some (typed) constants and ground every action and every predicate with every possible mapping. Typing is not a syntactic construct not originally part of STRIPS, but ubiquitous for contemporary domains and planners. We write a type t as ?t and can only replace types with matching constants during grounding. Typing replaced the need to design a unary predicate for each (implicit) type and add it to the preconditions of all relevant actions.

Formally, if we have a domain $D = (\mathcal{P}, \mathcal{A})$ and some constants c, we get all possible problems $\mathbf{P} = \langle \mathcal{F}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ with

$$\begin{aligned}
\mathcal{F} &= \{p(f) \mid p(v) \in \mathcal{P}, f \in map(v, c)\} \\
\mathcal{O} &= \bigcup_{a=\langle pre,add,del,v\rangle \in \mathcal{A}} \{a(f) \mid f \in map(v, c)\} \\
\mathcal{I} &= \mathcal{F}^{\mathbb{N}} \\
\mathcal{G} &= \mathcal{F}^{\mathbb{N}}
\end{aligned}$$

From there any single problem $P = \langle \mathcal{F}, \mathcal{O}, I \in \mathcal{I}, G \in \mathcal{G} \rangle$ can be selected.

Definition 4 *Predicate Types We call Predicates that do not appear in any action effects static predicates and all others fluent predicates or fluents. When grounding a problem, static predicates impose rules, while fluent predicates constitute the (grounded) facts. After grounding, we remove all static predicates from the problem.*

2.1.1 Normalization

Definition 5 (Well-Formed STRIPS Problem) *If a STRIPS Problem $P = \langle \mathcal{F}, \mathcal{O}, I, G \rangle$ contains the following properties, we call it well-formed:*

$$del(o) \subseteq pre(o) \qquad \forall o \in \mathcal{O} \qquad \text{(I)}$$

$$add(o_i \in \pi) \cap s_{i-1} = \emptyset \qquad \forall \pi \vDash P \qquad \text{(II)}$$

That means no operator may delete a fact that does not exist or add a fact that already does.

Corollary 1 *Every STRIPS Problem $P = \langle \mathcal{F}, \mathcal{O}, I, G \rangle$ is transformable into a well-formed problem. The well-formed problem has at most $\max_{o \in \mathcal{O}} |\mathcal{O}|^{|pre(o)/del(o)| \cdot |add(o)|}$ operators, though in practice that number is far smaller.*

Construction. Let $P = \langle \mathcal{F}, \mathcal{O}, I, G \rangle$ be a not necessarily well-formed problem. We transform a STRIPS problem into a well-formed STRIPS problem via normalization. Furthermore, let

$$\begin{aligned}
norm_{del}(o) &:= \left\{ \left\langle pre(o) \dot{\cup} x \dot{\cup} \overline{y}, add(o), del(o)/y \right\rangle \mid \langle x, y \rangle \in Part_2(del(o)/pre(o)) \right\} \\
norm_{add}(o) &:= \left\{ \left\langle pre(o) \dot{\cup} \overline{x} \dot{\cup} y, add(o)/y, del(o) \right\rangle \mid \langle x, y \rangle \in Part_2(add(o)) \right\} \\
norm(o) &:= \bigcup_{o' \in norm_{del}(o)} norm_{add}(o')
\end{aligned}$$

The problem $P' = \langle \mathcal{F}, \mathcal{O}', I, G \rangle$ with

$$\mathcal{O}' = \bigcup_{o \in \mathcal{O}} norm(o)$$

is well-formed. For every solution $\pi' = \langle o'_1, ..., o'_n \rangle$ of P', $\pi = \langle o_1, ..., o_n \rangle$ is a solution of P iff $o'_i \in norm(o_i), \forall 1 \leq i \leq n$

The idea behind the construction is to normalize the operators in two steps. First, for every operator, if its delete effects contain a fact that does not appear in its precondition, split that operator into two: once with the fact as a precondition and in the delete effects, and once with the negation of the fact as a precondition and without the fact in the delete effects. If there is more than one such fact, repeat the process for both newly created operators. Analogously for the add effects.

That P' is well-formed should be obvious. Since every operator in $\mathcal{O}$' has its add effects as negated preconditions and its delete effects as non-negated preconditions, it is ensured that all deleted facts existed before applying the operator and all added facts did not. We must show that the relationship between π and π' holds.

Proof. Let $s, s' \subseteq \mathcal{F}$ be states with $s' = o'(s), o' \in \mathcal{O}'$, $pre(o) \subseteq s$. We will show that for every operator $o \in \mathcal{O}$ with $pre(o) \subseteq s$,

$$o' \in norm(o) \implies s' = o(s)$$

We define

$$del(o/o') := del(o)/del(o')$$
$$add(o/o') := add(o)/add(o')$$

With both

$$\begin{aligned}
& o' \in norm(o) && \\
\implies & x \notin s, \forall x \in del(o/o') && \text{(o' is well-formed)} \\
\implies & x \notin s', \forall x \in del(o/o') && \text{(o is well-formed)} \\
\implies & x \notin s', \forall x \in del(o) && (del(o') \subseteq del(o)) \\
\implies & s' \subseteq o(s) && \text{(definition)}
\end{aligned}$$

and

$$
\begin{aligned}
& o' \in norm(o) && \\
\Longrightarrow\ & x \in s, \forall x \in add(o/o') && \text{(o' is well-formed)} \\
\Longrightarrow\ & x \in s', \forall x \in add(o/o') && \text{(o is well-formed)} \\
\Longrightarrow\ & x \in s', \forall x \in add(o) && (add(o') \subseteq add(o)) \\
\Longrightarrow\ & o(s) \subseteq s' && \text{(definition)}
\end{aligned}
$$

we see that $o(s) = s'$.

2.1.2 Reversed Problems

Reversing a problem allows us to do a forward search equivalent to a regression search of the original problem. Solving a reversed problem is usually slower because the states contain many more facts in practice.

Definition 6 (Reverse Problem) *Any STRIPS probelm $P = \langle \mathcal{F}, \mathcal{O}, I, G \rangle$ can be reversed into a problem $P_r = \langle \mathcal{F}, \mathcal{O}_r, \mathcal{F}/G, \mathcal{F}/I \rangle$ such that $\pi = \langle o_1, ..., o_n \rangle \vDash P$ iff $\pi_r = \langle h(o_n), ..., h(o_1) \rangle \vDash P_r$ with the mapping $h : \mathcal{O} \mapsto \mathcal{O}_r, \langle pre, add, del \rangle \rightarrow \langle del, add, pre \rangle$.*

The reversed problem is the complement of the original problem. The idea is that for each operator o in the solution π that transforms a state s into s', the operator h(o) transforms the state $\mathcal{F}$/s' into $\mathcal{F}$/s. The mapping h works because the operator o removes all facts in $del(o)$ from s, so none are in s'. Accordingly, all $del(o)$ facts are in $\mathcal{F}$/s'. Similarly, none of the $pre(o)$ facts are in $\mathcal{F}$/s. However, that means that $del(o)$ is always a precondition of any operator that transforms $\mathcal{F}$/s' into $\mathcal{F}$/s, while $pre(o)$ is always part of the delete effects. Furthermore, because no facts in $add(o)$ can be in $\mathcal{F}$/s', while they could be part of $\mathcal{F}$/o (if P is well-formed, they always are), the add effect of any such operator must include them (Massey, 1998).

2.1.3 Relaxed Problems

Relaxed problems are derived problems that do not preserve correctness towards the original problem. However, an optimal solution to a derived problem is never longer than any solution to its original problem and is computable in polynomial time. These attributes make them a viable (Bonet and Geffner, 1999) option to create an admissible heuristic.

Definition 7 (Relaxed Problem) *For any problem $P = \langle \mathcal{F}, \mathcal{O}, I, G \rangle$ we call $P' = \langle \mathcal{F}, \mathcal{O}', I, G \rangle$ it's relaxed problem, with*

$$\mathcal{O}' := \{\langle pre, add, \varnothing \rangle \mid \langle pre, add, del \rangle \in \mathcal{O}\}$$

Similarly, we call all operators of P' relaxed operators.

Since for any state s and all available relaxed operators o' the relationship $s \subseteq o(s)$ holds, all operators in $\mathcal{O}$' must be applied at most once to reach a goal state. That means at most $|\mathcal{O}'|^2$ states are reachable from any given state. Thus, the Dijkstra algorithm can compute the shortest path from any given state to its closest goal state in polynomial time.

3 Related Work

Tail Reasoning analyzes the end of a problem to enhance (forward) planning. This section explores existing research towards forward, regression, and bi-directional planning.

As introduced in Chapter 1, classical planning seeks a sequence of actions to achieve a goal state from an initial state. Heuristic search, particularly forward search guided by heuristics like HFF (Hoffmann and Nebel, 2011), is a dominant approach. However, forward search can struggle with implicit goal orderings, motivating the exploration of techniques incorporating information about the goal state. This chapter explores related work in heuristic search, bidirectional search, and the use of SAT solvers in planning.

3.1 Heuristic Search

Heuristic search plays a crucial role in classical planning. Planners like FF (Hoffmann, 2001) and HSP (Haslum and Geffner, 2000) rely on heuristics derived from relaxed problem representations (e.g., ignoring either the delete effects or preconditions of each action). The effectiveness of these heuristics directly impacts the planner's performance. Much research has focused on improving heuristic accuracy and efficiency (Liu *et al.*, 2002). While these approaches enhance the overall search process, they often do not explicitly address the challenges posed by implicit goal orderings. Our work complements these efforts by specifically incorporating goal-oriented information into the search process. Bonet and Geffner (Bonet and Geffner, 1999; Bonet and Geffner, 2001) explored heuristic search planners with a focus on a simple heuristic assuming independent preconditions. Like many others, their work highlights the importance of effective heuristics but does not directly tackle the goal-ordering problem we address.

3.2 SAT Search

Planning as satisfiability (Giunchiglia and Maratea, 2007; Wehrle and Rintanen, 2007; Rintanen, 2012) has been a successful approach, leveraging the power of SAT solvers to encode and solve planning problems. SATPLAN (Giunchiglia and Maratea, 2007) has been particularly effective. SATPLAN is able to use preferred operators and greatly improved upon (Kautz and Selman, 1992) by incorporating methods of previously used in graph-based planning.

3.3 Bidirectional Search

Bidirectional search algorithms, such as MM (Holte *et al.*, 2017), explore the state space from both the initial state and the goal state to meet in the middle. While bidirectional search can be effective in some cases, incorporating heuristics into bidirectional search has been a long-standing challenge. Our work draws inspiration from the concept of searching from both ends but utilizes a hybrid approach, combining forward heuristic search with backward reasoning using SAT solvers. This allows us to leverage the strengths of each approach, addressing different aspects of the planning problem. Kambhampati (Kambhampati and Srivastava, 1996) proposed a framework unifying state-space and plan-space planning, allowing for interleaving refinements. Our approach is a specific instance of this general idea, where we interleave forward state-space search with backward reasoning using SAT.

3.4 Tail Reasoning and Goal Ordering

Tail Reasoning relates to the final steps of a plan. While not explicitly named as such, this idea is implicitly present in some planning approaches. For example, planners that utilize regression search from the goal state inherently perform a form of tail reasoning. However, we focus on integrating this information into a forward search process. Our work is also related to research on goal ordering (Fiser and Komenda, 2018), which aims to identify and exploit dependencies between goals and state invariants. While some techniques for goal ordering may be applicable to our approach, our focus is more on leveraging information about the goal state to guide the forward search rather than explicitly determining the complete goal ordering. Our work is also related to landmark heuristics (Wichlacz *et al.*, 2022), which identify necessary actions or states for achieving the goal. While landmarks provide valuable guidance, our approach explicitly targets the use of goal-related information in forward search, offering a complementary perspective.

4 Methods

This section will present the two major approaches we want to analyze and some variations.

4.1 Reverse Forward Heuristic Search

We first decided to study relaxation heuristics and see if we could modify the standard relaxation - removing all delete effects - in a way that would obtain more impressive results for tail-heavy problems like blocksworld. Since plans for tail-heavy problems are shorter when planning on the reversed problem, we questioned whether something about the structure of the reversed operators is what led to that observation.

There are three differences between a problem and its reversed problem: the initial and goal states and the operators. The greatest challenge when solving a reversed problem is that all the new initial and goal states contain many more facts than the original problem. On the other hand, the number and size of operators do not change, but they have their preconditions and delete effects swapped. Our initial idea was to apply the same principle to HFF and swap the preconditions and delete effects before relaxing the operator.

4.1.1 Fast Forward Search

HFF, a widely-used system and state-of-the-art until 2009, uses a relaxation heuristic to estimate the cost of reaching the goal by ignoring the delete effects of operators and uses enforced hill-climbing search (EHS) to guide which state to reach next.

EHS works in a semi-greedy way, as shown in 1. It first calculates the relaxed distance between the current and goal states. It then does the same for adjacent states but stops when it finds an adjacent state with a shorter relaxed distance. The planner then advances to that state. If no adjacent state has a shorter relaxed distance, it randomly searches through states until it finds one with a lower relaxed distance.

Nowadays, EHS is mostly succeeded by A*. We still decided to stay with EHS since the underlying search is not part of this thesis, and any insights should be transferrable between different search systems.

Definition 8 (Reverse Relaxed Problem) *For any problem $P = \langle \mathcal{F}, \mathcal{O}, I, G \rangle$ we call the modified $P' = \langle \mathcal{F}, \mathcal{O}', I, G \rangle$ its reverse relaxed problem (RR-P), with*

$$\mathcal{O}' = \{\langle del(op), add(op), \varnothing \rangle \mid op \in \mathcal{O}\}$$

Algorithm 1 Fast Forward Search.

```
1  procedure ENFORCED HILL CLIMBING(state, 𝒪, goals, hmin)
2      hval ← hmin
3      if hval = 0 then
4          return state                                  We are in a goal state
5      while hmin ≤ hval do
6          op ← NEXT(𝒪, state)                           Obtain the next available operator
7          hval ← h_FF(op(state), 𝒪, goals)              Determines the relaxed distance
8      return ENFORCED HILL CLIMBING(op(state), 𝒪, goals, hval)
9  function NEXT(𝒪, state)
10     for op ∈ 𝒪 do
11         if pre(op) ⊆ state then
12             yield op                                  The function continues from here if called again.
13     for op ∈ 𝒪 do
14         if pre(op) ⊆ state then
15             yield op ∘ NEXT(𝒪, op(state))             Yields a chain of operators
```

Similarly, we call all operators of P' reverse relaxed operators (RR-OPs).

The RR-OP have the same general properties as relaxed operators, and we use them as drop-in replacements for HFF. To denote that RR-OPs are used in the heuristic, we call the modified planner Reverse Forward Heuristic Search (HRF).

That naive implementation suffered from difficulties with not well-formed problems since the heuristic was no longer admissible. We then tried a second modification of HFF that used the intersection between the precondition and the delete effects as the precondition.

Definition 9 (Symmetric Relaxed Problem) *For any problem $P = \langle \mathcal{F}, \mathcal{O}, I, G \rangle$ we call the modified $P' = \langle \mathcal{F}, \mathcal{O}', I, G \rangle$ its symmetric relaxed problem (SR-P), with*

$$\mathcal{O}' = \{\langle del(op) \cap pre(op), add(op), \emptyset \rangle \mid op \in \mathcal{O}\}$$

Similarly, we call all operators of P' symmetric relaxed operators (SR-OPs).

The results we obtained motivated us to make a further modification: Using an ensemble of the heuristics for HFF and HRF, which we call the Combined Forward Heuristic Search (HCF). We computed HCF by taking the average of HFF and HRF, so long as at least one of them can compute a result, or taking the lower of each otherwise. We present the algorithm in 2.

The idea was that since HFF computes much more often and is correct more frequently, we would use it as a fall-back if HRF fails, with the hope that HCF would preserve the rare cases when HRF was superior.

Algorithm 2 Combined Forward Heuristic Search.

```
1  procedure ENFORCED HILL CLIMBING(state, 𝒪, goals, hmin)
2      hval ← hmin
3      if hval = 0 then
4          return state                                  We are in a goal state
5      while hmin ≤ hval do
6          op ← NEXT(𝒪, state)                           Obtain the next available operator
7          hval_FF ← h_FF(op(state), 𝒪, goals)           Determines the relaxed distance
8          hval_RF ← h_RF(op(state), 𝒪, goals)           Determines the reverse relaxed distance
9          if hval_RF + hval_FF < infinity then
10             hval ← (hval_RF + hval_FF) / 2
11         else
12             hval ← min(hval_RF, hval_FF)
13     return ENFORCED HILL CLIMBING(op(state), 𝒪, goals, hval)
```

4.2 Hybrid Search

4.2.1 Planning as Satisfiability

Initially proposed by Kautz and Selman, 1992 was an alternative system. Instead of a state-space search, this approach transforms the planning problem into a boolean formula and passes it to any state-of-the-art satisfiability (SAT) solver. During the transformation, the solver guesses a length k for a plan. If the formula is satisfiable, a plan of length k exists for the planning problem, which the variable assignments for the formula allow to extract. If the plan is unsatisfiable, the solver guesses a different plan length, and the process repeats.

As long as the initial plan length starts with $k = 0$ and increases by one each time, the solution found is always optimal. Another observation is that satisfiability planning cannot determine if a planning problem is unsolvable, no matter how much time we give the planner. Compare this to state-space planning, which maps out the whole (finite) state space given enough time and memory. Therefore, we provide all satisfiability problems with a maximum number of steps before stopping, called the horizon.

The transformation creates variables $F_i, 0 \leq i \leq k$ for every fact and $OP_i, 1 \leq i \leq k$ for every operator. All variables F_0 represent the facts in the initial state of the planning problem, F_k are the facts in the goal state, and all other variables F_i are facts in intermediate states. Similarly, the variables OP_i represent the $i-th$ operator selected for the final plan.

Four types of literals enable this behavior. First, the boolean formula ϕ contains all initial and goal state facts as literals. All facts not part of the initial state are present as negated literals. In

this section, we use sets to represent a conjunction of literals and variables.

$$F \in I \rightarrow \{F_0\} \in \phi$$
$$F \in \mathcal{F}/I \rightarrow \neg\{F_0\} \in \phi$$
$$F \in G \rightarrow \{F_k\} \in \phi$$

Then, we must ensure that exactly one operator occurs at any step.

$$\forall i \leq k \rightarrow \{OP_i \text{ for } OP \in \mathcal{O}\} \in \phi$$
$$\forall OP \in \mathcal{O}, \forall i \leq k \rightarrow \{OP_i\} \cup \left\{\neg OP_i' \text{ for } OP' \in \mathcal{O}/\{OP\}\right\} \in \phi$$

Next, we fulfill the preconditions and effects of each operator by treating the effects as preconditions of the same step.

$$\begin{aligned}\forall OP \in \mathcal{O}, \forall i \leq k \rightarrow \neg(\{OP_i\} &\cup \{\neg F_{i-1} \text{ for } F \in pre(OP)\} \\ &\cup \{\neg F_i \text{ for } F \in add(OP)\} \\ &\cup \{F_i \text{ for } F \in del(OP)\}) \in \phi\end{aligned}$$

Finally, we state that each fact in state i was either carried over from state $i-1$ or added through the effect of an operator. Let $operators(f) \subseteq \mathcal{O}$ be the set that denotes all operators that add $f \in \mathcal{F}$.

$$\forall F \in \mathcal{F}, \forall i \leq k \rightarrow \neg(\{F_i, \neg F_{i-1}\} \cup \{\neg OP_i \text{ for } OP \in operators(F)\}) \in \phi$$

The focus of our studies belonged to the hybrid approach, where the problem is dynamically split into two parts: a back section where all goals are reached in quick succession and a front section that reaches a state from which that is possible. Each partial solution yields a partial plan, which is combined in the end.

Algorithm 3 Hybrid Search.

```
1 procedure HYBRIDSEARCH(F, O, I, G)
2     G', plan-fragment-tail ← TAIL-REASONING(F, O, I, G)
3     plan-fragment-head ← ENFORCED-HILL-CLIMBING(F, O, I, G')
4     return plan-fragment-head + plan-fragment-tail
```

We aimed to design an algorithm to find the intermediary goals G'. In our first naive implementation 4, we attempted a backward search from the set of goals to reach any (partial) state that contained no facts about the goals. In cases with more than one intermediary state, we would randomly pick one such state.

In many cases, selecting a random satisfying variable assignment led to unreachable intermediary states from the initial state. We needed a different method to evaluate the potential states. We came up with three methods that had a chance to rectify that.

Algorithm 4 Naive Tail Reasoning.

```
1  procedure TAIL REASONING(F, O, I, G)
2      t ← 0
3      φ(k) := SAT-PLAN(F, O_r, F/G, G, k)        k is the variable for plan length
4      while Plan π not found do
5          problem ← φ(F, O_r, F/G, G, t)
6          π ← SAT-SOLVE(problem)        False or a random satisfiable variable assignment
7          t ← t + 1
8      G' ← F/G                          The operators in π are reversed operators
9      for op ∈ π do
10         G' ← op(G')
11     plan-fragment-tail ← ⟨π_t, ..., π_1⟩                    π is reversed again
12     G' ← F/G'
13     return G', plan-fragment-tail
```

The sub-procedure SAT-SOLVE referenced in 4 calls a SAT solver to work with them. We used clingo (Gebser *et al.*, 2017) for our implementation which requires a program written in Answer Set Programming (ASP) and consists of first-order logic instead of boolean formulas.

Definition 10 (SAT-PLAN($\mathcal{F}, \mathcal{O}, I, G, k$)) *SATPLAN describes a generic schema on translating a problem $\langle \mathcal{F}, \mathcal{O}, I, G \rangle$ into a first-order formula $\phi(k)$. We use ASP syntax for operators. First, we encode the initial state, goals, and the operators.*

$$
\begin{aligned}
fact(f, 0). &\in \phi(k) && \forall f \in I \\
fact(f, k). &\in \phi(k) && \forall f \in G \\
pre(op, f). &\in \phi(k) && \forall op \in \mathcal{O}, \forall f \in op(pre) \\
add(op, f). &\in \phi(k) && \forall op \in \mathcal{O}, \forall f \in op(add) \\
del(op, f). &\in \phi(k) && \forall op \in \mathcal{O}, \forall f \in op(del)
\end{aligned}
$$

We add rules that ensure the application of precisely one operator per cycle.

$$
\begin{aligned}
\{\{op(o_1, T), ..., op(o_{|\mathcal{O}|})\}\} == 1 \text{ :- } & T = 1..k. \in \phi(k) \text{ with } o_1, ..., o_{|\mathcal{O}|} \in \mathcal{O} \\
& \text{:- } op(OPX, T), op(OPY, T), OPY! = OPX. \in \phi(k)
\end{aligned}
$$

We also add rules for the effects of any operator.

$$\begin{aligned} fact(X,T) &\text{:- } op(OP,T), add(OP,X). \in \phi(k) \\ fact(X,T) &\text{:- } op(OP,T+1), pre(OP,X). \in \phi(k) \\ &\text{:- } fact(X,T), op(OP,T), del(OP,X). \in \phi(k) \end{aligned}$$

Finally, we added maintenance rules that ensure facts do not change between steps without an operator's effects.

$$\begin{aligned} fact(X,T) &\text{:- } op(OP,T), notdel(OP,X), fact(X,T-1). \in \phi(k) \\ &\text{:- } fact(X,T), op(OP,T), notadd(OP,X), not\, fact(X,T-1). \in \phi(k) \end{aligned}$$

With these definitions, $\phi(k)$ is a generic model that solves every solvable problem with a plan of length k.

If the problem has a complete initial state, we can further extend $\phi(k)$ with the rule

$$\text{:- } fact(f,0). \in \phi(k) \qquad\qquad \forall f \in \mathcal{F}/I$$

.

4.2.2 Modified SAT-PLAN

During our tests, we modified TAIL REASONING 4 by replacing SAT-PLAN with one of the following options to improve performance.

Definition 11 (MAX-PLAN($\mathcal{F}, \mathcal{O}, I, G, W, k$)) *MAX-PLAN is a generalization for the model $\phi(k) = SAT - PLAN(\mathcal{F}, \mathcal{O}, I, G, k)$ with an additional set of facts W. MAX-PLAN has an additional constraint compared to SAT-PLAN; the final state must contain as many facts in W as possible. We can define this in ASP as a negative weight, i.e., that the fact is not part of the final state is penalized by a constant factor.*

$$\text{:}\sim fact(f,k).\left[-1, w_f\right] \in \phi(k) \qquad\qquad \forall f \in W$$

The additional weight constraints are only relevant after $\phi(k)$ is proven satisfiable. It provides an optimal satisfiable assignment regarding W instead of a random satisfiable assignment.

$$\phi(k) := MAX - PLAN(\mathcal{F}, \mathcal{O}_r, \mathcal{F}/G, G, \mathcal{F}/I, k)$$

Definition 12 (MUTEX-PLAN($\mathcal{F}, \mathcal{O}, \mathcal{V}, I, G, k$)) *MUTEX-PLAN is another generalization for the model $\phi(k) = SAT - PLAN(\mathcal{F}, \mathcal{O}, I, G, k)$ with an additional set of constraints $\mathcal{V}$. These invariants contain i-tuples of facts that may not appear together in any state. In ASP, we can add them as additional constraints.*

$$:\text{-}\ fact(f_1, T), ..., fact(f_i).. \in \phi(k) \qquad \forall \langle f_1, ..., f_i \rangle \in \mathcal{V}$$

During our tests MUTEX-PLAN is modified TAIL REASONING 4 by replacing SAT-PLAN with

$$\phi(k) := MUTEX - PLAN(\mathcal{F}, \mathcal{O}_r, \mathcal{F}/G, G, M, k)$$

where M was a hand-crafted set of mutual exclusive facts. While our planner was supposed to be domain-independent and not require any prior domain knowledge, we posit that mutual exclusive facts were only small impositions since the rules for invariants are made during the creation of a domain (Grundke *et al.*, 2024). So long as such rules exist, inferring the mutexes is trivial. Even without explicit invariants, it is possible to infer such rules (Fiser and Komenda, 2018), though that problem is in PSPACE, just like planning itself.

Due to the difficulty of designing invariants by hand, we provided such functionality in chapter 5.

4.2.3 Modified Fast Forward Search

Aside from the different versions of SAT transformations, we also considered changing the search instead. The alternative to modifying the backward search is to change the forward search. Instead of trying to reach a specific intermediate state, the forward search may try to reach any of the possible intermediate states discovered by SATPLAN or a (random) subset of them since the number of such intermediate states can be significant.

We ultimately had to abandon research in this direction due to time constraints.

5 Domains

This chapter will present and analyze the new tunnel domain and briefly overview the other benchmarks we will use.

5.1 Tunnel Domains

The design of the tunnel domain tests a planner's ability to reason about implicit goal ordering while being able to increase the instance size without affecting the complexity of the search. Sokoban inspired the domain. The problem consists of a grid with randomly placed blocks that an actor can push and pull. The goal is to fill an initially empty tunnel with these blocks. The catch is that the tunnel is only one space wide, so the actor can manipulate only the block at the entrance. Thus, the problem requires the planner to fill the tunnel in reverse order (from the final space backward).

Definition 13 (Tunnel) *The tunnel domain has five variable types - ?location, ?actor, ?block, ?direction, and ?entity. The latter is the abstract supertype of ?actor and ?block. The type ?direction consists of only the constants* ***above, below, left, right*** *while the type ?actor has the sole constant* ***act****.*

Tunnel further has three predicates

> *at(?entity, ?location) - represents the location of an entity*
> *dir(?location, ?direction, ?location) - represents the connections between locations*
> *free(?location) - shows if a location contains no entity*

with two rules for the predicate dir. If for some ?locations ***l1****, and* ***l1*** *the fact dir(****l1, above, l2****) is part of any initial state, then is dir(****l2, below, l1****) too. The same rule applies to dir(****l1, left, l2****) and dir(****l2, right, l1****).*

The domain has three actions

$push(l1, l2, l3 - ?location, b - ?block, d - ?direction)$:
pre : $\{at(\textbf{act}, l1), at(b, l2), free(l3), dir(l1, d, l2), dir(l2, d, l3)\}$
add : $\{free(l1), at(\textbf{act}, l2), at(b, l3)\}$
del : $\{at(\textbf{act}, l1), at(b, l2), free(l3)\}$
$pull(l1, l2, l3 - ?location, b - ?block, d - ?direction)$:
pre : $\{at(\textbf{act}, l2), at(b, l3), free(l1), dir(l1, d, l2), dir(l2, d, l3)\}$
add : $\{free(l3), at(\textbf{act}, l1), at(b, l2)\}$

$$
\begin{aligned}
&del : \{at(\boldsymbol{act}, l2), at(b, l3), free(l1)\} \\
move(l1, l2?location, d - ?direction)&: \\
&pre : \{at(\boldsymbol{act}, l1), free(l2), dir(l1, d, l2)\} \\
&add : \{free(l1), at(\boldsymbol{act}, l2)\} \\
&del : \{at(\boldsymbol{act}, l1), free(l2)\}
\end{aligned}
$$

Two variables define all problems in this domain: the field size $n \in \mathbb{N}$ and the tunnel length $m \in \mathbb{N}$ with $n^2 > m$ (the problems are not well defined with less than $m + 1$ spaces in the grid; this is not a significant restriction since all problems with $m - 1 = n^2$ are already unsolvable). We write them as $\mathbf{P}^{n,m}_{tun} = \langle \mathcal{F}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ with the additional constants

$$
\begin{aligned}
?location &: \left\{field_{i,j} \mid \forall i, j \leq n \right\} \cup \{tun_i \mid \forall i \leq m\} \\
?block &: \{block_i \mid \forall i \leq m\}
\end{aligned}
$$

There is only a single set of goals

$$
\mathcal{G} = \left\{\left\{\overline{free(tun_i)} \mid \forall i \leq m\right\}\right\}
$$

but a large number of initial states. All possible initial states share part of the same location layout. The field is a quadratic grid of size n, and the tunnel is a one-width pathway of length m that starts empty.

$$
\begin{aligned}
grid =& \left\{dir\left(field_{i,j}, left, field_{i,j+1} \mid \forall i \leq n, \forall j \leq n-1\right)\right\} \\
&\cup \left\{dir\left(field_{i+1,j}, above, field_{i,j} \mid \forall i \leq n-1, \forall j \leq n\right)\right\} \\
path =& \{dir(tunnel_i, left, tunnel_{i+1}) \mid \forall i \leq m-1\} \\
&\cup \{free(tunnel_i) \mid \forall i \leq m\}
\end{aligned}
$$

What differentiates the potential initial states is the connection of the grid and path, as well as the distribution of the entities. Each entity is at exactly one location. Each field contains either nothing or precisely one entity. Finally, a random field at the right edge of the grid connects to the first location of the tunnel. Let $\mathcal{E}$ be the set of entities and $\mathcal{L}$ the set of all fields.

$$
\begin{aligned}
\mathcal{I} = \{&\{at(block_i, l) \mid \forall i \leq m \exists l \in loc_b, \exists l \in \mathcal{L}\} \\
&\cup \left\{free\left(l \in loc_f\right)\right\} \\
&\cup \left\{dir\left(field_{k,n}, left, tunnel_1\right), at(act, l \in loc_a)\right\} \cup grid \cup path \\
&\mid \forall k \leq n, \left(loc_b, loc_a, loc_f\right) \in Part_3(\mathcal{L}), |loc_b| = m, |loc_a| = 1\}
\end{aligned}
$$

In addition to the tunnel domain, we have created a simplified version that only consists of the tunnel. It similarly requires goal ordering from the planner but does not contain as many facts

and cannot have its size increased without increasing complexity. It resembles a blocksworld problem where all blocks start on the table.

Definition 14 (Simplified Tunnel Domain) *The simplified tunnel domain only has two variable types - ?block and ?location. It also has a predefined constant - the tun_0 of type ?location.*

It has six predicates.

front(?block) - represents the first block in the tunnel
outside(?block) - represents all blocks outside the tunnel
order(?block, ?block) - the order of blocks in the tunnel
at(?block, ?location) - represents the actual location of a block in a tunnel
next(?location, ?location) - gives which locations are adjacent
free(?location) - shows if a location has no block in it

*Similar to the tunnel domain, if the fact $next(l1, l2)$ for some locations **l1, l2** exists in the initial state, so does the fact $next(l2, l1)$.*

The domain also defines three actions

$place(b1, b2 - ?block)$:
pre : $\{outside(b1), free(tun_0), front(b2)\}$
add : $\{order(b1, b2), front(b1), at(b1, tun_0)\}$
del : $\{outside(b1), front(b2), free(tun_0)\}$
$remove(b1, b2 - ?block)$:
pre : $del(place(b1, b2))$
add : $pre(place(b1, b2))$
del : $add(place(b1, b2))$
$move(l1, l2 - ?location, b - ?block)$:
pre : $\{at(b, l1), free(l2), next(l1, l2)\}$
add : $\{free(l1), at(b, l2)\}$
del : $\{at(b, l1), free(l2)\}$

Note that place and remove are designed to reverse each other exactly.

A single variable defines all problems in this domain: the tunnel length m. We write them as $\mathbf{P}^m_{tun} = \langle \mathcal{F}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ with the constants

$$?location : \{tun_i \mid \forall i \leq m\}$$
$$?block : \{block_i \mid \forall i \leq m\}$$

There is only a single set of goals.

$$\mathcal{G} = \left\{\left\{\overline{free(tun_i)} \mid \forall i \leq m\right\}\right\}$$

Further, there is only a single initial state where every block starts outside, except for one block already placed at the tunnel entrance.

$$\begin{aligned}\mathcal{I} = \{\{&free(tun_i), outside(block_i) \mid \forall 1 \leq i \leq m\} \\ &\cup \{next(tun_i, tun_{i+1}) \mid \forall i \leq m-1\} \\ &\cup \{front(block_0, at(block_0, tun_0))\}\}\end{aligned}$$

5.1.1 Properties of the Tunnel Domains

The Tunnel domain highlights the limitations of HFF, which struggles to recognize the need to fill the tunnel's final space first. Similarly, it highlights the problems of SAT planning, which has problems with large numbers of facts.

Corollary 2 (Simple Tunnel Complexity) *The optimal solution of any problem in $\mathbf{P}^m_{tun}$ has a length of $\frac{m^2+m}{2} \in \mathcal{O}(m^2)$.*

Proof. *Any solution requires $inside(block_i), i \in [1, n]$ to be part of the final state. The only action that adds the predicate $inside(X)$ is place. Therefore, any plan must contain place at least n times. Furthermore, since each tunnel only contains n spaces, and blocks cannot share a space, every space in the tunnel needs to contain a block. To move a block from $tutn_j$ to tun_k for $j < k \leq n$ requires $k - l$ uses of the push action. Since all blocks start in tun_0, we can finally compute the number of calls.*

$$\sum_{i=1}^{n}(i-1) = \sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$$

Altogether, that results in $n + \frac{n(n-1)}{2} = \frac{n^2+n}{2}$ actions at minimum.

Corollary 3 (Tunnel Complexity) *The optimal solution of any problem in $\mathbf{P}^{n,m}_{tun}$ has a length of at most $(9 + 4n + m)m \in \mathcal{O}(nm + m^2)$.*

Proof. *Consider first $\mathbf{P}^{n,m}_{tun}$ with $m = 1$. In this case, any block is at most $2n$ fields away from the tunnel, for example, with the tunnel in the upper right corner of the field and the block in the lower left corner. In that case, the actor must travel at most $2n$ locations to the block; moving the block to an adjacent field takes either a push/pull operation or two move operations, followed by a push/pull operation. Finally, the actor pushes the blocks down the tunnel, which requires one push operation for $4n + 6 + 1 \in \mathcal{O}(n)$ operations.*

For higher m, the planner repeats the process at most m times. For each additional block, the blocks need to enter the last location of the tunnel and the actor must leave the tunnel, leading to an additional $2m - 1$ operations. That leads to a total optimal plan length of at most $m(4n + 6 + 2m - 1) \in \mathcal{O}(nm + m^2)$

Corollary 4 (Tunnel Invariants) *All states in the tunnel must follow four rules. M is the set of mutually exclusive tuples in any given state. We use $A - ?type$ to show that A is of type $?type$ and M for the set of mutexes as described in 4.2.2.*

$$\forall A - ?entity, \forall L, L' - ?location, L \neq L' : \langle at(A, L), at(A, L')\rangle \in M$$
$$\forall A - ?entity, \forall L - ?location : \langle at(A, L), free(L)\rangle \in M$$

For tunnel problems with $\mathbf{P}^{n,m}_{tun}$ leads to a size of

$$\begin{aligned} |M| &= (m+1)\frac{(n+m)(n+m-1)}{2} + (m+1)(n+m) \\ &= \frac{mn^2 + 2m^2n + m^3 + nm + m^2}{2} + m + n \in \mathcal{O}\big(mn^2 + m^3\big) \end{aligned}$$

.

Corollary 5 (Simple Tunnel Size) *Any problem in $\mathbf{P}^{m}_{tun}$ has at most $\mathcal{O}\big(m^2\big)$ facts and $\mathcal{O}\big(m^2\big)$ operators.* ***Proof.*** *Let $m \in \mathbb{N}$, and $\mathbf{P}^{m}_{tun}$ the simple tunnel problem. Then $\mathbf{P}^{m}_{tun}$ had exactly m constants of each ?location and ?block. Each predicate and action with k parameters may produce, at most, m^k facts or operators. $\mathbf{P}^{m}_{tun}$ only has no predicates with $k = 3$ parameters, therefore $|\mathcal{F}| \in \mathcal{O}\big(m^2\big)$.*

$\mathbf{P}^{m}_{tun}$ has only one action with three parameters - the action move(?location, ?location, ?block). However, the precondition next(?location, ?location) constrains the combinations of the ?location parameters for move. Only $2m$ facts are in the initial state, and no operator can add them with their effects. Therefore, there are at most $2m^2$ move-operators and overall $|\mathcal{O}| \in \mathcal{O}\big(m^2\big)$

Corollary 6 (Tunnel Size) *Any problem in $\mathbf{P}^{m}_{tun}$ has at most $\mathcal{O}\big(mn^2\big)$ facts and $\mathcal{O}\big(mn^2\big)$ operators.* ***Proof.*** *Let $m \in \mathbb{N}$, and $\mathbf{P}^{n,m}_{tun}$ the tunnel problem. Then $\mathbf{P}^{m}_{tun}$ has m constants of types ?entity and ?block each, one constant of type ?actor, four constants of type ?direction, and $n^2 + m$ of type ?location. It follows that $\mathcal{F}$ may contain at most mn^2 at-facts, $n^2 + m$ free-facts. The construction constrains the number of dir-facts; each location may only appear in at most four dir-facts. Therefore, the number of dir-facts is bound by $4\big(n^2 + m\big)$. Overall, we have $\mathcal{F} \in \mathcal{O}\big(mn^2\big)$*

$\mathbf{P}^{n,m}_{tun}$ has three action, but push and pull clearly result in a similar number of operators. We will, therefore, first look at move(?location, ?location, ?direction). The direction constrains the number of possible combinations for the locations, similar to what we used in our proof for the simple tunnel size. We, therefore, have an upper bound of $4\big(n^2 + m\big)$ move-operators.

This structure also constrains the push (and pull) action. The three locations are bound by $4\big(n^2 + m\big)$ combinations, so we have at most $4\big(n^2 + m\big) \cdot m$ operators. Overall the total number

of operators is bound by $|\mathcal{O}| \in \mathcal{O}(m^2 + mn^2)$ *and since* $n^2 > m$ *by our construction requirement that further simplifies to* $|\mathcal{O}| \in \mathcal{O}(mn^2)$

5.2 Benchmark Domains

This section will briefly overview the various domains we use as benchmarks in chapter 6. Complexity results exist for the domains defined before 2006 (Helmert, 2003).

Year	Domain Name	Year	Domain Name
1998	Grid	2006	tpp
	Gripper	2011	Sokoban
	Logistics	2014	Child Snack
2002	Blocksworld		

Table 5.1: IPC Benchmark Domains

5.2.1 Blocksworld

The most famous STRIPS domain consists of a table with stacks of labeled blocks. A single operation can remove the topmost block from any stack, place a held block on the table or any stack, or pick a block from the table.

The goal is to create one or more towers with a specific block order. Achieving the goal requires inferring the (implicit) order of the subgoals.

Corollary 7 (Blocksworld Invariants) *Blocksworld contains four fluent predicates namely ontable(?block), clear(?block), handempty, and holding(?block) that need to follow four rules regarding mutual exclusion. M is the set of mutually exclusive tuples in any given state.*

$$\begin{aligned}
\forall A, B - ?block, A \neq B &: \langle on(A,B), on(B,A)\rangle \in M \\
\forall A, B - ?block, A \neq B &: \langle holding(A), holding(B)\rangle \in M \\
\forall A, B - ?block &: \langle ontable(A), on(A,B)\rangle \in M \\
\forall A - ?block &: \langle holding(A), handempty\rangle \in M
\end{aligned}$$

For blocksworld problems with n blocks, this leads to a size of

$$\begin{aligned}
|M| &= 3\frac{n(n-1)}{2} + n \\
&= 2n(n-1) \in \mathcal{O}(n^2)
\end{aligned}$$

5.2.2 Grid

There is a (square) grid of locations, each of which may be empty, contain a key, or be locked. A robot can move through orthogonally adjacent locations one field at a time. If the robot is in a room with a key, it can pick it up, and if it has the key to a matching locked room, it may open it if they are orthogonally adjacent. The goal is to travel to specific rooms or collect keys.

Corollary 8 (Grid Invariants) *Grid contains six fluent predicates, namely at(?key, ?loc), handempty, atrobot(?loc), locked(?loc), open(?loc), holding(?key), that need to follow six rules regarding mutual exclusion.*

$$\begin{aligned}
\forall K - ?key, \forall L, L' - ?loc, L \neq L' &: \langle at(K, L), at(K, L') \rangle \in M \\
\forall L, L' - ?loc, L \neq L' &: \langle atrobot(L), atrobot(L') \rangle \in M \\
\forall L - ?loc &: \langle open(L), closed(L) \rangle \in M \\
\forall K - ?key &: \langle holding(K), handempty \rangle \in M \\
\forall K - ?key, \forall L - ?loc &: \langle holding(K), at(K, L) \rangle \in M \\
\forall K, K' - ?key, K \neq K' &: \langle holding(K), holding(K') \rangle \in M
\end{aligned}$$

For grid problems with l locations and k keys, this leads to a size of

$$\begin{aligned}
|M| &= k\frac{l(l-1)}{2} + \frac{l(l-1)}{2} + kl + l + k + \frac{k(k-1)}{2} \\
&= (k+1)\frac{l(l-1)+k}{2} + l + kl \in \mathcal{O}\left(kl^2 + k^2\right)
\end{aligned}$$

5.2.3 Gripper

A two-handed robot can carry one ball in each hand and move between two rooms.

Each room contains n balls. The goal is to move all balls from one room to the other. The problem is straightforward and is a baseline every general planner should be able to solve.

Corollary 9 (Gripper Invariants) *Gripper contains four fluent predicates, namely at(?ball, ?loc), atrobot(?loc), free(?gripper), and carry(?ball, ?gripper), that need to follow six rules regarding*

mutual exclusion.

$$\begin{aligned}
\forall B - ?ball, \forall L, L' - ?loc, L \neq L' &: \langle at(B, L), at(B, L') \rangle \in M \\
\forall L, L' - ?loc, L \neq L' &: \langle atrobot(L), atrobot(L') \rangle \in M \\
\forall B - ?ball, \forall L - ?loc, \forall G - ?gripper &: \langle carry(B, G), at(B, L) \rangle \in M \\
\forall G - ?gripper, \forall B, B' - ?ball, B \neq B' &: \langle carry(B, G), carry(B', G) \rangle \in M \\
\forall B - ?ball, \forall G, G' - ?gripper, G \neq G' &: \langle carry(B, G), carry(B, G') \rangle \in M \\
\forall B - ?ball, \forall G - ?gripper &: \langle carry(B, G), free(G) \rangle
\end{aligned}$$

For the sake of brevity, we will be shortening our proofs from now on. For grid problems with l locations, g grippers, and b balls, this leads to a size of

$$\begin{aligned}
|M| \in &\mathcal{O}\big(bl^2 + l^2 + bl + gb^2 + bg^2 + bg\big) \\
= &\mathcal{O}\big(bl^2 + gb^2 + bg^2\big)
\end{aligned}$$

Normally, there are exactly two locations and two grippers, so in those cases, that simplifies into $\mathcal{O}\big(b^2\big)$

5.2.4 Logistics

Packages are stored at various locations throughout several cities and can be loaded and transported by trucks and airplanes. The catch is that trucks have random access to all locations in their city but cannot leave for other cities, while airplanes can only travel to unique locations marked as airports but may move between any of them. There is only one truck per city, but any number of airplanes.

The goal is to transport specific packages to specific locations. The difficulty in this domain does not lie in solving it but in finding an optimal or at least short solution.

Corollary 10 (Logistics Invariants) *Logistics contains only two fluent predicates at(?entity, ?loc) and in(?package, ?vehicle) that need to follow three rules regarding mutual exclusion.*

$$\begin{aligned}
\forall E - ?entity, \forall L, L' - ?loc, L \neq L' &: \langle at(E, L), at(E, L') \rangle \in M \\
\forall P - ?package, \forall V, V' - ?vehicle, V \neq V' &: \langle in(P, V), in(P, V') \rangle \in M \\
\forall P - ?package, \forall L - ?loc, \forall V - ?vehicle &: \langle at(P, L), in(P, V) \rangle \in M
\end{aligned}$$

For logistics problems with n packages, m vehicles, and l locations, this leads to a size of

$$|M| \in \mathcal{O}\big((n + m)l^2 + nm^2 + nml\big)$$

5.2.5 Sokoban

There is a grid where some spaces are blocked off. Inside the grid are several blocks and an actor. The actor can push (but not pull) blocks and move to adjacent fields on the grid. The actor may also only move one block at a time.

The goal is to have any of the blocks placed on specific fields. Because the actor cannot pull, making a wrong move can result in a dead-end state where no goal state is reachable anymore.

Corollary 11 (Sokoban Invariants) *Sokoban contains three fluent predicates: at(?entity, ?loc), at-goal(?block), and clear(?loc), which need to follow three rules regarding mutual exclusion. The definition further divides the type ?loc into ?nongoal and ?goal locations.*

$$\begin{aligned}\forall A - ?entity, \forall L, L' - ?loc, L \neq L' &: \langle at(A, L), at(A, L')\rangle \in M\\ \forall A - ?entity, \forall L - ?loc &: \langle at(A, L), clear(L)\rangle \in M\\ \forall A - ?entity, \forall L - ?nongoal &: \langle at(A, L), atgoal(A)\rangle \in M\end{aligned}$$

For sokoban problems with n locations and m+1 entities, this leads to a size of

$$\begin{aligned}|M| &\in \mathcal{O}\big(mn^2 + mn + m(n-m)\big)\\ &= \mathcal{O}\big(mn^2\big)\end{aligned}$$

5.2.6 Travelling Purchase Problem

The travelling purchase problem (TPP) is a planning problem from operations research (OR) and acts as a generalization of the travelling salesman problem (TSP) where we have some products and some markets. Each market sells a different number of those products at a different price, but traveling to a market also carries a cost. The goal is to buy a certain amount of each product while minimizing the cost. Like the TSP, the TPP is NP-Hard.

Corollary 12 (TPP Invariants) *TPP contains five fluent predicates, namely at(?truck, ?loc), loaded(?good, ?truck, ?level), ready(?good, ?market, ?level), on-sale(?good, ?market, ?level) and stored(?good, ?level) that need to follow five rules regarding mutual exclusion.*

$$\begin{aligned}\forall T - ?truck, \forall L, L' - ?loc, L \neq L' &: \langle at(T, L), at(T, L')\rangle \in M\\ \forall G - ?good, \forall M - ?market \forall L, L' - ?level, L \neq L' &: \langle onsale(G, M, L), onsale(G, M, L')\rangle \in M\\ \forall G - ?good, \forall M - ?market \forall L, L' - ?level, L \neq L' &: \langle ready(G, M, L), ready(G, M, L')\rangle \in M\\ \forall G - ?good, \forall T - ?truck, \forall L, L' - ?level, L \neq L' &: \langle loaded(G, T, L), loaded(G, T, L')\rangle \in M\\ \forall G - ?good \forall L, L' - ?level, L \neq L' &: \langle stored(G, L), stored(G, L')\rangle \in M\end{aligned}$$

For TPP problems with n markets, n+k locations, m goods, t trucks, and l levels, this leads to a size of

$$\begin{aligned}|M| \in &\mathcal{O}\big(t(n+k)^2 + 2gnl^2 + gtl^2 + gl^2\big)\\ &= \mathcal{O}\big(t(n+k)^2 + (t+n)gl^2\big)\end{aligned}$$

5.2.7 Child Snack

Sandwiches are served to hungry children. Each child needs a specific version, which can contain gluten or not. The sandwiches are created in a kitchen, put on a tray, and then carried to the children to feed them.

The goal is to feed all children. This domain contains many actions and a complex structure, making solving it difficult.

Corollary 13 (Childsnack Invariants) *Childsnack contains six fluent predicates, namely ontray(?sandwich, ?tray), notexist(?sandwich), nogluten(?foodstuff), gluten(?foodstuff), at(?tray, ?loc), atkitchen(?foodstuff) that need to follow three rules regarding mutual exclusion.*

$$\begin{aligned}\forall T - ?tray, \forall L, L' - ?loc, L \neq L' &: \langle at(T,L), at(T,L')\rangle \in M\\ \forall S - ?sandwich, \forall T - ?Tray &: \langle ontray(S,T), atkitchen(S)\rangle \in M\\ \forall F - ?foodstuff &: \langle gluten(F), nogluten(F)\rangle \in M\\ \forall S - ?sandwich &: \langle notexist(S), nogluten(S)\rangle \in M\\ \forall S - ?sandwich &: \langle gluten(S), notexist(S)\rangle \in M\\ \forall S - ?sandwich &: \langle atkitchen(S), notexist(S)\rangle \in M\\ \forall S - ?sandwich, \forall T - ?tray &: \langle ontray(S,T), notexist(S)\rangle \in M\end{aligned}$$

For Child Snack problems with n sandwiches, n+k foodstuff, t trays, and m locations, this leads to a size of

$$\begin{aligned}|M| \in &\mathcal{O}\big(tm^2 + tn + n + k + 3n + tn\big)\\ &= \mathcal{O}\big(tm^2 + tn + k\big)\end{aligned}$$

6 Results

We tested our various approaches on the domains presented in 5. We sourced all sets from previous planning competitions of the IPC, except the Tunnel domain. We used $\mathbf{P}_{tun}^{n,m}$ with all combination of $n \in \{4, 6, 8, 10\}, m \in \{2, 3, 4, 5, 6\}$ for a total of twenty problems. The table 6.1 presents the number of problems for all domains used. The raw results can be found in the Appendix A.1 We use our implementation for SAT and HFF. To ensure correctness, we used

Count	Domain Name	Count	Domain Name
5	Grid	30	TPP
20	Gripper	30	Sokoban
28	Logistics	20	Child Snack
35	Blocksworld	20	Tunnel

Table 6.1: number of problems per domain

pyperplan (Alkhazraji *et al.*, 2020) to compare our results against. Pyperplan does not offer state-of-the-art performance but focuses on (formal) correctness. We implemented all of our algorithms in Python and ASP; the latter we provided to the Python implementation of Clingo (Gebser *et al.*, 2017).

We gave each planner 60 seconds to solve each problem. If a planner failed to reach a goal state before that time, we would terminate the attempt after it finished its current search step. Since a plan exists for all problems in the benchmark domains, we treated not having a plan and prematurely terminating a planner equally; both count as unable to solve the problem.

We also determined that any problem containing more than 20000 operators would crash Clingo by running out of memory with our implementation. We provided 30GB of memory for each attempt, so exceeding that forced us to terminate the planner.

6.1 Performance Reverse Forward Heuristic Search

The table 6.1 shows the results of modifying HFF. The results are similar in blocksworld and Gripper, which is unsurprising. Especially the blocksworld domain is not well-formed, and its operators' variable preconditions and delete effects are the same. Conversely, we expect a good performance on Gripper since it is a straightforward domain.

Both HFF and HRF perform quite poorly on sokoban and child snack with the original search beating our modifications slightly in the first case. The performance on the tunnel domain is

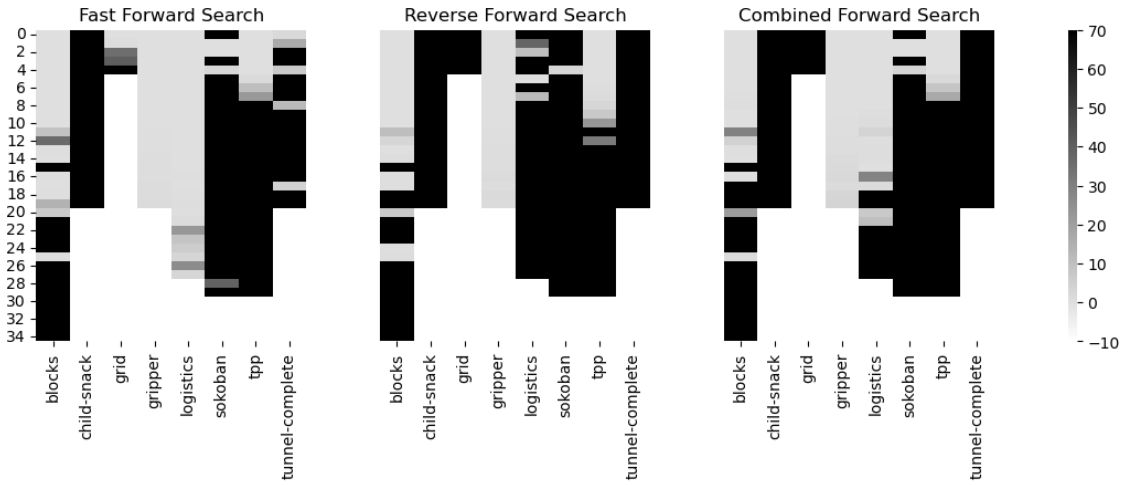

Figure 6.1: Runtime for modified HFF. Black indicates a problem that did not terminate. The color indicates the time it took to find a solution in seconds. Beige is close to 0 seconds; dark gray is close to 60 seconds. -10 and 70 on the right side are artifacts of seaborn and have no meaning; all values in the graph are between 0 and 60.

better for HFF since HRF did not complete a single problem in that domain, but even then, it only obtained minor success.

We see the biggest disparity in the logistics and tpp domains. HRF is completely unable to solve the logistics domain, while HFF managed to comfortably complete them. The first interesting result is that HRF managed to complete a much larger amount of the tpp domain compared to HFF. This result proved rather anomalous, and we have been unable to determine its cause.

The ensemble HCF did not produce the desired results; instead, the combined search performed worse on logistics than HFF and worse on tpp than HRF, with no significant changes on other domains. Instead of

6.1.1 Difficulties

After we studied the different benchmarks more closely, we determined that $del(o) \subset pre(o)$ held for most operators $o \in \mathcal{O}$. This result is not very interesting, as our HRF is simply a less expressive version of the original relaxation HFF. If we instead look at the operators where $del(o)/pre(o) \neq \varnothing$, we discover that HRF is unable to find a plan if it requires any such operator for the solution. We resolve this problem by normalizing the action, in which case all operators become SR-OPs

6.2 Hybrid Search

Initial results for Hybrid Search were more promising than they appear here. A user error had us initially consider all planner outputs that produced a plan as successful. We did

not catch on until very late that we would also provide partial plans whenever a problem terminated in advance. Such partial plans did not exist when the planning terminated during pre-processing, to which finding the intermediate state for tail reasoning counted. This result led us to falsely assume our results were more productive than they were since they lined up with our expectations of what the results should have been. Furthermore, the modifications we did to our algorithms improved those results, which led to us not catching the error until very late in the process.

We will still present our findings, but all insights gained need to be taken carefully. For each table, pyp-HFF and pyp-SAT are the Fast Forward and Satisfiability implementations of pyperplan, while HFF and SAT are our native algorithms. NAIVE is the basic Hybrid Search algorithm 3, while MAX and MUTEX are the variants described in 4.2.2.

Planner	# Solution	time	time (sol)	plan	plan (sol)	# no plan
pyp-HFF	4	11.84	14.79	34.40	43.00	0
pyp-SAT	0	0.00	0.00	0.00	0.00	0
HFF	0	3.38	0.00	0.00	0.00	5
SAT	0	5.83	0.00	0.00	0.00	1
NAIVE	0	22.53	0.00	0.60	0.00	1
MAX	0	1.96	0.00	0.20	0.00	0
MUTEX	0	22.58	0.00	0.00	0.00	3

Table 6.2: Aggregated grid results. Time and time (sol) are average planning times. The latter is only for solved problems. Plan and plan (sol) are the average length of the (partial) plans. The latter is only for solved problems. # no plan indicates how many solutions had a plan length of zero.

Grid 6.2 was a complex problem for all our algorithms, including Hybrid Search and plain SAT. Only the pyperplan implementation of HFF obtained any solutions, which may indicate a problem with our native implementation of HFF. Of note is that the Hybrid Search approaches NAIVE and MAX both yielded partial plans. In the latter case, those partial plans even existed for every problem, indicating that MAX improved NAIVE in selecting a more suitable intermediary state. That NAIVE had an instance with no plan indicates that the intermediary state was unreachable. The failure of MUTEX is more surprising since the additional constraints should have resulted in a more accurate solution instead of fewer solutions overall.

Results are better on the gripper domain 6.3, where both our algorithms remain competitive with those of pyperplan, even if they took longer to find solutions, the plans created with our HFF are considerably shorter. We can again see that MAX is a marked improvement over NAIVE, even if it still took longer to complete than either of its constituent algorithms. It also remains with a very short plan length, especially compared to HFF. MUTEX is again underwhelming, and its continued underperformance perhaps indicates a programming error we could not catch.

Planner	# Solution	time	time (sol)	plan	plan (sol)	# no plan
pyp-HFF	20	0.27	0.27	88.35	89.00	0
pyp-SAT	1	0.01	0.14	0.55	11.00	0
HFF	20	3.02	3.02	70.70	70.70	0
SAT	1	24.03	0.78	0.55	11.00	16
NAIVE	0	29.53	0.00	0.85	0.00	18
MAX	2	23.59	7.81	1.00	10.00	15
MUTEX	0	20.78	0.00	0.00	0.00	20

Table 6.3: Aggregated gripper results. Time and time (sol) are average planning times. The latter is only for solved problems. Plan and plan (sol) are the average length of the (partial) plans. The latter is only for solved problems. # no plan indicates how many solutions had a plan length of zero.

Planner	# Solution	time	time (sol)	plan	plan (sol)	# no plan
pyp-HFF	28	2.10	2.10	43.32	43.61	0
pyp-SAT	6	0.49	2.28	3.32	15.50	0
HFF	28	1.40	1.40	42.36	42.36	0
SAT	3	22.93	13.62	1.32	12.33	21
NAIVE	0	16.49	0.00	11.32	0.00	14
MAX	5	1.53	3.79	4.57	21.40	0
MUTEX	0	15.71	0.00	9.93	0.00	13

Table 6.4: Aggregated logistics results. Time and time (sol) are average planning times. The latter is only for solved problems. Plan and plan (sol) are the average length of the (partial) plans. The latter is only for solved problems. # no plan indicates how many solutions had a plan length of zero.

Results in the logistics domain 6.4 are similar to Gripper. We can again see that MAX outperforms all other variants of tail reasoning and remains highly competitive with plan lengths. It is consistently shorter than either version of pure Fast Forward Search.

Planner	# Solution	time	time (sol)	plan	plan (sol)	# no plan
pyp-HFF	21	1.48	2.47	19.09	33.33	0
pyp-SAT	13	0.73	1.97	5.14	13.85	0
HFF	18	9.71	1.74	30.29	33.00	0
SAT	11	17.39	9.42	3.94	12.55	21
NAIVE	0	17.05	0.00	6.14	0.00	21
MAX	1	19.27	0.10	4.77	6.00	20
MUTEX	0	19.70	0.00	7.06	0.00	21

Table 6.5: Aggregated blocksworld results. Time and time (sol) are average planning times. The latter is only for solved problems. Plan and plan (sol) are the average length of the (partial) plans. The latter is only for solved problems. # no plan indicates how many solutions had a plan length of zero.

We initially tested our algorithms in Blocksworld 6.5 since it is a prime candidate for a tail-heavy problem. The hope was for hybrid search to find an intermediate state roughly equivalent to

placing all blocks on the table, then having the forward search portion put them there from their initial stacks. It appears that our initial assumption was faulty, however, since both HFF and SAT performed exceptionally well in the domain. The Hybrid Search that should have dominated here could not solve more than a single problem, which puts our methodology into question.

Planner	# Solution	time	time (sol)	plan	plan (sol)	# no plan
pyp-HFF	3	5.78	10.99	5.63	56.33	22
pyp-SAT	0	0.00	0.00	0.00	0.00	0
HFF	4	6.65	8.68	36.57	71.25	2
SAT	0	19.63	0.00	0.00	0.00	25
NAIVE	0	11.41	0.00	29.13	0.00	8
MAX	0	9.18	0.00	12.83	0.00	3
MUTEX	0	24.77	0.00	6.30	0.00	22

Table 6.6: Aggregated sokoban results. Time and time (sol) are average planning times. The latter is only for solved problems. Plan and plan (sol) are the average length of the (partial) plans. The latter is only for solved problems. # no plan indicates how many solutions had a plan length of zero.

Planner	# Solution	time	time (sol)	plan	plan (sol)	# no plan
pyp-HFF	0	0.00	0.00	0.00	0.00	0
pyp-SAT	0	0.00	0.00	0.00	0.00	0
HFF	0	0.37	0.00	0.00	0.00	20
SAT	0	17.48	0.00	0.00	0.00	12
NAIVE	0	25.23	0.00	0.00	0.00	15
MAX	0	19.40	0.00	0.00	0.00	11
MUTEX	0	0.00	0.00	0.00	0.00	0

Table 6.7: Aggregated child snack results. Time and time (sol) are average planning times. The latter is only for solved problems. Plan and plan (sol) are the average length of the (partial) plans. The latter is only for solved problems. # no plan indicates how many solutions had a plan length of zero.

The results for Sokoban and Child Snack do not add new insights and have been included here for completion's sake. Child Snack was too difficult a problem for any of our planners, and it was solved only by HFF.

The results for TPP were similarly underwhelming. Only the implementations of pyperplan managed to obtain any solutions, though the time it took SAT to complete appears dubious as well.

Finally, we have the solutions for our new tunnel domain 6.9. The performance of HFF is in line with our expectations; after all, we designed the domain with the express intent to exemplify its shortcomings. That none of the other implementations could obtain even a partial plan is confusing and worrying.

Planner	# Solution	time	time (sol)	plan	plan (sol)	# no plan
pyp-HFF	8	0.69	2.60	5.20	24.25	0
pyp-SAT	4	0.01	0.08	1.27	9.50	0
HFF	0	0.65	0.00	0.00	0.00	20
SAT	0	11.83	0.00	0.00	0.00	21
NAIVE	0	19.22	0.00	0.83	0.00	19
MAX	0	0.03	0.00	0.40	0.00	0
MUTEX	0	0.00	0.00	0.00	0.00	0

Table 6.8: Aggregated tpp results. Time and time (sol) are average planning times. The latter is only for solved problems. Plan and plan (sol) are the average length of the (partial) plans. The latter is only for solved problems. # no plan indicates how many solutions had a plan length of zero.

Planner	# Solution	time	time (sol)	plan	plan (sol)	# no plan
pyp-HFF	4	4.76	19.32	5.80	29.00	1
pyp-SAT	0	0.00	0.00	0.00	0.00	0
HFF	0	0.38	0.00	0.00	0.00	20
SAT	0	21.57	0.00	0.00	0.00	20
NAIVE	0	23.17	0.00	0.00	0.00	20
MAX	0	21.69	0.00	0.00	0.00	20
MUTEX	0	24.97	0.00	0.00	0.00	20

Table 6.9: Aggregated tunnel results. Time and time (sol) are average planning times. The latter is only for solved problems. Plan and plan (sol) are the average length of the (partial) plans. The latter is only for solved problems. # no plan indicates how many solutions had a plan length of zero.

Overall, our tests' results were not very expressive. Whether due to user error, hardware limitations, or even design problems, none of our algorithms achieved sensible solutions.

7 Conclusion

7.1 Insights Gained

Despite the overall lackluster results, the theory behind our approach was sound. The newly designed tunnel domain performed as expected, being able to mostly stop HFF from solving it. We also reaffirmed that HFF was state of the art for a long time, with its performance on many older domains still outstanding. However, we showed empirically that it struggles with newer and more complex problems, just like we surmised.

Our research into Hybrid Search and Reverse Forward Search did not bear fruit, which is disheartening, but some of our implementations are salvageable. The theory and design of our algorithms are sound, and there are multiple additional avenues with which a Hybrid Search could still be optimized and realized.

That said, our research generally turned out to be a failure. Calculating an intermediary state was more complicated than initially expected, and even though we improved the selection of a valid intermediary state, the choice remained somewhat random and non-viable. We could also see that our algorithms utterly failed on problems where goals were not completable in a short span towards the end. In cases like blocksworld, where we could infer the existence of such an intermediary state, Hybrid Search performed better than on domains where such a state is not readily available, like the gripper domain. In cases where such a state is unavailable, a Hybrid Search is akin to satisfiability on the reversed problem, i.e., a regression search with SAT. Our results reaffirm that satisfiability is not a viable alternative for HFF in solving whole problems, so failure in those areas was expected.

It stands to debate whether an approach with a different backward planner would have been more effective. SAT seemed to be a good fit because of the inherent advantages satisfiability carries regarding invariants, but it proved too cumbersome. An entirely different approach based more heavily on regression search may have been the superior option and is something we would investigate if we were to continue this line of research.

7.2 Limitations

Ultimately, Hybrid Search does not attempt to figure out a goal topology directly; even in the best-case scenario, it merely aggregates goals near the end. That makes Hybrid Search brittle in

all domains, where subgoals must be solved sequentially. The further each goal can be partially solved, the more effective Hybrid Search becomes.

Another inherent limitation of Hybrid Search is that we require a problem to be neatly divisible into a head and tail. It is entirely possible to design domains that specifically counter this behavior. An example would be a domain akin to our tunnel domain, but after every block is in the tunnel, a *validate*(?*block*, ?*end*) action needs to be performed on every block, where ?*end* is a variable for some predicate that is only true after every block is in the tunnel. The action would add some *goal*(?*block*) predicate, and a problem's goal would naturally be *goal*(?*block*) for every block instead of a non-free tunnel. With such a modification, the closest (legal) state our hybrid approach would be able to find is precisely a filled tunnel, which is equivalent to the unmodified tunnel domain.

Combating this behavior requires some modification to select the intermediary state. One idea proposed was to recompute the intermediary state at regular intervals whenever the problem takes too long to complete or when HFF is stuck with some low h-value. The recomputed intermediary state must contain no facts about the goals or previous intermediary states. Whether such a modification is feasible or even desirable is not something we were able to test ourselves.

7.3 Summary of Contributions

This thesis explored different transformations for both satisfiability planning, HFF, and combinations between the two that, despite not being entirely successful, provide interesting results nonetheless. Modifying HFF yielded mixed results but with the performance improvement on TPP, which is a valuable problem to solve, not entirely without merit, despite us being unable to figure out the reason.

Our foray into the bidirectional combination of satisfiability and HFF may not have been able to beat HFF standing on its own. However, the modifications to selecting an intermediary state improved the naive approach. Perhaps with some more modifications, this approach may become competitive.

Finally, we provided the new tunnel domain with a detailed analysis of its complexity and invariants. We further provided invariants for many commonly used planning domains that can be used to conduct further research with them.

The source code for all our work is made freely available for anyone who wants to test things out for themselves and contains a way to generate problems in the new tunnel domains automatically.

7.4 Future Research Direction

From this point, we see several possible directions to take this. Given the results, abandoning the approach altogether is a possibility. On the other hand, the theory behind Hybrid Search remains solid, so we should consider what went wrong with our implementations. Many results indicate an implementation error, especially with the performance of HFF compared to pyperplan and the disappointing results of MUTEX.

Another thing to consider is a different modification for reaching and selecting the intermediary state. As we posited in 4, having HFF look for any of the possible intermediary states remains a possibility with quite some merit, so long as future research determines an efficient way to test for an explosive number of goal sets.

There is the opportunity for a dynamic approach that regularly recomputes the intermediate state, which solves the problem with not perfectly clustered goals. Similarly, a different approach could be used to determine when tail reasoning would be advantageous in the first place. This differentiation could be done by determining (heuristically) if goals can be solved in parallel or not. Hybrid Search obtained slightly better results when goals could be worked towards and only had to be completed sequentially towards the tail.

A Appendix

A.1 Raw Results

All results for every domain. T indicates the problem was succesfully solved, while F indicates a failure. The first number is the time it took to find a solution or until the planner was succesfully interrupted. A -1 indicates that pyperplan had to be shut down, a -2 indicates the problem could not be grounded due to memory issues. A -3 indicates that the test was not implemented. The second number is the length of the plan. If that number is zero, no plan was found. A number greater than zero with an F indicates a partial plan could be found.

#	Pyp-HFF	Pyp-SAT	HFF	SAT	Naive	Max	Mutex
01	F: 0.68, 0	F: -1.00, 0	F: 0.29, 26	F: 68.82, 0	F: 0.26, 36	F: 0.66, 36	F: 3.67, 32
02	F: 0.15, 0	F: -1.00, 0	F: 0.07, 13	F: 73.99, 0	F: 0.10, 15	F: 0.20, 23	F: 0.95, 26
03	T: 0.10, 69	F: -1.00, 0	T: 0.06, 51	F: 67.47, 0	F: 0.10, 44	F: 0.09, 44	F: 0.27, 44
04	F: 0.90, 0	F: -1.00, 0	F: 0.77, 112	F: 67.55, 0	F: 1.38, 68	F: 4.02, 50	F: 27.46, 0
05	T: 2.20, 53	F: -1.00, 0	T: 1.50, 53	F: 23.09, 0	F: 21.78, 0	F: 95.84, 0	F: 82.99, 0
06	F: 0.22, 0	F: -1.00, 0	F: 0.11, 14	F: 25.15, 0	F: 0.09, 21	F: 0.12, 15	F: 0.55, 35
07	F: -1.00, 0	F: -1.00, 0	T: 2.28, 89	F: 24.88, 0	F: 24.34, 138	F: 8.21, 38	F: 35.16, 0
08	F: 5.78, 0	F: -1.00, 0	F: 14.74, 33	F: 23.93, 0	F: 20.07, 20	F: 24.50, 0	F: 26.30, 0
09	F: 3.59, 0	F: -1.00, 0	T: 30.86, 92	F: 21.43, 0	F: 0.61, 31	F: 408.53, 18	F: 14.57, 52
10	F: 2.97, 0	F: -1.00, 0	F: 0.77, 8	F: 24.37, 0	F: 8.09, 12	F: 159.37, 5	F: 27.29, 0
11	F: 14.01, 0	F: -1.00, 0	F: 2.18, 74	F: 22.08, 0	F: 20.95, 52	F: 13.82, 83	F: 31.94, 0
12	F: -1.00, 0	F: -1.00, 0	F: 3.22, 54	F: 21.54, 0	F: 2.79, 33	F: 33.32, 4	F: 31.16, 0
13	F: -1.00, 0	F: -1.00, 0	F: 42.28, 25	F: 21.70, 0	F: 6.54, 36	F: 156.94, 5	F: 25.86, 0
14	F: 1.11, 0	F: -1.00, 0	F: 0.74, 80	F: 24.11, 0	F: 2.41, 89	F: 1.75, 13	F: 24.97, 0
15	F: 14.89, 0	F: -1.00, 0	F: 1.78, 29	F: 24.33, 0	F: 6.47, 32	F: 25.59, 4	F: 65.24, 0
16	F: 6.36, 0	F: -1.00, 0	F: 1.97, 53	F: 25.46, 0	F: 12.17, 81	F: 67.91, 5	F: 33.01, 0
17	F: 5.56, 0	F: -1.00, 0	F: 1.33, 21	F: 25.12, 0	F: 3.81, 12	F: 16.64, 23	F: 42.36, 0
18	F: 7.40, 0	F: -1.00, 0	F: 64.69, 272	F: 20.50, 0	F: 22.00, 0	F: 22.97, 0	F: 40.13, 0
19	F: 30.21, 0	F: -1.00, 0	F: 0.70, 24	F: 20.32, 0	F: 24.60, 0	F: 36.41, 0	F: 23.44, 0
20	F: -1.00, 0	F: -1.00, 0	F: 33.55, 0	F: 103.46, 0	F: 24.66, 0	F: 134.30, 0	F: 39.76, 0
21	F: 18.17, 0	F: -1.00, 0	F: 75.34, 39	F: 27.76, 0	F: 13.46, 73	F: 48.35, 48	F: 206.83, 54
22	F: 3.90, 0	F: -1.00, 0	F: 1.00, 32	F: 20.87, 0	F: 3.28, 34	F: 38.81, 4	F: 31.32, 0
23	F: 1.95, 0	F: -1.00, 0	F: 0.54, 15	F: 21.55, 0	F: 9.07, 18	F: 938.15, 5	F: 40.04, 0
24	F: 3.39, 0	F: -1.00, 0	F: 1.16, 34	F: 20.66, 0	F: 161.45, 76	F: -2.00, 0	F: 48.88, 0
25	F: 2.30, 0	F: -1.00, 0	F: 0.35, 18	F: 21.90, 0	F: 25.98, 0	F: -2.00, 0	F: 22.61, 0
26	F: 3.11, 0	F: -1.00, 0	F: 0.72, 39	F: 23.41, 0	F: 10.64, 29	F: -2.00, 0	F: 22.00, 0
27	F: 3.31, 0	F: -1.00, 0	F: 1.02, 49	F: 21.20, 0	F: 30.38, 0	F: -2.00, 0	F: 39.33, 0
28	F: 10.33, 0	F: -1.00, 0	F: 2.15, 40	F: 26.92, 0	F: 113.65, 0	F: -2.00, 0	F: 30.51, 0
29	T: 30.66, 47	F: -1.00, 0	F: 2.85, 19	F: 24.96, 0	F: 22.73, 0	F: -2.00, 0	F: 39.26, 0
30	F: -1.00, 0	F: -1.00, 0	F: 50.39, 0	F: 31.53, 0	F: 23.66, 0	F: -2.00, 0	F: 40.42, 0

Table A.1: All results on the sokoban domain. First value determine if the planner solved (T) or failed (F) the problem. Second value is the time until the planner stopped or was stopped in seconds. Last value is the length of the (partial) plan

#	Pyp-HFF	Pyp-SAT	HFF	SAT	Naive	Max	Mutex
01	T: 0.06, 14	F: -1.00, 0	F: 0.31, 0	F: 74.97, 0	F: 15.32, 1	F: 9.79, 1	F: 29.54, 0
02	T: 0.47, 30	F: -1.00, 0	F: 0.76, 0	F: 102.56, 0	F: 55.03, 2	F: 103.65, 2	F: 43.33, 0
03	T: 22.29, 71	F: -1.00, 0	F: 1.96, 0	F: 92.71, 0	F: 42.29, 0	F: 98.11, 0	F: 40.04, 0
04	T: 36.36, 57	F: -1.00, 0	F: 4.37, 0	F: 120.64, 0	F: 89.39, 0	F: 129.20, 0	F: 96.45, 0
05	F: -1.00, 0	F: -1.00, 0	F: 9.52, 0	F: 29.16, 0	F: 188.75, 0	F: 304.63, 0	F: 220.40, 0

Table A.2: All results on the grid domain. First value determine if the planner solved (T) or failed (F) the problem. Second value is the time until the planner stopped or was stopped in seconds. Last value is the length of the (partial) plan

#	Pyp-HFF	Pyp-SAT	HFF	SAT	Naive	Max	Mutex
01	T: 0.00, 13	T: 0.14, 11	T: 0.01, 13	T: 0.78, 11	F: 0.97, 7	T: 0.50, 8	F: 20.16, 0
02	T: 0.01, 21	F: -1.00, 0	T: 0.01, 17	F: 80.46, 0	F: 11.20, 10	T: 15.12, 12	F: 20.39, 0
03	T: 0.01, 29	F: -1.00, 0	T: 0.03, 23	F: 142.36, 0	F: 27.74, 0	F: 70.76, 0	F: 20.18, 0
04	T: 0.02, 37	F: -1.00, 0	T: 0.08, 29	F: 62.67, 0	F: 21.45, 0	F: 64.02, 0	F: 20.08, 0
05	T: 0.03, 45	F: -1.00, 0	T: 0.12, 35	F: 24.66, 0	F: 40.60, 0	F: 83.89, 0	F: 20.49, 0
06	T: 0.04, 53	F: -1.00, 0	T: 0.12, 41	F: 20.85, 0	F: 25.66, 0	F: 28.69, 0	F: 20.70, 0
07	T: 0.06, 61	F: -1.00, 0	T: 0.42, 47	F: 43.61, 0	F: 41.58, 0	F: 20.36, 0	F: 20.58, 0
08	T: 0.08, 69	F: -1.00, 0	T: 0.85, 55	F: 22.92, 0	F: 26.65, 0	F: 26.14, 0	F: 20.52, 0
09	T: 0.10, 77	F: -1.00, 0	T: 0.68, 59	F: 32.94, 0	F: 20.20, 0	F: 35.48, 0	F: 20.34, 0
10	T: 0.14, 85	F: -1.00, 0	T: 1.48, 69	F: 38.70, 0	F: 26.22, 0	F: 23.32, 0	F: 21.36, 0
11	T: 0.18, 93	F: -1.00, 0	T: 0.36, 71	F: 42.19, 0	F: 38.73, 0	F: 23.49, 0	F: 21.26, 0
12	T: 0.23, 101	F: -1.00, 0	T: 2.31, 77	F: 22.84, 0	F: 29.75, 0	F: 28.60, 0	F: 20.54, 0
13	T: 0.29, 109	F: -1.00, 0	T: 4.48, 85	F: 24.79, 0	F: 52.39, 0	F: 44.00, 0	F: 20.67, 0
14	T: 0.35, 117	F: -1.00, 0	T: 4.37, 103	F: 30.50, 0	F: 24.84, 0	F: 20.80, 0	F: 21.82, 0
15	T: 0.42, 125	F: -1.00, 0	T: 6.51, 95	F: 39.57, 0	F: 27.53, 0	F: 21.95, 0	F: 20.97, 0
16	T: 0.51, 133	F: -1.00, 0	T: 4.80, 101	F: 21.71, 0	F: 38.66, 0	F: 31.80, 0	F: 21.40, 0
17	T: 0.59, 141	F: -1.00, 0	T: 10.09, 125	F: 20.68, 0	F: 43.95, 0	F: 33.81, 0	F: 20.16, 0
18	T: 0.69, 149	F: -1.00, 0	T: 1.66, 115	F: 24.86, 0	F: 42.70, 0	F: 33.29, 0	F: 22.28, 0
19	T: 0.80, 157	F: -1.00, 0	T: 8.94, 119	F: 36.71, 0	F: 21.99, 0	F: 40.97, 0	F: 20.18, 0
20	T: 0.93, 165	F: -1.00, 0	T: 13.07, 135	F: 32.20, 0	F: 27.73, 0	F: 43.40, 0	F: 21.59, 0

Table A.3: All results on the gripper domain. First value determine if the planner solved (T) or failed (F) the problem. Second value is the time until the planner stopped or was stopped in seconds. Last value is the length of the (partial) plan

#	Pyp-HFF	Pyp-SAT	HFF	SAT	Naive	Max	Mutex
01	T: 1.25, 19	F: -1.00, 0	F: 0.04, 0	F: 20.40, 0	F: 20.86, 0	F: 21.13, 0	F: 20.61, 0
02	T: 8.74, 17	F: -1.00, 0	F: 0.12, 0	F: 20.40, 0	F: 21.11, 0	F: 20.12, 0	F: 23.97, 0
03	F: -1.00, 0	F: -1.00, 0	F: 0.30, 0	F: 21.99, 0	F: 22.49, 0	F: 23.37, 0	F: 20.60, 0
04	F: -1.00, 0	F: -1.00, 0	F: 0.64, 0	F: 23.22, 0	F: 23.80, 0	F: 21.68, 0	F: 29.29, 0
05	T: 24.89, 37	F: -1.00, 0	F: 0.05, 0	F: 20.00, 0	F: 22.04, 0	F: 20.47, 0	F: 20.60, 0
06	F: -1.00, 0	F: -1.00, 0	F: 0.15, 0	F: 20.27, 0	F: 21.71, 0	F: 20.57, 0	F: 21.64, 0
07	F: -1.00, 0	F: -1.00, 0	F: 0.36, 0	F: 20.13, 0	F: 23.33, 0	F: 22.00, 0	F: 21.80, 0
08	F: -1.00, 0	F: -1.00, 0	F: 0.75, 0	F: 22.31, 0	F: 22.74, 0	F: 21.44, 0	F: 21.69, 0
09	T: 42.39, 43	F: -1.00, 0	F: 0.06, 0	F: 20.83, 0	F: 20.54, 0	F: 21.65, 0	F: 21.80, 0
10	F: -1.00, 0	F: -1.00, 0	F: 0.18, 0	F: 20.73, 0	F: 22.10, 0	F: 20.09, 0	F: 22.56, 0
11	F: -1.00, 0	F: -1.00, 0	F: 0.42, 0	F: 20.13, 0	F: 20.65, 0	F: 20.86, 0	F: 24.22, 0
12	F: -1.00, 0	F: -1.00, 0	F: 0.84, 0	F: 21.84, 0	F: 20.31, 0	F: 27.21, 0	F: 30.74, 0
13	F: -1.00, 0	F: -1.00, 0	F: 0.08, 0	F: 20.24, 0	F: 22.20, 0	F: 22.69, 0	F: 21.37, 0
14	F: -1.00, 0	F: -1.00, 0	F: 0.22, 0	F: 20.76, 0	F: 22.70, 0	F: 21.53, 0	F: 24.22, 0
15	F: -1.00, 0	F: -1.00, 0	F: 0.48, 0	F: 22.83, 0	F: 23.76, 0	F: 21.23, 0	F: 25.71, 0
16	F: -1.00, 0	F: -1.00, 0	F: 0.97, 0	F: 22.63, 0	F: 29.61, 0	F: 23.72, 0	F: 38.11, 0
17	F: -1.00, 0	F: -1.00, 0	F: 0.09, 0	F: 20.22, 0	F: 20.64, 0	F: 22.09, 0	F: 25.61, 0
18	F: 17.87, 0	F: -1.00, 0	F: 0.26, 0	F: 22.68, 0	F: 23.38, 0	F: 20.81, 0	F: 28.04, 0
19	F: -1.00, 0	F: -1.00, 0	F: 0.54, 0	F: 27.23, 0	F: 29.40, 0	F: 20.67, 0	F: 27.00, 0
20	F: -1.00, 0	F: -1.00, 0	F: 1.03, 0	F: 22.47, 0	F: 30.02, 0	F: 20.41, 0	F: 29.87, 0

Table A.4: All results on the tunnel domain. First value determine if the planner solved (T) or failed (F) the problem. Second value is the time until the planner stopped or was stopped in seconds. Last value is the length of the (partial) plan

#	Pyp-HFF	Pyp-SAT	HFF	SAT	Naive	Max	Mutex
01	T: 0.01, 20	T: 3.16, 20	T: 0.03, 20	F: 69.94, 0	F: 0.23, 15	F: 1.14, 4	F: 0.27, 20
02	T: 0.01, 19	T: 2.88, 19	T: 0.05, 19	F: 63.44, 0	F: 0.25, 22	F: 1.64, 4	F: 0.24, 18
03	T: 0.01, 15	T: 1.80, 15	T: 0.03, 15	T: 23.98, 15	F: 0.06, 4	T: 0.17, 15	F: 0.07, 7
04	T: 0.01, 27	F: -1.00, 0	T: 0.05, 27	F: 76.13, 0	F: 0.38, 21	F: 9.70, 5	F: 0.46, 22
05	T: 0.01, 17	T: 2.82, 17	T: 0.04, 17	F: 60.64, 0	F: 0.23, 18	T: 3.04, 23	F: 0.29, 26
06	T: 0.00, 8	T: 0.67, 8	T: 0.01, 8	T: 3.86, 8	F: 0.15, 18	F: 0.28, 3	F: 0.13, 9
07	T: 0.01, 25	F: -1.00, 0	T: 0.06, 25	F: 23.25, 0	F: 0.44, 30	T: 10.69, 28	F: 0.46, 23
08	T: 0.01, 14	T: 2.33, 14	T: 0.03, 14	T: 13.02, 14	F: 0.12, 11	T: 2.42, 16	F: 0.16, 20
09	T: 0.01, 25	F: -1.00, 0	T: 0.07, 26	F: 20.17, 0	F: 0.38, 19	F: 11.04, 5	F: 0.53, 24
10	T: 0.01, 24	F: -1.00, 0	T: 0.04, 24	F: 21.64, 0	F: 0.24, 21	T: 2.64, 25	F: 0.25, 24
11	T: 0.06, 37	F: -1.00, 0	T: 0.26, 36	F: 27.49, 0	F: 7.79, 28	F: -2.00, 0	F: 121.18, 0
12	T: 0.08, 45	F: -1.00, 0	T: 0.27, 44	F: 31.62, 0	F: 25.74, 0	F: -2.00, 0	F: 516.81, 0
13	T: 0.08, 35	F: -1.00, 0	T: 0.24, 32	F: 31.64, 0	F: 9.44, 34	F: -2.00, 0	F: 12.25, 28
14	T: 0.11, 47	F: -1.00, 0	T: 0.37, 46	F: 25.07, 0	F: 21.85, 0	F: -2.00, 0	F: 22.24, 0
15	T: 0.10, 38	F: -1.00, 0	T: 0.26, 38	F: 26.54, 0	F: 8.67, 38	F: -2.00, 0	F: 11.66, 28
16	T: 0.04, 31	F: -1.00, 0	T: 0.17, 31	F: 21.15, 0	F: 8.92, 38	F: -2.00, 0	F: 10.78, 29
17	T: 0.30, 48	F: -1.00, 0	T: 1.33, 48	F: 24.08, 0	F: 38.02, 0	F: -2.00, 0	F: 33.79, 0
18	T: 0.19, 44	F: -1.00, 0	T: 0.82, 43	F: 24.82, 0	F: 32.54, 0	F: -2.00, 0	F: 31.72, 0
19	T: 0.33, 52	F: -1.00, 0	T: 0.84, 49	F: 24.69, 0	F: 35.53, 0	F: -2.00, 0	F: 39.15, 0
20	T: 0.57, 65	F: -1.00, 0	T: 1.42, 66	F: 24.16, 0	F: 35.63, 0	F: -2.00, 0	F: 40.14, 0
21	T: 0.29, 44	F: -1.00, 0	T: 0.52, 43	F: 24.23, 0	F: 28.03, 0	F: -2.00, 0	F: 33.92, 0
22	T: 0.88, 76	F: -1.00, 0	T: 2.08, 73	F: 24.40, 0	F: 49.64, 0	F: -2.00, 0	F: 28.31, 0
23	T: 15.91, 90	F: -1.00, 0	T: 4.96, 82	F: 37.13, 0	F: 25.25, 0	F: -2.00, 0	F: 29.18, 0
24	T: 1.18, 65	F: -1.00, 0	T: 6.78, 68	F: 37.98, 0	F: 26.78, 0	F: -2.00, 0	F: 29.77, 0
25	T: 4.65, 68	F: -1.00, 0	T: 4.30, 65	F: 37.65, 0	F: 26.16, 0	F: -2.00, 0	F: 28.02, 0
26	T: 16.38, 84	F: -1.00, 0	T: 3.21, 76	F: 37.64, 0	F: 26.69, 0	F: -2.00, 0	F: 29.75, 0
27	T: 15.16, 84	F: -1.00, 0	T: 6.07, 80	F: 37.95, 0	F: 26.26, 0	F: -2.00, 0	F: 28.88, 0
28	T: 2.37, 74	F: -1.00, 0	T: 4.88, 71	F: 37.79, 0	F: 26.41, 0	F: -2.00, 0	F: 27.49, 0

Table A.5: All results on the logistics domain. First value determine if the planner solved (T) or failed (F) the problem. Second value is the time until the planner stopped or was stopped in seconds. Last value is the length of the (partial) plan

#	Pyp-HFF	Pyp-SAT	HFF	SAT	Naive	Max	Mutex
01	T: 0.01, 14	T: 0.08, 6	T: 0.02, 14	T: 0.07, 6	F: 0.06, 9	T: 0.10, 6	F: 0.13, 9
02	T: 0.00, 10	T: 0.19, 10	T: 0.01, 10	T: 0.14, 10	F: 0.03, 5	F: 0.05, 5	F: 0.06, 5
03	T: 0.00, 6	T: 0.07, 6	T: 0.01, 10	T: 0.06, 6	F: 0.05, 5	F: 0.11, 5	F: 0.14, 15
04	T: 0.00, 16	T: 0.54, 12	T: 0.02, 18	T: 0.95, 12	F: 0.23, 13	F: 0.48, 7	F: 0.77, 14
05	T: 0.04, 26	T: 0.39, 10	T: 0.12, 24	T: 0.48, 10	F: 0.22, 11	F: 0.61, 15	F: 0.61, 11
06	T: 0.02, 32	T: 0.94, 16	T: 0.05, 26	T: 3.07, 16	F: 0.46, 15	F: 1.16, 21	F: 1.11, 21
07	T: 0.01, 16	T: 0.98, 12	T: 0.07, 20	T: 3.66, 12	F: 3.75, 17	F: 8.99, 11	F: 11.96, 17
08	T: 0.07, 26	T: 0.70, 10	T: 0.03, 14	T: 2.80, 10	F: 4.10, 13	F: 7.82, 11	F: 12.16, 21
09	T: 0.14, 36	T: 2.66, 20	T: 1.83, 34	T: 25.13, 20	F: 5.87, 20	F: 9.59, 15	F: 18.91, 24
10	T: 0.02, 26	T: 4.49, 20	F: 4.27, 16	T: 38.24, 20	F: 7.92, 27	F: 33.93, 13	F: 14.84, 32
11	T: 0.07, 30	T: 5.41, 22	T: 0.26, 36	F: 76.25, 0	F: 4.64, 23	F: 11.12, 15	F: 12.98, 22
12	T: 6.55, 50	T: 4.47, 20	F: 4.82, 30	F: 68.07, 0	F: 5.05, 21	F: 9.23, 19	F: 11.86, 25
13	T: 34.72, 50	F: -1.00, 0	T: 3.33, 34	F: 78.34, 0	F: 13.78, 25	F: 36.92, 11	F: 26.12, 18
14	T: 0.21, 32	F: -1.00, 0	T: 0.69, 32	F: 22.88, 0	F: 7.13, 11	F: 24.46, 13	F: 17.50, 13
15	T: 0.17, 32	T: 4.72, 16	F: 6.77, 18	T: 29.03, 16	F: 20.30, 0	F: 84.65, 13	F: 23.84, 0
16	F: -1.00, 0	F: -1.00, 0	F: 9.75, 34	F: 21.59, 0	F: 24.86, 0	F: 27.95, 0	F: 28.56, 0
17	T: 0.19, 44	F: -1.00, 0	T: 0.55, 46	F: 25.63, 0	F: 22.94, 0	F: 29.55, 0	F: 25.99, 0
18	T: 1.05, 44	F: -1.00, 0	T: 0.81, 44	F: 25.61, 0	F: 23.19, 0	F: 33.41, 0	F: 29.90, 0
19	T: 0.32, 48	F: -1.00, 0	T: 7.36, 58	F: 22.17, 0	F: 23.07, 0	F: 34.33, 0	F: 21.07, 0
20	F: -1.00, 0	F: -1.00, 0	F: 17.45, 38	F: 28.83, 0	F: 24.36, 0	F: 23.13, 0	F: 27.38, 0
21	T: 7.40, 60	F: -1.00, 0	T: 9.00, 56	F: 21.31, 0	F: 24.30, 0	F: 20.27, 0	F: 30.39, 0
22	F: -1.00, 0	F: -1.00, 0	F: 13.26, 12	F: 20.07, 0	F: 29.90, 0	F: 29.42, 0	F: 32.12, 0
23	F: -1.00, 0	F: -1.00, 0	F: 16.52, 40	F: 25.81, 0	F: 24.19, 0	F: 31.21, 0	F: 29.77, 0
24	F: -1.00, 0	F: -1.00, 0	T: 5.42, 60	F: 21.58, 0	F: 23.75, 0	F: 28.06, 0	F: 30.50, 0
25	T: 0.33, 48	F: -1.00, 0	F: 30.49, 46	F: 23.22, 0	F: 23.24, 0	F: 24.79, 0	F: 26.36, 0
26	T: 0.48, 54	F: -1.00, 0	T: 1.81, 58	F: 21.98, 0	F: 23.48, 0	F: 28.37, 0	F: 28.96, 0
27	F: -1.00, 0	F: -1.00, 0	F: 33.76, 48	F: 25.31, 0	F: 27.03, 0	F: 21.29, 0	F: 20.20, 0
28	F: -1.00, 0	F: -1.00, 0	F: 20.23, 34	F: 21.12, 0	F: 21.40, 0	F: 22.09, 0	F: 23.66, 0
29	F: -1.00, 0	F: -1.00, 0	F: 28.24, 33	F: 26.92, 0	F: 30.18, 0	F: 26.92, 0	F: 30.14, 0
30	F: -1.00, 0	F: -1.00, 0	F: 27.66, 30	F: 26.66, 0	F: 30.10, 0	F: 36.62, 0	F: 29.97, 0
31	F: -1.00, 0	F: -1.00, 0	F: 27.84, 32	F: 31.86, 0	F: 35.64, 0	F: 21.88, 0	F: 20.98, 0
32	F: -1.00, 0	F: -1.00, 0	F: 24.04, 6	F: 22.42, 0	F: 33.59, 0	F: 22.12, 0	F: 25.21, 0
33	F: -1.00, 0	F: -1.00, 0	F: 26.28, 39	F: 24.09, 0	F: 28.40, 0	F: 20.15, 0	F: 27.51, 0
34	F: -1.00, 0	F: -1.00, 0	F: 17.05, 10	F: 25.17, 0	F: 21.67, 0	F: 26.64, 0	F: 26.90, 0
35	F: -1.00, 0	F: -1.00, 0	F: 67.59, 19	F: 20.77, 0	F: 27.97, 0	F: 21.70, 0	F: 21.06, 0

Table A.6: All results on the blocksworld domain. First value determine if the planner solved (T) or failed (F) the problem. Second value is the time until the planner stopped or was stopped in seconds. Last value is the length of the (partial) plan

#	Pyp-HFF	Pyp-SAT	HFF	SAT	Naive	Max	Mutex
01	T: 0.00, 5	T: 0.03, 5	F: 0.00, 0	F: 60.05, 0	F: 0.00, 1	F: 0.00, 1	F: -3.00, 0
02	T: 0.00, 8	T: 0.03, 8	F: 0.00, 0	F: 60.04, 0	F: 0.00, 2	F: 0.00, 2	F: -3.00, 0
03	T: 0.00, 11	T: 0.07, 11	F: 0.00, 0	F: 60.18, 0	F: 0.01, 3	F: 0.01, 3	F: -3.00, 0
04	T: 0.00, 14	T: 0.18, 14	F: 0.00, 0	F: 60.06, 0	F: 0.01, 4	F: 0.02, 4	F: -3.00, 0
05	T: 0.01, 19	F: -1.00, 0	F: 0.00, 0	F: 60.77, 0	F: 0.08, 5	F: 0.76, 5	F: -3.00, 0
06	T: 0.90, 31	F: -1.00, 0	F: 0.02, 0	F: 22.92, 0	F: 2.49, 6	F: 528.37, 6	F: -3.00, 0
07	T: 5.71, 48	F: -1.00, 0	F: 0.01, 0	F: 22.78, 0	F: 9.93, 7	F: -2.00, 0	F: -3.00, 0
08	T: 14.20, 58	F: -1.00, 0	F: 0.01, 0	F: 23.21, 0	F: 24.94, 0	F: -2.00, 0	F: -3.00, 0
09	F: -1.00, 0	F: -1.00, 0	F: 0.03, 0	F: 26.85, 0	F: 43.02, 0	F: -2.00, 0	F: -3.00, 0
10	F: -1.00, 0	F: -1.00, 0	F: 0.03, 0	F: 22.11, 0	F: 21.90, 0	F: -2.00, 0	F: -3.00, 0
11	F: -1.00, 0	F: -1.00, 0	F: 0.04, 0	F: 20.52, 0	F: 31.86, 0	F: -2.00, 0	F: -3.00, 0
12	F: -1.00, 0	F: -1.00, 0	F: 0.05, 0	F: 26.40, 0	F: 52.16, 0	F: -2.00, 0	F: -3.00, 0
13	F: -1.00, 0	F: -1.00, 0	F: 0.11, 0	F: 33.76, 0	F: 30.36, 0	F: -2.00, 0	F: -3.00, 0
14	F: -1.00, 0	F: -1.00, 0	F: 0.10, 0	F: 20.32, 0	F: 39.50, 0	F: -2.00, 0	F: -3.00, 0
15	F: -1.00, 0	F: -1.00, 0	F: 0.10, 0	F: 27.58, 0	F: 22.75, 0	F: -2.00, 0	F: -3.00, 0
16	F: -1.00, 0	F: -1.00, 0	F: 160.45, 58	F: 34.30, 0	F: 24.34, 0	F: -2.00, 0	F: -3.00, 0
17	F: -1.00, 0	F: -1.00, 0	F: 329.29, 50	F: 99.69, 0	F: 69.76, 0	F: -2.00, 0	F: -3.00, 0
18	F: -1.00, 0	F: -1.00, 0	F: 170.10, 33	F: 120.25, 0	F: 90.42, 0	F: -2.00, 0	F: -3.00, 0
19	F: -1.00, 0	F: -1.00, 0	F: 1.56, 0	F: 143.23, 0	F: 103.58, 0	F: -2.00, 0	F: -3.00, 0
20	F: -1.00, 0	F: -1.00, 0	F: 1.70, 0	F: 147.19, 0	F: 119.37, 0	F: -2.00, 0	F: -3.00, 0
21	F: -1.00, 0	F: -1.00, 0	F: 732.67, 36	F: 3.33, 0	F: 23.05, 0	F: -2.00, 0	F: -3.00, 0
22	F: -1.00, 0	F: -1.00, 0	F: 263.76, 48	F: 3.37, 0	F: 23.38, 0	F: -2.00, 0	F: -3.00, 0
23	F: -1.00, 0	F: -1.00, 0	F: 0.80, 0	F: 3.64, 0	F: 23.58, 0	F: -2.00, 0	F: -3.00, 0
24	F: -1.00, 0	F: -1.00, 0	F: 0.89, 0	F: 3.92, 0	F: 23.74, 0	F: -2.00, 0	F: -3.00, 0
25	F: -1.00, 0	F: -1.00, 0	F: 1.29, 0	F: 5.57, 0	F: 25.41, 0	F: -2.00, 0	F: -3.00, 0
26	F: -1.00, 0	F: -1.00, 0	F: 1.99, 0	F: 8.62, 0	F: 28.74, 0	F: -2.00, 0	F: -3.00, 0
27	F: -1.00, 0	F: -1.00, 0	F: 2.13, 0	F: 9.23, 0	F: 29.05, 0	F: -2.00, 0	F: -3.00, 0
28	F: -1.00, 0	F: -1.00, 0	F: 2.30, 0	F: 9.68, 0	F: 29.98, 0	F: -2.00, 0	F: -3.00, 0
29	F: -1.00, 0	F: -1.00, 0	F: 3.15, 0	F: 13.10, 0	F: 32.84, 0	F: -2.00, 0	F: -3.00, 0
30	F: -1.00, 0	F: -1.00, 0	F: 3.26, 0	F: 13.80, 0	F: 33.58, 0	F: -2.00, 0	F: -3.00, 0

Table A.7: All results on the travelling purchase problem domain. First value determine if the planner solved (T) or failed (F) the problem. Second value is the time until the planner stopped or was stopped in seconds. Last value is the length of the (partial) plan

#	Pyp-HFF	Pyp-SAT	HFF	SAT	Naive	Max	Mutex
01	F: -1.00, 0	F: -1.00, 0	F: 0.07, 0	F: 75.31, 0	F: 32.65, 0	F: 64.79, 0	F: -3.00, 0
02	F: -1.00, 0	F: -1.00, 0	F: 0.07, 0	F: 73.77, 0	F: 32.66, 0	F: 62.23, 0	F: -3.00, 0
03	F: -1.00, 0	F: -1.00, 0	F: 0.08, 0	F: 101.51, 0	F: 28.34, 0	F: 65.39, 0	F: -3.00, 0
04	F: -1.00, 0	F: -1.00, 0	F: 0.06, 0	F: 36.74, 0	F: 42.22, 0	F: 94.06, 0	F: -3.00, 0
05	F: -1.00, 0	F: -1.00, 0	F: 0.14, 0	F: 28.08, 0	F: 30.08, 0	F: 31.47, 0	F: -3.00, 0
06	F: -1.00, 0	F: -1.00, 0	F: 0.14, 0	F: 29.56, 0	F: 30.26, 0	F: 33.22, 0	F: -3.00, 0
07	F: -1.00, 0	F: -1.00, 0	F: 0.19, 0	F: 46.05, 0	F: 51.21, 0	F: 57.37, 0	F: -3.00, 0
08	F: -1.00, 0	F: -1.00, 0	F: 0.18, 0	F: 47.13, 0	F: 51.76, 0	F: 55.50, 0	F: -3.00, 0
09	F: -1.00, 0	F: -1.00, 0	F: 0.22, 0	F: 22.35, 0	F: 21.89, 0	F: 24.18, 0	F: -3.00, 0
10	F: -1.00, 0	F: -1.00, 0	F: 0.26, 0	F: 21.99, 0	F: 23.01, 0	F: 23.40, 0	F: -3.00, 0
11	F: -1.00, 0	F: -1.00, 0	F: 0.28, 0	F: 31.99, 0	F: 32.24, 0	F: 31.39, 0	F: -3.00, 0
12	F: -1.00, 0	F: -1.00, 0	F: 0.28, 0	F: 30.76, 0	F: 33.60, 0	F: 34.73, 0	F: -3.00, 0
13	F: -1.00, 0	F: -1.00, 0	F: 0.34, 0	F: 50.03, 0	F: 49.89, 0	F: 52.00, 0	F: -3.00, 0
14	F: -1.00, 0	F: -1.00, 0	F: 0.43, 0	F: 69.12, 0	F: 72.92, 0	F: 80.85, 0	F: -3.00, 0
15	F: -1.00, 0	F: -1.00, 0	F: 0.41, 0	F: 72.60, 0	F: 75.14, 0	F: 81.80, 0	F: -3.00, 0
16	F: -1.00, 0	F: -1.00, 0	F: 0.49, 0	F: 93.57, 0	F: 98.26, 0	F: 106.09, 0	F: -3.00, 0
17	F: -1.00, 0	F: -1.00, 0	F: 0.65, 0	F: 126.00, 0	F: 132.90, 0	F: 145.50, 0	F: -3.00, 0
18	F: -1.00, 0	F: -1.00, 0	F: 0.74, 0	F: 175.33, 0	F: 191.28, 0	F: 207.56, 0	F: -3.00, 0
19	F: -1.00, 0	F: -1.00, 0	F: 1.15, 0	F: 2.54, 0	F: 22.47, 0	F: 22.38, 0	F: -3.00, 0
20	F: -1.00, 0	F: -1.00, 0	F: 1.12, 0	F: 2.41, 0	F: 22.30, 0	F: 22.28, 0	F: -3.00, 0

Table A.8: All results on the childsnack domain. First value determine if the planner solved (T) or failed (F) the problem. Second value is the time until the planner stopped or was stopped in seconds. Last value is the length of the (partial) plan

List of Acronyms

ASP	Answer Set Programming
EHS	enforced hill-climbing search
HCF	Combined Forward Heuristic Search
HFF	Fast Forward Heuristic Search
HRF	Reverse Forward Heuristic Search
OR	operations research
RR-OP	reverse relaxed operator
RR-P	reverse relaxed problem
SAT	satisfiability
SR-OP	symmetric relaxed operator
SR-P	symmetric relaxed problem
STRIPS	Stanford Research Institute Problem Solver
TPP	travelling purchase problem
TSP	travelling salesman problem
wff	well-formed formula

List of Figures

List of Tables

List of Algorithms

List of References

[1] Y. Alkhazraji, M. Frorath, M. Grützner, M. Helmert, T. Liebetraut, R. Mattmüller, M. Ortlieb, J. Seipp, T. Springenberg, P. Stahl, and J. Wülfing, *Pyperplan*, https://doi.org/10.5281/zenodo.3700819, 2020. DOI: 10.5281/zenodo.3700819. [Online]. Available: https://doi.org/10.5281/zenodo.3700819.

[2] B. Bonet and H. Geffner, "Planning as heuristic search: New results," in *European Conference on Planning*, 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:5527200.

[3] B. Bonet and H. Geffner, "Planning as heuristic search," *Artif. Intell.*, vol. 129, pp. 5–33, 2001. [Online]. Available: https://api.semanticscholar.org/CorpusID:11337237.

[4] R. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artif. Intell.*, vol. 2, pp. 189–208, 1971. [Online]. Available: https://api.semanticscholar.org/CorpusID:8623866.

[5] D. Fiser and A. Komenda, "Fact-alternating mutex groups for classical planning," *J. Artif. Intell. Res.*, vol. 61, pp. 475–521, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:4095667.

[6] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, "Multi-shot asp solving with clingo," *Theory and Practice of Logic Programming*, vol. 19, pp. 27–82, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:3994108.

[7] E. Giunchiglia and M. Maratea, "Planning as satisfiability with preferences," in *AAAI Conference on Artificial Intelligence*, 2007. [Online]. Available: https://api.semanticscholar.org/CorpusID:2535533.

[8] C. Grundke, G. Röger, and M. Helmert, "Formal representations of classical planning domains," in *International Conference on Automated Planning and Scheduling*, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:270170904.

[9] N. Gupta and D. S. Nau, "On the complexity of blocks-world planning," *Artificial intelligence*, vol. 56, no. 2-3, pp. 223–254, 1992.

[10] P. Haslum and H. Geffner, "Admissible heuristics for optimal planning," in *International Conference on Artificial Intelligence Planning Systems*, 2000. [Online]. Available: https://api.semanticscholar.org/CorpusID:196186.

[11] M. Helmert, "Complexity results for standard benchmark domains in planning," *Artif. Intell.*, vol. 143, pp. 219–262, 2003. [Online]. Available: https://api.semanticscholar.org/CorpusID:33047101.

[12] J. Hoffmann, "Ff: The fast-forward planning system," *AI Mag.*, vol. 22, pp. 57–62, 2001. [Online]. Available: https://api.semanticscholar.org/CorpusID:9968823.

[13] J. Hoffmann and B. Nebel, "The ff planning system: Fast plan generation through heuristic search," *ArXiv*, vol. abs/1106.0675, 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:6552475.

[14] R. C. Holte, A. Felner, G. Sharon, N. R. Sturtevant, and J. Chen, "Mm: A bidirectional search algorithm that is guaranteed to meet in the middle," *Artif. Intell.*, vol. 252, pp. 232–266, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:31304550.

[15] S. Kambhampati and B. Srivastava, "Universal classical planner: An algorithm for unifying state-space and plan-space planning," 1996. [Online]. Available: https://api.semanticscholar.org/CorpusID:16627212.

[16] S. Kambhampati and B. Srivastava, "Unifying classical planning approaches," *Arizona State University ASU CSE TR*, pp. 96–006, 1996.

[17] H. A. Kautz and B. Selman, "Planning as satisfiability," in *European Conference on Artificial Intelligence*, 1992. [Online]. Available: https://api.semanticscholar.org/CorpusID:42462267.

[18] Y. Liu, S. Koenig, and D. Furcy, "Speeding up the calculation of heuristics for heuristic search-based planning," in *AAAI/IAAI*, 2002. [Online]. Available: https://api.semanticscholar.org/CorpusID:1532117.

[19] B. C. Massey, "Propositional strips planning problem reversal," 1998. [Online]. Available: https://api.semanticscholar.org/CorpusID:15193806.

[20] J. Rintanen, "Planning as satisfiability: Heuristics," *Artif. Intell.*, vol. 193, pp. 45–86, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:1426037.

[21] M. Wehrle and J. Rintanen, "Planning as satisfiability with relaxed $-step plans," in *Australian Conference on Artificial Intelligence*, 2007. [Online]. Available: https://api.semanticscholar.org/CorpusID:3233938.

[22] J. Wichlacz, D. Höller, and J. Hoffmann, "Landmark heuristics for lifted classical planning," in *International Joint Conference on Artificial Intelligence*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:250637264.