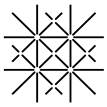

University of Basel

# Constrained Pattern Databases

Master's thesis

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

Examiner: Prof. Dr. Malte Helmert
Supervisor: Dr. Salomé Eriksson

Takudzwa Togarepi
t.togarepi@stud.unibas.ch
21-066-436

29-March-2025

# Acknowledgments

I would like to start by thanking the head of the Artificial Intelligence Research Group at the University of Basel, Prof. Dr. Helmert for allowing me the opportunity to write my Master's thesis within his research group. Furthermore, I extend my gratitude to my supervisor Dr. Salomé Eriksson who was always available throughout the whole process providing timely and helpful feedback. I also would like to thank my family for their support during the thesis writing process. Finally, I thank all my friends who were always available when I needed someone to share my experience with.

# Abstract

Optimal planning is a subfield within the domain of classical planning that aims to find not only valid plans to a problem but the most efficient plan according to a given cost metric. One way to achieve this is by making use of guided search along with an admissible heuristic, and one of the commonly used guided search algorithms is A$^*$. A common way to get this admissible heuristic is by using Pattern Databases which are based on state abstractions. Pattern Databases provide a fast lookup table with the heuristic estimates for abstract states. In this thesis, we extend Pattern Databases by enforcing mutex constraints which forbid certain facts from occurring at the same time. To this end, we will evaluate three different methods of inferring the mutexes and compare which one works best with Pattern Databases.

# Table of Contents

# 1
# Introduction

Classical planning is used to solve a variety of challenges, ranging from logistics optimizations to more recreational tasks such as solving a Rubik's cube [13]. Optimal planning is a subfield within the domain of classical planning that aims to find not only valid plans to a problem but the most efficient plan according to a given cost metric. In simple terms, a plan refers to a sequence of steps an agent takes to move from its initial state to a desired state. One way optimal planning achieves this is by making use of guided search. Guided search refers to search algorithms that make use of heuristics to guide it to explore promising states that are likely to lead to a goal state. A function which maps a state to a number which estimates the distance of the current state to the goal state is what is known as a heuristic.

One way to guarantee optimal solutions is to use a specific guided search algorithm called A$^*$, together with so-called admissible heuristics. Informally, an admissible heuristic is one that never overestimates the true goal distance from a state. A common method used for constructing admissible heuristics is Pattern Databases (PDBs) [4].

Common classical planning systems like Fast Downward make use of PDBs to come up with heuristic estimates. As proposed by Haslum et al. [8] we can make use of an enhanced version of PDBs known as constrained pattern databases (CPDBs) to compute heuristics also. The goal of this thesis is to implement CPDBs and evaluate their performance within the Fast Downward planner. The main reason for using CPDBs is it gives us better heuristic estimates as compared to normal PDBs and they also can save memory during heuristic computation. This is mainly due to the way we compute the heuristic in both cases. A CPDB heuristic takes the original problem and identifies rules or constraints within that problem [8], these constraints then help us fine-tune our estimates from the PDB. For example, we might know that in the original problem certain actions cannot be applied in any reachable state or the action leads to a state where two facts are true for which we know they cannot occur concurrently. Then we can ignore those moves when computing our estimates or alternatively we assign value infinity to them. This makes our estimates stronger and more reliable, that is the planner is able to find the goal more efficiently. All this whilst CPDBs still provide us with admissible heuristics. On the other hand PDBs do not ignore such unreachable actions or states, which leads to a potentially lower heuristic estimate.

For our implementation of CPDBs we will use different sources of mutex information, the goal also will be to determine which performs best with PDBs. In this thesis, the first chapter is an introductory chapter which gives a brief overview of the thesis. The second chapter is where we introduce some concepts relevant to the thesis and the third chapter we introduce CPDBs formally. Our experiment setup and results will be discussed in the fourth chapter. Finally, the fifth chapter concludes this thesis as well as suggest what could be done in future to further improve our implementation.

# 2
# Background

This chapter introduces the notion of planning tasks, pattern databases and invariants which are all foundational concepts crucial for understanding the concept of constrained pattern databases which will be the main topic of this thesis.

## 2.1 SAS$^+$ Planning Task

In classical planning, SAS$^+$ planning tasks are a formal way of representing real world complex environments using multi-valued state variables. It is an important concept for understanding how invariants, which will be discussed later, work. The formal representation of SAS$^+$ used in this paper is an adoption of the one used already by Bäckström and Nebel [2].

**Definition 1.** ***(SAS$^+$ Planning Task)*** *A SAS$^+$ planning task* $\Pi$ *is defined as a tuple with 4 elements, that is* $\Pi = \langle \boldsymbol{V, O, I, G} \rangle$ *where*

- $\mathbf{V}$ is a finite set of state variables $\mathbf{V} = \{x_1, x_2, x_3, \ldots, x_n\}$ and each variable is associated with some finite domain $\text{dom}(x_i) = \{d_1, d_2, d_3, \ldots, d_n\}$. A state is a full assignment to all state variables, whereas a partial state is a partial assignment of the state variables meaning just a subset of the variables are assigned. $\mathcal{S}$ is the set of all states.
- $\mathbf{O}$ is a set of operators or actions which are in the form $\langle \text{pre}(o), \text{eff}(o) \rangle$ where $\text{pre}(o)$ and $\text{eff}(o)$ are partial assignments over the state variables $\mathbf{V}$ to their respective domain. $\text{pre}(o)$ and $\text{eff}(o)$ are preconditions and effects respectively.
- $\mathbf{I}$ is the initial state or the start state, $\mathbf{I} \in \mathcal{S}$.
- $\mathbf{G}$ is a partial assignment of the state variables which represents the goal condition. A state is a goal state if for all variables $x_i \in \mathbf{V}$ where $\mathbf{G}(x_i) = d_i$ is defined, we have $s(x_i) = d_i$ (they agree on the partial assignment $\mathbf{G}$).

In SAS$^+$ we also make use of what is known as a **fact** which is a variable-value pair $(x_i, d_i)$ where $x_i$ is a variable and $d_i$ a value from $x_i$'s domain. A partial assignment $s$ can be translated to a set of facts $\mathcal{F} = \{(x_i, d_i) | s(x_i) = d_i\}$. Also, we have what is known as a **cost**

in planning tasks, which is a function that assigns a non-negative number to each operator $\text{cost}(o) : \mathbf{O} \rightarrow \mathbb{R}_0^+$. An operator is said to be applicable in any given state, if the operator precondition is true in the state in which it has to be applied in. An operator is said to be applicable in any given state $s$ if the operator precondition is true in $s$, meaning that for all $x_i$ where $\text{pre}(o)$ is defined that $s(x_i) = \text{pre}(o)(x_i)$. State $s'$ is the resulting state after applying an operator $o$ in state $s$ which can be formerly written as $s' = s[o]$ where:

$$s[o](x_i) = \begin{cases} \text{eff}(o)(x_i) & \text{if eff}(o) \text{ is defined on } x_i \\ s(x_i) & \text{otherwise} \end{cases}$$

A sequence of operators $\pi = \langle o_1, \ldots, o_n \rangle$ which are applicable to state $s$ resulting in state $s'$ is known as a **path** from $s$ to $s'$. A state $s$ is said to be **reachable** if there exists a path from the initial state to $s$. Subsequently, a **plan** $\pi$ is the path from the initial state to a goal state. Each plan $\pi$ is associated with a cost which we will now denote as $C(\pi)$. The cost of a plan is a summation of all operator costs within that plan, that is for plan $\pi = \langle o_1, \ldots, o_n \rangle$ we have $C(\pi) = \sum_{i=1}^{n} cost(o_i)$. Furthermore, an **optimal plan** $\pi_{opt}$ is a plan with the least cost among all possible plans in a planning task, that is, $C(\pi_{opt}) \leq C(\pi)$ for all plans $\pi$.

## 2.2 Transition systems

In practice, planning tasks can have different types of representations, and for some techniques it is more suitable to represent planning tasks as a transition system.

**Definition 2.** ***(Transition system)*** *A transition system $\mathcal{T}$ is 5 tuple $\mathcal{T} = \langle \mathbf{S}, \mathbf{L}, \mathbf{T}, \mathbf{s}_0, \mathbf{s}_* \rangle$ where*

- **S** is a finite set of states.
- **L** is a finite set of transition labels. Each label $l$ in **L** has a corresponding cost assigned to it that is $\text{c}(l) \in \mathbb{R}_0^+$.
- **T** represents a set of labeled transitions. **T** $\subseteq$ **S** x **L** x **S**.
- $\mathbf{s}_0$ defined as $\text{s}_0 \in$ **S** represents the initial state.
- $\mathbf{s}_* \subseteq$ **S** represents the goal condition of the transition system.

Following the definition of a transition system $\mathcal{T}$, planning task $\Pi = \langle \mathbf{V}, \mathbf{O}, \mathbf{I}, \mathbf{G} \rangle$ can now be represented by a transition system $\mathcal{T}(\Pi)$ as follows:

**Definition 3.** ***(Transition system induced by a planning task)*** *A planning task $\Pi$ induces a Transition system $\mathcal{T}(\Pi)$ where*

- **S** is the set of all states over **V**.
- **L** is the set of operators **O** and each label is associated with a cost.

- $\mathbf{T}$ with $\langle s, o, s' \rangle \in \mathbf{T}$ iff operator $o$ is applicable in state $s$ and results in state s′.
- $\mathbf{s}_0 = \mathbf{I}$
- $\mathbf{s}_* = \{s \in \mathbf{S} \mid s \text{ agrees on } \mathbf{G}\}$

At this point we would like to introduce the concept of abstraction which is an essential concept of PDBs. In relation to transition systems (transition graphs) abstraction refers to the concept in which we create smaller versions of $\mathcal{T}$ by ignoring some information in $\mathcal{T}$. Essential to the concept of abstraction is $\alpha$ which is a function known as an abstraction mapping such that $\alpha : \mathbf{S} \rightarrow \mathbf{S}^\alpha$ is defined on states of $\mathcal{T}$ [11].

**Definition 4.** *(**Abstraction**). For any $\mathcal{T}$ and $\alpha$ the abstract transition system $\mathcal{T}^\alpha$ is given by $\mathcal{T}^\alpha = \langle \mathbf{S}^\alpha, \mathbf{L}, \mathbf{T}^\alpha, \alpha(\boldsymbol{s}_0), \mathbf{s}^\alpha_* \rangle$ where:*

- $\mathbf{S}^\alpha = \{\alpha(s) \mid s \in \mathbf{S}\}$
- $\mathbf{T}^\alpha = \{\langle \alpha(s), o, \alpha(s') \rangle \mid \langle s, o, s' \rangle \in \mathbf{T}\}$
- $\mathbf{s}^\alpha_* = \{\alpha(s) \mid s \in \mathbf{s}_*\}$

Since an abstraction takes a transition system $\mathcal{T}$ and makes a smaller transition system $\mathcal{T}^\alpha$, we solve that smaller abstract transition system and get a heuristic which is commonly referred to as the abstraction heuristic.

**Definition 5.** *(**Abstraction heuristics**) Given a transition system $\mathcal{T}$ and its abstraction mapping $\alpha$, the abstraction heuristic for $\mathcal{T}$ is given by $h^\alpha$. $h^\alpha$ is a function which assigns to each state $s \in \mathcal{S}$ the cost of the cheapest path in $\mathcal{T}^\alpha$ from $\alpha(s)$ to a goal state in $\mathcal{T}^\alpha$, where $\mathcal{S}$ is a set of reachable states in $\mathcal{T}$ [11].*

Apart from the general abstraction defined in Definition 5 there also exists a special type of abstraction known as a projection which is what the pattern databases are based on.

**Definition 6.** *(**Projection**) Let $\Pi$ be a SAS$^+$ planning task with state set $\mathcal{S}$ and let $\mathcal{P} \subseteq \mathbf{V}$ be the subset of variables. A projection onto a variable set $\mathcal{P}$ is a function defined by $\alpha^\mathcal{P}$ such that $\alpha(s) = \alpha(s')$ if and only if $s(p) = s'(p)$ for all $p \in \mathcal{P}$ [11].*

## 2.3 Pattern Databases

Pattern Databases (PDBs) introduced by Culberson and Schaeffer [3], are an important approach used in classical planning to come up with admissible heuristics. PDBs can be defined as a dictionary which stores heuristic estimates of abstract state to goal distances [3] [4]. This is done by first creating a special type of abstract planning task which we will call $\Pi^\mathcal{P}$. The projected planning task $\Pi^\mathcal{P}$ is created by projecting a pattern $\mathcal{P}$ on to the original planning task $\Pi$, this results in $\Pi^\mathcal{P} = \langle \mathcal{P}, \alpha^\mathcal{P}(\mathbf{I}), \mathbf{O}^\mathcal{P}, G^\mathcal{P} \rangle$ where $\mathcal{P} \subseteq \mathbf{V}$. Projections map states of the original planning task $\Pi$ into a smaller one otherwise called an abstract state. A solution path in $\Pi$ would also be a solution path in $\Pi^\mathcal{P}$ [3] [8]. In other words $\Pi^\mathcal{P}$ is

a smaller and relaxed version of Π whereby we only focus on variables in the pattern and abstract everything else that is not in the pattern.

**Definition 7.** ***(Pattern databases)*** *PDBs exist as a set of pairs with one of the elements being an abstract state and the other being the cheapest path between the abstract state and a set of abstract goal states* $PBD(\Pi^{\mathcal{P}}) = \{(s_{\mathcal{P}}, d(s_{\mathcal{P}}, G^{\mathcal{P}}) \mid s_{\mathcal{P}} \in \mathcal{S}^{\mathcal{P}}\}$ [5].

The given equation shows how a typical structure of a PDB where:

- $s_{\mathcal{P}}$ is an abstract state and is an element of $\mathcal{S}^{\mathcal{P}}$ (set of states of $\Pi^{\mathcal{P}}$)
- $d(s_{\mathcal{P}}, G^{\mathcal{P}})$ is the cost of the cheapest path from $s_{\mathcal{P}}$ to any goal state in $\Pi^{\mathcal{P}}$
- $G^{\mathcal{P}}$ represents an abstract goal state.

To illustrate the concept of pattern projections, we will look at the Blocksworld problem. In the Blocksworld problem, we are given a task to stack blocks, one on top of another in a particular order defined in the problem: We are given the initial configuration of the blocks that is their initial position and we are also given operators we can apply as well as the goal configuration [7]. Operators tell us how to move around the blocks from one configuration to another. Let us assume we have a problem with 3 blocks labeled $b_1$, $b_2$ and $b_3$ and the problem is defined by the planning task Π as follows:
$\Pi = \langle$**V, I, O, G**$\rangle$ where:

- **V** $= \{b_1, b_2, b_3, clear_1, clear_2, clear_3\}$
  - dom($b_i$) = {on-table, on-$b_j$} where $i, j = \{1, 2, 3\}$ and $i \neq j$
  - dom($clear_i$) $= \{\top, \bot\}$ where $i \in \{1, 2, 3\}$
- **I** $= \{b_1 \mapsto$ on-table, $b_2 \mapsto$ on-table, $b_3 \mapsto$ on-table, $clear_1 \mapsto \top$, $clear_2 \mapsto \top$, $clear_3 \mapsto \top\}$
- **O** $= \{$put-on-table-$b_i \mid i \in \{1, 2, 3\}\} \cup \{$stack-$b_i$-on-$b_j \mid i \in \{1, 2, 3\}, j \in \{1, 2, 3\}, i \neq j\} \cup \{$from-$b_k$-stack-$b_i$-on-$b_j \mid i, j, k \in \{1, 2, 3\}$ and $k \neq i, i \neq j$ and $k \neq j\}$, where:
  - put-on-table-$b_i = \langle\{clear_i \mapsto \top,\ b_i \mapsto$ on-$b_j\}, \{b_i \mapsto$ on-table$\}, \{clear_j \mapsto \top\}\rangle$
  - stack-$b_i$-on-$b_j = \langle\{clear_i \mapsto \top,\ clear_j \mapsto \top,\ b_i \mapsto$ on-table$\}, \{b_i \mapsto$ on-$b_j,\ clear_j \mapsto \bot\}\rangle$
  - from-$b_k$-stack-$b_i$-on-$b_j = \langle\{clear_k \mapsto \bot,\ clear_i \mapsto \top,\ clear_j \mapsto \top,\ b_i \mapsto$ on-$b_k\}, \{clear_j \mapsto \bot,\ clear_k \mapsto \top,\ b_i \mapsto$ on-$b_j\}\rangle$
- **G**$= \{b_1 \mapsto$ on-$b_2$, $b_2 \mapsto$ on-table, $b_3 \mapsto$ on-$b_1\}$

Figure 2.1 shows a graphical representation of the 3 Blocksworld task. The blocks are labeled as $b_1$, $b_2$, and $b_3$ and for a more clearer representation, they are assigned to colors blue, orange and red respectively. The initial state is the state with all 3 blocks on the table labeled **I** and the goal state is labeled as **G**. Only the reachable states are shown in

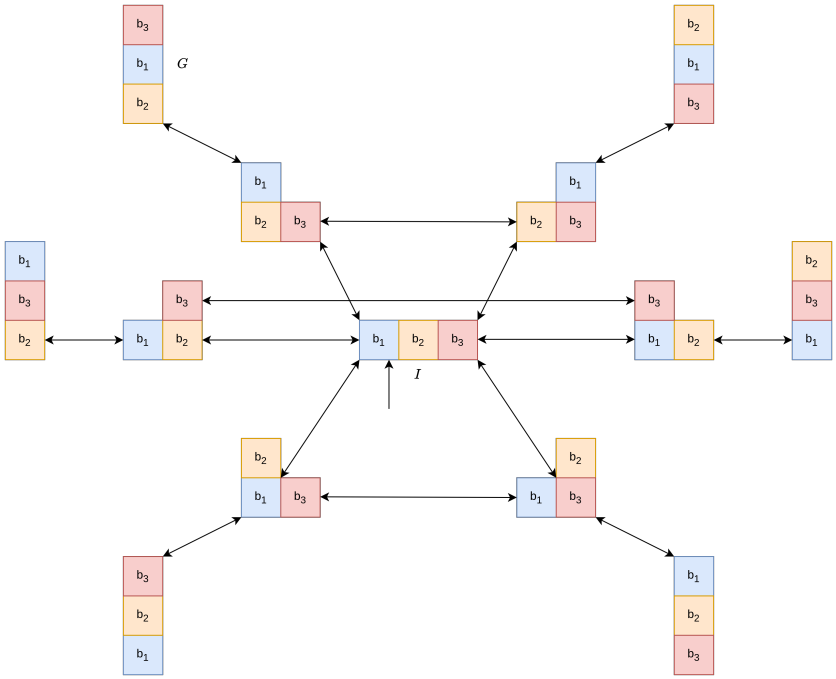


**Figure 2.1:** State space representation of the original planning task

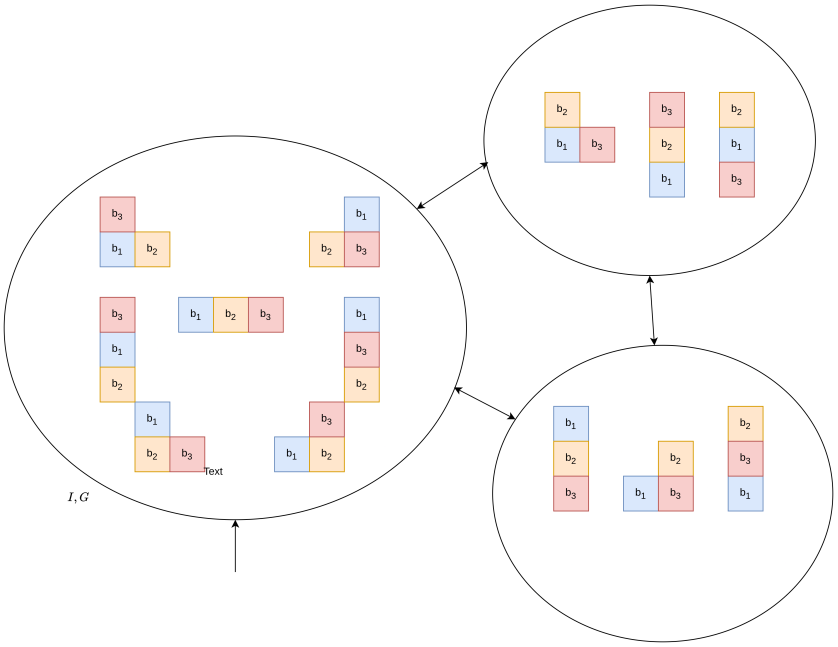


**Figure 2.2:** State space representation of the pattern projection on pattern ($b_2$)

Figure 2.1, the unreachable states which would otherwise be syntactically valid states have been omitted. An example of a syntactically valid but unreachable state would be one where $b_3$ is on the table and there is a cyclic stack between blocks $b_1$ and $b_2$ that is, $b_2$ is on $b_1$ and $b_1$ is on $b_2$.

On the other hand, Figure 2.2 shows a pattern projection of the original task on pattern $b_2$.

In other words this means that Figure 2.2 depicts a scenario where all states in which $b_2$ is in the same position have been grouped into one abstract state. For example, the state on the left shows an abstract state where $b_2$ is on the table. This pattern only cares for the position of the variable in question which is $b_2$ in this case and disregards the positions of the other blocks.

To come up with the PDBs we use breadth-first search in reverse, that is starting from a set of goal states and traversing up the graph [5]. An advantage of PDBs is that they offer a fast lookup once we have calculated the heuristic estimates of an abstract state. The PDB can now be used as a heuristic by abstracting the state according to the pattern, looking up that abstracted state in the PDB, and using the stored distance as a heuristic estimate. However, one main drawback with PDBs is that a single pattern usually results in loss of information because the pattern only covers a small portion of the original problem [14]. To address the aforementioned drawback we can make use of pattern collections, which work by using multiple patterns which cover different parts of the problem hence retaining more information on the original task [14][12].

## 2.4 Invariants

Within the field of classical planning, it is crucial to identify properties that remain constant throughout the whole plan. These properties are known as invariants and they hold in all reachable states.

**Definition 8.** ***(Invariants)***. *As defined by Alcázar and Torralba [1] an invariant refers to a formula over facts of the planning task which is satisfied by all states reachable from the initial state. In other words these are conditions that hold true for all reachable states and if they are not true in the goal state then the planning task is unsolvable.*

**Definition 9.** ***(Mutex)***. *Two facts ($x_i$,$d_i$) and ($x_j$,$d_j$) are said to be mutually exclusive (mutex) if there is no reachable state s with $s(x_i) = d_i$ and $s(x_j) = d_j$ [6].*

In other words, mutex information can be used to detect if a state is reachable. This gives rise to mutex groups which then specify that within a group at most one of the facts can occur. As an example, we look at the Blocksworld problem as we defined it in Section 2.3 where we were given a task with three blocks and had to stack them in an order defined in the planning task $\Pi$. In this Blocksworld example, $s_n = \{b_2 \mapsto \text{on-}b_1, b_1 \mapsto \text{on-}b_2\}$ is an example of a partial assignment containing mutex facts. The reason why we say $s_n$ contains mutex facts is we cannot have a cyclic stack that is $b_1$ being on $b_2$ at the same time as $b_2$ is on $b_1$. Furthermore, the partial assignment $s_{n1} = \{b_2 \mapsto \text{on-}b_1, \text{clear}_1 \mapsto \top\}$ also contains mutex facts since there is a contradiction as $b_1$ cannot be clear while $b_2$ is also on top of it.

# 3

# Constrained pattern databases

As an extension to the PDBs, Haslum et al. [8] introduced a way for PBDs to use mutex information to get better heuristics. If we recall the example that we gave of the Blocksworld problem in the section of computing invariants, we said $s_n = \{b_2 \mapsto \text{on-}b_1, b_1 \mapsto \text{on-}b_2\}$ and $s_{n1} = \{b_2 \mapsto \text{on-}b_1, \text{clear}_1 \mapsto \top\}$ both contain mutex facts. However, this information is currently not utilized during PDB computation. Incorporating it into our PDB computation would result in a more informed heuristic as we get a better heuristic which is still admissible. Constrained pattern databases (CPDBs) take advantage of mutex invariants to improve the search by excluding states and transitions in the abstract space that are impossible to reach in the original planning task $\Pi$.

To implement this idea of CPDBs, we assume we have a set of mutex groups $\mathcal{C} = \{ \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \ldots, \mathcal{C}_m \}$ where $\mathcal{C}_i$ is a set of facts. For any reachable state $s$ the constraint $|\mathcal{C}_i \cap s| \leq 1$ must hold for all $\mathcal{C}_i \in \mathcal{C}$, meaning to say in every reachable state at most 1 fact from $\mathcal{C}_i$ should be true. In addition, we exclude transitions from $s'$ to $s$ where constraint $|(s' \cup \text{pre}(o)) \cap \mathcal{C}_i| \leq 1$ is violated [8]. This means that a transition from $s'$ to $s$ only exists if $s'$ and the precondition of operator $o$, do not violate the mutex constraint.

Just like PDBs, CPBDs are computed based on an abstract planning task $\Pi^{\mathcal{P}}$ using breadth-first search in reverse.

**Definition 10.** *(**Constrained transition system induced by an abstract task**) An abstract planning task $\Pi^{\mathcal{P}}$ induces a constrained transition system $\mathcal{T}_C(\Pi^{\mathcal{P}}) = \langle \boldsymbol{S}_C, \mathbf{L}, \boldsymbol{T}_C, \boldsymbol{s}_0^{\mathcal{P}}, \boldsymbol{s}_{*C} \rangle$ where:*

- $S_C = \{s \mid |s \cap \mathcal{C}_i| \leq 1 \text{ for all } \mathcal{C}_i \in \mathcal{C}, s \in \mathcal{S}^{\mathcal{P}}\}$ where $\mathcal{S}^{\mathcal{P}}$ is the set of all states from $\Pi^{\mathcal{P}}$.
- $T_C = \{\langle s', o, s \rangle | s, s' \in S_C, o \text{ applicable in } s', s'[o] = s, |(s' \cup \text{pre}(o)) \cap \mathcal{C}_i| \leq 1 \text{ for all } \mathcal{C}_i \in \mathcal{C}\}$
- $s_{*C} = \{s \mid s \in (s_*^{\mathcal{P}} \cap S_C)\}$

The search space of the constrained transition system induced by projection $\mathcal{P}$ denoted by $\mathcal{T}_C(\Pi^{\mathcal{P}})$ is a subgraph of $\mathcal{T}(\Pi^{\mathcal{P}})$ which consists only of states and transitions that do not violate the mutex constraints explained before. The transition system of the abstracted planning task $\mathcal{T}(\Pi^{\mathcal{P}})$ maintains all valid solution paths in the original transition system $\mathcal{T}(\Pi)$. CPDBs ensure that all states and transitions within $\mathcal{T}_C(\Pi^{\mathcal{P}})$ are reachable and explicitly adhere to the constraints.

PDBs are known to be admissible heuristics, meaning that their heuristic estimates never overestimate the true state to goal distance that is $h^{\alpha}(s) \leq h^*(s)$ where $h^*(s)$ is the true (perfect) heuristic estimate and $h^{\alpha}(s)$ is the estimate from our PDB. We denote the heuristic estimates of CPDBs as $h^{\alpha}_C(s)$. Haslum et al.[8] mentioned that $h^{\alpha}(s) \leq h^{\alpha}_C(s) \leq h^*(s)$ . For $h^{\alpha}(s) \leq h^{\alpha}_C(s)$ a plan $\pi$ with minimum cost in $\mathcal{T}_C(\Pi^{\mathcal{P}})$ is also a plan in $\mathcal{T}(\Pi^{\mathcal{P}})$ which in turn means that the minimum cost plan in $\mathcal{T}(\Pi^{\mathcal{P}})$ can at most be $C(\pi)$ (but could potentially be lower). PDBs being admissible we already know a valid plan $\pi$ in $\mathcal{T}(\Pi)$ is also a valid plan $\pi$ in $\mathcal{T}(\Pi^{\mathcal{P}})$. This also means that $\pi$ should also be a valid plan in $\mathcal{T}_C(\Pi^{\mathcal{P}})$ except for cases where an abstract state or transition that was visited by the plan was removed from $\mathcal{T}(\Pi^{\mathcal{P}})$ by enforcing mutexes. However, that would mean that the abstract state or transition violated a mutex, which is not possible since that would imply the original plan violated a mutex.

## 3.1 Sources of mutex information

As we have seen, for us to implement CPDBs we need to have some mutex information. There are currently several ways in which one can come up with mutex information to use for CPDBs. In this section, we discuss only but a few of those methods. These are also the sources of mutex information we have used in this thesis.

### 3.1.1 Fast Downward mutexes

Fast Downward has two main components, namely translation and search. The translation process mainly deals with converting our planning task defined in Planning Domain Definition Language (PDDL) to a $\text{SAS}^+$ planning task. In contrast, the search process mainly is responsible for exploring the planning state space so as to find an optimal solution if one exists. The translation process mainly comprises of 4 steps and the second step is invariant synthesis step [10]. Instead of detailing all four steps, in this thesis we only focus on the invariant synthesis step which computes mutex relations between atoms. To generate these mutex invariant a guess, check and repair approach is utilized which checks if an initial candidate is mutex invariant or not. Firstly, candidate invariant groups are generated and to check whether this is an invariant or not we make sure no operator adds multiple facts from the same group, and if it adds a fact from the same group it also needs to delete at least one fact from the same group. In essence, the number of added facts should be less than or equal to the ones deleted. If the above holds and in the initial state at most one fact of the group is true, then the candidate is indeed an invariant.

### 3.1.2 Fact Alternating mutex groups (fam-groups)

Another alternative way to generate mutex information is by inferring mutex groups using linear programming [6]. The main steps of the process on how to infer mutexes using this method are outlined in Algorithm 1.

**Definition 11.** ***(fact-alternating mutex groups)***. *A fam-group denoted by $M$ is a subset of $\mathcal{F}$, (facts of the planning task). $M$ has constraints that it can not contain more than one fact from the initial state and also that the number of facts that an operator adds are less or equal to the ones that are preconditions and deleted by that operator [6].*

In SAS$^+$ a set of all facts assigned by eff($o$) are known as add effects add($o$) whereas, the delete effects del($o$) are all facts $(x_i, d_i)$ such that there is a fact $(x_j, d_j)$ in eff($o$) with $x_i = x_j$ and $d_i \neq d_j$. An integer linear program (ILP) based approach can be used to infer fam-groups. The ILP approach starts by translating the fam-group constraints in the definition into ILP constraints. In the setup each ILP variable $x_i$ corresponds to a fact $f_i$ within the original planning task. The fam-group definition is written in ILP constraints using equation 3.1 and equation 3.2 respectively as follows,

$$\sum_{f_i \in \mathbf{I}} x_i \leq 1 \tag{3.1}$$

$$\sum_{f_i \in add(o)} x_i \leq \sum_{f_i \in del(o) \cap pre(o)} x_i \tag{3.2}$$

The ILP variables are binary in nature that is they can only be one of two values 0 or 1. A value of 0 denotes absence of the fact within the fam-group whereas a value 1 denotes presence of the corresponding fact in the fam-group. Since we want to find as large groups as possible, as they give more information it makes sense to maximize the following objective function,

$$\sum_{f_i \in \mathcal{F}} x_i \tag{3.3}$$

By solving the ILP we get a fam-group that satisfies our constraints. However, the solution to the ILP is only one fam-group. To infer more mutex groups, we can repeatedly solve the ILP and add additional constraints (equation 3.4) that forbid previously found solutions [6].

$$\sum_{f_i \notin M} x_i \geq 1 \tag{3.4}$$

In equation 3.4 $M$ is the fam-group that was just found. We go with this cycle until the newly found fam-group has at most one fact. By maximizing the objective function and excluding already identified fam-groups and their subsets from the solution, we get a unique fam-group at every step.

**Algorithm 1:** Inference algorithm for fact-alternating mutex groups using ILP [6]

**Input:** Planning task $\Pi = \langle \mathbf{V, O, I, G} \rangle$
**Output:** A set of fam-groups $\mathcal{M}$

Initialize ILP with constraints according to Equations 3.1 and 3.2;
Set objective function of ILP to maximize Equation 3.3;
Solve ILP and save the resulting fam-group into $M$;
**while** $|M| \geq 1$ **do**
    Add $M$ to the output set $\mathcal{M}$;
    Add constraint according to Equation 3.4;
    $M \leftarrow \emptyset$;
    Solve ILP and if a solution was found, save the resulting fam-group into $M$;
**end**

### 3.1.3 $h^2$ mutexes

As suggested by Alcázar and Torralba [1] one of the most common ways to find mutexes is the $h^m$ heuristic. The $h^m$ heuristic performs a reachability analysis on a semi-relaxed version of the original planning task $P^m$. $P^m$ encodes facts of size $m$ into a single fact. If after doing a reachability analysis on $P^m$ we find a fact in $P^m$ that is unreachable, this would mean the corresponding set of facts are also unreachable in the original planning task. For us to get mutex groups we usually restrict the size of $m$ to 2 meaning we have $P^2$. This would result in a mutex group of size 2, this would mean no state has both these facts true meaning the two facts are mutex. Normally, invariant computation in $h^2$ is done from the initial state and inferring information in a forward direction. However, Alcázar and Torralba introduced an $h^2$ preprocessor which computes invariants in both directions that is it can also start in the goal state and find invariants in a regression manner. Also there's potential of inferring additional by for example computing $h^2$ in the backward direction then computing in the forward direction afterwards [1]. The whole process is an iterative process and it is usually beneficial to do forwards to backward searches iteratively since each step might give more information for the next one.

# 4

# Experimental results

## 4.1 Experimental setup

We ran our experiments with the Fast Downward planning system [9]. In order to test the accuracy of our implementation we ran it against the IPC benchmarks provided by the University of Basel AI research group[1]. We however did not test on all benchmarks available there, instead we only tested against the optimal STRIPS planning track. For running our benchmarks, we made use of the Downward Lab framework [15]. When we had to implement the fam-group inference algorithm, we used the LP solver interface which uses IBM's CPLEX version 22.1.1[2]. In addition, calculations were performed at sciCORE[3] scientific computing center at University of Basel. Our configurations were run on an Intel Xeon Silver 4114 2.2 GHz processor, for each run we also had 3584MB of memory available to us and we limited the runtime to 30 minutes.

### 4.1.1 Experimental configurations

In this section we will introduce the different configurations we used for the experiments in this thesis. In total we had 10 configurations in which we did an in-depth analysis on the **coverage, expansions, pdb_time, search_time and planner_wall_clock_time**. Coverage is the number of successfully solved tasks whereas expansions are the number of states the planner fully generates successor states for. The total time taken to compute the PDB is denoted as pdb_time and search_time is time taken to search through the state space. The time taken for both PDB computations (plus general initialization) and search is total_time and planner_wall_clock_time is the total_time plus time taken for the translation process, that is, the total time taken from translate and the search process as well as any other preprocessing steps.

- **ipdb-uc**. This is the ipdb configuration as it originally is in Fast Downward which we will call the unconstrained version (uc). This served as our baseline for the ipdb

[1] https://github.com/aibasel/downward-benchmarks
[2] https://www.ibm.com/products/ilog-cplex-optimization-studio
[3] http://scicore.unibas.ch/

variant.

- **fd-ipdb-c**. This is the constrained variant of the ipdb which uses the Fast Downward mutexes from translation.
- **fam-ipdb-c**. This is also a constrained variant of the ipdb however, it uses the mutex information inferred by fam-group inference [6]. Also Fišer and Komenda proved that fam-groups discover all mutex groups already generated by Fast Downward during translation, therefore we seeded the fam-groups with these mutex groups so that we do not rediscover them again which reduces computation time.
- **h2-ipdb-uc**. This is an unconstrained version of ipdb which uses the $h^2$ preprocessor. This was our baseline for configurations involving the $h^2$ preprocessor, we needed to have a separate baseline for these configurations since the preprocessor changes the task.
- **h2-ipdb-c**. Constrained version of ipdb which uses mutex generated by the $h^2$ pre-processor.

We also carried out experiments on PDBs with only a single pattern which are known as just pdb in Fast Downward.

- **pdb-uc**. This is the pdb configuration as it originally is in Fast Downward which we will call the unconstrained version (uc).
- **fd-pdb-c**. This is the constrained variant of the pdb which uses the Fast Downward mutexes from translation.
- **fam-pdb-c**. This is also a constrained variant of the pdb however, it uses the mutex information inferred by fam-group inference [6]. We also seeded with the mutex groups from Fast Downward translation.
- **h2-pdb-uc**. This is an unconstrained version of pdb which uses $h^2$ pre-processor.
- **h2-pdb-c**. Constrained version of pdb which uses mutex generated by $h^2$ pre-processor.

## 4.2 Results

In this section we will present our findings from the experiments as well as discuss the results. We will mainly focus on the ipdb results since the pdb results were similar, but we will highlight some differences at the end of the section.

### 4.2.1 Configurations without the $h^2$ preprocessor

The summary in Table 4.1 shows us that when comparing the results of the experiments in which we had the baseline version (ipdb-uc), fd-ipdb-c and fam-ipdb-c, ipdb-uc yielded the highest coverage 946 meaning to say it successfully solved 946 tasks. The two constrained

| Summary | ipdb-uc | fd-ipdb-c | fam-ipdb-c |
|---|---|---|---|
| coverage - Sum | **946** | 927 | 866 |
| error-search-out-of-memory - Sum | 651 | 537 | **391** |
| error-search-out-of-time - Sum | **231** | 364 | 570 |

**Table 4.1:** Shows a summary of number of successfully solved tasks, number of tasks that failed due to running out of time or memory of the different configurations. The best result for each attribute is highlighted in bold.

| domain | ipdb-uc | fd-ipdb-c | fam-ipdb-c |
|---|---|---|---|
| airport (50) | **35** | 29 | 22 |
| depot (22) | **11** | 9 | 9 |
| floortile-opt11-strips (20) | 2 | **9** | **9** |
| floortile-opt14-strips (20) | 0 | **9** | **9** |
| freecell (80) | **21** | 12 | 12 |
| ged-opt14-strips (20) | 19 | **20** | **20** |
| mprime (35) | **24** | **24** | 23 |
| mystery (30) | **17** | **17** | 14 |
| nomystery-opt11-strips (20) | **20** | **20** | 18 |
| organic-synthesis-split-opt18-strips (20) | **7** | **7** | 1 |
| parcprinter-08-strips (30) | **21** | **21** | 14 |
| parcprinter-opt11-strips (20) | **16** | **16** | 10 |
| parking-opt11-strips (20) | **7** | **7** | 4 |
| pipesworld-notankage (50) | **21** | 20 | 15 |
| pipesworld-tankage (50) | **17** | 9 | 9 |
| quantum-layout-opt23-strips (20) | **13** | **13** | 8 |
| scanalyzer-08-strips (30) | **13** | 11 | 11 |
| scanalyzer-opt11-strips (20) | **10** | 8 | 8 |
| snake-opt18-strips (20) | **11** | 1 | 0 |
| spider-opt18-strips (20) | **15** | **15** | 0 |
| tetris-opt14-strips (17) | **1** | 0 | 0 |
| trucks-strips (30) | 9 | **11** | **11** |
| woodworking-opt08-strips (30) | 12 | **14** | **14** |
| woodworking-opt11-strips (20) | 8 | **9** | **9** |
| Sum (694) | **330** | 311 | 250 |

**Table 4.2:** Shows a summary of only domains where we observed a difference in coverage between the 3 configurations. Domains in which we had the same coverage throughout the 3 configurations have been omitted. For a full summary refer to Table A.1

versions got less coverage with fam-ipdb-c performing the worst in terms of coverage for the three configurations. However, in terms of failed runs due to memory limits the constrained versions performed better than ipdb-uc, fam-ipdb-c recorded the least number of failures followed by fd-ipdb-c and lastly ipdb-uc. Looking at Table 4.2, we can see that ipdb-uc and fd-ipdb-c solved the same number of tasks in 9 domains while outperforming the fam-ipdb-c configuration in the process. Comparing just the constrained variants, that is, fd-ipdb-c and fam-ipdb-c we see that fd-ipdb-c always performed at least as good as fam-ipdb-c. Also, what was interesting was the fact that both constrained variants seemed to perform much better than ipdb-uc in the floortile domains both floortile-opt11-strips and floortile-opt14-strips. For us to understand why the constrained versions performed better we will first give a brief explanation about the floortile domain. Within the floortile domain, we are given a grid of tiles which need to be painted in a certain color. In addition, we are

also given robots that will perform the task of painting the tiles and they can only paint tiles above or below their position. The robots cannot move onto a painted tile. From our experiments, the pattern generated by the unconstrained version did not contain the robot position. The problem with this was that the unconstrained version could not figure out that, for example, painting tiles above and below a robot's current location would lead to a dead end if the tile currently occupied by the robot needs to be painted as well. However, the constrained version was much better at recognizing these dead ends in cases where the two painted tiles were in the pattern. This meant ipdb-uc ran out of memory in most tasks within the floortile domain due to the large number of expansions. All but two tasks did not complete the search. It is in tasks like these, the constrained variants really outperform the ipdb-uc in terms of coverage. The constrained versions also slightly outperformed ipdb-uc in 4 other domains although the difference was not as large as that in floortile. On the other hand ipdb-uc and fd-ipdb-c both outperformed fam-ipdb-c in spider-opt18-strips domain, out of a possible 20 tasks they both managed to solve 15 each whereas fam-ipdb-c could not solve a single task in that domain. Within that domain, fam-ipdb-c could not finish the pattern database generation as it ran out of time in all the tasks hence it could not carry out the search, therefore it was unable to solve the tasks. In the tetris-opt14-strips domain ipdb-uc only managed to solve 1 task in that domain whilst the other configurations could not solve any task there. Except for the 1 task solved by ipdb-uc all other tasks across all three configurations failed due to timeout errors, the pattern database computations timed out.

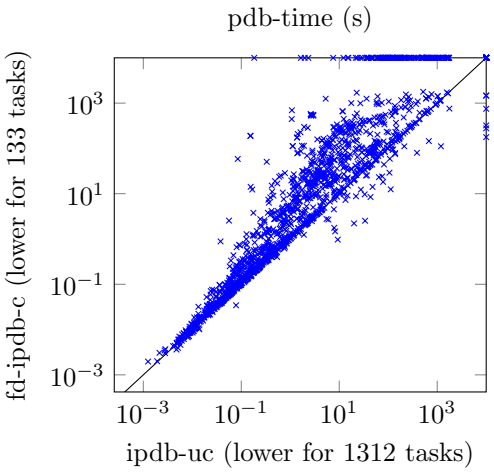


**Figure 4.1:** Comparison of pdb time of ipdb-uc and fd-ipdb-c

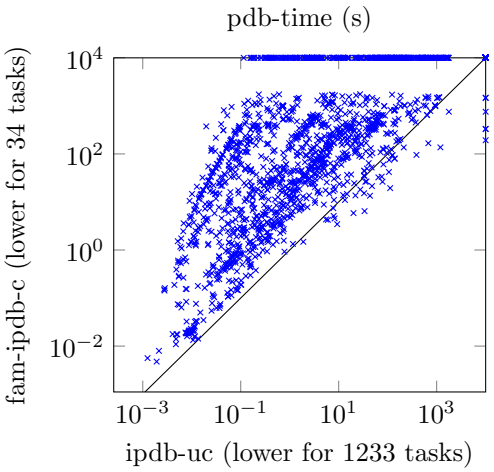


**Figure 4.2:** Comparison of pdb time of ipdb-uc and fam-ipdb-c

Figures 4.1, 4.2 and 4.3 show the comparison of the pattern database generation times of the following three configurations ipdb-uc, fd-ipdb-c and fam-ipdb-c. It is evident that the pattern database generation time of ipdb-uc was the lowest among the three configurations, this is because in the constrained variants each state will have to checked to see if it respects the mutex constraints which introduces an overhead in computation time. However, cutting off transitions and states can also lead to less states explored overall in the backwards reach-

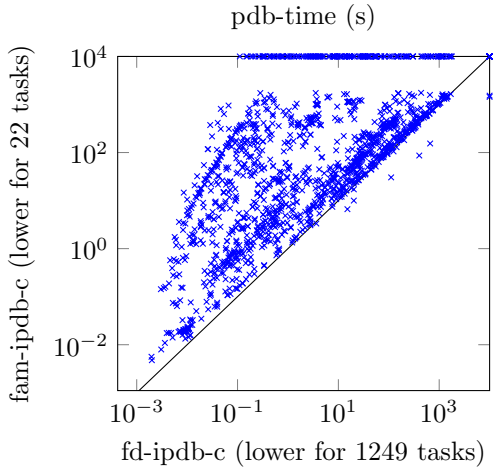


**Figure 4.3:** Comparison of pdb time of fd-ipdb-c and fam-ipdb-c

ability analysis, hence some of the tasks in the constrained variants had a lower computation time than the unconstrained one. Comparing the two constrained variants, fam-ipdb-c took the longest for the pattern database generation, it generated the pattern database faster than fd-ipdb-c in 22 tasks only as compared to 1249 tasks in which fd-ipdb-c was quicker. The fam-ipdb-c has two sources of overhead compared to fd-ipdb-c, first it needs to compute the mutexes using LPs and then generate the pattern database. This means that the pbd time overhead is larger the more mutexes we finds since fam-ipdb-c always finds at least as many mutexes as fd-ipdb-c. This overhead explains the result in Table 4.1 where fam-ipdb-c ran into timeout errors a total of 570 times, the most of the three configurations.

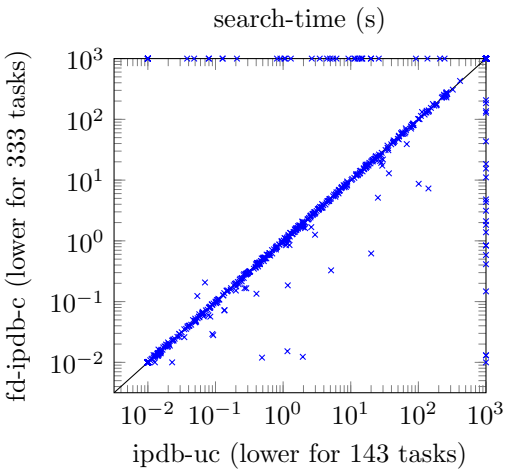


**Figure 4.4:** ipdb-c vs fd-ipdb-c search time

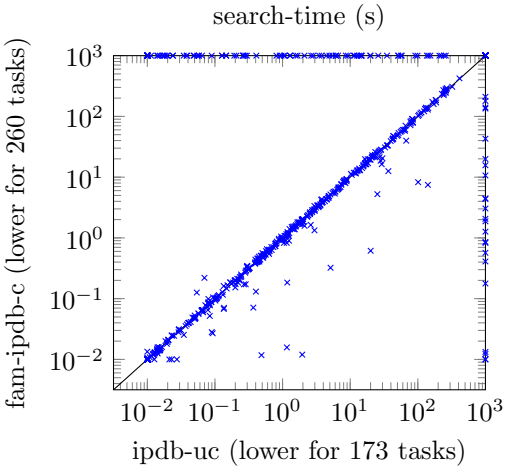


**Figure 4.5:** ipdb-c vs fam-ipdb-c search time

In terms of search time, the constrained variants performed better than ipdb-uc, suggesting that with a more efficient pattern database generation technique we could really harness the full advantages of using constrained abstractions. The search time was lower in constrained

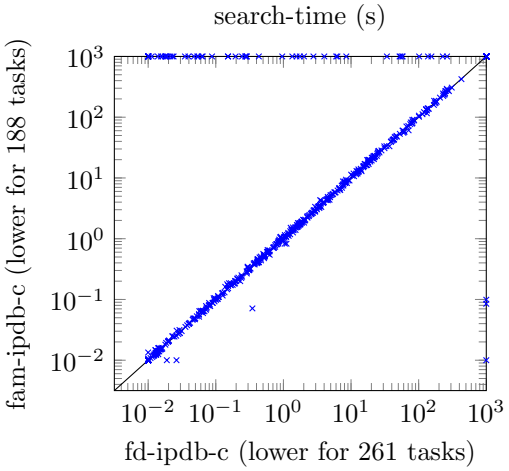


**Figure 4.6:** Comparison of search time of fd-ipdb-c and fam-ipdb-c

versions as a result of using a more informed heuristic than the unconstrained version. This is evident if we compare the search times of the constrained versions to the unconstrained versions we see that the difference between fd-ipdb-c and fam-ipdb-c is less than that of comparing any of the two aforementioned configurations to ipdb-c.

The reduction in search time in the constrained versions as compared to the unconstrained version is a result of lower number of expansions in the constrained versions. The attribute search time includes expanding nodes as well as computing heuristics. The computation of heuristics is equally as expensive for the constrained and unconstrained versions, however, the constrained versions managed to expand fewer states than the unconstrained version. This had the effect of using less memory as to the constrained version which had to expand more states. This is because if we expand fewer states we would need less memory to keep track of the states that are expanded than if we expanded more as we need to keep track of all the expanded states to avoid re-expansion.

To compare the perfomance of the different configurations, we took the number of tasks a particular configuration was better in and divide by the total number of tasks. As for the planner wall clock time ipdb-uc was faster in 71% of the tasks compared to fd-ipdb-c, and ipdb-uc was faster in 88% of the tasks compared to than fam-ipdb-c. This shows that although the constrained variants both had lower search times compared to ipdb-uc, it was not enough to offset the overhead from the pattern database computations hence the total runtime took longer in the constrained variants. Comparison of fd-ipdb-c and fam-ipdb-c showed that the planner wall clock time for fd-ipdb-c was faster in 86% of the tasks compared to of fam-ipdb-c. Not only did fam-ipdb-c manage to solve the fewest task but it also took the longest among the three configurations whereas ipdb-uc solved the most tasks and also in the shortest planner wall clock time.

For the results on the coverage and errors for the pdb configuration we refer to the summary in Table 4.3.

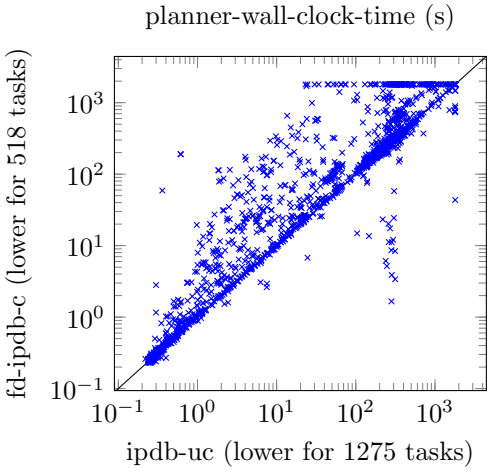


**Figure 4.7:** Comparison of planner_wall_clock_time of ipdb-uc and fd-ipdb-c

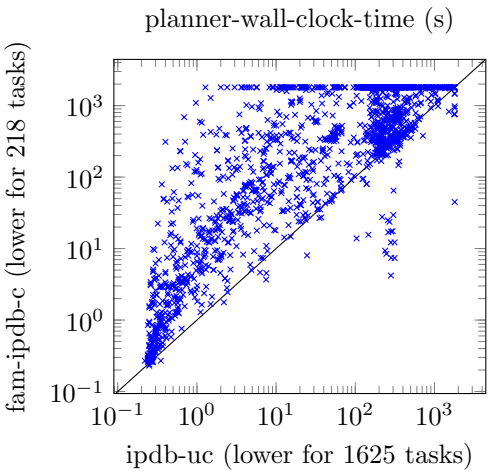


**Figure 4.8:** Comparison of planner_wall_clock_time

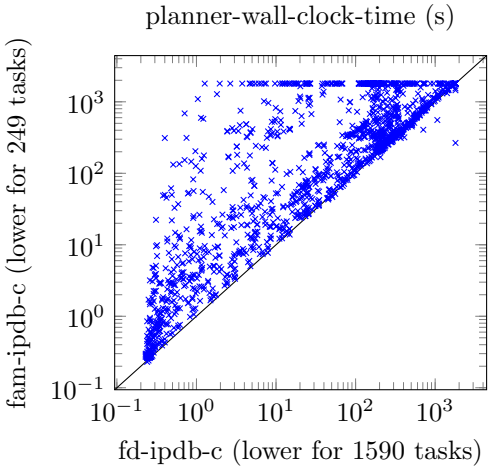


**Figure 4.9:** Comparison of planner_wall_clock_time of fd-ipdb-c and fam-ipdb-c

| Summary | pdb-uc | fd-pdb-c | fam-pdb-c |
|---|---|---|---|
| coverage - Sum | 801 | **809** | 788 |
| error-search-out-of-memory - Sum | 1029 | 1010 | **960** |
| error-search-out-of-time - Sum | **0** | 11 | 82 |

**Table 4.3:** Shows the summary of coverage and errors for the pdb variants. The best value for each attribute is highlighted in bold.

We also ran some tests on the pdb variants that is, pdb-uc, fd-pdb-c and fam-pdb-c. From Table 4.3, we see that fd-pdb-c yielded the highest coverage of the three configurations with a total of 809 successfully solved tasks. It is unlike in the ipdb variants highlighted in Table 4.1 where the unconstrained version had the most coverage. However, just like in the ipdb variants fam-pdb-c still had the lowest coverage here as well whilst having the highest failures due to out of time errors just like fam-ipdb-c. Although the pdb-uc had the most search errors due to high memory usage, it did not record a single failure due to

| domain | pdb-uc | fd-pdb-c | fam-pdb-c |
|---|---|---|---|
| floortile-opt11-strips (20) | 2 | **4** | **4** |
| floortile-opt14-strips (20) | 0 | **2** | **2** |
| organic-synthesis-split-opt18-strips (20) | **10** | **10** | 1 |
| petri-net-alignment-opt18-strips (20) | **4** | **4** | 0 |
| pipesworld-tankage (50) | 17 | **18** | 17 |
| spider-opt18-strips (20) | **11** | **11** | 8 |
| storage (30) | 15 | **16** | **16** |
| tetris-opt14-strips (17) | **10** | **10** | 6 |
| woodworking-opt08-strips (30) | 11 | **12** | **12** |
| woodworking-opt11-strips (20) | 6 | **7** | **7** |
| Sum (247) | 86 | **94** | 73 |

**Table 4.4:** Shows a summary coverage. Domains in which all three got the same coverage is ommitted and can be seen in Table A.1. The best result for each attribute is highlighted in bold.

time limitations. All but two tasks in the floortile domain failed due to high memory usage errors in pdb-c whereas the two constrained versions recorded the same coverage which is a similar trend as to what we've seen before in the ipdb variants. Figures with the results on pdb_time, search_time and planner_wall_clock_time of the pdb variants can be found in the appendix section.

### 4.2.2 Configurations with the $h2$ preprocessor

In this section we will shift our attention to the configurations that used the $h^2$ preprocessor.

| Summary | h2-ipdb-uc | h2-ipdb-c |
|---|---|---|
| coverage - Sum | **990** | 888 |
| error-search-out-of-memory - Sum | 654 | **439** |
| error-search-out-of-time - Sum | **175** | 492 |
| expansions - Sum | 1'439'751'127 | **1'268'326'792** |

**Table 4.5:** Shows a summary coverage, errors due to timeouts or memory limitations as well as a sum of expansions for each configuration. The best result for each attribute is highlighted in bold.

In reference to Table 4.5, we notice the same trend as in Table 4.1 where the unconstrained version had managed to successfully solve more tasks than the unconstrained version. Out of all the ten different configurations that we had, h2-ipdb-uc had the highest coverage with **990** tasks. Just like in Table 4.1, the constrained version had fewer tasks failing due to search out of memory, while having more tasks failing due to search out of time errors as compared to the unconstrained version. Out of a total of 1847 tasks, h2-ipbd-uc managed to solve 54.6% correctly while h2-ipdb-c managed only 48% of the tasks.

An in-depth review on coverage shows that h2-ipdb-c outperformed h2-ipdb-uc in three domains floortile-opt11-strips, floortile-opt14-strips and trucks-strips. The largest difference in coverage was observed in the floortile-opt14-strips domain, where the constrained variant solved 16 tasks successfully compared to the 8 of the unconstrained version. Similarly to the

| domain | h2-ipdb-uc | h2-ipdb-c |
|---|---|---|
| agricola-opt18-strips (20) | **3** | 0 |
| airport (50) | **31** | 17 |
| depot (22) | **11** | 7 |
| floortile-opt11-strips (20) | 8 | **13** |
| floortile-opt14-strips (20) | 8 | **16** |
| freecell (80) | **21** | 8 |
| ged-opt14-strips (20) | **19** | 15 |
| mprime (35) | **24** | 23 |
| mystery (30) | **17** | 13 |
| nomystery-opt11-strips (20) | **20** | 13 |
| organic-synthesis-split-opt18-strips (20) | **13** | 7 |
| pipesworld-notankage (50) | **21** | 13 |
| pipesworld-tankage (50) | **17** | 8 |
| scanalyzer-08-strips (30) | **13** | 12 |
| scanalyzer-opt11-strips (20) | **10** | 9 |
| snake-opt18-strips (20) | **12** | 0 |
| spider-opt18-strips (20) | **15** | 6 |
| storage (30) | **16** | 15 |
| tetris-opt14-strips (17) | **11** | 0 |
| tidybot-opt11-strips (20) | **14** | 13 |
| tidybot-opt14-strips (20) | **9** | 5 |
| trucks-strips (30) | 9 | **11** |
| woodworking-opt08-strips (30) | **16** | 14 |
| woodworking-opt11-strips (20) | **11** | 9 |
| Sum (694) | **349** | 247 |

**Table 4.6:** Shows a summary of only domains where we observed a difference in coverage between the h2-ipdb-uc and h2-ipdb-c configurations. Domains in which we had the same coverage for the two configurations have been omitted. A full summary can be found in Table A.1

configurations without the $h^2$, the unconstrained version h2-ipdb-uc could not find dead ends as opposed to h2-ipdb-c which managed to find the dead ends in most tasks. Furthermore, the constrained version had fewer expansions compared to the unconstrained version, which was further highlighted by a higher number of tasks failing due to memory issues in the unconstrained version. For three domains, h2-ipdb-c could not find a solution while h2-ipdb-uc managed to find solutions in these domains. One of these domains was snake-opt18-strips, in this domain h2-ipdb-c ran out of time during pattern database computation. This was also the observation in Table 4.2 where the unconstrained version performed better than the constrained versions.

Comparing the pdb-time of h2-ipdb-uc and h2-ipdb-c we noticed that the unconstrained variant h2-ipdb-uc, computed the pattern database faster in 93% of the tasks. The search time was about 1.49 times faster in the constrained variant, this is due to the fewer expansions by h2-ipdb-c since search time comprises of state expansions and heuristic computation, and the heuristic computation is equally expensive in both variants. Table 4.5 clearly shows this difference in expansions. However, this speedup in search time was not enough to lower the planner wall clock time of h2-ipdb-c since the pattern database generation time was significantly slower in h2-ipdb-c.

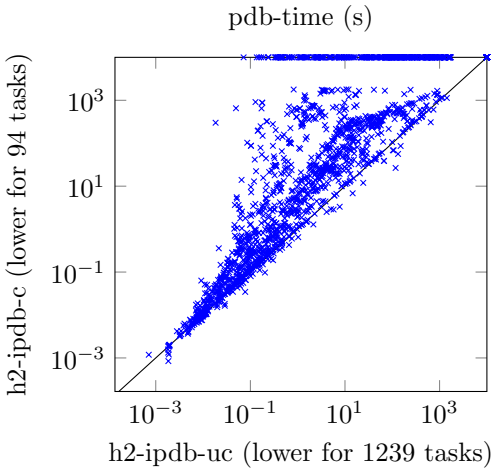


**Figure 4.10:** Comparison of pdb time

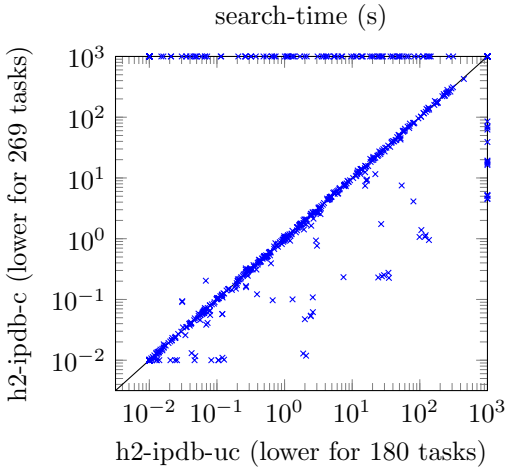


**Figure 4.11:** Comparison of search_time of h2-ipdb-uc and h2-ipdb-c

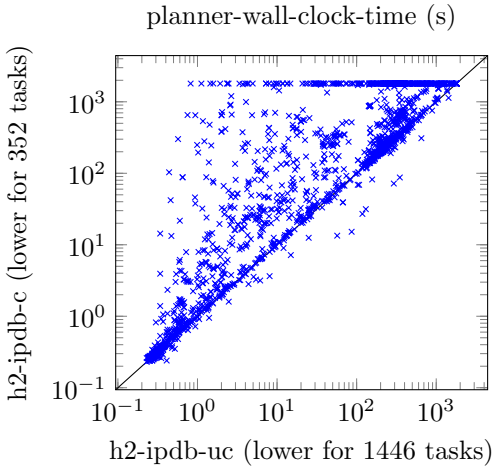


**Figure 4.12:** Comparison of planner_wall_clock_time of h2-ipdb-uc and h2-ipdb-c

Lastly, we will take a look at the performance of the pdb variant that used the $h^2$ preprocessor, our discussion will be based on the results shown in Table 4.7.

| Summary | h2-pdb-uc | h2-pdb-c |
|---|---|---|
| coverage - Sum | **839** | 818 |
| error-search-out-of-memory - Sum | 978 | **776** |
| error-search-out-of-time - Sum | **0** | 223 |
| expansions - Sum | 2'368'553'061 | **2'077'711'155** |

**Table 4.7:** Summary of coverage, errors and expansions of h2-pdb-uc and h2-pdb-c

Comparing the pdb and ipdb versions for $h^2$ we noticed that the unconstrained versions yielded the highest coverage in both the pdb and ipdb versions. Tasks that failed due to memory and time limitations also followed the same trend as in the ipdb variants. Across the

floortile domain we noticed the same trend as in the ipdb variants whereby the constrained version outperformed the unconstrained version. Furthermore, the unconstrained version had more tasks failing due to running out of memory, whilst the constrained version had more tasks failing to running out of time. Due to the way ipdbs work, the significant time difference between the pdb and ipdb models was to be expected as ipdbs tend to use more than just one pattern like pdbs do. In terms of expansions we also noticed a similar trend to the ipdb variant whereby the constrained version expanded overall fewer states than the unconstrained version. This also explains why the constrained version had fewer tasks failing due to running out of memory since it did not have as many states as the unconstrained version to expand.

# 5 Conclusion

The main objective of the thesis was to implement CPDBs and test the performance of the different configurations and see which one would work best. We tested the implementation on the more general pdb as well as ipdb to see how it differs in both settings. We also had one implementation which used the $h^2$ preprocessor and since it changes the task we compared the constrained and unconstrained versions of the $h^2$ separately from the rest of the other implementations.

From our findings, we can conclude that using the mutex groups that are already generated by Fast Downward during translation is sufficient, and this is evident from the fact that fd-ipdb-c was the version with the highest coverage of all the CPDB implementations we had. The results also show that it is almost never beneficial to use the fact-alternating mutex groups (fam-groups), this is because using mutex groups from Fast Downward we never get coverage that is worse than what we get from fam-groups while having better performance in pattern database generation time.

On the other hand of all the ten configurations h2-ipdb-uc got the best coverage. However, our implementation of CPDBs using mutex groups generated by the $h^2$ preprocessor (h2-ipdb-c) yielded lower coverage as compared to the CPDBs which used mutex groups from Fast Downward. Not only did we get lower coverage in h2-ipdb-c but we also had longer pattern database generation time as compared to fd-ipdb-c. In light of all this, we would say of all the CPDB implementations (inclusive of the ones from $h^2$) fd-ipdb-c would be the best implementation as it gets better coverage to planner_wall_clock_time ratio.

## 5.1 Future work

In most domains that we tested, we noticed that the pattern database generation time in the CPDBs was a bottleneck, although other attributes such as search time, expansions and memory usage were better there than their PDB counterparts. This means that future efforts can be made in an attempt to optimize the already existing implementation of PDBs. This is likely to reduce the overhead in time introduced by considering mutex information in the CPDBs, and that along with an efficient search would result in CPDBs solving more

tasks since we would had fewer tasks failing due to running out of time.

Secondly, figuring out a good criteria to know when to apply mutex constraints or allow flexibility would also be beneficial. We noticed that in some tasks although we had mutex information, it was not really beneficial to us to solve the tasks so this just introduced overhead in computation time without the benefits from the mutex information. To this end, we can try and look into operator applicability and check if each precondition variable is in the pattern or not, and if there are mutex groups with this variable. The idea is if a precondition variable is not in the pattern and there exist mutex groups including the variable then we should enforce mutex constraining.

In conclusion, we can also extend the concept of mutex constraining to other abstraction heuristics as like Merge and Shrink [16]. Since we also use the concept of abstraction in Merge and Shrink, we can also enforce mutex constraints to get a more informed heuristic which would also then improve our search and potentially the performance of the Merge and Shrink.

# Bibliography


[1] Vidal Alcázar and Álvaro Torralba. A reminder about the importance of computing and exploiting invariants in planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling*, pages 2–6, 2015.

[2] Christer Bäckström and Bernhard Nebel. Complexity results for $SAS^+$ planning. *Computational Intelligence*, pages 625–655, 1995.

[3] Joseph C. Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

[4] Stefan Edelkamp. Automated creation of pattern database search heuristics. In *International Workshop on Model Checking and Artificial Intelligence*, pages 35–50, 2006.

[5] Stefan Edelkamp. Planning with pattern databases. In *Proceedings of the 6th European Conference on Planning*, pages 84–90, 2014.

[6] Daniel Fišer and Antonín Komenda. Fact-alternating mutex groups for classical planning. *Journal of Artificial Intelligence Research*, 61:475–521, 2018.

[7] Naresh Gupta and Dana S. Nau. On the complexity of blocks-world planning. *Artificial Intelligence*, 56:223–254, 1992.

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

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

[10] Malte Helmert. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173(5):503–535, 2009.

[11] Malte Helmert, Patrik Haslum, and Jörg Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling*, pages 176–183, 2007.

[12] Robert C. Holte, Jack Newton, Ariel Felner, Ram Meshulam, and David Furcy. Multiple pattern databases. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling*, pages 122–131, 2004.

[13] Richard E Korf. Finding optimal solutions to rubik's cube using pattern databases. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 700–705, 1997.

[14] Florian Pommerening, Gabriele Röger, and Malte Helmert. Getting the most out of pattern databases for classical planning. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pages 2357–2364, 2013.

[15] Jendrik Seipp, Florian Pommerening, Silvan Sievers, and Malte Helmert. Downward Lab, 2017.

[16] Silvan Sievers and Malte Helmert. Merge-and-Shrink: A compositional theory of transformations of factored transition systems. *Journal of Artificial Intelligence Research*, 71:781–883, 2021.

# A

# Appendix

| Summary | fam-ipdb-c | fam-pdb-c | fd-ipdb-c | fd-pdb-c | h2-ipdb-c | h2-ipdb-uc | h2-pdb-c | h2-pdb-uc | ipdb-uc | pdb-uc |
|---|---|---|---|---|---|---|---|---|---|---|
| coverage - Sum | 866 | 788 | 927 | 809 | 888 | 990 | 818 | 839 | 946 | 801 |
| dead_ends - Sum | 1943081 | 44371498 | 1942937 | 44342532 | 538017 | 1024261 | 1042634 | 966869 | 3313006 | 32835023 |
| error-exitcode-232 - Sum | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 4 | 0 | 0 |
| error-exitcode--15 - Sum | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 |
| error-search-out-of-memory - Sum | 391 | 960 | 537 | 1010 | 439 | 654 | 776 | 978 | 651 | 1029 |
| error-search-out-of-time - Sum | 570 | 82 | 364 | 11 | 492 | 175 | 223 | 0 | 231 | 0 |
| error-search-unsolvable-incomplete - Sum | 7 | 4 | 6 | 4 | 11 | 11 | 11 | 11 | 6 | 4 |
| error-sigkill - Sum | 0 | 3 | 1 | 2 | 3 | 1 | 1 | 1 | 1 | 1 |
| error-success - Sum | 866 | 788 | 927 | 809 | 888 | 990 | 818 | 839 | 946 | 801 |
| error-translate-out-of-memory - Sum | 13 | 10 | 12 | 11 | 10 | 12 | 12 | 12 | 12 | 12 |
| expansions - Geometric mean | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| memory - Sum | 107446096 | 169488912 | 105742660 | 168710232 | 99992628 | 97333912 | 150825684 | 156606356 | 102120512 | 174811836 |
| pdb_time - Sum | 215314.47 | 7.81 | 69239.55 | 7.63 | 124008.61 | 25065.92 | 4.47 | 4.52 | 29343.52 | 7.53 |
| planner_wall_clock_time - Sum | 1398392.18 | 567552.98 | 920105.21 | 400017.05 | 1169372.02 | 616962.73 | 776868.18 | 284023.38 | 686679.96 | 274361.52 |
| search_time - Geometric mean | 0.07 | 0.25 | 0.07 | 0.25 | 0.07 | 0.08 | 0.20 | 0.22 | 0.08 | 0.28 |
| total_time - Geometric mean | 14.99 | 3.58 | 2.28 | 2.59 | 3.16 | 0.82 | 2.26 | 0.72 | 1.20 | 0.95 |

**Figure A.1:** Summary of all configurations

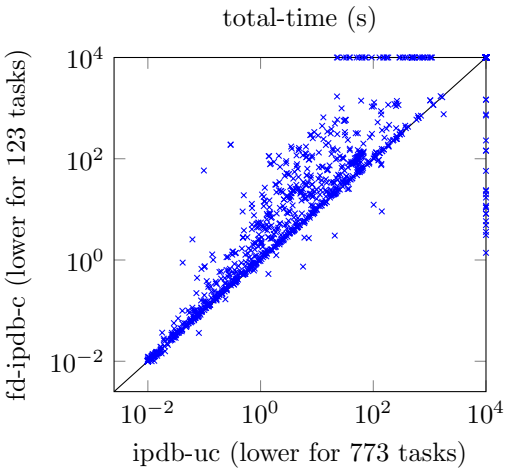


**Figure A.2:** Comparison of total_time of ipdb-uc and fd-ipdb-c

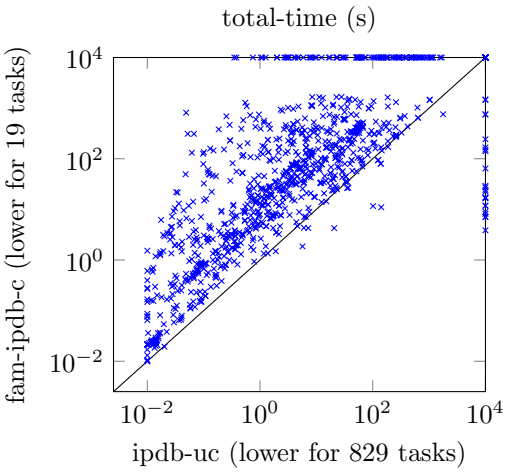


**Figure A.3:** Comparison of total_time of ipdb-uc and fam-ipdb-c

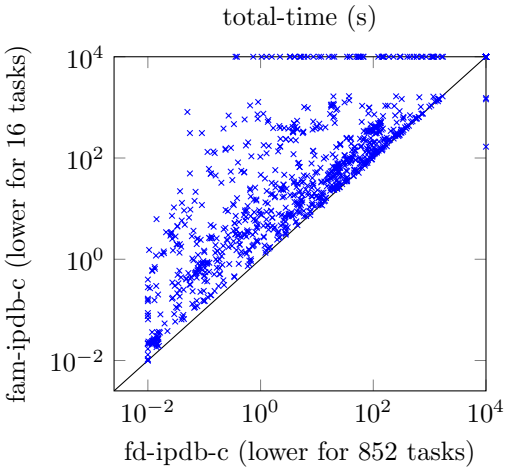


**Figure A.4:** Comparison of total_time of fd-ipdb-c and fam-ipdb-c

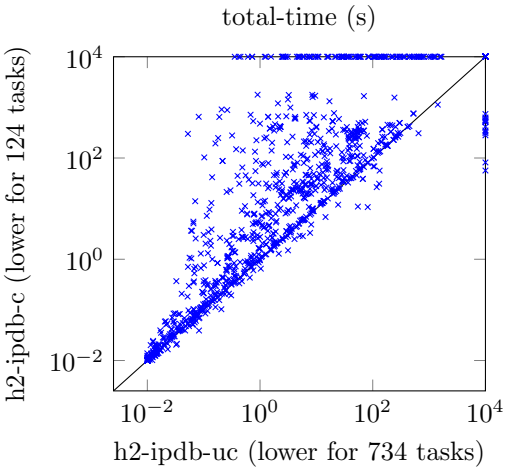


**Figure A.5:** Comparison of total_time of h2-ipdb-uc and h2-ipdb-c

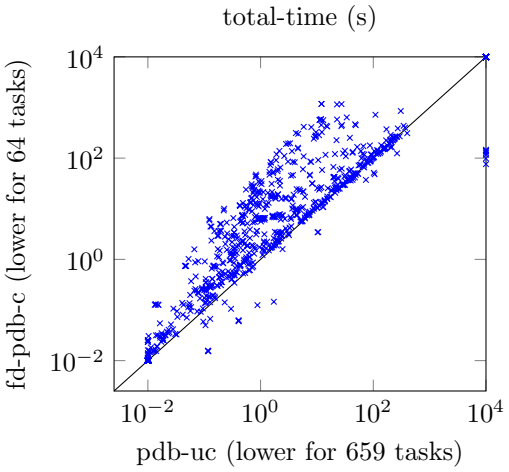


**Figure A.6:** Comparison of total_time of pdb-uc and fd-pdb-c

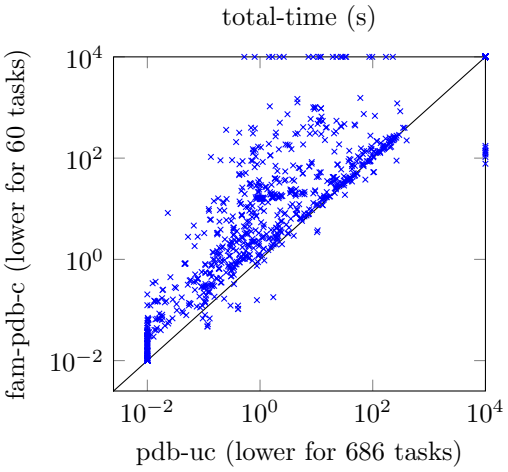


**Figure A.7:** Comparison of total_time of pdb-uc and fam-pdb-c

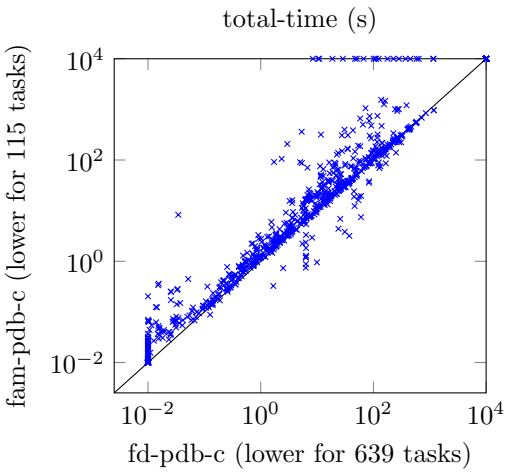


**Figure A.8:** Comparison of total_time of fd-pdb-c and fam-pdb-c

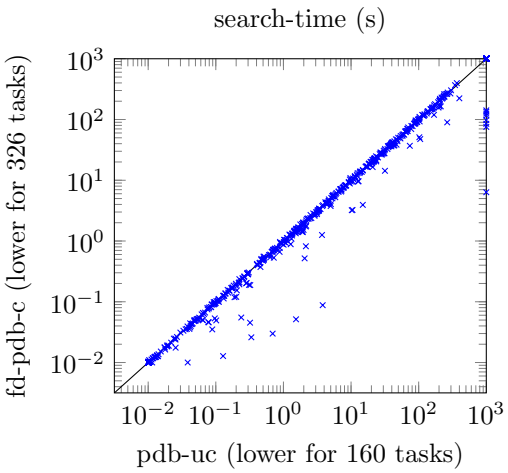


**Figure A.9:** Comparison of search_time of pdb-uc and fd-pdb-c

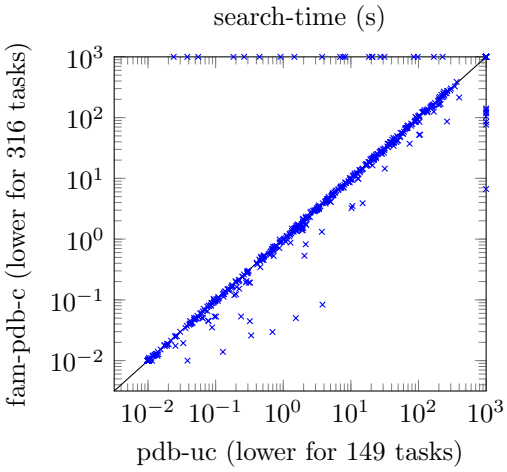


**Figure A.10:** Comparison of search_time of pdb-uc and fam-pdb-c

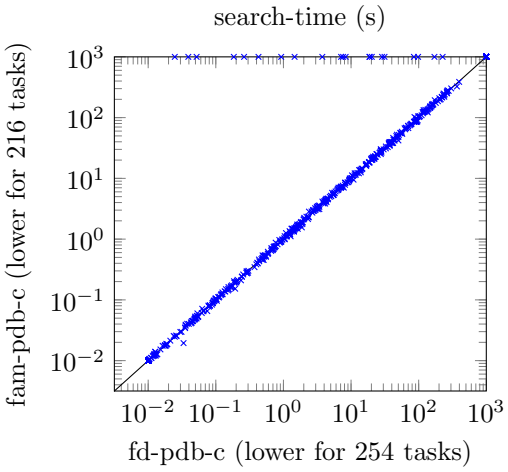


**Figure A.11:** Comparison of search_time of fd-pdb-c and fam-pdb-c

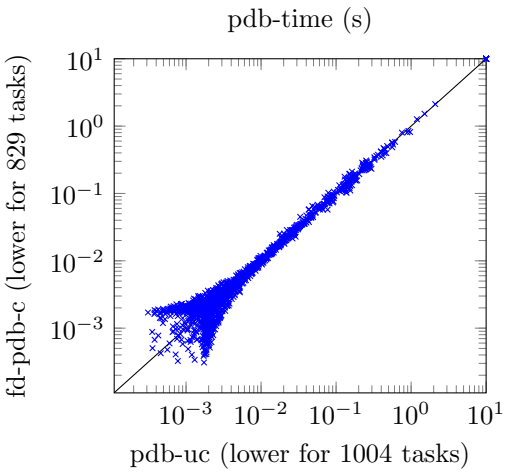


**Figure A.12:** Comparison of pdb_time of pdb-uc and fd-pdb-c

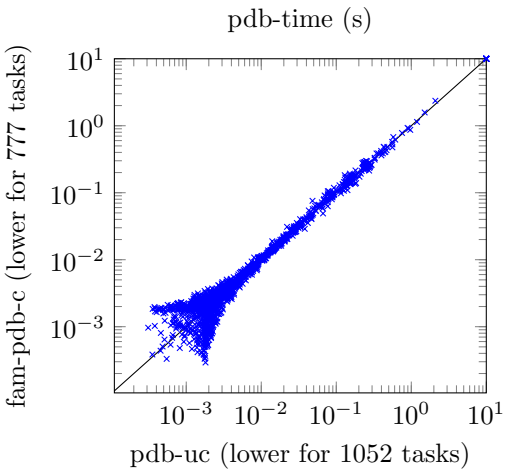


**Figure A.13:** Comparison of pdb_time of pdb-uc and fam-pdb-c

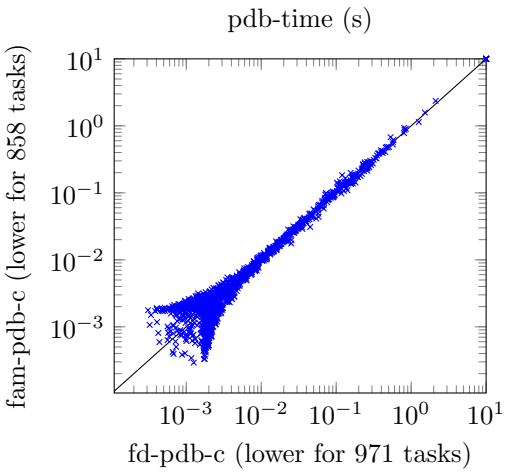


**Figure A.14:** Comparison of pdb_time of fd-pdb-uc and fam-pdb-c

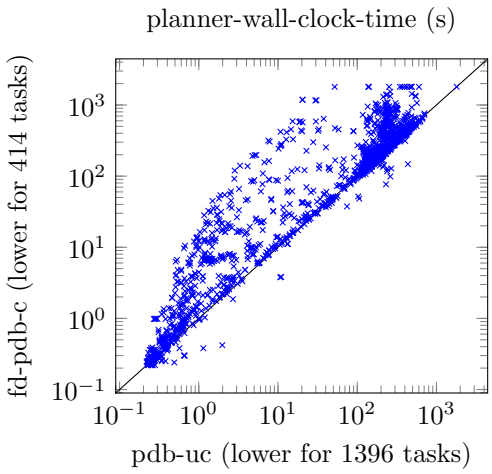


**Figure A.15:** Comparison of planner_wall_clock_time of pdb-uc and fd-pdb-c

| domain | fam-ipdb-c | fam-pdb-c | fd-ipdb-c | fd-pdb-c | h2-ipdb-c | h2-ipdb-uc | h2-pdb-c | h2-pdb-uc | ipdb-uc | pdb-uc |
|---|---|---|---|---|---|---|---|---|---|---|
| agricola-opt18-strips (20) | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 |
| airport (50) | 22 | 23 | 29 | 23 | 17 | 31 | 27 | 27 | 35 | 23 |
| barman-opt11-strips (20) | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| barman-opt14-strips (14) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| blocks (35) | 28 | 21 | 28 | 21 | 28 | 28 | 21 | 21 | 28 | 21 |
| childsnack-opt14-strips (20) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| data-network-opt18-strips (20) | 12 | 9 | 12 | 9 | 12 | 12 | 9 | 9 | 12 | 9 |
| depot (22) | 9 | 7 | 9 | 7 | 7 | 11 | 7 | 7 | 11 | 7 |
| driverlog (20) | 13 | 10 | 13 | 10 | 13 | 13 | 10 | 10 | 13 | 10 |
| elevators-opt08-strips (30) | 23 | 14 | 23 | 14 | 23 | 23 | 14 | 14 | 23 | 14 |
| elevators-opt11-strips (20) | 18 | 12 | 18 | 12 | 18 | 18 | 12 | 12 | 18 | 12 |
| floortile-opt11-strips (20) | 9 | 4 | 9 | 4 | 13 | 8 | 10 | 8 | 2 | 2 |
| floortile-opt14-strips (20) | 9 | 2 | 9 | 2 | 16 | 8 | 11 | 8 | 0 | 0 |
| freecell (80) | 12 | 20 | 12 | 20 | 8 | 21 | 20 | 20 | 21 | 20 |
| ged-opt14-strips (20) | 20 | 15 | 20 | 15 | 15 | 19 | 15 | 15 | 19 | 15 |
| grid (5) | 3 | 2 | 3 | 2 | 3 | 3 | 2 | 2 | 3 | 2 |
| gripper (20) | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| hiking-opt14-strips (20) | 13 | 14 | 13 | 14 | 13 | 13 | 14 | 14 | 13 | 14 |
| logistics00 (28) | 25 | 16 | 25 | 16 | 23 | 23 | 15 | 15 | 25 | 16 |
| logistics98 (35) | 6 | 3 | 6 | 3 | 6 | 6 | 4 | 4 | 6 | 3 |
| miconic (150) | 69 | 55 | 69 | 55 | 69 | 69 | 55 | 55 | 69 | 55 |
| movie (30) | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 |
| mprime (35) | 23 | 23 | 24 | 23 | 23 | 24 | 21 | 23 | 24 | 23 |
| mystery (30) | 14 | 17 | 17 | 17 | 13 | 17 | 14 | 17 | 17 | 17 |
| nomystery-opt11-strips (20) | 18 | 10 | 20 | 10 | 13 | 20 | 10 | 10 | 20 | 10 |
| openstacks-opt08-strips (30) | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 |
| openstacks-opt11-strips (20) | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 |
| openstacks-opt14-strips (20) | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| openstacks-strips (30) | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| organic-synthesis-opt18-strips (20) | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| organic-synthesis-split-opt18-strips (20) | 1 | 1 | 7 | 10 | 7 | 13 | 13 | 13 | 7 | 10 |
| parcprinter-08-strips (30) | 14 | 15 | 21 | 15 | 28 | 28 | 20 | 20 | 21 | 15 |
| parcprinter-opt11-strips (20) | 10 | 11 | 16 | 11 | 18 | 18 | 15 | 15 | 16 | 11 |
| parking-opt11-strips (20) | 4 | 0 | 7 | 0 | 7 | 7 | 1 | 1 | 7 | 0 |
| parking-opt14-strips (20) | 6 | 0 | 6 | 0 | 6 | 6 | 0 | 0 | 6 | 0 |
| pathways (30) | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| pegsol-08-strips (30) | 30 | 27 | 30 | 27 | 30 | 30 | 27 | 27 | 30 | 27 |
| pegsol-opt11-strips (20) | 20 | 17 | 20 | 17 | 20 | 20 | 17 | 17 | 20 | 17 |
| petri-net-alignment-opt18-strips (20) | 0 | 0 | 0 | 4 | 0 | 0 | 6 | 4 | 0 | 4 |
| pipesworld-notankage (50) | 15 | 22 | 20 | 22 | 13 | 21 | 22 | 23 | 21 | 22 |
| pipesworld-tankage (50) | 9 | 17 | 9 | 18 | 8 | 17 | 12 | 17 | 17 | 17 |
| psr-small (50) | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| quantum-layout-opt23-strips (20) | 8 | 10 | 13 | 10 | 13 | 13 | 10 | 10 | 13 | 10 |
| rovers (40) | 8 | 7 | 8 | 7 | 8 | 8 | 7 | 7 | 8 | 7 |
| satellite (36) | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| scanalyzer-08-strips (30) | 11 | 13 | 11 | 13 | 12 | 13 | 13 | 13 | 13 | 13 |
| scanalyzer-opt11-strips (20) | 8 | 10 | 8 | 10 | 9 | 10 | 10 | 10 | 10 | 10 |
| snake-opt18-strips (20) | 0 | 12 | 1 | 12 | 0 | 12 | 9 | 12 | 11 | 12 |
| sokoban-opt08-strips (30) | 30 | 27 | 30 | 27 | 30 | 30 | 30 | 30 | 30 | 27 |
| sokoban-opt11-strips (20) | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 |
| spider-opt18-strips (20) | 0 | 8 | 15 | 11 | 6 | 15 | 1 | 11 | 15 | 11 |
| storage (30) | 16 | 16 | 16 | 16 | 15 | 16 | 16 | 15 | 16 | 15 |
| termes-opt18-strips (20) | 13 | 12 | 13 | 12 | 13 | 13 | 12 | 12 | 13 | 12 |
| tetris-opt14-strips (17) | 0 | 6 | 0 | 10 | 0 | 11 | 11 | 11 | 1 | 10 |
| tidybot-opt11-strips (20) | 14 | 14 | 14 | 14 | 13 | 14 | 13 | 14 | 14 | 14 |
| tidybot-opt14-strips (20) | 9 | 9 | 9 | 9 | 5 | 9 | 5 | 9 | 9 | 9 |
| tpp (30) | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| transport-opt08-strips (30) | 14 | 11 | 14 | 11 | 14 | 14 | 11 | 11 | 14 | 11 |
| transport-opt11-strips (20) | 12 | 6 | 12 | 6 | 12 | 12 | 6 | 6 | 12 | 6 |
| transport-opt14-strips (20) | 9 | 7 | 9 | 7 | 9 | 9 | 7 | 7 | 9 | 7 |
| trucks-strips (30) | 11 | 8 | 11 | 8 | 11 | 9 | 8 | 8 | 9 | 8 |
| visitall-opt11-strips (20) | 16 | 9 | 16 | 9 | 16 | 16 | 9 | 9 | 16 | 9 |
| visitall-opt14-strips (20) | 12 | 4 | 12 | 4 | 12 | 12 | 4 | 4 | 12 | 4 |
| woodworking-opt08-strips (30) | 14 | 12 | 14 | 12 | 14 | 16 | 12 | 12 | 12 | 11 |
| woodworking-opt11-strips (20) | 9 | 7 | 9 | 7 | 9 | 11 | 7 | 7 | 8 | 6 |
| zenotravel (20) | 13 | 9 | 13 | 9 | 13 | 13 | 9 | 9 | 13 | 9 |
| Sum (1847) | 866 | 788 | 927 | 809 | 888 | **990** | 818 | 839 | 946 | 801 |

**Table A.1:** Coverage results for all tested domains in all the different configurations

| domain | fam-ipdb-c | fam-pdb-c | fd-ipdb-c | fd-pdb-c | h2-ipdb-c | h2-ipdb-uc | h2-pdb-c | h2-pdb-uc | ipdb-uc | pdb-uc |
|---|---|---|---|---|---|---|---|---|---|---|
| agricola-opt18-strips | 0 | 17 | 0 | 20 | 0 | 17 | 0 | 20 | 9 | 20 |
| airport | 0 | 27 | 1 | 27 | 0 | 15 | 10 | 23 | 2 | 27 |
| barman-opt11-strips | 15 | 16 | 15 | 16 | 7 | 15 | 16 | 16 | 15 | 16 |
| barman-opt14-strips | 7 | 14 | 7 | 14 | 2 | 7 | 14 | 14 | 7 | 14 |
| blocks | 7 | 14 | 7 | 14 | 7 | 7 | 14 | 14 | 7 | 14 |
| childsnack-opt14-strips | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 |
| data-network-opt18-strips | 6 | 11 | 7 | 11 | 7 | 8 | 11 | 11 | 8 | 11 |
| depot | 0 | 15 | 0 | 15 | 0 | 9 | 15 | 15 | 9 | 15 |
| driverlog | 2 | 10 | 4 | 10 | 4 | 5 | 10 | 10 | 6 | 10 |
| elevators-opt08-strips | 7 | 16 | 7 | 16 | 7 | 7 | 16 | 16 | 7 | 16 |
| elevators-opt11-strips | 2 | 8 | 2 | 8 | 2 | 2 | 8 | 8 | 2 | 8 |
| floortile-opt11-strips | 7 | 16 | 7 | 16 | 0 | 8 | 10 | 12 | 13 | 18 |
| floortile-opt14-strips | 11 | 18 | 11 | 18 | 0 | 12 | 9 | 12 | 18 | 20 |
| freecell | 24 | 60 | 24 | 60 | 0 | 59 | 22 | 60 | 59 | 60 |
| ged-opt14-strips | 0 | 5 | 0 | 5 | 0 | 1 | 5 | 5 | 1 | 5 |
| grid | 1 | 3 | 2 | 3 | 2 | 2 | 3 | 3 | 2 | 3 |
| gripper | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 |
| hiking-opt14-strips | 7 | 6 | 7 | 6 | 7 | 7 | 6 | 6 | 7 | 6 |
| logistics00 | 3 | 12 | 3 | 12 | 5 | 5 | 13 | 13 | 3 | 12 |
| logistics98 | 5 | 32 | 11 | 32 | 11 | 11 | 31 | 31 | 11 | 32 |
| miconic | 45 | 95 | 46 | 95 | 46 | 50 | 95 | 95 | 49 | 95 |
| movie | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| mprime | 5 | 12 | 11 | 12 | 11 | 11 | 7 | 12 | 11 | 12 |
| mystery | 3 | 8 | 7 | 9 | 1 | 2 | 0 | 2 | 7 | 9 |
| nomystery-opt11-strips | 0 | 10 | 0 | 10 | 0 | 0 | 10 | 10 | 0 | 10 |
| openstacks-opt08-strips | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| openstacks-opt11-strips | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| openstacks-opt14-strips | 13 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 |
| openstacks-strips | 6 | 21 | 7 | 23 | 7 | 7 | 21 | 23 | 7 | 23 |
| organic-synthesis-opt18-strips | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| organic-synthesis-split-opt18-strips | 0 | 0 | 7 | 8 | 1 | 1 | 0 | 3 | 7 | 10 |
| parcprinter-08-strips | 0 | 15 | 0 | 15 | 2 | 2 | 9 | 9 | 0 | 15 |
| parcprinter-opt11-strips | 0 | 9 | 0 | 9 | 2 | 2 | 4 | 4 | 0 | 9 |
| parking-opt11-strips | 2 | 20 | 13 | 20 | 8 | 13 | 15 | 19 | 13 | 20 |
| parking-opt14-strips | 2 | 20 | 14 | 20 | 10 | 14 | 16 | 20 | 14 | 20 |
| pathways | 26 | 26 | 26 | 26 | 26 | 26 | 26 | 26 | 26 | 26 |
| pegsol-08-strips | 0 | 3 | 0 | 3 | 0 | 0 | 3 | 3 | 0 | 3 |
| pegsol-opt11-strips | 0 | 3 | 0 | 3 | 0 | 0 | 3 | 3 | 0 | 3 |
| petri-net-alignment-opt18-strips | 0 | 0 | 20 | 16 | 20 | 20 | 14 | 16 | 20 | 16 |
| pipesworld-notankage | 12 | 27 | 28 | 28 | 20 | 29 | 11 | 27 | 28 | 28 |
| pipesworld-tankage | 1 | 30 | 2 | 31 | 0 | 19 | 7 | 33 | 21 | 33 |
| psr-small | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| quantum-layout-opt23-strips | 1 | 10 | 7 | 10 | 7 | 7 | 10 | 10 | 7 | 10 |
| rovers | 11 | 31 | 11 | 33 | 11 | 11 | 33 | 33 | 11 | 33 |
| satellite | 17 | 30 | 30 | 30 | 29 | 29 | 30 | 30 | 30 | 30 |
| scanalyzer-08-strips | 8 | 15 | 13 | 15 | 13 | 17 | 14 | 17 | 17 | 17 |
| scanalyzer-opt11-strips | 4 | 8 | 6 | 8 | 6 | 10 | 8 | 10 | 10 | 10 |
| snake-opt18-strips | 0 | 5 | 0 | 8 | 0 | 8 | 0 | 8 | 9 | 8 |
| sokoban-opt08-strips | 0 | 3 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 3 |
| sokoban-opt11-strips | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| spider-opt18-strips | 0 | 2 | 5 | 9 | 0 | 5 | 0 | 9 | 5 | 9 |
| storage | 2 | 12 | 2 | 14 | 0 | 2 | 3 | 15 | 2 | 15 |
| termes-opt18-strips | 7 | 8 | 7 | 8 | 7 | 7 | 8 | 8 | 7 | 8 |
| tetris-opt14-strips | 0 | 0 | 0 | 7 | 0 | 0 | 2 | 6 | 0 | 7 |
| tidybot-opt11-strips | 6 | 6 | 6 | 6 | 1 | 6 | 2 | 6 | 6 | 6 |
| tidybot-opt14-strips | 10 | 11 | 11 | 11 | 5 | 11 | 2 | 11 | 11 | 11 |
| tpp | 9 | 23 | 24 | 24 | 24 | 24 | 24 | 24 | 24 | 24 |
| transport-opt08-strips | 14 | 19 | 16 | 19 | 16 | 16 | 19 | 19 | 16 | 19 |
| transport-opt11-strips | 8 | 14 | 8 | 14 | 8 | 8 | 14 | 14 | 8 | 14 |
| transport-opt14-strips | 9 | 13 | 11 | 13 | 11 | 11 | 13 | 13 | 11 | 13 |
| trucks-strips | 11 | 22 | 12 | 18 | 12 | 14 | 11 | 22 | 14 | 22 |
| visitall-opt11-strips | 4 | 11 | 4 | 11 | 4 | 4 | 11 | 11 | 4 | 11 |
| visitall-opt14-strips | 8 | 16 | 8 | 16 | 8 | 8 | 16 | 16 | 8 | 16 |
| woodworking-opt08-strips | 0 | 18 | 0 | 18 | 1 | 1 | 18 | 18 | 0 | 19 |
| woodworking-opt11-strips | 0 | 13 | 0 | 13 | 1 | 1 | 13 | 13 | 0 | 14 |
| zenotravel | 0 | 11 | 0 | 11 | 0 | 1 | 11 | 11 | 2 | 11 |
| Sum | **391** | 960 | 537 | 1010 | 439 | 654 | 776 | 978 | 651 | 1029 |

**Table A.2:** Error search out of memory of all domains

| domain | fam-ipdb-c | fam-pdb-c | fd-ipdb-c | fd-pdb-c | h2-ipdb-c | h2-ipdb-uc | h2-pdb-c | h2-pdb-uc | ipdb-uc | pdb-uc |
|---|---|---|---|---|---|---|---|---|---|---|
| agricola-opt18-strips | 20 | 3 | 20 | 0 | 20 | 0 | 20 | 0 | 11 | 0 |
| airport | 28 | 0 | 20 | 0 | 33 | 4 | 13 | 0 | 13 | 0 |
| barman-opt11-strips | 1 | 0 | 1 | 0 | 9 | 1 | 0 | 0 | 1 | 0 |
| barman-opt14-strips | 7 | 0 | 7 | 0 | 12 | 7 | 0 | 0 | 7 | 0 |
| blocks | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| childsnack-opt14-strips | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| data-network-opt18-strips | 2 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| depot | 13 | 0 | 13 | 0 | 15 | 2 | 0 | 0 | 2 | 0 |
| driverlog | 5 | 0 | 3 | 0 | 3 | 2 | 0 | 0 | 1 | 0 |
| elevators-opt08-strips | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| elevators-opt11-strips | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| floortile-opt11-strips | 4 | 0 | 4 | 0 | 7 | 4 | 0 | 0 | 5 | 0 |
| floortile-opt14-strips | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 2 | 0 |
| freecell | 44 | 0 | 44 | 0 | 72 | 0 | 38 | 0 | 0 | 0 |
| ged-opt14-strips | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 |
| grid | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| gripper | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| hiking-opt14-strips | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| logistics00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| logistics98 | 24 | 0 | 18 | 0 | 18 | 18 | 0 | 0 | 18 | 0 |
| miconic | 36 | 0 | 35 | 0 | 35 | 31 | 0 | 0 | 32 | 0 |
| movie | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| mprime | 7 | 0 | 0 | 0 | 1 | 0 | 7 | 0 | 0 | 0 |
| mystery | 6 | 1 | 0 | 0 | 5 | 0 | 5 | 0 | 0 | 0 |
| nomystery-opt11-strips | 2 | 0 | 0 | 0 | 7 | 0 | 0 | 0 | 0 | 0 |
| openstacks-opt08-strips | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| openstacks-opt11-strips | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| openstacks-opt14-strips | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| openstacks-strips | 17 | 2 | 16 | 0 | 16 | 16 | 2 | 0 | 16 | 0 |
| organic-synthesis-opt18-strips | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| organic-synthesis-split-opt18-strips | 19 | 19 | 6 | 2 | 8 | 2 | 3 | 0 | 6 | 0 |
| parcprinter-08-strips | 16 | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 9 | 0 |
| parcprinter-opt11-strips | 10 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 4 | 0 |
| parking-opt11-strips | 14 | 0 | 0 | 0 | 5 | 0 | 4 | 0 | 0 | 0 |
| parking-opt14-strips | 12 | 0 | 0 | 0 | 4 | 0 | 4 | 0 | 0 | 0 |
| pathways | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| pegsol-08-strips | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| pegsol-opt11-strips | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| petri-net-alignment-opt18-strips | 20 | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| pipesworld-notankage | 23 | 1 | 2 | 0 | 17 | 0 | 17 | 0 | 1 | 0 |
| pipesworld-tankage | 40 | 3 | 39 | 1 | 42 | 14 | 31 | 0 | 12 | 0 |
| psr-small | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| quantum-layout-opt23-strips | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| rovers | 21 | 2 | 21 | 0 | 21 | 21 | 0 | 0 | 21 | 0 |
| satellite | 13 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| scanalyzer-08-strips | 11 | 2 | 6 | 2 | 5 | 0 | 3 | 0 | 0 | 0 |
| scanalyzer-opt11-strips | 8 | 2 | 6 | 2 | 5 | 0 | 2 | 0 | 0 | 0 |
| snake-opt18-strips | 20 | 3 | 19 | 0 | 20 | 0 | 11 | 0 | 0 | 0 |
| sokoban-opt08-strips | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| sokoban-opt11-strips | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| spider-opt18-strips | 20 | 10 | 0 | 0 | 14 | 0 | 19 | 0 | 0 | 0 |
| storage | 12 | 2 | 12 | 0 | 15 | 12 | 11 | 0 | 12 | 0 |
| termes-opt18-strips | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| tetris-opt14-strips | 17 | 11 | 17 | 0 | 17 | 6 | 4 | 0 | 16 | 0 |
| tidybot-opt11-strips | 0 | 0 | 0 | 0 | 6 | 0 | 5 | 0 | 0 | 0 |
| tidybot-opt14-strips | 1 | 0 | 0 | 0 | 10 | 0 | 13 | 0 | 0 | 0 |
| tpp | 15 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| transport-opt08-strips | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| transport-opt11-strips | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| transport-opt14-strips | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| trucks-strips | 8 | 0 | 7 | 4 | 7 | 7 | 11 | 0 | 7 | 0 |
| visitall-opt11-strips | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| visitall-opt14-strips | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| woodworking-opt08-strips | 16 | 0 | 16 | 0 | 15 | 13 | 0 | 0 | 18 | 0 |
| woodworking-opt11-strips | 11 | 0 | 11 | 0 | 10 | 8 | 0 | 0 | 12 | 0 |
| zenotravel | 7 | 0 | 7 | 0 | 7 | 6 | 0 | 0 | 5 | 0 |
| Sum | 570 | 82 | 364 | 11 | 492 | 175 | 223 | **0** | 231 | **0** |

**Table A.3:** Error search out of time for all domains

Universität Basel

Philosophisch-Naturwissenschaftliche Fakultät

## Erklärung zur wissenschaftlichen Redlichkeit und Veröffentlichung der Arbeit (beinhaltet Erklärung zu KI, Plagiat und Betrug)

Titel der Arbeit: CONSTRAINED PATTERN DATABASES

Name Beurteiler*in: Prof. MALTE HELMERT

Name Student*in: TAKUDZWA TOGAREPI

Matrikelnummer: 21-066-436

Ich bezeuge mit meiner Unterschrift, dass ich meine Arbeit selbständig ohne fremde Hilfe verfasst habe und meine Angaben über die bei der Abfassung meiner Arbeit benützten Quellen in jeder Hinsicht der Wahrheit entsprechen und vollständig sind. Alle Quellen, die wörtlich oder sinngemäss übernommen wurden, habe ich als solche gekennzeichnet.
Des Weiteren versichere ich, sämtliche Textpassagen, die unter Zuhilfenahme KI-gestützter Programme verfasst wurden, entsprechend gekennzeichnet sowie mit einem Hinweis auf das verwendete KI-gestützte Programm versehen zu haben.
Eine Überprüfung der Arbeit auf Plagiate und KI-gestützte Programme – unter Einsatz entsprechender Software – darf vorgenommen werden. Ich habe zur Kenntnis genommen, dass unlauteres Verhalten zu einer Bewertung der betroffenen Arbeit mit einer Note 1 oder mit «nicht bestanden» bzw. «fail» oder zum Ausschluss vom Studium führen kann.
Ich habe zur Kenntnis genommen, dass ich bei begründetem Verdacht auf eine unerlaubte oder nicht gekennzeichnete Anwendung von KI bei schriftlichen Leistungsüberprüfungen auf Einladung hin verpflichtet bin, an der Klärung des Verdachts mitzuwirken, z.B. durch Teilnahme an einem Gespräch.

Ort, Datum: BASEL, 28.03.25 Student*in: Togarepi

Wird diese Arbeit oder Teile davon veröffentlicht?

Nein ✓

Ja. Mit meiner Unterschrift bestätige ich, dass ich mit einer Veröffentlichung der Arbeit (print/digital) in der Bibliothek, auf der Forschungsdatenbank der Universität Basel und/oder auf dem Dokumentenserver des Departements / des Fachbereichs einverstanden bin. Ebenso bin ich mit dem bibliographischen Nachweis im Katalog SLSP (Swiss Library Service Platform) einverstanden. (nicht Zutreffendes streichen)

Veröffentlichung ab: _____

Ort, Datum: _____ Student*in: _____

Ort, Datum: _____ Beurteiler*in: _____

*Diese Erklärung ist in die Bachelor-, resp. Masterarbeit einzufügen.*

Februar 2025